Computer Security Symposium 2019
21 - 24 October 2019

Privacy Enhanced Slot Machine based on Smart Contract

PoChu, Hsu[1,†1,a),b)] Hideaki, Miyaji[1,c)] Atsuko, Miyaji[1,d)] ShihWei, Liao[†1,e)]

Abstract: Slot game is a popular gambling game. We want to build a smart contract protocol for slot game and preserve its financial fairness and the privacy of gaming parameters. In this research, we will focus on the most critical one, the winning probability.
Blockchain is a distributed system with no privacy. Traditionally, people will use trusted third party such as Intel SGX explained in Town Crier to keep critical parameters privately. However, in most cases, the correctness of the gaming result cannot be verified by smart contract. Zero knowledge proof based protocol such as Hawk might be able to solve this problem, but it takes more time to generate proof and it contains possibly not secure initial key generation phase.
In this research, without using trusted third party and zero knowledge proof, we use commitment scheme to keep the winning probability privately and maintain its verifiability by smart contract.

Keywords: Blockchain, Commitment, Smart Contract

1. Introduction

Slot is a popular gambling game over the world. In casino, there are many slot machines with different hidden gaming parameters. Among these gaming parameters, players care most about the winning probability. Experienced player will try to find the machine with highest winning probability.

In order to build an online slot game, which is intrinsically an online gambling service, it is necessary to integrate a reliable monetary system such as Bitcoin [3]. Bitcoin is a decentralized monetary system with limited scripting language. In blockchain, every transactions are broadcasted to full nodes for validation. The consensus algorithm, proof of work, ensures the correctness of the validation process.

Ethereum [4] provides a more sophisticated scripting functionality called smart contract. It contains an account belongs to the contract and can use turing complete programming language such as solidity to create control flow to manipulate the account. This design makes smart contract more suitable to build games. Except the turing complete property, it also provides a clock, which can be used to set timeout for preventing participants abort the game.

As a gambling game, conflict of interest exists between player and manager. Not only the bet by player or the stack deposited by manager, but also the transaction fee paid by manager, everything cost money. Therefore, the design of stack is necessary to preserve the financial fairness. In this research, we used some cryptographic techniques to minimize the stack. Makes manager don't need to store full stack for all players at the same time.

The design of blockchain allows every participants to verify its correctness. Therefore, every transaction are in plain text. In order to preserve privacy, ZCash [1] used non-interactive zero knowledge proof (NIZK) to provide privacy to the transaction. However, currently, they didn't provide smart contract functionality. In this research, we use probabilistic additive homomorphic encryption to preserve the privacy of winning probability and proof the number of winning tokens are same between each rounds.

Besides the privacy issue, malicious manager and player are also able to harm the protocol by abort the game or violate the protocol. For example, in the scratch-off ticket case, manager can pretend as a player and buy all winning ticket. In the slot game case, manager will also trying to create the illusion of high winning probability to attract players to join the game.

Traditionally, people use trusted third party to keep secretes. Town Crier [5] provides an example of financial derivative game. Showing how to use Intel SGX to preserve user credentials such as user name and password from knowing by others. Compared to Town Crier, our protocol doesn't use any trusted hardware such as Intel SGX and includes more sophisticate design to defense malicious manager.

An other approach is zero knowledge proof. Hawk [2] use

In this research, we proposed a smart contract protocol

[1] Osaka University
[†1] Presently with National Taiwan University
[a)] hsu@cy2sec.comm.eng.osaka-u.ac.jp
[b)] r05922177@ntu.edu.tw
[c)] hideaki@cy2sec.comm.eng.osaka-u.ac.jp
[d)] miyaji@comm.eng.osaka-u.ac.jp
[e)] liao@csie.ntu.edu.tw

to hide the winning probability of slot machine from player. We will give a brief introduction on common gambling games and blockchain on section 2, comparing our relative works in section 3.

2. Background

2.1 Common Gambling Games

2.1.1 Lotto

Lotto is a game based on probability. It supports unlimited players to join the game. Every players can select $n \in \mathbb{N}$ numbers from $m \in \mathbb{N}, m > n$ numbers. The lotto institution will issue tickets contains the selected n numbers as a certificate to prove the player chose these numbers before the winning numbers revealed.

In each round, every players can buy as many tickets as they want. At the end, the institution will use a fair machine to generate n winning numbers. This machine is designed for generating uniformly distributed numbers. Tickets which includes n numbers identical to the winning number can get the largest reward. Tickets which includes less than n identical numbers can also earn smaller reward.

The constant n and m are part of the rules, which makes the winning probability and the expectation value public to every player.

2.1.2 Scratch-off Lottery Ticket

Scratch-off lottery ticket is different from lotto. Usually, the probability of winning the game and the expectation value of each ticket are public by the lottery institution.

On top of the ticket, there is a special coating to prevent players from knowing the result before they buy the ticket. After buying the ticket, players can use coins or other tools to scratch this coatings off to know if he wins the price or not. This is the reason why this type of lottery ticket called scratch-off lottery ticket. This property makes some player feels more convenient.

Different from the lotto game, players doesn't need to wait the reveal of the winning number. They can know the result immediately after they buying the ticket. However, comparing to lotto, it is more boring since the problem is simplified from which lucky number you love to which ticket you want.

2.1.3 Slot Machine

Slot machine is a common gambling machine in casino. Usually, there are three identical rollers in front of the machine. On each roller, there are several symbols. When the game starts, three rollers will start to roll in a very fast but not identical rolling speed. Players can stop the rollers at anytime by pulling the handle. Usually, the rollers will slow down one by one. If the symbols on these three rollers are identical in one line, players can get the reward. The amount of reward is depends on what symbol you makes into one line. However, the probability to win the game is not totally depends on the player's behavior.

In order to make profit, there are several hidden parameters in each slot machine. Casino manager can change these parameters to change the winning probability. Usually, these parameters are fixed during the day. Machines with high probability make players winning money more easily, and vice versa. The key point for players to maximize their income is to find a machine with high winning probability. Experienced players will try to find a slot machine with high winning probability by their experience based on the outcome of some test rounds or by the statistic board on top of each machine in some casino.

Table 1 Winning probability of common gambling games

	Beforehand	Halfway	Afterward
Lotto	Public	Public	Public
Scratch-off ticket	Public	Public	Public
Slot game	Private	Private	Private

2.1.4 Comparison

Table 1 shows the comparison of winning probability between different types of common gambling game. The winning probability of lotto and scratch-off ticket are always public. The winning probability of slot game is always private.

2.2 Blockchain

2.2.1 Bitcoin

Bitcoin [3] is the first blockchain based cryptocurrency. It introduced the concept of Proof of Work (PoW), and the details of how to use PoW to reach consensus and secured cryptocurrency system. It provides primitive scripting language, but mainly used for access control.

2.2.2 Ethereum

Ethereum is similar to bitcoin, but its design philosophy is different. Instead of building a secure cryptocurrency system, the Ethereum development team wants to use Blockchain technology to build a decentralized deterministic state machine, in other words a world computer.

Due to the difference of design philosophy, the meaning of transaction is different. In Bitcoin, the purpose of transaction is transferring cryptocurrencies from one account to another account. However, in Ethereum, the purpose of transaction is state transition.

2.2.3 Smart Contract

Smart contract is the program stored in blockchain such as Ethereum. It is similar to stored scripts in database. With smart contracts, we can write programs which will be executed and verified by all full nodes in the blockchain network. When we issue a function call to smart contract, every computation and money transfer will be included in the same transaction. This atomic property make smart contract able to prevent various attacks.

2.2.4 Transaction Fee (Gas)

Even though Ethereum is a world computer, people needs to pay for the computation. Different from stored scripts in database, smart contracts can only triggered by someone who can afford to the computation. Transaction fee is the necessary fee to execute smart contract or money transfer. The amount of fee depends on the computation and storage usages.

2.3 Trusted Third Party

Trusted third party (TTP) is the party which cab be trusted by everyone. There are different type of trust. For example, the trust of not disclose any secret, the trust of always execute some program correctly or the trust of not abscond with money.

2.4 Trusted Execution Environment

Trusted execution environment (TEE) is the execution environment which can be trusted by everyone. Normal user space programs can be easy affected by other programs with higher privilege such as operating system. For example, memory, disk, network, everything not fully controlled by the program is vulnerable. TEE is the environment that the program will not be affected by any other programs.

2.5 Commitment Scheme

Commitment scheme is a common way to proof some secret not changed before revealing. There are two phase of commitment scheme, Commit phase and reveal phase. In the commit phase, user will use one way function such as hash function to create a digest from a secret. The digest can be shared to any party. After the commit phase, user will reveal the secret. This is so called reveal phase. Commitment scheme can be used in various places, for example, rock-paper-scissors. Assume there are two player, Alice and Bob. In the commit phase, Alice and Bob share the digest of their choice to each other, rock, paper or scissors. Then, in reveal phase, they share their real choice to each other. Both of them can verify the justness of the game by examining if each others choice equals to their previous digest or not.

2.6 Zero Knowledge Proof

Zero knowledge proof is similar to commitment scheme, but verifier can verify it without revealing secrete. For example, with public g, Alice can prove to Bob that she knows $y = g^x$ without revealing x.

Generally speaking, a practical general purposed zero knowledge proof can prove a NP statement C be True, with public input in and a secret witness w. For example, Alice can prove to Bob that she knows $w = x = 3$ makes $in = (g, y) = (2, 8)$ such that $C(w, in) = C(x, (g, y)) = (y == g^x)$.

3. Relative Works

3.1 Town Crier

Town crier [5] is a system which use Intel Software Guard Extensions (SGX), a trusted execution environment (TEE), to build a trusted third party (TTP). The main purpose of Town Crier is to provide secured data feeds from HTTPS enabled websites to smart contract, solving the problem that smart contract cannot access HTTPS enabled websites.

There are two main parts in the Town Crier system, Town Crier smart contract and Town Crier server. In the server side, the main program is running in a TEE called enclave created by Intel SGX. Everyone can ask the Intel attesting service if the program is really running in a valid Intel SGX enclave.

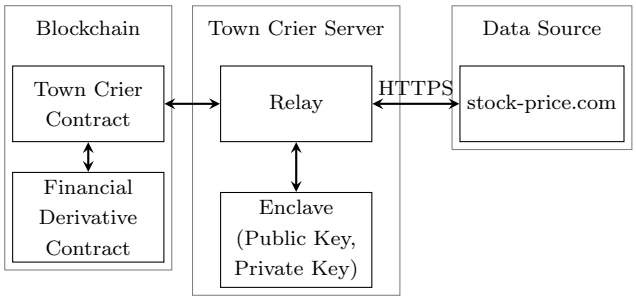

Fig. 1 Town crier system structure

Town Crier can not only fetch normal websites, it can also fetch private data from websites which needs user credentials. For example, user can use Town Crier server's public key to encrypt their user name and password. After Town Crier fetch the request from smart contract, it can decrypt the user name and password within the enclave, then encrypt the user name and password by HTTPS and login to the website.

Practically, it provides an example of how to use Town Crier to create a gambling contract based on HTTPS enabled website's data feeds as figure 1. First, the gambling contract sends a request to Town Crier contract for a specific stock price. Second, programs in Town Crier server enclave fetch the request. Third, the program fetch stock price from website. Forth, the program sends the result back to Town Crier contract and calls the gambling contract.

Even though the design of Town Crier is remarkable since it can hide private data in the enclave, but its security is based on the trust to Intel hardware, not based on the trust to cryptography.

3.2 Hawk

Hawk [2] is a smart contract protocol to provide privacy on smart contract. There are one manager and several players. It assume manager will never disclose any information submitted by player, in other words manager is a TTP.

Different from Town Crier, Hawk use non-interactive zero knowledge proof (NIZK) as its cryptographic primitive. Both player and manager can to use NIZK to generate proof for smart contract validation without revealing their private data.

It provides an example of how to use this protocol to build a rock-paper-scissors game. The work flow is showed in figure 2. First, player 1 and player 2 choose their input rock, paper or scissor respectively and encrypt it by manager's public key. Then compile the whole operation then into NIZK proof π_1 and π_2 and submit to smart contract for validation. Second, manager calculate the result based on players' input and submit it with the NIZK proof π_3 of the whole calculation to smart contract. Therefore, smart contract can validate the whole process by π_1, π_2 and π_3 without knowing players' choice.

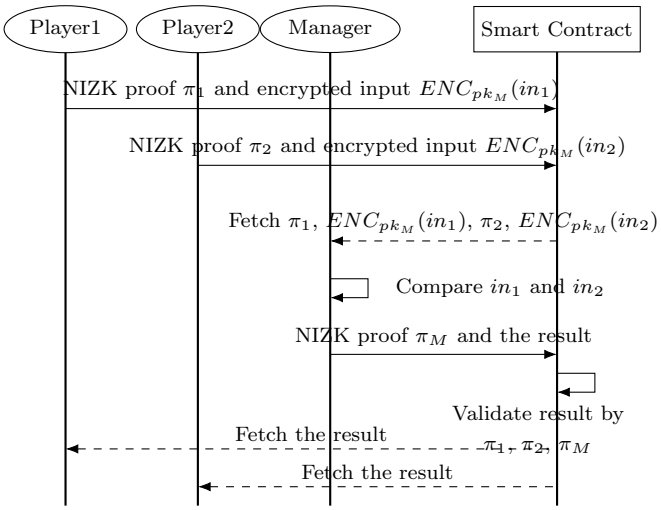

Fig. 2 Hawk protocol

The design of Hawk protocol seems perfect, but still based on a strong assumption that manager will not disclose any player's information. In other words, manager will not collude with any other player or even pretend as a player by himself.

4. Difficulties

4.1 Financial Fairness

In a gambling game, players need to pay money to participate it. Not only player pays money, both player and manager needs to pay transaction fee to send messages to smart contract. If either player or manager violates the protocol or aborts the game, they need to be punished financially.

In the design of Ethereum smart contract, the smart contract itself includes an account. To ensure the financial fairness, before the game starts, player and manager are required to deposit money in to the smart contract account. The deposit amount is larger than or equal to another party's maximum loss.

Ethereum smart contract also includes an incremental counter. Based on the timer, if any player or manager didn't respond within a given time window, they will be financially punished. The deposits stored in the smart contract will be transferred to others as compensation.

4.2 Preserving Privacy

As described in section 2.2, a Blockchain system with smart contract can ensures the output of the game binds with monetary transfer in a decentralized manner. This procedure will be verified by all full nodes over the Blockchain network. However, spread information to all full node means there are no secrets on Blockchain. One way to preserve privacy is to limit the information we put into the smart contract. However, limited information makes full nodes hard to understand the work flow and cannot verify it for us.

To protect the confidentiality of critical information, traditional solution is TTP. TTP promise the correctness of the execution results and ensures that they will not disclose critical information to others.

4.2.1 TTP Disadvantage : Correctness of Execution Result

Even though zero knowledge proof can ensure the verifiability by smart contract, the usage of stack is still necessary. To encourage TTP generate correct results, one common way is to ask them to lock some stacks in smart contract. The usage of stack can not only make up loss to their clients, but also punish these third party for giving incorrect results. However, excessive stack might actually reduce the efficiency of the services provided by trusted third party since they only have limited amount of money.

4.2.2 TTP Disadvantage : Information Leakage

Information leakage is critical especially TTP can make profit from it. For example, a gambling game includes many participants with conflicting of interest. If TTP leak any critical information to any player, the fairness of the game can broke easily. Even though zero knowledge proof can ensure the correctness of execution result, it still cannot ensure that TTP will not leak information to others.

4.3 Transaction Fee (Gas)

As we mentioned in section 2, in order to maintain the efficiency of the Blockchain system and prevent some malicious contracts from occupying the computation resources, the gas on Ethereum smart contract is determined by the amount of computation resources used by the transaction. Thus, reducing gas usage is an important issue especially when the protocol includes complicated cryptographic techniques.

4.4 Security Issues

Even though blockchain and smart contract already provides reliable access control and protect the protocol from many common security issues such as replay attacks. There are still several security issues need to take care about.

4.4.1 Security Issues Caused by Manager

4.4.1.1 Manager pretends as a player

Manager knows all secrets. In the scratch-off lottery ticket case, he can pretend himself as one of the players to join the game and buy all winning tickets. This attack is hard to identify and is one of the most critical problem needs to be solved.

4.4.1.2 Manager changes gaming parameters

Manager is the only person who knows secret gaming parameter, for example, the winning probability. In order to attract players to join the game, manager might set high winning probability at first. After several rounds, manager then lower the probability to make profit.

4.4.1.3 Manager aborts the game

In most cases, manager make profit by operating the smart contract, providing service to players. However, in some cases, if manager predict that he will lose money, he might want to abort the game temporarily.

4.4.2 Security Issues Caused by Player

Different from manager, player cannot play so many tricks. However, player can still aborts the game if he pre-

dict that he will lose money.

5. Our Protocol

5.1 Overview

In this section, we will explain the simplified slot game model, the roles in the protocol, the smart contracts which provide services, the tokens used by the smart contract, the life cycle of the game and two different design philosophies.

5.1.1 Slot game

As explained in section 2.1.3, slot game is a common gambling game in casino. To simplify the game and make the rule more clear, we made following assumptions.

- The winning probability of each slot machine will not changed during the same day.
- Only the manager can change the winning probability before the start of each day.

5.1.2 Life cycle

In order to better describe the life cycle, we need to define the game day.

Game Day Game day is a time period. In the same game day, the probability between each round of games will not be changed.

In each game day, the basic life cycle is as follows.

(1) Game day start.
(2) Manager set probability and do necessary operation.
(3) Manager start the game.
(4) Players join the game.
(5) Manager close the game.
(6) Game day ends.

5.1.3 Roles

Manager Manager is the person who own the smart contract. He needs to decide the winning probability of the slot machine smart contract and keep the probability privately until it is necessary to reveal.

Manager2 Manager2 is the person who do the permutation of encrypted tokens. We assume he will not share any information with Manager.

Player Player is the person who want to play the game. He needs to choose which lucky number he wants and communicate with the manager through smart contract.

Slot machine smart contract Slot machine contract is the contract which manager provide the service. As described in section 2.2.3, smart contract can be treated as a TTP with no privacy. Smart contract is a fixed program which control stacks and verify if participants follows the protocol. It is also the platform which players and manager communicate with each others.

Financial manager smart contract Financial manager contract is the contract which handles financial related affairs. It provides three main functionality.

(1) Issues tokens for the operation of the slot machine contract.
(2) Exchange token and ether used by the slot machine contract.
(3) A shuffling service for the players who want to preserve their privacy.

5.1.4 Tokens

Ether, a currency created by ethereum team, is the token designed for the reward of mining new blocks. It is created by the beginning of the ethereum blockchain and can not generated as our wish. In order to create more financial flexibility and follow the convention of real world casino. Instead of using ethers to play the game, issuing new tokens is necessary. Financial manager smart contract is the smart contract which provides this functionality.

5.1.5 Different design philosophies

In real world, the winning probability is always hidden from players. In order to maintain the fairness of the game, there are some international gambling institutions which certifies the fairness of the machine, and do spot checks periodically.

There are two different design philosophies depends on the needs of manager.

Disclose private parameter In this design, the winning probability will eventually be disclosed to the public. Every player wants to know the real winning probability especially after they lose the game. If a manager sets an extremely low winning probability, there might be no players to join the game anymore. A good manager will set winning probabilities in a reasonable range to maintain the relationship with players. After the winning probability be disclosed to the public, every player can verify if the probability is really used in the past games. If the manager lies to the players, he will be punished by giving his stacks to players as compensation.

Never disclose private parameter In this design, the winning probability will never be disclosed to the public. Even though disclose winning probability might create better user experience, some managers might still want to keep the winning probability privately to protect their interests. In this case, it needs more complicated ways to ensure the hidden winning probability is identical between each rounds. This might cause more gas since it needs more computation resources.

In the following section, we proposed two scheme. Scheme A adopts "disclose private parameter" design philosophy and scheme B adopts "never disclose private parameter" design philosophy.

5.2 Features

- Manager can refuse to accept some player's enrollment request.
- The protocol doesn't use trusted third party (TTP).
- The protocol doesn't use trusted execution environment (TEE).
- The protocol doesn't use zero knowledge proof.
- The historical data, such as winning probabilities, of each smart contract is public.

5.2.1 Financial fairness

- Both players and manager needs to deposit stacks into smart contract to ensure financial fairness.

5.2.2 Preserving privacy

- The privacy of the winning probability is protected by the additive homomorphic encryption.
- Depends on if the manager wants to decrypt all tokens or not, it can achieve both "disclose private parameter" and "never disclose private parameter" design philosophy.
- Before the acceptance of an enrollment, manager cannot know the content of the request.

5.2.3 Security

- Manager cannot increase the statistic value of the winning probability by pretending as a player.
- Manager cannot get take advantage by pretending as a player.
- Smart contract can verify if manager changes the winning probability or not in each round.
- If necessary, smart contract can compensate player in each round.
- Both manager and player cannot take advantage by aborting the game.

5.3 Assumptions

5.3.0.1 Cryptographic hash function assumption:

There is a one-way function, which is practically unfeasible to invert and can keep strong collision resistance.

5.3.0.2 Assumption of digital signature:

There exists a secure digital signature algorithm, which can deterministically generate signature from each messages.

5.3.0.3 Assumption of probabilistic additive homomorphic encryption:

There exists a secure probabilistic additive homomorphic encryption algorithm. Such that $\forall a, b \in \mathbb{F}_p, E(a) + E(b) \equiv E(a+b)$ and $E(a) \neq E(b)$ if $a = b$, where $\mathbb{F}_p$ is a finite field.

5.3.0.4 Assumption of secure permutation

There exists a secure permutation algorithm.

5.4 Life cycle of each game day

We assume the smart contract is already deployed on blockchain before initialization, and will not be destroyed by the end of the life cycle.

(1) Initialization phase
(2) Game phase (repeat multiple times)
  (a) Initialization phase
  (b) Enrollment phase
  (c) Reveal phase
(3) Reveal phase (optional)

5.5 Notations

5.5.1 Cryptographic functions

- $H()$: cryptographic hash function.
- $Sign()$: deterministic digital signature function.
- pk_M, sk_M : manager's public and secret key pair used for signature.
- $E()$: probabilistic additive homomorphic encryption function.
- $Perm()$: secure permutation algorithm.

5.5.2 Game parameters

- N : number of tokens in each round.
- $M \leq N$: number of winning tokens in each round.
- T_i : $T_i = 1$ if it is a winning token, $T_i = 0$ if it is a losing token, where i is the ith token.
- f : fine.
- Z : $Z \leq f \times$ (number of rounds in game phase) stack deposited by manager when game day starts.
- d_{min}, d_{max} : the range of bet.
- d : $d_{min} \leq d \leq d_{max}$ amount of money player want to bet.
- r : odds. Player get $r \times d$ if he wins.
- t_w : time window to accept enrollment.
- $h_G = H(N, f, d_{min}, d_{max}, r, t_w)$.

5.6 The protocol

5.6.1 Initialization phase

(1) Manager initialize smart contract with pk_M.
(2) Manager decide $N, M, Z, f, d_{min}, d_{max}, r, t_w$ and calculate h_G.
(3) Manager deposit stack Z and use game parameter $N, f, d_{min}, d_{max}, r, t_w, h_G$ to initialize the game.

5.6.2 Game phase

5.6.2.1 Initialization phase

(1) Manager generate N tokens $T_1, T_2, ..., T_N$ and encrypt them as $E(T_1), E(T_2), ..., E(T_N)$. There are exactly M winning token and $N - M$ losing tokens.
(2) Manager submit $E(T_1), E(T_2), ..., E(T_N)$ and $ZKP\left[\sum_{i=1}^{N} T_i = \sum_{i=1}^{N} \widehat{T}_i \middle| E\left(\sum_{i=1}^{N} T_i\right), E\left(\sum_{i=1}^{N} \widehat{T}_i\right)\right]$ to smart contract.
(3) Smart contract verify the $E(\sum_{i=1}^{N} T_i)$ equals to $E(\sum_{i=1}^{N} \widehat{T}_i)$

5.6.2.2 Enrollment phase

(1) Player fetch game parameter $N, f, d_{min}, d_{max}, r, t_w, h_G$ from smart contract to check if he wants to join or not.
(2) Player fetch all tokens $E(T_1), E(T_2), ..., E(T_N)$ from smart contract.
(3) Player choose lucky number s.
(4) Player sends enrollment request $Comm(s), d$ to smart contract.
(5) If manager doesn't have enough stack $Z \leq f$, smart contract will not accept the request.
(6) Manager fetch the request and accept enrollment.
(7) Smart contract verify the validity and lock $d \times r$, the maximum reward for player from stack Z.

5.6.2.3 Reveal phase

(1) Player reveal s to smart contract.
(2) Manager fetch s from smart contract, reply $s' = Sign_{pk_M}(H(R + s))$ and send $T_{(s' \bmod N)+1}$ to smart contract.
(3) Smart contract verify $T_{(s' \bmod N)+1}$ by $E(T_{(s' \bmod N)+1})$.
(4) If it is a winning token, smart contract sends $(d \times r)$ to player. Otherwise, smart contract unlock manager's stack $(d \times r)$.

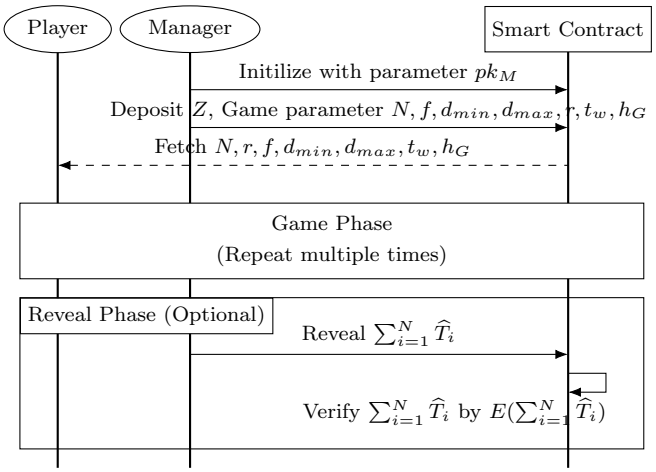

Fig. 3 Game day

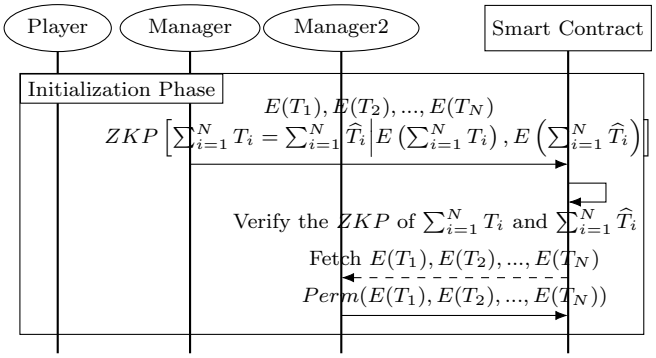

Fig. 4 Initialization phase of game phase

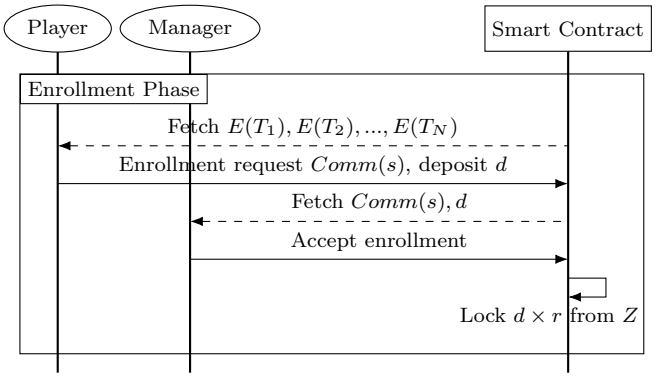

Fig. 5 Enrollment phase of game phase

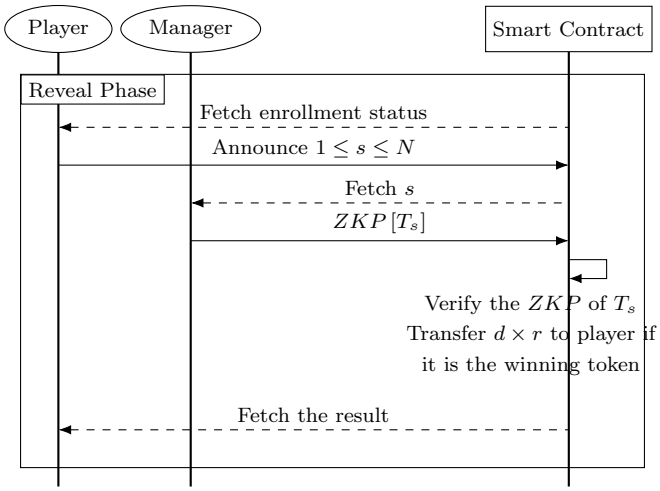

Fig. 6 Reveal phase of game phase

5.6.3 Reveal phase (optional)

The reveal phase is optional. Manager can decide if he wants to reveal true winning probability or not by himself.

(1) Announce $\sum_{i=1}^{N} \widehat{T}_i$ by revealing $E(\sum_{i=1}^{N} \widehat{T}_i)$

(2) Smart Contract checks if it is a valid proof.

(3) If manager lies, he will be punished by $f \times$ (number of rounds).

6. Discussion

6.1 Financial fairness

The design of timeout t_w, stack Z and fine f are used for prevent both manager and player aborts the game or violate the protocol. Since smart contract can verify all behavior of manager in each round, the stack pool Z doesn't need to contain large amount of money. It only needs to contain

$d_{max} \times r \times n$ for manager to maintain the operation of at least n rounds.

6.2 Preserving privacy

The usage of probabilistic additive homomorphic encryption algorithm makes it possible to proof the sum of all tokens are same between each rounds. Such that $\sum_{i=1}^{N} E(T_i) \equiv E(\sum_{i=1}^{N} T_i)$ and $E(T_i) \neq E(T_j)$ if $T_i = T_j$. Also, the winning probability equals to $(\sum_{i=1}^{N} T_i)/N$.

6.3 Security issue : Manager may pretends as a player

The design of using new set of tokens in each round can prevent the manager from taking advantages by pretend as a player, in other words, prevent manager from buying all winning tokens by himself as the scratch-off ticket case explained in section 4.4.1.1.

However, manager can still pretend as a player to join the game and buy winning tokens to increase the statistical winning probability. Including secure random source R, such as block header, in the selection of token can increase the difficulty for manager to buy winning token by himself. Prevent manager from creating an illusion that the winning probability is high. Without this mechanism, by pretending as a player, manager can always buy winning tokens to increase the statistic value of winning probability.

6.4 Security issue : Manager may abort the game

In the protocol, it adopts a commitment scheme to hide the lucky number s chosen by player. The purpose of this mechanism is to provide a opportunity for manager to refuse some player's enrollment before knowing the player's selection. Without the commitment scheme, even though we included random source R, there are still some possibility for manager to calculate s' before the acceptance of enrollment and refuse it if he is tending to lose the game.

6.5 Comparison with Town Crier

As described in section 3.2, Town crier is a system which use TEE to construct TTP to fullfill the gap between smart contract and HTTPS enabled websites. Compared with Town Crier's financial derivative game, the design of our protocol has following advantages.

- This protocol doesn't rely on TEE such as Intel SGX.
- Before the acceptance of an enrollment, manager cannot know the content of the request.
- Manager cannot take advantages by pretending as a player such as buying all winning token or increasing the statistical value of winning probability. In Town Crier, manager can pretend as a player to join the game and do these things as their wish.

6.6 Comparison with Hawk

Another relative work, Hawk, a smart contract protocol used NIZK to preserve privacy of player's request and proof the correctness of manager's execution result, provides an example of rock-paper-scissors game. Compared with Hawk, the design of our protocol has following advantages.

Table 2 Compare with relative works

	Our Protocol	Hawk	Town Crier
TTP	No	Yes	Yes
TEE	No	Optional	Yes
Trusted Hardware	No	Optional	Yes

- This protocol doesn't need TTP. In hawk, manager is a TTP, trusted not spreading information to other players.
- Before the acceptance of an enrollment, manager cannot know the content of the request.
- Manager cannot take advantages by pretending as a player such as buying all winning token or increasing the statistical value of winning probability. In Hawk, manager can pretend as a player to join the game and do these things as their wish.

7. Conclusion

In this work, we proposed a smart contract protocol for a slot game and achieves all features explained in section 5. In our design, we used cryptographic primitives such as probabilistic additive homomorphic encryption to make the validation of winning probability in each round possible. We also considered several malicious behavior of manager and player in section 4. However, we still have many future works need to be done.

8. Future Works

- Comparison of several probabilistic additive homomorphic encryption algorithm.
- Optimization of the encryption algorithm.
- Implementation of the web interface for player.
- Implementation of the server used by manager.
- Implementation of the smart contract.

References

[1] Daira Hopwood et al. "Zcash protocol specification". In: *Tech. rep. 2016–1.10. Zerocoin Electric Coin Company, Tech. Rep.* (2016).

[2] Ahmed Kosba et al. "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts". In: *2016 IEEE symposium on security and privacy (SP)*. IEEE. 2016, pp. 839–858.

[3] Satoshi Nakamoto et al. "Bitcoin: A peer-to-peer electronic cash system". In: (2008).

[4] Gavin Wood et al. "Ethereum: A secure decentralised generalised transaction ledger". In: *Ethereum project yellow paper* 151 (2014), pp. 1–32.

[5] Fan Zhang et al. "Town crier: An authenticated data feed for smart contracts". In: *Proceedings of the 2016 aCM sIGSAC conference on computer and communications security*. ACM. 2016, pp. 270–282.