BlockLot: Blockchain-based Verifiable Lottery

Yongrae Jo* and Chanik Park †
Department of Computer Science and Engineering
Pohang University of Science and Technology
*memex@postech.ac.kr †cipark@postech.ac.kr

***Abstract*—We propose BlockLot, a blockchain based verifiable lottery. BlockLot provides transparent, immutable, fair, and verifiable lottery services enhanced by recent blockchain technologies such as append-only (replicated) distributed ledger and smart contract. In addition, BlockLot allows all participants to perform various verification to ensure that the system is actually working as expected.**

We implement BlockLot services which includes open, query, subscribe, and draw in smart contracts. We also develop web-based user interface for using the lottery services provided by BlockLot. The web interface allows the user to verify the lottery as well.

***Index Terms*—Blockchain, Lottery, Hyperledger/Fabric**

I. Introduction

Lottery events occur frequently. A lottery event requires trust between participants and event organizers. In general, people participate in the event because they believe the probability of winning lottery is fair. But the trust between the two is sometimes broken. For example, an event organizer can manipulate the results of the winner.

To solve the problem of fraudulent lottery, computer software can be used to compute winners without human intervention. Lottery program, which is determined solely by input and output, is believed to be more reliable, fair, transparent and verifiable than human-involved one.

The existing lottery systems adopt a centralized approach, in which calculations are performed on a single computer [1], [2], [13], [16]. However, in such a case, a person who gains privileged access to the computer(e.g., Hacker), on which the lottery program is run, may harm its integrity, transparency, and fairness. For example, an attacker who gains privileged access can manipulate the winners after the results are issued. Therefore, the centralized lottery system is unreliable in nature.

We define three attributes of the unreliable lottery which includes predictability, modifiability and information hiding. Predictability means that the random seed used in the lottery can be predicted. Modifiability means manipulation of event information, lottery result, random seed, lottery algorithm. Information hiding represents hiding important details of the lottery(e.g., changing a prize).

Against the attributes of unreliable lottery system, we also defined four attributes of reliable lottery which includes fairness, transparency, immutability, and verifiability. Fairness means that the winner should not be predictable in advance. Transparency indicates that the method of drawing winners and the results of the draw should be transparent to all participants. Immutability means that the registered information can not be modified at ones disposal and no one can change the result after it is announced. Verifiability represents that the lottery system should satisfy the aforementioned attributes and that the results should be verifiable.

We introduce BlockLot, a reliable lottery system based on blockchain. BlockLot enables reliable lottery in a way of fair, transparent, immutable, and verifiable. BlockLot uses a Bitcoin blockchain as a random beacon to choose winners, which it is unpredictable and publicly verifiable [11], [12]. BlockLot also uses a distributed-replicated ledger based on a blockchain and all services in BlockLot are supported by smart contracts in distributed manner. The lottery services provided by BlockLot includes *open*, *query*, *subscribe*, *draw*, *check*, and *verify*. The event organizer can open a lottery by providing required information such as the announcement date, number of winners, and list of prizes. Participants can query the lottery events registered and subscribe to the event to participate in it. When conditions for drawing winners are satisfied, an event organizer draw winners and participants can check and verify it.

The paper is organized as follows. Section II describes design and implementation of BlockLot. Section III presents discussion and related work. Section IV includes concluding remarks.

II. Design and Implementation

A. Overview

We developed BlockLot on Hyperledger/Fabric, a popular permissioned blockchain platform for enterprise [3]. Hyperledger/Fabric uses peer nodes to maintain a distributed ledger and execute chaincodes(a.k.a smart contracts). The Web server and the Hyperledger/Fabric SDK interact with the user (web client) and the blockchain network. Upon receiving the user's transaction, the web server sends it to the peer node and informs the user of the chaincode execution result. Also, it fetches the target block number to be used as a random seed from the Bitcoin random beacon. (See Fig. 1)

BlockLot users can be divided into participants and event organizers. As shown in the Fig. 2, an event organizer can register the lottery event and draw the winner with authentication. Participants can query the event, subscribe to the lottery, and check and verify the lottery result.

Chaincode is a special program that processes transactions and performs I/O with blockchain database in Hyper-

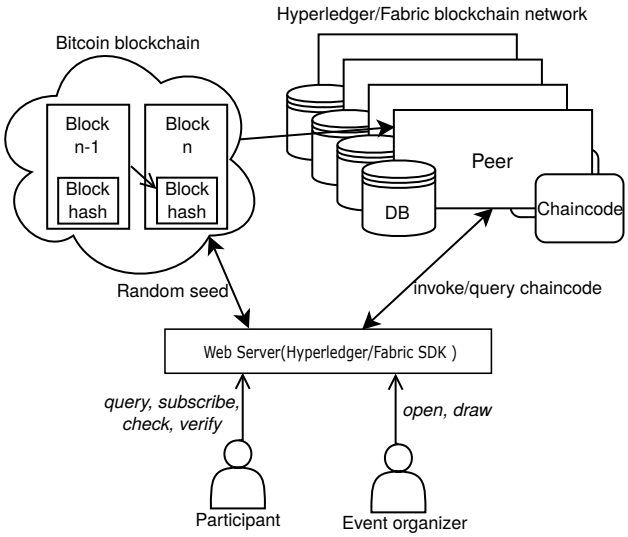

Fig. 1: Overview of BlockLot

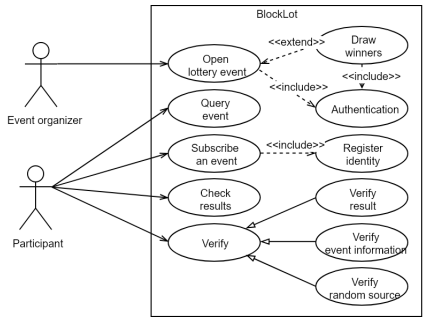

Fig. 2: Usecase diagram of BlockLot

ledger/Fabric. The types of transaction of BlockLot consists of query, open, draw, and subscribe, each of which is implemented by chaincode. We used GoLang to write chaincode because Hyperledger/Fabric supports GoLang as a primary language.

B. Lottery sequence

In this section, we introduce the detailed implementation of the lottery service of BlockLot as in 4.

1) Open: A open transaction registers a lottery in the blockchain. An event organizer provides event information such as name, announcement date, number of winners, block offset. The block offset represents the offset from the most recent block height in the main chain of the Bitcoin blockchain at the time of registration(namely target block). We used developer API provided by [4]. During the process, a unique ID of the event is generated and inserted into the key/value store of ledger with related information.

2) Query: A query transaction queries all registered events in the blockchain. We use the range query function GetStateByRange(), which is provided by the chain code shim interface, to retrieve the strings of (key, value) pairs and display the concatenated values to the participant.

3) Subscribe: To subscribe to an event, a participant chooses an event, enters their identity(e.g., email address) and gets a authentication token. The token is used to prove that the participant is actually a winner after the drawing. Subscribe transaction appends a participant to the list of participants of given event. The hash value of the registration name concatenated by authentication token is recorded on the blockchain, ensuring the anonymity of the participant.

4) Draw: A draw transaction can be invoked after the announcement date has arrived and a target block has been published. The event organizer enters a pre-issued authentication token, and if the verification is passed, winners will be selected. Once the winners are selected, all the information about the lottery event is fixed, so the information obtained by using the hash function is used as a *verifiable key*.(Fig. 3). The winner list and the verifiable key are stored in the blockchain. The script for draw uses the *Fisher-Yates random shuffle algorithm*, which uses the block hash of the target block as a random seed, to output the winner array in a deterministic way.

5) Check: After the draw transaction has been completed, the participant can confirm the winner by entering the name and authentication token. Like the participant list, the winner list is also expressed in hash values.

6) Verify: Participants also need to verify the result to ensure that the lottery is conducted without compromising the attributes of a reliable lottery. Our assumptions on an attacker contains a) predicting and manipulating random seed, b) fabricating event information arbitrarily, c) Manipulating winner list after draw.(i.e., recording a fraudulent winner list), d) malicious lottery scripts(e.g., draw winners regardless of random seed).

a) Preventing an attack on random seed and verifying its integrity: We use Bitcoin block hash, which is inherently unpredictable and immutable. Bitcoin miners spread around the world compete to get reward by solving random-based hash puzzles, so they generate unpredictable random numbers. Also, Bitcoin is the largest global blockchain at the moment, which it is immune to control block hash. Using Bitcoin blockchain as randomness source is described in [11], [12]. In Bitcoin, block hash is calculated using its *version, previousHash, merkle_root, timestamp, bits, nonce* fields of the block. So we calculate the hash value of the block and compare it with the one in the lottery in BlockLot.

b) Verifying event information integrity: We also recalculate and compare hash values with information determined at the time of event registration and draw to verify integrity. Given the participant list, random seed, and draw script, participant can calculate the winner list and compare it to the winner list on the blockchain.

c) Verifying response results from multiple peers: A peer that holds a blockchain can be crashed or hacked, so we need to query all peers in the network, and compare the response results. If more than half of the peers return the same response results, users can accept it.

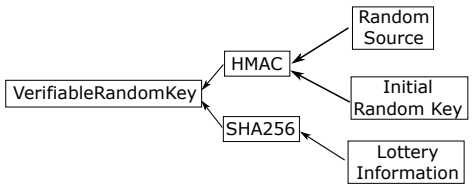

Fig. 3: Deriving verifiable random key

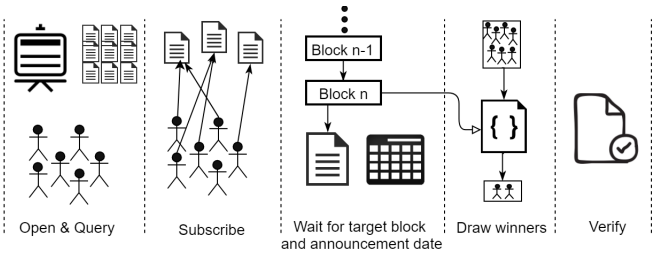

Fig. 4: Lottery sequence

d) Verifying statistical fairness: The draw script is verified by z-test [15] to verify that a particular participant does not win more than a average number of times. Given n times of running lottery, and when the probability of winning lottery is p (number of winners / number of participants), its binomial distribution should follow a normal distribution if n is large enough. If each participant's z-score is less than a pre-defined maximum z-value, the verification is successful, otherwise it fails.

C. Chaincode Implementation

Hereby we explain how to handle lottery transactions with chaincode implementation in Hyperledger/Fabric. Shim package provides simple interface to access and modify ledger. For example, GetState() is used to access to the ledger and PutState() is used to modify it. Unless otherwise stated, all of the functions below are included in the stub interface.

A lottery event in chaincode is represented as in Fig. 5.

1) Open: Implementing an *open* transaction is straightforward. It creates a new lottery_event struct, fills fields in the struct, marshals the struct, and inserts it into a ledger using PutState() with its identifier as a key. Since the ledger is maintained per channel in Hyperledger/Fabric, *ChannelID* should be specified when the event is created. The ID of *open* transaction is recorded as well. GetChannel() and GetTxID() is used to do these. *DrawTxID* and *SubscribeTxIDs* are initialized as UNDEFINED. Status field is initialized as REGISTERED.

```
type lottery_event struct {
        Status              string `json:Status`
        EventName           string `json:EventName`
        IssueDate           string `json:IssueDate`
        Duedate             string `json:"Duedate"`
        AnnouncementDate    string `json:"AnnouncementDate"`
        NumMax              string `json:"NumMax"`
        NumOfWinners        string `json:"NumOfWinners"`
        NumOfRegistered     string `json:"NumOfRegistered"`
        MemberList          string `json:"MemberList"`
        RandomKey           string `json:"RandomKey"`
        Identifier          string `json:"Identifier"`
        FutureBlockHeight   string `json:"FutureBlockHeight"`
        WinnerList          string `json:"WinnerList"`
        Script              string `json:"Script"`
        VerifiableRandomkey string `json:"VerifiableRandomkey"`
        LotteryNote         string `json:"LotteryNote"`
        DrawTxID            string `json:"DrawTxID"`
        DrawTxTimestamp     string `json:"DrawTxTimestamp"`
        OpenTxID            string `json:"OpenTxID"`
        OpenTxTimestamp     string `json:"OpenTxTimestamp"`
        SubscribeTxIDs      string `json:"SubscribeTxID"`
        ChannelID           string `json:"ChannelID"`
        OpenClientIdentity  string `json:"OpenClientIdentity"`
}
```

Fig. 5: Lottery event representation in chaincode

2) Query: We used GetStateByRange() to iterate all registered lottery events in the ledger. By iterating all keys in the ledger, we can combine those key/value pairs into one byte array. The byte array is returned to client.

3) Subscribe: A *subscribe* transaction contains identifiers of event and member as function arguments. It retrieves an event using GetState(), checks if the member is already registered, compares timeliness(i.e., due date). If successful, it adds the member to *MemberList* in lottery event. After that, it also records *subscribeTxID* returned from GetTxID().

4) Draw: It fetches target block from [4], extracts hash value from it, and uses it as a random seed for the Fisher-Yates random shuffle algorithm [14]. We implement the algorithm in GoLang. The algorithm receives the number of participantsP, number of winners(W), and random source as arguments. Since the algorithm requires a fresh random seed in each iteration step to swap two elements in an array with size of P, we newly design randomOracle() to generate new random values from the random source(i.e., block hash). randomOracle() simply hashes the random source and extracts the first four 32-bit integers from the hashed value, and adds them together to get a single value. Then, the result value is used to choose two elements randomly in the array. After iterating the array by P times, the algorithm is terminated. As a result, first W elements from the array are selected as winners.

Draw also creates *VerifiableRandomKey* from all fields in lottery event struct (Fig. 5). Since the winner list is determined, no information in the lottery event is changed. Therefore, the key for verifying integrity can be derived from these values.(Fig. 3)

D. User Interface

User interface for BlockLot is shown in Fig. 6. Lotteries are displayed as tables, which contain event name, announcement date, number of participants, number of winners, buttons for subscribe, draw, check, verify, and info.

An event organizer can register the lottery event by pressing the big button with a plus sign at the bottom of the table. A user can participate in a lottery event by clicking the subs. button and providing his/her name or email address. User interface for open and subscribe is in Fig. 7.

We implemented full-fledged BlockLot and it is publicly available on [10]. More implementation details as well as user interfaces that is not covered here can be found on the github repository.

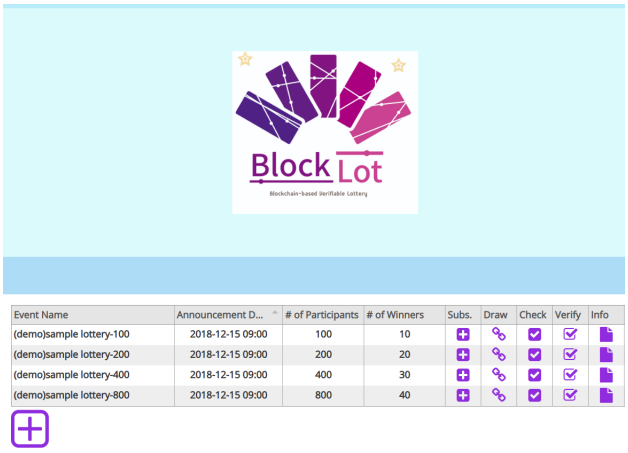

Fig. 6: Main user interface

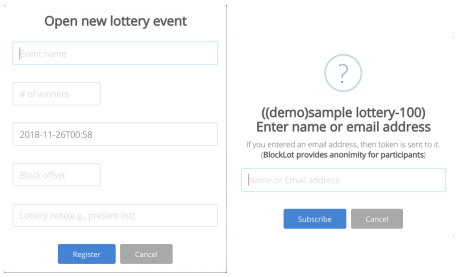

Fig. 7: Open & Subscribe Interface

III. Discussion and Related work

A. Security

a) Anonymity: BlockLot assumes that the participants identities are known offline in prior, meaning that their privacy may not be strictly guaranteed. For example, participation information to an event or the final results can be publicly available. We believe that this assumption is not a big problem because many lottery events provided by an enterprise in real-world often require participant's identity. (e.g., customers name).

b) Duplicate participation: Current implementation does not strictly prevent one user from participating in the same lottery multiple times. Because the user authentication is done only by name or e-mail, a malicious user can easily generate multiple identities to increase the chances of winning lottery. However, this concern will not be a problem if participant's identities are known offline to the event organizer.

c) Lagged blockchain view: We used developer APIs from [4], a freely available blockchain service provider to read the latest Bitcoin block. One significant problem here is that the main server which receives blocks on that site sometimes see a lagged blockchain view (e.g., temporary network delay). In that case, an attacker can predict the random seed, and exploit this information to be winner. But this limitation can be relaxed by choosing a target block which is much higher than currently perceived latest block.

B. Deployment environment

BlockLot focuses on lottery in situations where participants' identities are known in advance. For example, when a company performs a lottery for promotional purposes, the customer provides his/her identity to participate in the lottery.

C. Related work

In traditional centralized lottery system, a lottery is executed in a single computer. But this approach is vulnerable to a single point of failure as well as to the existence of a hacker who can gain privileged access to the system. Our work is basically a decentralized system, which means that a single participant cannot manipulate any data on the blockchain.

Also, blockchain-based lottery systems such as Kibo [5], FireLotto [6], Quanta [7], and SmartBilions [8] are based on Ethereum blockchain [9]. Ethereum provides a turing-complete smart contract platform which is suitable for executing lottery services. In contrast, our system is based on a permission-based blockchain, mainly for enterprise, which is orthogonal to Ethereum-based methods because our system requires identity to be known to participate in a lottery.

IV. Conclusion

In this paper, we present BlockLot, a Blockchain based verifiable lottery that records all information about the lottery event in a distributed ledger, and conducts the event in a transparent and immutable manner. BlockLot uses Bitcoin block hash as a unpredictable(i.e., fairness) random seed to draw winners. BlockLot also offers a variety of verification methods to ensure that the lottery is actually reliable. In conclusion, BlockLot is expected to significantly reduce social costs due to fraudulent lotteries.

Acknowledgment

This work was partially supported by The Institute for Information & Communications Technology Promotion (IITP) in South Korea, 2018, funded by the Korea government (Ministry of Science and ICT) (No.2017-0-00652, Development of High-performance and High-reliable Blockchain for Distributed Autonomous IoT Platform). We also would like to thank undergraduate students in POSTECH. Hyunseung Lee gave supplementary comments on this original document, Byoungyun Park helped with testing BlockLot, and Minsoo Koo designed BlockLot logo.

References

[1] http://www.unipicker.com/unipicker/main.do.
[2] https://cse.snu.ac.kr/node/18358.
[3] https://www.hyperledger.org/projects/fabric.
[4] https://www.blockchain.com/ko/api.
[5] https://kiboplatform.net/_alldata/files/kibowhitepaper.pdf.
[6] https://firelotto.io/whitepaper_en.pdf.
[7] https://www.quantaplc.im/Quanta.pdf.
[8] https://smartbillions.com/docs/SmartBillions_Smartpaper.pdf.
[9] https://ethereum.github.io/yellowpaper/paper.pdf.
[10] Blocklot. https://github.com/rupc/BlockLot, 2018.
[11] Iddo Bentov, Ariel Gabizon, and David Zuckerman. Bitcoin beacon. *arXiv preprint arXiv:1605.04559*, 2016.

[12] Joseph Bonneau, Jeremy Clark, and Steven Goldfeder. On bitcoin as a public randomness source. *IACR Cryptology ePrint Archive*, 2015:1015, 2015.

[13] David M Goldschlag and Stuart G Stubblebine. Publicly verifiable lotteries: Applications of delaying functions. In *International Conference on Financial Cryptography*, pages 214–226. Springer, 1998.

[14] Wikipedia contributors. Fisheryates shuffle — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Fisher%E2%80%93Yates_shuffle&oldid=864477677, 2018. [Online; accessed 25-November-2018].

[15] Wikipedia contributors. Z-test — Wikipedia, the free encyclopedia, 2018. [Online; accessed 29-November-2018].

[16] Jianying Zhou and Chunfu Tan. Playing lottery on the internet. In *International Conference on Information and Communications Security*, pages 189–201. Springer, 2001.