PureLottery: Fair Leader Election without Decentralized Random Number Generation

Jonas Ballweg, Zhuo Cai, Amir Kafshdar Goharshady

PureLottery: Fair Leader Election without Decentralized Random Number Generation

Jonas Ballweg
Technical University of Munich
Munich, Germany
jonas.ballweg@tum.de

Zhuo Cai Amir Kafshdar Goharshady
Hong Kong University of Science and Technology
Hong Kong SAR, China
{zcaiam,goharshady}@cse.ust.hk

***Abstract*—Given n participants, leader election (LE) is the process of designating one of them as the leader or coordinator. LE is ubiquitous in distributed computing and blockchain and has a wide variety of applications ranging from lotteries to proof-of-stake (PoS) protocols and decentralized autonomous organizations (DAOs). In a blockchain setting, we normally require our LE protocol to be decentralized, lead to a consensus about the elected leader, and be fair, i.e. elect each participant with the same probability $1/n$. Traditional blockchain LE solutions reduce the problem to decentralized uniform sampling from the set $\{1, 2, \ldots, n\}$. This reduction leads to approaches which are either vulnerable to manipulation, or fail to guarantee fairness, or require inefficient procedures such as verifiable delay functions (VDFs) and publicly-verifiable secret sharing (PVSS), thus making them gas-inefficient and costly when implemented as smart contracts.**

In this work, we observe that fair leader election can be achieved without explicit decentralized random number generation (RNG). In other words, the intuition behind our work is that RNG is a strictly harder problem than LE. This is because every LE participant is assumed to prefer to be chosen as the leader, e.g. the participant prefers to win the lottery or be the miner for the next PoS block. Thus, they will refrain from actions that would reduce their chance of being selected. This game-theoretic incentive can be exploited to design much simpler protocols for LE in comparison to RNG, by ensuring that dishonest behavior can only reduce the winning chances of the participant.

Specifically, we propose PureLottery, a protocol inspired by single-elimination knockout tournaments in sports such as football. We show that PureLottery selects the winner uniformly at random and provides strong game-theoretic guarantees to incentivize honest behavior. PureLottery is also strongly bias-resistant in the sense that every honest participant is guaranteed to win the lottery with probability at least $1/n$, even if an adversary controls all of the other $n-1$ players. In other words, dishonest behavior will never increase an adversary's chances of being elected, even if the adversary controls all but one of the participants. Finally, PureLottery is a simple protocol that can be efficiently implemented as a smart contract and uses a small amount of gas in practice. We provide an open-source implementation of the protocol, dedicated to the public domain with no copyright.

***Index Terms*—Leader Election, Blockchain, Decentralized Random Number Generation, Mechanism Design**

I. Introduction

Distributed Leader Election. Leader election (LE) protocols have been a central research topic in distributed systems for decades [1], [2]. Given a set $\{1, 2, \ldots, n\}$ of participants, the goal of LE is to choose one of them as the designated leader. In this work, we focus on LE in the context of blockchain. In this context, three major applications of LE are as follows:

- *Proof-of-Stake (PoS):* Currently, the most viable solution to replace Proof-of-work (PoW) and avoid its catastrophic costs and environmental damage is PoS [3]. In PoS protocols, a miner's chance of being chosen to add the next block is proportional to their stake in the currency or, in some variants, to the amount of money that they have locked for PoS. Currently, there are many functional PoS protocols [4]–[8] and Ethereum, the second largest cryptocurrency by market cap, has already switched to PoS [9], [10]. What all these protocols have in common is that they need a mechanism to choose the next block's miner in a fair, transparent and decentralized manner. Thus, they all have their own LE protocols.
- *Lotteries:* Lotteries are the quintessential leader election procedures in the real world. n people take part in a lottery by buying tickets and our goal is to elect one winner in a fair manner, ensuring that each participant has the same $1/n$ probability of being chosen. Lotteries are a large industry and had a market value of more than $300 billion in 2021 [11]. Traditional lotteries are centralized and the winner is chosen by the organizers. However, this is highly unreliable. For example, in the Hot Lotto fraud scandal of 2017, an IT director abused his power to tamper with the random number generator and successfully helped his friend win the $783,000 first prize. Therefore, lotteries should be implemented in a decentralized system, such as on a blockchain.
- *DAOs:* Decentralized Autonomous Organizations (DAOs) are organizations that are mainly or partly managed by smart contracts [12]. In such organizations, the members have voting powers and exercise them on the blockchain. Although all members have the same voting power, to propose new actions or investments and to initiate the voting process, it is necessary to first elect a coordinator among the members. Thus, virtually all real-world DAOs require LE protocols.

Of course, LE has many other applications as well, both in blockchain and in other distributed settings.

Leader Election on the Blockchain. There are many LE/lottery schemes for blockchains in the literature [4]–[6], [13]–[19],

The research was partially supported by the Hong Kong Research Grants Council ECS Project Number 26208122. Z. Cai was supported by the Hong Kong PhD Fellowship Scheme. Authors are ordered alphabetically.

[19], [20], [20]–[26]. See Section II for an overview of the techniques used in these approaches. To the best of our knowledge, all existing LE schemes select the winners based on decentralized random number generation (RNG). Indeed, they do not distinguish between LE and RNG even when the protocol is explicitly constructed for a lottery. The random number is either (i) produced by a pseudo-random number generator or verifiable random functions using blockchain states, block hashes or transaction data as seeds, or (ii) taken from an independent distributed protocol that outputs random numbers using a combination of commitment schemes, verifiable delay functions and secret sharing.

Despite being in wide use, both solutions have significant limitations. In the former, the security and fairness of the LE rely on the assumption that blockchain states and real-time transaction data are unpredictable and random, so that they are eligible to serve as seeds for pseudo-random number generators. However, this assumption might not hold, especially when a lottery can deliver a huge prize that incentives powerful miners and transaction owners to bias the seeds. In the latter, efficiency is sacrificed to solve a much harder problem, i.e. secure and tamper-proof RNG. Distributed randomness protocols usually require a committee of n participants to jointly contribute to the creation of the resulting random number. In order to ensure bias-resistance and other desired properties, these protocols either have a high communication complexity, e.g. $\Theta(n^2)$ or more for protocols that use publicly-verifiable secret sharing schemes, or assume that a significant proportion of participants are honest. In short, their techniques are too complicated and an overkill for a simple problem such as LE.

Our Contribution. In contrast to all existing LE protocols for blockchain which use decentralized random number generation (RNG), we propose PureLottery, a simple and novel alternative approach to LE which does not require the decentralized generation of an explicit random number that is subject to consensus. Instead of RNG, PureLottery selects the winner through a tree-shaped process which resembles knockout tournaments in games such as football. Our approach achieves the following desired properties:

1) **Fairness and Uniform Randomness**: If all participants honestly follow the protocol, every participant has an equal probability $1/n$ of being chosen as the winner.
   Note that, in contrast to our approach, there is no rigorous guarantee that pseudo-random number generators output numbers whose distribution is exactly uniform. Therefore, most previous approaches cannot guarantee strictly uniform randomness in winner selection as we do.
2) **Strong Bias-resistance**: Even with arbitrary malicious collusion, any subset of $t \leq n-1$ participants cannot increase their overall probability to win the prize. More precisely, if t participants act honestly, the probability that the winner is among them is t/n. We guarantee that no set of t participants can collude and act dishonestly so as to increase their overall winning probability to $t/n + \varepsilon$ for any $\varepsilon > 0$. Similarly, we guarantee that any honest participant has at least a $1/n$ probability of being chosen as the winner, even in the presence of $n-1$ colluding adversaries, i.e. even if every other participant is dishonest and maliciously trying to make the honest participant lose. In contrast, previous approaches that use blockchain data as seeds are vulnerable to tampering by the miners.
3) **Liveness**: No subset of $t < n$ participants can collaborate to terminate the protocol prematurely.
   Liveness is closely related to bias-resistance. For example, in random number generators using commitment schemes [24], adversaries can bias the output by leaving the protocol before completing the reveal phase. Other random number generators use publicly-verifiable secret sharing (PVSS) to ensure that the protocol cannot be terminated halfway [21]. However, PVSS requires a majority of participants to be honest. Therefore, the liveness in such protocols is preserved only against a minority of $t < n/2$ adversaries.

PureLottery achieves the aforementioned guarantees without verifiable delay functions, PVSS or other complicated primitives. It thus avoids operations that use huge amounts of gas and can therefore be easily implemented as a cost-efficient smart contract. Indeed, we provide an open-source implementation in Solidity at https://zenodo.org/doi/10.5281/zenodo.10065782. This implementation is dedicated to the public domain with no copyright.

PureLottery has a $\Theta(\log n)$ worst-case communication complexity between each lottery participant and the smart contract, i.e. a participant has to pay $\Theta(\log n)$ units of gas in the worst case. However, the expected complexity is $O(1)$. More specifically, each participant sends 4 messages to the smart contract in expectation.

Intuition. The main intuition behind our work is the realization that LE is generally a simpler problem than RNG due to the existence of natural incentives for the participants to maximize their chances of being selected. For example, an RNG participant might choose to leave the protocol prematurely, thus tampering with the distribution of the generated random number. However, an LE participant can be forced to choose between completing the protocol or not being elected. These incentives can be exploited in mechanism design to develop much more straightforward solutions for LE than the existing RNG techniques. We emphasize that the concept that explicit RNG is unnecessary for LE is itself a new and important realization. This critical distinction between LE and RNG was overlooked by previous blockchain-based LE schemes.

Organization. We review the necessary cryptographic preliminaries and related works about RNG and lottery schemes in Section II. In Section III, we present our PureLottery protocol. This is then followed by a security and complexity analysis in Section IV.

II. Related Works

A. Distributed Random Number Generation

Distributed random number generation (RNG) considers a network of n participants who want to jointly sample a uniform random number and reach consensus regarding the sampled output. The participants do not trust each other, so the protocol

must ensure no party can tamper with the value or bias its distribution. A secure distributed RNG protocol immediately provides a solution to distributed LE, but the opposite direction does not necessarily hold.

Naive Solution. A naive solution for RNG is to have every participant $i \in \{1, \ldots, n\}$ choose her own random value, e.g. a random string $x_i \in \{0,1\}^l$. Then, they all broadcast their values and compute $y = x_1 \oplus x_2 \oplus \cdots \oplus x_n$ as the common output. Here, $\oplus$ is the bitwise exclusive or (xor) operator. If all the values are chosen independently from each other and at least one of the value is really sampled from a uniform distribution, then the output y is a uniformly random string. In the real world, this protocol is fundamentally flawed because the messages are not sent simultaneously. Whoever announces her value after all other $n-1$ participants has the advantage to unilaterally determine the output. Without loss of generality, assume that participant n submits her value last and wishes the output to be $a \in \{0,1\}^l$. She can choose $x_n := x_1 \oplus x_2 \oplus \cdots \oplus x_{n-1} \oplus a$ and thus ensure the protocol's output is $y = a$.

Commitment Schemes. A standard solution to avoid domination by the last participant and mimic simultaneous action is to use a commitment scheme. A commitment scheme is a cryptographic primitive that allows a sender to commit a chosen value while keeping it hidden to the receiver. On the other hand, the receiver has the ability to verify the committed value later if the sender provides a hint. A commitment scheme has two phases. In the *commit* phase, the sender holds a message x and chooses a random string $r \in \{0,1\}^\kappa$. The sender encodes them to c and sends c to the receiver. Usually, a hash function is used at this point and we have $c := \texttt{hash}(x, r)$. In the reveal phase, the sender sends a hint string k to the receiver. Having the knowledge of k, the receiver can open the commitment c to get and verify x. For example, if a hash function is used for the commitment, we can use the hint $k := (x, r)$. Formally, a commitment scheme should satisfy the following two security properties:

- *Hiding:* Receiving a commitment c should leak no information about message x. Formally, for $\forall x_0,\ x_1$, let $p_0 \sim \{(r,c) | r \xleftarrow{\$} \{0,1\}^\kappa, c \xleftarrow{\$} \mathbf{Commit}(x_0, r)\}$, $p_1 \sim \{(r,c) | r \xleftarrow{\$} \{0,1\}^\kappa, c \xleftarrow{\$} \mathbf{Commit}(x_1, r)\}$. Then the distribution of p_0 and p_1 should be computationally indistinguishable.
- *Binding:* Values other than x should not be encoded into the same commitment c. Formally, for all non-uniform probabilitistic polynomial time algorithms that output x_0, x_1 and r_0, r_1, the probability that $x_0 \neq x_1$ and $\mathbf{Commit}(x_0, r_0) = \mathbf{Commit}(x_1, r_1)$ is a negligible function in κ, the length of r_0 and r_1.

With a commitment scheme, the participants of RNG can commit their values in the commit phase and announce the values in the reveal phase after every participant has already committed. The last participant does not know the values of other participants when she chooses her value in the commit phase, hence she cannot arbitrarily tamper with the output. However, she might choose not to reveal her value in the reveal phase and it is computationally infeasible to open the commitment. If the protocol computes the output without her value, then she has successfully biased the distribution. In the case of a random bit, she can dominate the output value by committing to 1 and then maliciously choosing whether to reveal or not. Even if we design a punishment scheme by claiming a prespecified amount of deposit for such malicious behavior, it is not effective if the output has significant economic consequences and the benefits from tampering the output outweigh the confiscated deposits. Moreover, rerunning the protocol in cases where a party does not reveal would also lead to success in biasing the output. The last participant can compute the value and choose not to reveal unless the resulting output is in her favor.

RANDAO. RANDAO [24] is a family of Ethereum smart contracts that produce distributed random numbers as a service. RANDAO adopts a commitment scheme and achieves tamper-resistance through verifiable delay functions (VDF) [27]. A VDF is a function whose value can only be evaluated after a preset amount of non-parallelizable serial computations. Thus, VDFs ensure that the output can be computed only after a prescribed amount of time. Even with massive parallel processors, the computation time cannot be significantly reduced. After the computation, an efficient proof of the output can be generated so that the expensive computation is no longer required for others to verify the output. Therefore, if we apply a VDF to the xor result of RNG, the new output cannot be biased by the malicious participants because they cannot predict the VDF output during the reveal phase. However, this incurs additional gas-inefficient computations on the blockchain in order to verify the result, and a huge amount of off-chain computation to run the sequential algorithm to evaluate the VDF. Finally, the output of a VDF is not guaranteed to preserve the uniform distribution unless we assume that hash functions are idealized random oracles.

PVSS. An alternative paradigm to tackle the issue of malicious manipulation is using a publicly-verifiable secret sharing (PVSS) scheme [21], [28]. With a PVSS scheme, each participant chooses a value as her secret and sends shares to other participants in the first phase. In the second phase, either the participant can release her value and other participants can check the correctness, or the participant maliciously hides the secret and other participants can collaborate to reconstruct her value based on the secret shares. Unfortunately, PVSS requires quadratic, i.e. $\Theta(n^2)$, communication complexity and additionally relies on the assumption that a majority of the participants are honest in order to successfully reconstruct the values in presence of adversaries.

Proof-of-stake protocols, such as [4]–[6] use a combination of the above methods to choose the next miner. Some of them also use blockchain data, such as hashes of previous blocks, as seeds for pseudo-random number generators. Thus, they suffer from the same limitations. Specifically, (i) VDFs and PVSS schemes have high computational or communication complexity; (ii) hash functions, VDFs and pseudo-random number generators are not guaranteed to output uniform randomness and achieve fairness; and (iii) blockchain data, such as block hashes, can be manipulated by malicious miners

if they can gain an advantage by tampering with the LE result.

B. Single Secret Leader Selection

Another line of research studies the problem of single secret leader selection [29], where (i) exactly one leader is selected among the participants and (ii) the identity of leader is unknown to other participants unless the leader reveals her success. For example, [30] uses a tree-based multiple-round protocols similar to our tournament protocol. In the weighted setting where participants have different chance of winning, [30] further represents the input of each participant using a tree of depth $O(\log(S))$, where they assume weights are integers and S is the weight. Compared with [30], PureLottery is suitable for weighted setting without extra overhead. Our protocol does not aim for secrecy of leader selection, hence having much lower computation and communication complexity.

C. Online Lottery Protocols

Another family of related works are online lottery schemes which do not necessarily use blockchains. Chow [31] proposed an online lottery scheme using a hash chain to link the lottery tickets of participants. A verifiable random function (VRF) is applied to extract verifiable randomness from the hash chain and a verifiable delay function (VDF) is applied to avoid the malicious adversary of the last participant. Similar to commitment schemes with VDFs, the sequential evaluation time of VDFs brings efficiency concerns and is also unable to guarantee fairness.

Lee [32] designed a scheme that uses the Chinese Remainder Theorem and blind signatures [33] in the lottery tickets. For the randomness to select the winner, Lee's scheme uses a pseudo-random number generator and seeds it with the sequential modular sum of random values submitted by all the participants. These random values are sent to the lottery dealer under encryption. In this scheme, the last participant can collude with the lottery dealer to tamper the random seed by decrypting other participants' values. Liu [34] improved this protocol by changing the random seed to a Lagrange interpolation result that depends on random values of all participants. The random values are committed in the first phase. Liu's scheme does not address the issue of maliciously hiding the random values, as in the commitment schemes. Grumbach [35] follows the paradigm of using delay functions to handle manipulation. The contribution is to use a Merkle tree structure to achieve fast probabilistic verification in large-scale lottery systems, which is orthogonal to our contribution. Xia [36] proposed a lottery scheme using symmetric bivariate polynomials to share random secrets among different lottery centers. This distributed randomness is similar to distributed RNG using PVSS schemes.

The approaches above are not decentralized. Instead, they all have a centralized structure with a dealer who receives values from all participants. In many cases, the dealer may be replaced by a smart contract, leading to methods similar to those of the following section.

D. Blockchain-based Lottery Schemes

Blockchain and smart contracts are well suited for designing distributed protocols without trusted third-parties. Thus, it is natural that there are many LE and lottery schemes using smart contracts. To the best of our knowledge, all existing blockchain-based lottery schemes [13]–[15], [17], [17] use RNG, either through (i) pseudo-random number generators or random oracles that use blockchain states as seeds, or (ii) an external decentralized random number generation protocol. Therefore, they all inherit the limitations of existing decentralized random number generation methods. Some of them present solutions to achieve orthogonal properties such as further privacy protections, but none can avoid the limitations in RNG protocols.

III. Our Protocol

In this section, we present PureLottery, our new LE protocol that does not rely on decentralized RNG. More specifically, all random numbers mentioned below are local randomness and generated on a participant's own machine. We never need to generate a random number that is subject to consensus and a mixture of submissions from different parties as is required in decentralized RNG methods. Our protocol is surprisingly simple and does not use any gas-inefficient cryptographic primitives such as VDF or PVSS.

In the sequel, we assume there are $n = 2^m$ participants in the leader election protocol. Exactly one participant will ultimately be selected as the leader. If the LE protocol is used to implement a lottery, this participant is the winner and can claim the entire money. Other participants lose the lottery game and cannot claim any money back. The LE protocol is run by a smart contract and is entirely decentralized, i.e. all participants have the same roles and abilities. We assume that each participant corresponds to an account on the underlying blockchain. Thus, every participant has a secret key and a public key and can perform standard operations such as hashing and encryption.

A. Two-player Case

In PureLottery, our two-player case coincides with classical approaches based on commitment schemes, but the extension to more players is done in a significantly different manner. For simplicity, we first present the basic case where there are only 2 participants, i.e., $m = 1$. One of these two participants should be elected as the leader and the election should be fair. As is common in commitment schemes, our PureLottery protocol consists of two phases: a commit phase and a reveal phase. In our smart contracts, time is measured by block numbers and each phase corresponds to a predetermined interval of blocks.

Commit Phase. In the commit phase, the participants register and send the commitment message to the contract. The commit message is valid if (i) a pre-specified deposit is transferred to the contract in the transaction, and (ii) a valid commitment is attached in the message. The commitment of participant i is of the form $\langle \texttt{hash}(x_i||r_i) \rangle$, where `hash` is a predetermined cryptographic hash function, $x_i \in \{0,1\}$ is a uniformly-sampled random bit and $r_i \in \{0,1\}^\kappa$ is a random string used as a salt to hide x_i. Here, $||$ denotes string concatenation. Registration and commitment can be done at the same time or separately, in two messages to the contract. It is also possible to

divide them in two subphases and first have all the participants register and then ask them to commit only after the registration deadline has passed.

Each participant can only send one commit message during the commit phase. If they send more than one message or not send a valid message, they are considered malicious and lose the LE, forfeiting it to the other player who will be announced as the leader by default. We number the participants in the order of their registration with the contract. Note that this order is unambiguous and is inherited from the order of transactions in the blockchain. Alternatively, if the protocol is implemented without an underlying blockchain, e.g. for PoS, the participants can be ordered by their public key or its hash. The order does not give any advantage to the participants.

Reveal Phase. As in a normal commitment scheme, both participants should send a reveal message to the contract within the reveal phase. The reveal message of participant i should be $\langle x_i, r_i \rangle$. The contract computes $\texttt{hash}(x_i||r_i)$ and compares it with the commitment of participant i. If the hash outputs do not match, then the reveal message is ignored and the revelation is not accepted. A participant who does not reveal correctly and in time (during the reveal phase) automatically forfeits the leadership position to the other participant.

After reveal phase ends, if a participant has failed to commit or reveal within the deadline, then the other participant is declared as the winner. If no one is malicious, the contract computes $y = x_1 \oplus x_2$. Participant i wins if $y \equiv i - 1 \mod 2$. In other words, the first participant wins if $y = 0$ and the second participant wins if $y = 1$.

Note that, in the simple protocol above, an honest participant who truly samples x_i from a uniform distribution has at least a $1/2$ probability of winning, no matter what the opponent does. A dishonest opponent can only increase this probability by forfeiting the game. Thus, no game-theoretically rational participant would deviate from the Nash equilibrium of submitting an x_i sampled from the uniform distribution.

B. PureLottery with more than Two Players

We now present our PureLottery protocol in the general case with $n = 2^m$ players. The intuition is similar to a knockout tournament in sports such as football. The basic two-player case is used as a building block and the protocol now consists of $m+1$ rounds. In each round, every pair of adjacent participants compete with each other by playing a commit-reveal game as in the previous section. The loser is eliminated and the winner proceeds to the next round. If there is no winner, i.e. if both participants violate the protocol, a dummy player is created and proceeds to the next round. The dummy players always lose. Since the number of players is halved in each round, we will have a single elected leader at the end of the $m+1$ rounds of the game.

Registration and Commitment. The first round of our general PureLottery protocol is similar to the commit phase of the previous section. Each player i chooses $m+1$ uniformly-sampled random bits $x_i^1, x_i^2, \ldots, x_i^{m+1}$ and $m+1$ random salts $r_i^1, r_i^2, \ldots, r_i^{m+1}$. She then computes the following hashes:

$$\begin{aligned} h_i^{m+1} &= \texttt{hash}(x_i^{m+1}||r_i^{m+1}) \\ h_i^{m} &= \texttt{hash}(x_i^{m}||r_i^{m}||h_i^{m+1}) \\ &\vdots \\ h_i^{j} &= \texttt{hash}(x_i^{j}||r_i^{j}||h_i^{j+1}) \\ &\vdots \\ h_i^{1} &= \texttt{hash}(x_i^{1}||r_i^{1}||h_i^{2}). \end{aligned}$$

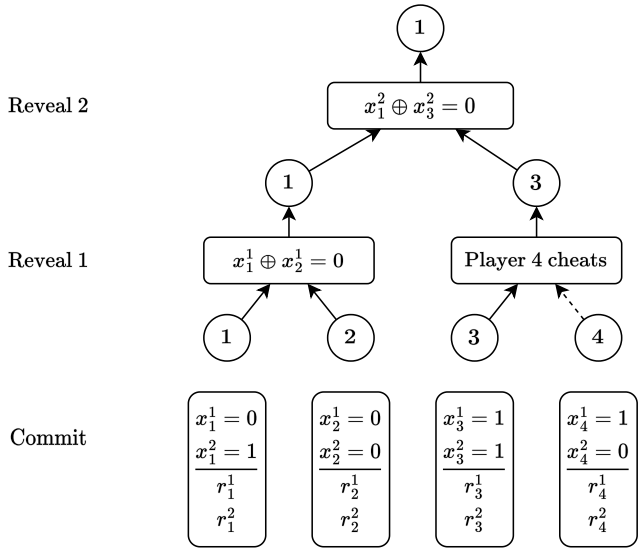

Fig. 1. Example of PureLottery with 4 Players

Intuitively, we aim to use the hash h_i^j in the j-th round of the protocol. The last hash h_i^{m+1} is simply a commitment to the random bit x_i^{m+1} as in the previous section. However, every other hash h_i^j is a commitment not only to x_i^j but also to the following hash h_i^{j+1}. Thus, any tampering with any of the x_i^j's would lead to a different value for h_i^1. In other words, h_i^1 is a commitment to all of the bits chosen by player i.

Each participant must send a message to the smart contract in the commit phase, providing a deposit and the commitment h_i^1. As before, a player who fails to provide the deposit or a valid commitment is automatically assumed to have lost and replaced with a dummy player for the next rounds. The players are ordered based on their registration time with the contract. As before, the order is arbitrary and can be replaced by an ordering based on public keys or any other well-defined ordering. Changing the order does not give any player an advantage.

Reveal Rounds. After the commit phase is over, our reveal phase begins, which consists of $m+1$ rounds. Each round has a predetermined deadline. In round j, we have $n_j = \frac{n}{2^{j-1}}$ remaining participants. The participants are considered in sorted order and divided into pairs of adjacent players who should face each other. Each player i must send a message to the smart contract which contains $\langle x_i^j, r_i^j, h_i^{j+1} \rangle$. The contract records

these values and checks whether the value previously declared for h_i^j by player i matches $\texttt{hash}(x_i^j||r_i^j||h_i^{j+1})$. If the hash does not match, the reveal message is unsuccessful and ignored.

Suppose player i is matched with player i' and $i' > i$. If either of the players fails to reveal a correct value, they automatically forfeit the game and the opponent wins. Otherwise, if both players have revealed correctly, the contract computes $y_{i,i'} = x_i^j \oplus x_{i'}^j$. Player i wins if $y_{i,i'} = 0$ and player i' wins if the value is 1.

In the unlikely and strange case that both players i and i' fail to reveal, they both lose the game. In this scenario, a dummy player with the same number i is created and proceeds to the next round. Dummy players are controlled by the smart contract itself. They lose in head-to-head encounters with any non-dummy player who reveals correctly. However, they win if their opponent fails to reveal. This ensures that the number of players halves at every round and we will have exactly n_j players at round j.

Elected Leader. The elected leader is the unique remaining player after round $m + 1$. In a normal LE procedure, any player who has not cheated can get their deposit back, but the cheaters' deposits are confiscated and burnt, i.e. they get stuck in the contract and cannot be withdrawn. In a lottery, each deposit has two parts: one to incentivize honesty as above and the other as a contribution to the lottery, i.e. ticket price. In this case, all lottery contributions are paid to the elected leader, who is also the lottery winner.

Example. Figure 1 shows an example of PureLottery with 4 players. Initially, the players choose their random bits x_i^j and salts r_i^j and commit to them. In the first round, 1 plays against 2 and 3 against 4. Since x_1^1 and x_2^1 are both 0, we have $y_{1,2} = 0$ and thus player 1 wins in this round and is promoted to the next round. In the other game of round 1, player 3 reveals that $x_3^1 = 1$. At this point, player 4 knows that he will lose the game since he knows x_4^1. So, he decides not to reveal. However, this does not affect the game and he loses by default as he failed to reveal. Players 1 and 3 proceed with the next round. In this round, the smart contract already knows h_1^2 and h_3^2 since these values were revealed in the previous round. Thus, each player $i \in \{1, 3\}$ has to reveal x_i^2 and r_i^2. The smart contract checks the hashes and computes $x_1^2 \oplus x_3^2$ concluding that 1 is the ultimate winner and elected leader.

An alternative scenario is shown in Figure 2. In this case, both players 3 and 4 fail to reveal their value in the first round. Thus, they are both eliminated and a dummy player, shown by $\perp$, advances to the next round. In the game between 1 and $\perp$, the non-dummy player 1 wins if and only if she reveals a correct value. In other words, $\perp$ is assumed to lose unless its opponent cheats.

C. PureLottery when n is not a Power of 2

For elegance and brevity, we presented our PureLottery protocol with the assumption that the number of players n is a power of 2 and defined $m := \log_2 n$. This assumption makes it much easier to present our protocol and also to analyze its security and complexity in Section IV. However, it can be easily relaxed without affecting the correctness or any of the analyses.

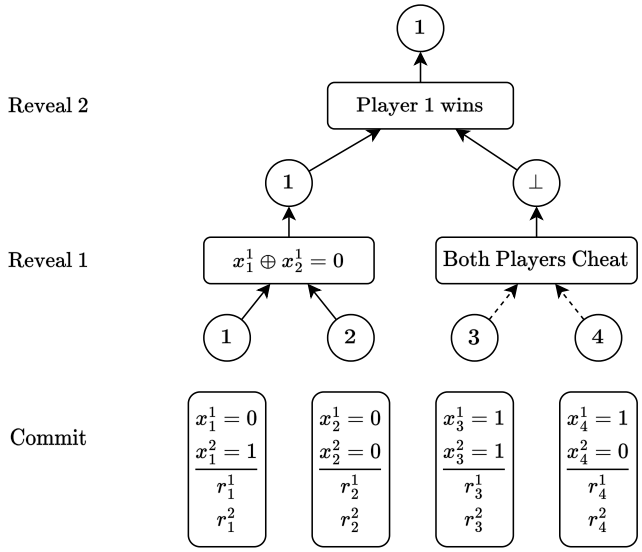

Fig. 2. Example of PureLottery when Players 3 and 4 Both Decide to Cheat

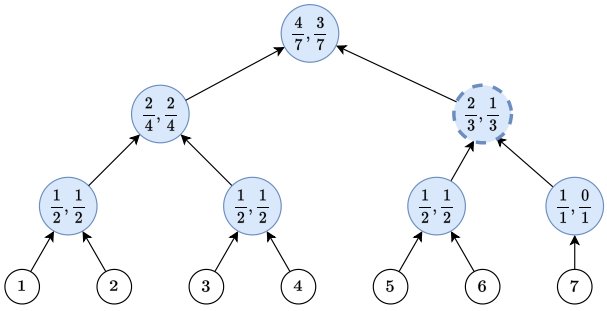

Fig. 3. PureLottery with 7 Players

Suppose n is not a power of 2. For example, $n = 7$. We can still create a binary tree of our players as in Figure 3. However, in this tree the game played at each internal (blue) node is a bit different. Consider the node shown with a dashed outline. Player 7 reaches this node automatically, without having to win against any other opponents. However, the winner of 5 and 6 reaches the same node after having defeated an opponent. Thus, it would be unfair to let 7 have the same probability of advancing to the next round as the winner of 5 and 6. Instead, we should have a game in which the winner of 5 and 6 has a $2/3$ probability of advancement and 7 has only $1/3$ probability. This would ensure that each of $5, 6$ and 7 have the same probability $1/2 \times 2/3 = 1/1 \times 1/3$ of reaching the last round. More generally, we have to ensure that the multiplication of winning probabilities from each leaf (player) to the root is exactly $1/n$.

This property is easy to achieve. Consider any internal node u of the tree. Let k_u be the number of leaves that are descendants of u. Similarly, let l_u be the number of descendants of the left child of u and r_u the number of descendants of its right child.

We annotate u with the probabilities $\frac{l_u}{k_u}, \frac{r_u}{k_u}$. See Figure 3. These are the probabilities that the winner at the node u is the winner of the left or right subgame, respectively. We have to design a game between these two players that achieves these exact winning probabilities. We call this a skewed commitment game.

Skewed Commitment Game. Suppose that two players play a game in which player 1 should win with probability l/k and player 2 with probability r/k where $r = k - l$. In the first phase, each player i chooses a uniformly-sampled random integer x_i from the set $\{0, 1, \ldots, k-1\}$ and commits to it. In the second phase, the players reveal their commitments and the value $y = x_1 + x_2 \mod k$ is computed. If $y < l$, then player 1 wins. Otherwise, player 2 wins.

We can easily plug in the skewed commitment game above into our PureLottery protocol of Section III-C. The only change is that we use modular sum instead of xor and the x_i^j's are no longer bits, but longer numbers. More specifically, if the ancestor of player i at round j has k descendant leaves, then x_i^j must be in the range $\{0, 1, \ldots, k-1\}$. As an example, in Figure 3, player 3 must choose $x_3^1 \in \{0, 1\}, x_3^2 \in \{0, 1, 2, 3\}$ and $x_3^3 \in \{0, 1, \ldots, 6\}$.

IV. Analysis of the Protocol

A. Fairness Uniform Randomness

If every participant is honest, then everyone sends valid messages in every round. In each round and within each pair, no participants lose before the end of reveal phase. The contract always receives valid reveal messages and computes the xor result to decide exactly one winner and one loser. For each such basic case, since both participants select a bit x_i^j uniformly at random, the xor result is also a uniform random bit. For each participant, the probability to survive each reveal round is $1/2$. Since independent random values are used for different rounds, the probability to survive all m rounds is $1/2^m = 1/n$, which is the desired fair probability. Note that the fairness of every round is not dependent on the assumption that *both* participants are honest. Even if one participant is honest and reveals their value correctly, they have at least a $1/2$ probability of advancing to the next round. The same analysis can be extended to the general case where n is not a power of 2 by observing that each player's probability of being the elected leader is equal to the product of probabilities from their leaf to the root and all such products are equal to $1/n$ by construction.

B. Strong Bias-resistance

First, consider the case where $n = 2$. If both participants are dishonest, it is easy to show that their overall winning probability cannot exceed 1. If both participants are honest, then we have already shown that they each have the same $1/2$ probability of winning. Suppose player 1 is honest but player 2 is a dishonest adversary.

- The adversary might not sample her value x_2 uniformly. However, due to the hiding property of commitment schemes, she has no information about x_1, which is drawn uniformly at random. This implies that x_2 is chosen independently from x_1. Consequently, $y = x_1 \oplus x_2$ follows a uniform distribution, no matter what the distribution of x_2 is.
- If the adversary does not honestly reveal, she forfeits the game and her probability of winning is 0. She also forfeits her deposit. In this case, since player 1 honestly follows the protocol, he wins the lottery with probability $1 > 1/2$.

The exact same analysis can be applied to a skewed commitment game, ensuring that an adversary cannot decrease an honest player's chances of winning the game.

We now show that strong bias-resistance, i.e., bias-resistance even when $n-1$ adversaries collude, holds for the PureLottery protocol with $n = 2^m$ participants by showing that each honest participant has at least probability $1/2$ to survive each reveal round. In this case, if two adversaries play against each other, they can choose who wins the round. Presumably, the choice is made so as to maximize their total probability of being elected. However, if the winner eventually has to play an honest player i at a later round j, the adversaries have no way of knowing this honest player's x_i^j, which remains hidden until player i reveals it at round j. Thus, if player i is honest and generates x_i^j uniformly at random, then her probability of winning at any given round is not affected. This also holds for the case where n is not a power of two.

In summary, the design of multiple independent reveal rounds ensures that the competition between a pair is independent from previous rounds from the perspective of honest participants. The opponent's value is committed before the current round while her own value is not revealed until the current round starts. Therefore, each honest participant has probability at least $1/n$ to be elected. In other words, dishonesty can only decrease the adversaries' chance of winning. Thus, no rational player or group of players would play dishonestly, ensuring that every player is elected with probability exactly $1/n$.

C. Liveness

The PureLottery protocol does not abort when a malicious participant violates the protocol by sending invalid messages or withholding values and refusing to reveal. Note that such a participant would immediately lose. This is safe because of strong bias-resistance. A malicious participant, or group of participants, might bias the probability distribution of the elected leader, but not to their own benefit. Thus, no rational participant or group of participants would fail to reveal their numbers in practice. This is a strong game-theoretic guarantee since revealing is always a dominant strategy. Essentially, by deviating from the protocol, the best that an adversary can do is to transfer its probability of winning to its opponent, the adjacent participant in its current pair. At the time of making decisions, it has no information about future rounds except for the participants controlled by itself. So, even if it is in collusion with its opponent, such dishonesty has no benefit and is strictly disincentivized since it causes a loss of deposit.

D. Complexity

Communication and Gas. In PureLottery, each participant sends $m + 3 = O(\log n)$ messages to the contract in the worst case. This includes a registration message, a commitment

message providing h_i^1, and up to $m+1$ messages for each of the reveal phases. The winner (elected leader) is never eliminated throughout the entire protocol and is required to send a message in each round. However, most participants lose in the early rounds and do not need to send any more messages. For a fixed participant, the expected number of rounds he participates in is $1 + 1/2 + 1/2^2 + \cdots + 1/2^m < 2$. Thus, in expectation, each participants sends no more than 4 constant-length messages to the smart contract and has to pay only $O(1)$ in gas fees since every function in the smart contract performs $O(1)$ on-chain computation. Note that the total gas usage of all participants is $\Theta(n)$, which is asymptotically optimal since we have to at least register every participant and the registration alone takes this much gas. Moreover, our contract is gas-efficient in practice, too, since it only performs hashes, which have a small fixed gas usage in programmable blockchains such as Ethereum [9]. We intentionally avoided gas-inefficient operations such as VDFs, PVSS and zk-SNARKs.

Off-chain Computation. In PureLottery, each participant should perform $O(\log n)$ hash operations to compute the h_i^j values. This can be computed on her own machine off-chain.

V. Conclusion

We presented PureLottery, a novel decentralized leader election scheme that does not rely on an explicit random number source. To the best of our knowledge, all previous LE schemes either require an external decentralized random number generation protocol, or assume that blockchain states are unpredictable and that random oracles can output randomness based on the unpredictable information on blockchains. We are the first to claim that decentralized random number generation is not required for a fair LE and lottery scheme. In PureLottery, the winner is selected through a novel multiple-round protocol of binary competition similar to knockout tournaments in sports. We showed that PureLottery is fair and selects the winner uniformly at random when participants are honest. Furthermore, honest behavior is well-incentivized. PureLottery is also strongly bias-resistant in the sense that no set of dishonest participants can increase their overall chance of winning by collusion. Moreover, no subset of participants can collaborate to terminate the protocol maliciously or to reduce an honest participant's chance of being elected. PureLottery has an asymptotically optimal gas usage and requires only 4 constant-length messages from each participant in expectation.

References

[1] E. Korach, S. Kutten, and S. Moran, "A modular technique for the design of efficient distributed leader finding algorithms," in *PODC*, 1985, pp. 163–174.

[2] H. Attiya and J. Welch, *Distributed computing: fundamentals, simulations, and advanced topics*. Wiley, 2004.

[3] L. Badea and M. C. Mungiu-Pupazan, "The economic and environmental impact of Bitcoin," *IEEE access*, vol. 9, pp. 48 091–48 104, 2021.

[4] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *SOSP*, 2017, pp. 51–68.

[5] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *CRYPTO*, 2017, pp. 357–388.

[6] B. David, P. Gazi, A. Kiayias, and A. Russell, "Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain," in *EUROCRYPT*, 2018, pp. 66–98.

[7] S. King and S. Nadal, "Ppcoin: Peer-to-peer crypto-currency with proof-of-stake," 2012.

[8] W. Zhao, S. Yang, X. Luo, and J. Zhou, "On PeerCoin proof of stake for blockchain consensus," in *ICBCT*, 2021, pp. 129–134.

[9] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, 2023.

[10] U. Pavloff, Y. Amoussou-Guenou, and S. Tucci Piergiovanni, "Ethereum proof-of-stake under scrutiny," in *SAC*, 2023, pp. 212–221.

[11] "Lottery market: Global opportunity analysis and industry forecast, 2021-2031," 2022. [Online]. Available: https://www.alliedmarketresearch.com/online-lottery-market-A14339

[12] S. Wang, W. Ding, J. Li, Y. Yuan, L. Ouyang, and F.-Y. Wang, "Decentralized autonomous organizations: Concept, model, and applications," *IEEE Transactions on Computational Social Systems*, vol. 6, no. 5, pp. 870–878, 2019.

[13] Y. Pan, Y. Zhao, X. Liu, G. Wang, and M. Su, "FPLotto: A fair blockchain-based lottery scheme for privacy protection," in *Blockchain*, 2022, pp. 21–28.

[14] J. Li, Z. Zhang, and M. Li, "BanFEL: A blockchain based smart contract for fair and efficient lottery scheme," in *DSC*, 2019, pp. 1–8.

[15] Y. Chen, S. Hsu, T. Chang, and T. Wu, "Lottery DApp from multi-randomness extraction," in *IEEE ICBC*, 2019, pp. 78–80.

[16] Z. Jia, R. Chen, and J. Li, "DeLottery: A novel decentralized lottery system based on blockchain technology," *CoRR*, vol. abs/1911.02392, 2019.

[17] Y. Jo and C. Park, "BlockLot: Blockchain based verifiable lottery," *CoRR*, vol. abs/1912.00642, 2019.

[18] C. Cachin, K. Kursawe, and V. Shoup, "Random oracles in Constantinople: Practical asynchronous byzantine agreement using cryptography," *J. Cryptol.*, vol. 18, no. 3, pp. 219–246, 2005.

[19] Z. Cai and A. K. Goharshady, "Trustless and bias-resistant game-theoretic distributed randomness," in *ICBC*, 2023, pp. 1–3.

[20] K. Chatterjee, A. K. Goharshady, and A. Pourdamghani, "Probabilistic smart contracts: Secure randomness on the blockchain," in *ICBC*, 2019, pp. 403–412.

[21] E. Syta, P. Jovanovic, E. Kokoris-Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford, "Scalable bias-resistant distributed randomness," in *SP*, 2017, pp. 444–460.

[22] P. Schindler, A. Judmayer, N. Stifter, and E. R. Weippl, "HydRand: Efficient continuous distributed randomness," in *SP*, 2020, pp. 73–89.

[23] G. Wang and M. Nixon, "RandChain: Practical scalable decentralized randomness attested by blockchain," in *IEEE Blockchain*, 2020, pp. 442–449.

[24] "RANDAO: A DAO working as RNG of Ethereum," 2019. [Online]. Available: https://github.com/randao/randao

[25] Z. Cai and A. K. Goharshady, "Game-theoretic randomness for proof-of-stake," in *MARBLE*, 2023.

[26] P. Fatemi and A. K. Goharshady, "Secure and decentralized generation of secret random numbers on the blockchain," in *BCCA*, 2023.

[27] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, "Verifiable delay functions," in *CRYPTO*, 2018, pp. 757–788.

[28] B. Schoenmakers, "A simple publicly verifiable secret sharing scheme and its application to electronic voting," in *CRYPTO*, 1999, pp. 148–164.

[29] D. Boneh, S. Eskandarian, L. Hanzlik, and N. Greco, "Single secret leader election," in *AFT*, 2020, pp. 12–24.

[30] M. Backes, P. Berrang, L. Hanzlik, and I. Pryvalov, "A framework for constructing single secret leader election from MPC," in *ESORICS*, 2022, pp. 672–691.

[31] S. S. M. Chow, L. C. K. Hui, S. Yiu, and K. Chow, "An e-lottery scheme using verifiable random function," in *ICCSA*, 2005, pp. 651–660.

[32] J. Lee and C. Chang, "Design of electronic t-out-of-n lotteries on the internet," *Comput. Stand. Interfaces*, pp. 395–400, 2009.

[33] D. Chaum, "Blind signatures for untraceable payments," in *CRYPTO*, 1982, pp. 199–203.

[34] Y. Liu, D. Lin, C. Cheng, H. Chen, and T. Jiang, "An improved *t*-out-of-*n* e-lottery protocol," *Int. J. Commun. Syst.*, pp. 3223–3231, 2014.

[35] S. Grumbach and R. Riemann, "Distributed random process for a large-scale peer-to-peer lottery," in *DAIS*, 2017, pp. 34–48.

[36] Z. Xia, Y. Liu, C. Hsu, and C. Chang, "An information theoretically secure e-lottery scheme based on symmetric bivariate polynomials," *Symmetry*, p. 88, 2019.