Fortuna: A Game-theoretic Protocol to Generate Secret Randomness on the Blockchain (Long Version)

Pouria Fatemi, Amir Kafshdar Goharshady

Fortuna: A Game-theoretic Protocol to Generate Secret Randomness on the Blockchain (Long Version)

Pouria Fatemi
Technical University of Munich
Munich, Germany
pouria.fatemi@tum.de

Amir Kafshdar Goharshady
University of Oxford
Oxford, United Kingdom
amir.goharshady@cs.ox.ac.uk

***Abstract*—In this work, we consider the problem of decentralized generation of *secret* random numbers. This is similar to classical random number generation (RNG) on the blockchain, except that the final result should only be known to a particular client. More formally a client, CASSIE, requests a secret random number r, kickstarting a decentralized RNG in which anyone on the network can contribute to the creation of r. As in traditional RNG, r should be unpredictable and tamper-proof. However, unlike traditional RNG, r is not publicized and in the end only CASSIE will know its value. Additionally, if/when she chooses, CASSIE can reveal r and provide a proof that it was indeed produced by our protocol.**

One of the main applications of secret RNG is to provide efficient and verifiable randomness for online gambling. If CASSIE operates a casino, she can request a single secret number each day and use it as a seed for a pseudo-random number generator that determines all betting outcomes. At the end of the day, she can prove that this seed was generated through a decentralized and tamper-proof process, demonstrating that the results were fair and not manipulated.

Both RNG and secret RNG have been widely studied in the literature. Existing protocols often allow anyone on the network to contribute an input and then aggregate all inputs together, e.g. using xor, to generate r. This ensures that as long as at least one participant is honest and samples their share from the uniform distribution, r is also uniformly-distributed. While having at least one honest participant has now become a standard assumption in this area, there are no game-theoretic incentives for honesty. In all previous secret RNG protocols, a participant can simply contribute a default value, such as zero, as their input and still receive the same rewards as an honest participant who submits uniform random shares.

In this work, we integrate a game-theoretic reward framework into our secret RNG protocol, ensuring that the only equilibrium of the game is when all participants submit uniformly random shares. Thus, honesty is not just assumed but also incentivized.

***Index Terms*—Random Number Generation, Secret Randomness, Blockchain, Game Theory**

I. INTRODUCTION

The Challenge of Secure Random Number Generation. Secure generation of unpredictable, tamper-proof, and verifiable random numbers is a fundamental challenge in blockchain and distributed systems [1]–[3]. Ideally, the process must be decentralized, ensuring no single entity can dominate the outcome. Additionally, it should be tamper-proof, preventing manipulation that could benefit certain participants. Transparency and auditability are also crucial, i.e. anyone should be able to verify the fairness of the process and the generated output.

Applications of RNG. Robust RNG mechanisms are essential for several key blockchain applications, including the following:

1) *Leader and Committee Selection:* Many blockchain protocols require a fair and randomized leader or committee selection process. For instance, proof-of-stake systems like Algorand [4] and Ouroboros [5] use randomness to determine validators, ensuring a participant's chance of selection is proportional to their stake in the currency.
2) *Probabilistic Smart Contracts [3], [6]–[8]:* Adding randomness to smart contracts expands their capabilities beyond deterministic functions, enabling more expressive applications.
3) *Online Gambling and Casinos [9]–[14]:* Blockchain-based gambling platforms must guarantee fair and unbiased betting outcomes. Despite the industry's massive multi-billion-dollar valuation, existing blockchain gambling solutions struggle to provide proofs of fairness and bias-resistance, even though these are often required for regulatory compliance.

Secret RNG. Several works in the literature, such as [15], [16], focus on the third use-case above. Unlike conventional RNG methods, they propose *secret* random number generation protocols where only the initiator, i.e. the casino operator, learns the generated number. Additionally, such protocols must guarantee *auditability*, allowing the casino to unmask the random number at a time of her choosing and prove that it was generated by the protocol.

Motivation for Secrecy [15], [16]. For a blockchain-based casino to be trustworthy, it must demonstrate that bets are fair and not manipulated. A naïve approach would involve generating a fresh random number for each bet using existing blockchain RNG techniques. See Section VII. However, this method is impractical due to high gas costs and processing delays. For example, Ethereum-based implementations of RNG

This work was partially supported by the Ethereum Foundation Research Grant FY24-1793.

protocols require several minutes to generate each random number. They also incur significant gas fees of several hundred USD. Thus, for a casino that processes tens or hundreds of thousands of bets per day, generating fresh randomness for each bet is both prohibitively expensive and excessively time-consuming. To address this, secret RNG provides a more efficient solution: at the start of the day, the casino (CASSIE) requests a single secret random number r using an on-chain decentralized RNG. This number acts as a seed for a pseudo-random number generator $\texttt{Rand}$. Each bettor (BETTY) provides a nonce n_{BETTY}, and the casino computes $\texttt{Rand}(r, n_{\text{BETTY}})$ to determine the outcome of their bet. This approach is only secure if r remains hidden. If it were publicly known, a bettor could manipulate the outcome by selecting a nonce n_{BETTY} that guarantees a favorable result. Unfortunately, classical blockchain RNG solutions reveal their outputs publicly, making them unsuitable. In contrast, secret RNG protocols ensure that only CASSIE knows r.

Motivation for Auditability [15], [16]. At the end of the day, CASSIE must be able to disclose r and prove that it was generated correctly, i.e. that she has not tampered with the r generated by the protocol. This proof, combined with the record of all bets, ensures that the casino did not manipulate any bet outcomes. More formally, during the day, CASSIE keeps r hidden and only reveals $\texttt{Rand}(r, n_{\text{BETTY}})$ for each bet. If BETTY holds a signed record of $(n_{\text{BETTY}}, \texttt{Rand}(r, n_{\text{BETTY}}))$, she can later verify whether CASSIE tampered with r. If tampering is detected, BETTY can present the signed record as proof, potentially triggering penalties through a smart contract. While such enforcement mechanisms are beyond the scope of this paper, they highlight the importance of auditability.

Our Contributions. We present a novel blockchain-based protocol that leverages homomorphic encryption, secret sharing, and mechanism design to generate tamper-proof *secret* random numbers in a trustless, decentralized, and auditable manner. Our method ensures that the random number remains hidden until the requester chooses to reveal it, at which point its authenticity can be verified by anyone. Unlike previous approaches that assume the existence of at least one honest participant, which is a problematic and unreasonable assumption in adversarial settings like casinos, our protocol eliminates the need for such trust by leveraging game-theoretic rationality. Instead of assuming that there is at least one honest participant, we design incentives that ensure rational participants who aim to maximize their own payoffs will follow the protocol honestly.

We initially present our protocol for the case where the generated secret random number is only one bit long. Then, we introduce optimizations that improve efficiency and extend the protocol's capability, enabling it to generate multiple secret random bits in parallel. Our approach can be straightforwardly implemented as a smart contract and intentionally avoids computationally expensive cryptographic primitives such as SNARKs, ensuring that gas costs remain minimal. Since reducing transaction fees is a major priority [8], [17]–[21], our protocol is well-suited for deployment in real-world blockchain environments.

II. PRELIMINARIES

Our protocol, which is presented in Section IV, brings together ideas from homomorphic encryption, secret sharing, game theory and mechanism design. In this section, we provide a concise overview of the necessary preliminaries and refer to [22]–[26] for more thorough treatment.

Homomorphic Encryption [23]. An encryption scheme is called homomorphic if it permits calculations to be carried out directly on encrypted data, without requiring decryption. While homomorphic encryption spans many advanced techniques, our needs are served by a simple form of partially homomorphic encryption known as the Goldwasser-Micali Cryptosystem). Suppose $\texttt{Enc}$ is a randomized encryption algorithm. For two bits $b_1, b_2 \in \{0, 1\}$ and their encryptions $\texttt{Enc}(b_1)$ and $\texttt{Enc}(b_2)$, we want to have:

$$\texttt{Enc}(b_1 \oplus b_2) = \texttt{Enc}(b_1) \cdot \texttt{Enc}(b_2)$$

where $\oplus$ is the XOR operator. This property lets us combine bits under encryption to produce the XOR of those bits, keeping the original bits hidden throughout the process.

Goldwasser–Micali Cryptosystem (GMC) [27]. One of the first provably secure public-key encryption schemes, the Goldwasser–Micali Cryptosystem (GMC) also provides our desired property. GMC consists of the following steps:

- **Key Generation:** An intended recipient ALICE picks two large primes p and q and sets $N := p \cdot q$. She then chooses an integer x that is *not* a quadratic residue modulo p or q, i.e. there is no integer y such that $y^2 = x \mod p$ or $y^2 = x \mod q$. Note that this means x is not a quadratic residue modulo N, either. The public key is (N, x) and the private key is (p, q).
- **Encryption:** To encrypt a bit $b \in \{0, 1\}$ for ALICE, a sender BOB picks a random integer y with $\gcd(y, N) = 1$ and then computes:

  $$e := \texttt{Enc}(b) := y^2 \cdot x^b \pmod N.$$

  If multiple bits need encryption, each bit is encrypted separately using different random values y.
- **Decryption:** Because x is a non-residue, e is a quadratic residue modulo N if and only if $b = 0$. Thus, ALICE checks whether e is a quadratic residue modulo both p and q. By Fermat's little theorem, e is a qudartic residue modulo p if and only if $e^{\frac{p-1}{2}} = 1 \mod p$. The same applies to q. If e is a quadratic residue modulo both p and q, then the bit b is 0. Otherwise, it is 1. Without knowing p and q, deciding if e is a quadratic residue modulo N is believed to be computationally difficult [28] and there are no known efficient solutions for this problem.

GMC has the partial homomorphic property we need: multiplying the encryptions of bits b_1 and b_2, with different randomizers y_1 and y_2, yields an encryption of $b_1 \oplus b_2$. This is by definition chasing:

$$\begin{aligned}\texttt{Enc}(b_1) \cdot \texttt{Enc}(b_2) &= y_1^2 \cdot x^{b_1} \cdot y_2^2 \cdot x^{b_2} \\ &= (y_1 \cdot y_2)^2 \cdot x^{b_1 + b_2} = \texttt{Enc}(b_1 \oplus b_2).\end{aligned}$$

Paillier Cryptosystem (PC) [29]. The Paillier cryptosystem

is a probabilistic public-key encryption scheme that provides additive homomorphism. It generalizes the Goldwasser–Micali cryptosystem by enabling the encryption of arbitrary integers rather than just single bits. PC consists of the following steps:

- **Key Generation:** An intended recipient ALICE randomly selects two large prime numbers p and q and computes $N = p \cdot q$. She then computes $\lambda = \mathrm{lcm}(p-1, q-1)$ and selects an integer $g \in \mathbb{Z}^*_{N^2}$. We assume $\gcd(N, \lambda) = 1$. If not, ALICE simply changes her choice of p and q. Finally, she computes

$$\mu = (L(g^\lambda \bmod N^2))^{-1} \bmod N,$$

  where the function L is defined as:

$$L(x) = \lfloor \frac{x-1}{N} \rfloor.$$

  If μ does not exist, ALICE changes her choice of p and q. The public key is (N, g), and the private key is (λ, μ).
- **Encryption:** To encrypt a message $m \in \mathbb{Z}_N$, a sender BOB selects a random integer $y \in \mathbb{Z}^*_N$ and computes:

$$c := \mathtt{Enc}(m) := g^m \cdot y^N \bmod N^2.$$

- **Decryption:** Given a ciphertext c, ALICE computes:

$$m := L(c^\lambda \bmod N^2) \cdot \mu \bmod N.$$

  This recovers the original plaintext message m.

The Paillier cryptosystem exhibits an additive homomorphic property: multiplying two ciphertexts of messages m_1 and m_2 (with different randomizers y_1 and y_2) results in a ciphertext encrypting $m_1 + m_2 \mod N$:

$$\mathtt{Enc}(m_1) \cdot \mathtt{Enc}(m_2) = (g^{m_1} \cdot y_1^N) \cdot (g^{m_2} \cdot y_2^N) \bmod N^2$$
$$= g^{(m_1+m_2)}(y_1 \cdot y_2)^N \bmod N^2 = \mathtt{Enc}(m_1 + m_2).$$

Shamir Secret Sharing [24]. Let n and t be positive integers with $t \leq n$. Suppose ALICE wishes to distribute a secret s among n parties so that *any* subset of t or more parties can reconstruct s, but any subset of $t-1$ or fewer parties learns *nothing* about s. Shamir Secret Sharing (SSS) achieves this by the following steps:

- **Share Creation:** ALICE chooses a large prime p and creates a random polynomial

$$g(x) := s + a_1 \cdot x + a_2 \cdot x^2 + \cdots + a_{t-1} \cdot x^{t-1} \pmod p,$$

  where s is the secret and $a_1, a_2, \ldots, a_{t-1}$ are random coefficients. Each party i receives the share $s_i := g(i)$.
- **Secret Reconstruction:** Because $s = g(0)$, any set of t or more parties can use Lagrangian interpolation to recover the polynomial g using their t points and compute $s = g(0)$. Any set of $t-1$ or fewer parties gains no information about s, as many polynomials, with all possible values at 0, could fit their partial data.

We now provide the necessary background from game theory.

Probability Distributions. Consider a finite set $Z = \{z_1, z_2, \ldots, z_k\}$. A probability distribution over Z is a function $\delta : Z \to [0, 1]$ such that $\delta(z_1)+\delta(z_2)+\cdots+\delta(z_k) = 1$. The set of all such probability distributions on Z is denoted by $\Delta(Z)$.

One-Shot Games. A *one-shot game* with n players is defined as $G = (S_1, S_2, \ldots, S_n, u_1, u_2, \ldots, u_n)$, where:

- S_i is the set of *strategies* for player i,
- $\mathcal{S} = S_1 \times S_2 \times \cdots \times S_n$ is the set of all *outcomes*,
- $u_i : \mathcal{S} \to \mathbb{R}$ is the *utility function* for player i, mapping outcomes to real values.

In a play of the game, each player i simultaneously selects one strategy s_i from S_i, without knowing the choices of others. Player i then receives the utility $u_i(s_1, s_2, \ldots, s_n)$ [26]. This means each player's payoff depends on every chosen strategy, not just their own.

Mixed Strategies. A *mixed strategy* σ_i for player i is simply a probability distribution over the strategies in S_i. Intuitively, σ_i describes how player i randomly chooses a strategy to play. A mixed strategy profile is a collection $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ that specifies one mixed strategy for each player. We denote by $u_i(\sigma)$ the *expected utility* of player i under σ, defined as the expectation of u_i over the randomly selected strategies according to $(\sigma_1, \sigma_2, \ldots, \sigma_n)$ [26].

Nash Equilibria. A *Nash equilibrium* in G is a mixed strategy profile σ where no player gains by unilaterally switching to a different mixed strategy. Formally, let σ_{-i} represent the strategies of all players other than i. Then σ is a Nash equilibrium if, for all $\tilde{\sigma}_i \in \Delta(S_i)$,

$$u_i(\sigma) \;\geq\; u_i(\tilde{\sigma}_i,\, \sigma_{-i}).$$

Hence, once a Nash equilibrium is fixed, no single player can do better by altering their strategy alone. By Nash's theorem, every finite game has at least one such equilibrium [30]. This concept captures stable outcomes in which each player is strictly motivated by maximizing their own expected payoff. However, when examining blockchain games, players operate under pseudonyms, meaning their true identities remain unknown. As a result, it is possible for different players to actually be the same person, leading to potential cooperation among them. In these situations, the concepts of strong and quasi-strong Nash equilibria become relevant.

Strong Nash Equilibria [31]. A strong Nash equilibrium is a mixed strategy profile in which *no group* of players can simultaneously change their strategies to make every member of the group better off. Formally, let P be a non-empty subset of players and $\tilde{\sigma}_P$ be a profile of new strategies for those players. If, for every $p \in P$,

$$u_p(\sigma) \;\geq\; u_p(\tilde{\sigma}_P,\, \sigma_{-P}),$$

then no coalition can jointly deviate in a way that improves the utility of all its members. However, if players are free to redistribute utility among themselves, an even stronger solution concept is necessary.

Quasi-strong Nash Equilibria [32]. A quasi-strong Nash equilibrium is a mixed strategy profile σ such that *any* non-empty set of players P cannot improve *their total utility* by deviating. Specifically, if $u_P = \sum_{p \in P} u_p$ is the sum of utilities for all players in P, then for any other strategy profile $\tilde{\sigma}_P$,

$$u_P(\sigma) \;\geq\; u_P(\tilde{\sigma}_P,\, \sigma_{-P}).$$

This is the appropriate notion in blockchain contexts, where one entity might control multiple players. Because they can pool or share their revenues, they will deviate as a group if it benefits them overall. Quasi-strong Nash equilibria thus offer the strongest guarantee that no unified coalition of rational players will deviate from the equilibrium. By definition, a quasi-strong Nash equilibrium is also a strong Nash equilibrium, and every strong equilibrium is a Nash equilibrium. Thus, if a quasi-strong Nash equilibrium exists in a game, it implies particularly robust stability.

Random Bit Generation Game (RBG) [32]. We adapt a special game designed in [32] to guarantee the right incentives in our protocol. Let n be the total number of players. An *RBG game* G with n players is specified as follows:

- Each player i has a strategy set:

$$S_i = \begin{cases} \{0,2\} & \text{if } i \text{ is even,} \\ \{1,3\} & \text{if } i \text{ is odd.} \end{cases}$$

- An outcome of this game is a tuple $s = (s_1, s_2, \ldots, s_n)$, where $s_i \in S_i$. The utility for player i at outcome s is

$$u_i(s) \;=\; \sum_{j \neq i} f(s_i, s_j),$$

where

$$f(s_i, s_j) \;=\; \begin{cases} 1 & \text{if } s_i \equiv s_j + 1 \pmod 4, \\ -1 & \text{if } s_i \equiv s_j - 1 \pmod 4, \\ 0 & \text{otherwise.} \end{cases}$$

In an RBG game, all players simultaneously pick one of their two allowed strategies. Then, for every pair (i, j) whose indices have different parity (one even, one odd), we imagine a small head-to-head contest based on the chosen numbers. A player scores $+1$ if her number is exactly one more (mod 4) than her opponent's, and -1 if it is one less. Each player's total utility is the sum of all these pairwise scores.

RBG games have a unique quasi-strong Nash equilibrium in which every player plays uniformly. Formally, suppose an RBG game G has at least two players. Define a mixed strategy profile $\bar{\sigma}$ where $\bar{\sigma}_i$ picks each action in S_i with probability 0.5. Then, $\bar{\sigma}$ is the *only* quasi-strong equilibrium of G [32].

In other words, when some players may be controlled by the same entity, as is the case on a blockchain, the *only* stable, rational way to play is to pick actions uniformly at random. Even-numbered players mix between $\{0, 2\}$, while odd-numbered players mix between $\{1, 3\}$. Thus, an RBG game strongly motivates participants to produce uniform random bits, perfectly aligning with our needs for decentralized randomness.

We now have all the required ingredients for our secret RNG protocol.

III. Requirements

In this section, we begin by outlining the conventional *xor-based approach* from prior blockchain RNG schemes and then highlight the additional challenges that arise when the output must remain secret. Next, we introduce the required security guarantees and game-theoretic incentives that our protocol must fulfill. Section IV provides the details of our protocol.

The Xor-based Approach. A standard decentralization strategy in many blockchain-based RNG protocols such as [3], [33] is to let anyone in the network participate by submitting an individually sampled random bit. If there are n participants and player i provides a bit b_i, the final random output is computed by

$$r \;=\; \bigoplus_{i=1}^{n} b_i,$$

where $\oplus$ is the bitwise xor operator. Of course, the same method can be extended to the case where each b_i is more than one bit long. This ensures that, if *at least one* participant samples their b_i from the uniform distribution, the overall outcome r will also be uniformly random.

Requirements for Secret RNG. In our scenario, a casino Cassie requests a random number with stronger requirements than the usual open xor-based method. In addition to decentralization and uniformity of output, we need:

1) *Secrecy.* Only Cassie should see r when it is generated. Even if all but one participant collude, they must be unable to learn r.
   Of course, if *all* participants collude, they trivially recover r by xoring their b_i's. However, since the system is decentralized and open, Cassie herself can take part as a participant and thus refuse to leak her b_i.
2) *Tamper-Resistance.* No group of participants, including or excluding Cassie, should be able to exit the protocol early or act in a way that alters r or its probability distribution.
3) *Auditability.* After the RNG protocol is over, Cassie must have the ability to reveal r whenever she deems necessary and to provide a publicly-checkable proof that r was indeed generated by the protocol and not changed by her.
4) *Accountability.* If a player i reveals her input b_i to another party, this dishonest act must be identifiable and traced back to i. Moreover, nobody other than player i should be capable of learning b_i. This includes Cassie.

Motivations Behind Accountability [15].. While properties 1–3 above are natural, accountability might seem strange at first glance. There are two motivations behind requiring accountability:

- We want to discourage players from revealing their numbers, thus preventing a disastrous situation where each player i is bribed by a bettor Betty to disclose b_i. In that scenario, Betty would learn r and win every bet against the casino Cassie. To avoid this, each player i must place a deposit that is awarded to anyone proving that i leaked b_i. Hence, if i gives b_i to Betty, Betty can collect i's deposit. Even if only one other participant remains honest and refuses to leak, Betty cannot deduce r, but can still seize i's deposit—which she will do if she wants to maximize her own payoff. Therefore, leaking is extremely risky, since a player can lose their deposit as soon as a single other participant chooses not to collude. Note that Cassie might also participate as a player, guaranteeing

there is at least one person who will not leak, given the casino would incur huge losses if r became public.

- We need to ensure that if r is leaked, CASSIE alone is held responsible, and if a particular b_i is leaked, that specific player i is blamed. If CASSIE were able to view all b_i's, she could do the following: if the final random value r works against her interests and she loses many bets during the day, she could deliberately leak every b_i (and thus r) and then allege that the players leaked them. She could then demand a fresh random value and void every bet settled with the compromised r. Moreover, it would be impossible to tell whether CASSIE herself leaked the bits or the players were at fault.

While previous approaches to secret random number generation, such as [15], [16], achieve all or most of the requirements above, they do so by relying on the assumption that at least one participant is honest and submits a b_i sampled from the uniform distribution. One of our main contributions in this work is to lift this assumption and guarantee the following game-theoretic requirement.

Game-Theoretic Requirements. Beyond the properties enumerated above, our protocol must incentivize honest participation, rather than assuming it. Specifically, we have the following game-theoretic requirements:

- *Prohibitively Expensive Manipulation.* Given a security parameter ν, altering the final output r or its distribution must cost at least ν units of currency in penalties, ensuring any attempt to change r is too expensive to be worthwhile. This is a game-theoretic representation of tamper-resistance.
- *Uniform Bits as a Quasi-strong Equilibrium.* Players should be motivated to submit *truly uniform* random bits b_i. Formally, choosing the b_i uniformly and feeding it into the protocol must constitute a quasi-strong equilibrium, so that any coalition of participants gains nothing by deviating from honest uniform random number generation.

IV. FORTUNA PROTOCOL

We now present Fortuna, a decentralized secret random number generation protocol that can be implemented as a smart contract and guarantees the requirements mentioned in the previous section. Our protocol integrates homomorphic encryption, secret sharing and mechanism design.

Intuition behind Fortuna. We build upon previous random number generation protocols such as [3], [15], [16]. In our protocol, each player still picks a bit b_i, as was the case in earlier methods, and the final random output r is the xor of all b_i's. However, to keep r secret, every bit b_i must be encrypted under CASSIE's Goldwasser–Micali (GMC) public key. Instead of sending $b_1, \ldots, b_n$ to CASSIE, players submit $\texttt{Enc}(b_1), \texttt{Enc}(b_2), \ldots, \texttt{Enc}(b_n)$. By multiplying these encrypted bits, one obtains

$$\prod_{i=1}^{n} \texttt{Enc}(b_i) \;=\; \texttt{Enc}\Big(\bigoplus_{i=1}^{n} b_i\Big) \;=\; \texttt{Enc}(r).$$

CASSIE can decrypt the latter to recover r. Simply letting each player post $\texttt{Enc}(b_i)$ on a smart contract does ensure secrecy and auditability, but not tamper-resistance. If CASSIE controls one or more players, she could decrypt their bits as they arrive, then adjust her own bits to force a desired r. Therefore, each player must commit to their bit before seeing anyone else's choice and cannot change it afterward. We also need to ensure that CASSIE never sees any of the $\texttt{Enc}(b_i)$ values, but only their product. We use Shamir Secret Sharing to guarantee this property. Finally, we use a Random Bit Generation Game (RBG) to pay rewards to the participants. In this game, each b_i is used to determine participant i's strategy. Since the only quasi-strong equilibrium in RBG is to play uniformly, it incentivizes every player to submit a uniform b_i.

We are now ready to present the Fortuna protocol in detail. Our protocol consists of the following steps:

Step 1. Initialization. CASSIE begins by calling an initialization function in the contract to request a random number. She deposits ϕ, the total reward to be shared among honest participants. Next, CASSIE generates two large primes p and q, sets $N = p \cdot q$, and chooses a primitive root $\gamma \bmod N$. She then picks a random integer χ so that

$$x := \gamma^{\chi} \bmod N$$

is a non-residue modulo both p and q. She then publicly announces N, γ, χ and x by recording them in the contract. She keeps the private key (p, q) secret. For Shamir Secret Sharing (SSS), CASSIE fixes:

- A threshold t^*.
- A prime p' that is significantly bigger than $N^{\dagger}$.
- Time limits, in terms of number of blocks, for each step below.

Finally, she fixes a penalty value ν that applies to dishonest behavior.

Step 2. Player Registration. Anyone on the blockchain can register as a player by depositing ν. We call the total number of participants n. After successful completion of the protocol, each honest player recovers their deposit and earns a share of ϕ. To identify participants, the contract records their public keys. We assume that CASSIE herself signs up as a player/participant as well.

Messaging. When player i sends a message m to player j, we assume m is encrypted so that only j can read it. We also assume it is signed by i, allowing j to verify its origin and prove the message's existence if needed. As is standard in such protocols, we assume the players communicate off-chain through secure, authenticated channels. If a message does not arrive on time, the intended recipient can request it on-chain, forcing the sender to post it on-chain before a deadline or lose their deposit. This approach keeps on-chain costs low while still ensuring dependable message delivery.

Step 3. Encryption and Secret Sharing. Each player i does the following:

- Randomly picks a bit $b_i \in \{0, 1\}$

*In practice, since the number of players n is not known at this point, it might be more natural to set a value for $n - t$.

†It suffices to have $p' > 3 \cdot N \cdot n$, where n is the number of players.

- Chooses a random integer v_i and sets $y_i = \gamma^{v_i} \bmod N$ in a way that $\gcd(y_i, N) = 1$
- Computes

$$e_i \;=\; \texttt{Enc}(b_i) \;=\; y_i^2 \;\cdot\; x^{b_i} \;(\text{mod } N),$$

using CASSIE's Goldwasser–Micali public key (N, x).
- To *commit* to b_i and enable its recovery without revealing it to CASSIE we rely on secret sharing. Player i defines

$$\lambda_i \;=\; \log_\gamma e_i \;=\; 2 \cdot v_i + b_i \cdot \chi.$$

They then apply Shamir Secret Sharing to λ_i. Concretely, they pick random coefficients $a_{i,1}, \ldots, a_{i,t-1}$ and form a polynomial:

$$g_i(x) \;=\; \lambda_i \;+\; a_{i,1} \cdot x \;+\; \ldots \;+\; a_{i,t-1} \cdot x^{t-1} \;(\text{mod } p').$$

They send each other participant $j \neq i$ the share $g_i(j)$. The player also commits to b_i by choosing a random salt s_i and sending $\texttt{Hash}(b_i, s_i)$ to the contract, which records it. Here, Hash is a cryptographic hash function.

Step 4. Aggregation. At the end of Step 3, every player i has received one share from every other player. Define a global polynomial

$$g(x) \;=\; \sum_{j=1}^{n} g_j(x) \;(\text{mod } p').^{\ddagger}$$

Player i can compute $g(i)$ by adding up all received shares $g_j(i)$. Then, player i sends $\texttt{Enc}\big(g(i)\big)$, encrypted using CASSIE's public key, which may be GMC or another scheme to the contract. This lets CASSIE read and decrypt $g(i)$ afterwards.

Step 5. Obtaining r. Once CASSIE has at least t of these encrypted values $g(i)$, she decrypts them and uses Lagrange interpolation to recover the polynomial g. Then, she computes $g(0)$, which satisfies:

$$g(0) = \sum_{i=1}^{n} g_i(0) = \sum_{i=1}^{n} \lambda_i = \sum_{i=1}^{n} \log_\gamma e_i = \log_\gamma \left(\prod_{i=1}^{n} e_i \right) =$$

$$\log_\gamma \left(\prod_{i=1}^{n} \texttt{Enc}(b_i) \right) = \log_\gamma \texttt{Enc} \left(\bigoplus_{i=1}^{n} b_i \right) = \log_\gamma \texttt{Enc}(r).$$

Hence, CASSIE computes

$$\texttt{Enc}(r) \;=\; \gamma^{\,g(0)} \;(\text{mod } N),$$

and then decrypts with her private key (p, q) to learn the secret random number r.

Step 6. Audit. At a later time of her choosing, e.g. at the day's end, CASSIE reveals r in the contract. She discloses the decrypted values $g(i)$ and her GMC private key (p, q). Anyone can verify these match $\texttt{Enc}(r)$ and confirm that CASSIE has not tampered with the final random value.

Step 7. Returning Deposits. Each player i must reveal their bit b_i and the salt s_i to the contract. If these values are consistent with the comitted hash value, the player gets their deposit ν back. Otherwise, the deposit goes to CASSIE. The contract also keeps track of the following five statistics for use in the final reward calculation:

- m: Number of participants who completed the protocol without being caught for cheating
- m_0: Number of even-indexed participants i who were not caught for cheating and reported $b_i = 0$
- m_1: Number of odd-indexed participants i who were not caught for cheating and reported $b_i = 0$
- m_2: Number of even-indexed participants i who were not caught for cheating and reported $b_i = 1$
- m_3: Number of odd-indexed participants i who were not caught for cheating and reported $b_i = 1$

Step 8. The RBG Game. In this step, we use the fee ϕ paid by CASSIE to reward participants for submitting inputs and contributing to the random number generation. The total fee ϕ is split among those who have correctly revealed their choices. Any participant found in m_k during Step 7 is treated as having played strategy k in the RBG game. In the RBG, if a participant i plays strategy k, her utility is

$$u_i(s) = m_{(k-1 \bmod 4)} - m_{(k+1 \bmod 4)}.$$

Each participant i earns a reward of

$$R_i := \frac{\phi}{m} \cdot \Big(1 + \frac{u_i(s)}{m}\Big).$$

Put simply, a participant's reward depends on her utility in the RBG game, and all rewards add up to ϕ. This concludes the Fortuna protocol. See Section VI below for extensions, including support for more than one random bit.

V. ANALYSIS OF THE PROTOCOL

We now explain why this method satisfies the required properties for a blockchain-based secret RNG.

Decentralization. Our protocol is fully open. In Step 2, anyone on the network can join as a player, and their bit b_i contributes to the final random outcome $r = \bigoplus_{j=1}^{n} b_j$. Since no central authority exists and all nodes have equal rights to participate, the system remains decentralized.

Uniform Distribution. Because r is computed as $\bigoplus_{j=1}^{n} b_j$, a single honest player who picks a uniformly random bit b_i ensures the entire result r follows a uniform distribution. Thus, a bettor BETTY concerned about potential cheating by CASSIE can become a player, choose an honest random bit sampled from the uniform distribution, and therefore guarantee uniformity.

Secrecy. Each player i produces shares of $\lambda_i = \log_\gamma\big(\texttt{Enc}(b_i)\big)$ in Step 3. Even if all other players collaborate, they can only obtain $\texttt{Enc}(b_i)$, which only CASSIE can decrypt. Therefore, those players cannot learn b_i, and consequently cannot figure out r. Note that we assume CASSIE herself is a player, too, and she would not want to leak r.

Tamper-resistance. Every player i fixes their bit b_i in Step 3. At that stage, player i lacks any knowledge of non-colluding bits b_j for $j \neq i$. This remains true even if i is CASSIE, because uncovering b_j would require CASSIE plus at least t colluding

[‡]We have to ensure p' is large enough to avoid overflows in $g(0)$. It suffices to set $p' > n \cdot 3 \cdot N$ since each $g_i(0)$ is at most $3 \cdot N$ by definition.

players to retrieve $\texttt{Enc}(b_j)$ and decrypt it. As long as there is no such collusion of t players plus CASSIE, the protocol remains tamper-proof.

Specifically, a player has no incentive to cheat in Steps 3 and 4 because they know nothing about r yet. Leaving the protocol early or acting dishonestly only causes them to lose their deposit. If a player quits in Step 3, they never contributed a bit b_i and thus have no influence on r. If they leave in Step 4, the protocol can still proceed to Step 5 as long as at least t players remain.

Failures. The protocol fails only if fewer than t players submit valid data in Step 4 or if certain players provide incorrect values, blocking CASSIE from running Lagrange interpolation. In practice, dishonest players gain nothing by sabotaging, and they lose their deposits as a penalty. The contract can have a fallback: if CASSIE cannot interpolate in Step 5, she discloses the partial $g(i)$ values, and the players reveal their hidden numbers. This exposes the dishonest parties, who are penalized. Crucially, such misconduct happens before any party knows r. Hence, we can penalize them and restart the protocol.

Auditability. Step 6 provides an inherent audit. CASSIE reveals her Goldwasser–Micali private key so anyone can replicate the computations from Step 5 to confirm r independently.

Accountability. Suppose a player i has provided an input number b_i of length k bits (with $k \geq 32$ in practice). Our contract features a *leak report function* that lets anyone accuse i of leaking b_i by submitting the alleged value. If such a report is filed by ALICE, an accountability process begins after Step 6, i.e. when r is already public and no longer secret. The purpose of this process is to determine whether ALICE's claimed b_i truly matches player i's input and, if so, penalize i.

During this procedure, player i must reveal all secret data generated while running the protocol: namely b_i, y_i, v_i, $a_{i,1}, \ldots, a_{i,t-1}$, plus the shares $g_i(j)$ sent to the other players and the value $g(i)$ submitted to CASSIE. These disclosures are placed on-chain. If any of the revealed items conflict with what was previously sent to another player or CASSIE, the recipient can prove the discrepancy and earn a portion of the penalty. Otherwise, the blockchain now holds all of player i's messages, enabling anyone (including the contract) to replicate i's actions in the protocol and check if the exposed b_i matches ALICE's claim. Only if it does match is player i penalized, and their deposit goes to ALICE and anyone else who helped expose the misconduct. Because this process consumes extra gas, ALICE must also post a deposit with her accusation. If the accusation is false, this deposit is forfeited to compensate i for the gas costs of demonstrating innocence.

All of these rules rest on the premise that only player i can know b_i, so if ALICE possesses b_i, it must have come from i. We justify this by considering every party:

- ALICE cannot simply guess b_i, because b_i is chosen uniformly at random and guessing has only a 2^{-k} chance of success.
- The other players cannot retrieve b_i (as established by the secrecy argument).
- CASSIE obtains only shares of g, not g_i, so she cannot reconstruct b_i unless she collaborates with the t players who hold shares of g_i. By design, t is nearly as large as n, meaning CASSIE would need to control almost all participants. This contradicts the open, decentralized nature of the protocol, which lets anyone on the network join as a player.

Prohibitively Expensive Manipulation. If a participant do not reveal their values correctly or attempts to manipulate the output in any steps, as we have *leak report function* in our contract their deposit of ν units will be forfeited. Since participants are assumed to act rationally, such behavior will not take place.

Uniform Bits as a Quasi-strong Equilibrium. The rewards for participants are directly linked to their utilities in the RBG game (Step 8). The unique quasi-strong equilibrium in this game is achieved when every player chooses their bit uniformly at random. Thus, if the players are rational, they are all incentivized to choose their b_i's uniformly at random. Unlike previous protocols, we do not assume that there is an honest participant who plays uniformly, but instead incentivize this behavior, ensuring that rational players will play uniformly.

Communication Complexity. The overall communication complexity of our protocol is $O(n^2)$, because each player must distribute shares to every other player. However, these exchanges happen off-chain, and we only require $O(n)$ on-chain messages. This design makes the protocol efficient regarding gas costs. In fact, a total on-chain cost of $\Omega(n)$ is unavoidable, since every player at least has to register in the protocol. Thus, our approach uses minimal gas.

VI. EXTENSIONS

The Fortuna protocol, as described so far, is designed to generate a single random bit. However, in practical applications, we often require a full random number instead of just one bit. In this section, we introduce methods to efficiently obtain k random bits, instead of just one.

A straightforward approach would be to run the protocol k times to generate k random bits. However, this method incurs high gas costs. To address this, we present alternative strategies that are significantly more efficient.

Extension 1. Multiple Random Bits in SSS. In Step 3 of our protocol, instead of selecting only one random bit, each player i picks k random bits $b_{i,0}, \ldots, b_{i,k-1}$, where $k \leq t$. These bits are then encrypted using the Goldwasser–Micali encryption scheme as described in Step 3, producing the encrypted values $e_{i,0}, \ldots, e_{i,k-1}$. The corresponding logarithmic values are computed as $\lambda_{i,j} = \log_\gamma e_{i,j}$ for each j.

Next, player i selects additional random coefficients $a_{i,k}, \ldots, a_{i,t-1}$ and constructs the polynomial:

$$g_i(x) = \lambda_{i,0} + \lambda_{i,1} \cdot x + \cdots + \lambda_{i,k-1} \cdot x^{k-1} + a_{i,k} \cdot x^k + \cdots + a_{i,t-1} \cdot x^{t-1} \pmod{p'}.$$

Each player then distributes shares $g_i(j)$ to all other participants $j \neq i$. Additionally, player i commits to their selected bits $b_{i,0}, \ldots, b_{i,k-1}$ using salts $s_{i,0}, \ldots, s_{i,k-1}$ and a predefined hash function.

In Step 5, CASSIE applies Lagrange interpolation to reconstruct the polynomial $g(x)$. Rather than extracting only $g(0)$,

she retrieves the first k coefficients of $g(x)$, as they are derived using the same process as $g(0)$ and are also secret bits. This extension allows CASSIE to generate k secret random bits instead of just one.

To fairly distribute rewards, we define a new per-bit fee $\phi' = \frac{\phi}{k}$. Instead of a single RBG game, we now run k separate RBG games, each with fee ϕ', ensuring that participants are rewarded separately for each bit they contribute. It is evident that this extension preserves all aspects of our protocol's analysis and maintains its validity.

Extension 2. Using the Paillier Cryptosystem. Another way to improve the efficiency of our Fortuna protocol is by using a cryptosystem that supports homomorphic operations on integers rather than individual bits. To achieve this, we replace the Goldwasser–Micali cryptosystem with the Paillier cryptosystem, where the final secret random number r is obtained as the sum of the individual random numbers b_i. The steps and analysis of the protocol change in a predictable manner, as detailed below.

Definition of r. Suppose each player i has submitted an integer b_i which is k bits long. Instead of using bitwise xor, we can use modular sum and define $r' = \sum_{i=1}^{n} b_i$ and $r = r' \mod 2^k$. This preserves the crucial property that r is uniformly distributed if at least one of the b_i's is uniformly distributed.

Key Generation. In Step 1, CASSIE selects two large prime numbers p and q and computes $N = p \cdot q$. She then calculates $\beta = \operatorname{lcm}(p-1, q-1)$ and chooses a random integer χ such that $x := \gamma^{\chi} \bmod N^2$ where $x \in \mathbb{Z}^*_{N^2}$. Finally, she computes the decryption key μ as:

$$\mu = (L(x^{\beta} \bmod N^2))^{-1} \bmod N,$$

where the function $L(x)$ is defined as: $L(x) := \lfloor \frac{x-1}{N} \rfloor$. She then publicly announces N, x, γ, and χ while keeping (β, μ) private. We also define $k := \left\lfloor \log_2 \left(N^2 + 1\right) \right\rfloor$. This is the length, i.e. number of bits, of the generated random number.

Encryption and Secret Sharing. In Step 3, each player i selects a random number b_i from the range $[0, 2^k - 1]$. They then choose a random integer υ_i and compute:

$$y_i = \gamma^{\upsilon_i} \bmod N,$$

ensuring that $\gcd(y_i, N) = 1$. The encryption of b_i using CASSIE's Paillier public key (N, x) is then computed as:

$$e_i = \texttt{Enc}(b_i) = y_i^N \cdot x^{b_i} \bmod N^2.$$

Player i then defines:

$$\lambda_i = \log_\gamma e_i = N \cdot \upsilon_i + b_i \cdot \chi.$$

They apply Shamir Secret Sharing to λ_i by selecting random coefficients $a_{i,1}, \ldots, a_{i,t-1}$ and constructing the polynomial:

$$g_i(x) = \lambda_i + a_{i,1} \cdot x + \cdots + a_{i,t-1} \cdot x^{t-1} \bmod p'.$$

Each player sends shares $g_i(j)$ to the other participants $j \neq i$. Additionally, they commit to b_i using a salt s_i and a predefined hash function.

Obtaining r. In Step 5, the reconstructed value $g(0)$ satisfies:

$$g(0) = \sum_{i=1}^{n} g_i(0) = \sum_{i=1}^{n} \lambda_i = \sum_{i=1}^{n} \log_\gamma e_i = \log_\gamma \left(\prod_{i=1}^{n} e_i \right) =$$

$$\log_\gamma \left(\prod_{i=1}^{n} \texttt{Enc}(b_i) \right) = \log_\gamma \texttt{Enc} \left(\sum_{i=1}^{n} b_i \right) = \log_\gamma \texttt{Enc}(r').$$

All calculations are performed modulo N^2. CASSIE computes:

$$\texttt{Enc}(r') = \gamma^{g(0)} \bmod N^2.$$

She then decrypts using her private key (β, μ) to obtain the secret random number r'. The final random number r is:

$$r = r' \bmod 2^k.$$

Reward Distribution. Since the protocol now produces k-bit numbers rather than single bits, Steps 7 and 8 must operate at the bit level. When selecting a uniformly random number from $[0, 2^k - 1]$, its binary representation consists of independent and uniformly distributed bits. Thus, we convert each player's number into binary and execute Step 7 and Step 8 using these bits. Since there are now k parallel RBG games, we define the per-bit fee as $\phi' = \frac{\phi}{k}$. Instead of a single RBG game, we run k separate games, each with fee ϕ', ensuring participants are rewarded for each bit they contribute. In this game, playing uniformly is a quasi-strong equilibrium, but it is not necessarily the unique equilibrium.

Verifying Properties. We must ensure that this extension satisfies all required properties. Most properties remain unchanged, but two need verification:

- *Uniform Distribution:* Since $r = \sum_{i=1}^{n} b_i \bmod 2^k$, if at least one player selects b_i uniformly from $[0, 2^k - 1]$, then the final random number r will also be uniformly distributed.
- *Uniform Bits as a Quasi-strong Equilibrium:* Since each player's final reward depends on their utility in the RBG game, they are incentivized to pick b_i so that all its bits are uniformly distributed. This leads to a quasi-strong equilibrium, which is stable.

Thus, all required properties are satisfied.

Extension 3: Combining Paillier Cryptosystem with Multiple Random Numbers in SSS. The final extension integrates both of the previous enhancements simultaneously. This means that we follow the same initial steps as in Extension 2. However, in Step 3, instead of selecting a single random number, each participant i chooses k' random numbers $b_{i,0}, \ldots, b_{i,k'-1}$, where $k' \leq t$. These numbers are then encrypted using the Paillier cryptosystem as described in Extension 2, resulting in encrypted values $e_{i,0}, \ldots, e_{i,k'-1}$. The corresponding logarithmic values are then computed as:

$$\lambda_{i,j} = \log_\gamma e_{i,j}, \quad \text{for each } j.$$

Next, participant i selects additional random coefficients $a_{i,k'}, \ldots, a_{i,t-1}$ and constructs the polynomial:

$$g_i(x) = \lambda_{i,0} + \lambda_{i,1} \cdot x + \cdots + \lambda_{i,k'-1} \cdot x^{k'-1} + a_{i,k'} \cdot x^{k'} + \cdots + a_{i,t-1} \cdot x^{t-1} \pmod{p'}.$$

Each participant then distributes shares $g_i(j)$ to all other participants $j \neq i$. Additionally, player i commits to their selected numbers $b_{i,0}, \ldots, b_{i,k'-1}$ using salts $s_{i,0}, \ldots, s_{i,k'-1}$ and a predefined hash function.

In Step 5, CASSIE reconstructs the polynomial $g(x)$ using Lagrange interpolation. Instead of extracting only $g(0)$, she retrieves the first k' coefficients of $g(x)$, as they are computed in the same way as $g(0)$. This extension enables CASSIE to obtain k' secret random numbers instead of just one. To distribute rewards fairly, we define a new per-bit fee: $\phi' = \frac{\phi}{k \cdot k'}$ where $k = \lfloor \log_2 (N^2 + 1) \rfloor$ (as defined in Extension 2). Instead of running a single RBG game, we now execute $k \cdot k'$ parallel RBG games, each with fee ϕ', ensuring that participants receive compensation for every bit they contribute. This extension preserves all the properties and guarantees established in our original protocol. Note that the extensions above keep the communication complexity unchanged but increase the number of random bits generated in one execution of the protocol.

VII. RELATED WORKS

Using Hashes or Timestamps. One common approach for generating randomness in smart contracts is to employ a pseudo-random number generator and use block attributes, such as the block hash or timestamp, as a seed. The goal is to ensure that the seed is not under the direct control of the function caller. However, this method is highly vulnerable since miners can manipulate the block attributes to influence the outcome. Additionally, it does not guarantee a uniform distribution of the seed, making it unreliable. Despite these well-known weaknesses and documented attacks, this method remains widely used in Ethereum smart contracts [34], [35].

Oracles [3], [36]. Another approach involves using external oracles to provide random numbers. Since smart contracts can only access on-chain data, an oracle acts as an intermediary that brings information from external sources to the blockchain. For example, services like Oraclize [36] can relay random numbers generated by external sources such as random.org. However, this approach is fundamentally centralized, as it relies on a trusted oracle operator. The entity controlling the oracle can manipulate the randomness.

Commitment-based Schemes [3], [33], [37]. A different class of RNG methods, including RANDAO [33], one of the most widely used RNGs on Ethereum, employs commitment schemes to ensure participation-based randomness. Here, each of the n players commits to their random number b_i by selecting a nonce z_i and recording the hash $\texttt{Hash}(b_i, z_i)$ on-chain. Since commitments are submitted before any player sees others' values, the process is resistant to direct tampering. Later, all players reveal their b_i and z_i, with penalties for failure to disclose values that match the initial hash. The final random number is computed as the xor of all b_i values. However, in our setting, this method has two critical flaws: (1) the output is not secret, and (2) CASSIE can manipulate the process by refusing to reveal her b_i if the final result is unfavorable. This gives her partial control over the output distribution, even if the protocol is restarted. Beyond these smart contract-based methods, several RNG techniques operate at the blockchain protocol level, primarily for use in proof-of-stake systems. Although these solutions are not directly comparable to ours, we summarize some of the most relevant approaches.

Verifiable Delay Functions. Many blockchain protocols enhance security by leveraging verifiable delay functions (VDFs) [38]–[40]. A VDF is a function f that requires a large number of sequential computational steps to evaluate, preventing attackers from predicting the result in advance. If f is applied to inputs $b_1, \ldots, b_n$, the final output $r = f(b_1, \ldots, b_n)$ remains unpredictable for a long period, making it difficult for any party to manipulate the selection of their b_i strategically. The work [16] obtains a secret RNG protocol based on VDFs.

Verifiable Random Functions (VRFs). Several proof-of-stake protocols, including Algorand [4] and Ouroboros [5], rely on verifiable random functions (VRFs) [41]–[44]. A VRF enables a party to locally compute a random value, which determines eligibility for tasks such as mining the next block or joining a committee. Additionally, the VRF produces a proof that allows others to verify that the computed value is correct. While this approach provides some level of secrecy—since a miner knows their eligibility before revealing it—it does not apply to our setting, where a specific user (CASSIE) must be the sole recipient of the secret random value.

Hydrand [2]. Hydrand employs publicly verifiable secret sharing to generate continuous streams of tamper-proof random numbers. Hydrand provides similar security guarantees and has comparable communication complexity to our method. However, a key distinction is that Hydrand's output is always publicly visible, making it incompatible with our secrecy requirements. Our work builds upon secret-sharing techniques used in Hydrand while integrating homomorphic encryption to ensure that the random output remains confidential. To the best of our knowledge, this is the first blockchain-based RNG that leverages homomorphic encryption.

Other Approaches. Numerous other methods exist for generating publicly verifiable random numbers [21], [45]–[51]. A comprehensive comparison of these techniques can be found in [2, Table I]. However, none of these approaches provide *secret* randomness.

REFERENCES

[1] T. Nguyen-Van, T. Nguyen-Anh, T. Le, M. Nguyen-Ho, T. Nguyen-Van, N. Le, and K. Nguyen-An, "Scalable distributed random number generation based on homomorphic encryption," in *Blockchain*, 2019, pp. 572–579.

[2] P. Schindler, A. Judmayer, N. Stifter, and E. R. Weippl, "Hydrand: Efficient continuous distributed randomness," in *S&P*, 2020, pp. 73–89.

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

[4] J. Chen and S. Micali, "Algorand: A secure and efficient distributed ledger," *Theor. Comput. Sci.*, vol. 777, pp. 155–183, 2019.

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

[7] ——, "Trustless and bias-resistant game-theoretic distributed randomness," in *ICBC*, 2023, pp. 1–3.

[8] K. Chatterjee, A. K. Goharshady, and Y. Velner, "Quantitative analysis of smart contracts," in *ESOP*, 2018, pp. 739–767.

[9] O. J. Scholten, D. Zendle, and J. A. Walker, "Inside the decentralised casino: A longitudinal study of actual cryptocurrency gambling transactions," *PloS one*, vol. 15, no. 10, 2020.

[10] S. M. Gainsbury and A. Blaszczynski, "How blockchain and cryptocurrency technology could revolutionize online gambling," *Gaming Law Review*, vol. 21, no. 7, pp. 482–492, 2017.

[11] C. White, "Betting on blockchain," *Colo. Tech. LJ*, vol. 17, p. 421, 2018.

[12] A. K. Goharshady, A. Behrouz, and K. Chatterjee, "Secure credit reporting on the blockchain," in *Blockchain*, 2018, pp. 1343–1348.

[13] J. Ballweg, Z. Cai, and A. K. Goharshady, "PureLottery: Fair leader election without decentralized random number generation," in *Blockchain*, 2023, pp. 273–280.

[14] J. Ballweg, A. K. Goharshady, and Z. Lin, "Fast and gas-efficient private sealed-bid auctions," in *PODC*, 2025.

[15] P. Fatemi and A. K. Goharshady, "Secure and decentralized generation of secret random numbers on the blockchain," in *BCCA*, 2023, pp. 511–517.

[16] T. Barakbayeva, Z. Cai, and A. K. Goharshady, "SRNG: an efficient decentralized approach for secret random number generation," in *ICBC*, 2024, pp. 615–619.

[17] Z. Cai, S. Farokhnia, A. K. Goharshady, and S. Hitarth, "Asparagus: Automated synthesis of parametric gas upper-bounds for smart contracts," in *OOPSLA*, 2023.

[18] E. Albert, J. Correas, P. Gordillo, G. Román-Díez, and A. Rubio, "Don't run on fumes—parametric gas bounds for smart contracts," *Journal of Systems and Software*, vol. 176, p. 110923, 2021.

[19] S. Farokhnia and A. K. Goharshady, "Reducing the gas usage of ethereum smart contracts without a sidechain," in *ICBC*, 2023, pp. 1–3.

[20] ——, "Alleviating high gas costs by secure and trustless off-chain execution of smart contracts," in *SAC*, 2023, pp. 258–261.

[21] K. Chatterjee, A. K. Goharshady, R. Ibsen-Jensen, and Y. Velner, "Ergodic mean-payoff games for the analysis of attacks in crypto-currencies," in *CONCUR*, 2018, pp. 11:1–11:17.

[22] J. Hoffstein, J. Pipher, J. H. Silverman, and J. H. Silverman, *An introduction to mathematical cryptography*. Springer, 2008.

[23] S. Halevi, "Homomorphic encryption," in *Tutorials on the Foundations of Cryptography*, 2017, pp. 219–276.

[24] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[25] T. Roughgarden, *Twenty lectures on algorithmic game theory*. Cambridge University Press, 2016.

[26] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani, "Algorithmic game theory, 2007," *Book available for free online*, 2007.

[27] S. Goldwasser and S. Micali, "Probabilistic encryption and how to play mental poker keeping secret all partial information," in *STOC*, 1982, pp. 365–377.

[28] L. M. Adleman, "On distinguishing prime numbers from composite numbers (abstract)," in *FOCS*, 1980, pp. 387–406.

[29] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *EUROCRYPT*, J. Stern, Ed., 1999, pp. 223–238.

[30] J. Nash, "Non-cooperative games," *Annals of mathematics*, pp. 286–295, 1951.

[31] R. J. Aumann, "Acceptable points in general cooperative n-person games," *Contributions to the Theory of Games*, vol. 4, no. 40, pp. 287–324, 1959.

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

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

[34] A. Reutov, "Predicting random numbers in ethereum smart contracts," 2018. [Online]. Available: https://blog.positive.com/predicting-random-numbers-in-ethereum-smart-contracts-e5358c6b8620

[35] Ethereum StackExchange, "When can blockhash be safely used for a random number? when would it be unsafe?" 2016.

[36] Oraclize, "A scalable architecture for on-demand, untrusted delivery of entropy," 2019.

[37] G. Brassard, D. Chaum, and C. Crépeau, "Minimum disclosure proofs of knowledge," *Journal of computer and system sciences*, vol. 37, no. 2, pp. 156–189, 1988.

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

[39] B. Wesolowski, "Efficient verifiable delay functions," in *EUROCRYPT*, 2019, pp. 379–407.

[40] K. Pietrzak, "Simple verifiable delay functions," in *ITCS*, 2018.

[41] S. Micali, M. O. Rabin, and S. P. Vadhan, "Verifiable random functions," in *FOCS*, 1999, pp. 120–130.

[42] Y. Dodis, "Efficient construction of (distributed) verifiable random functions," in *PKC*, 2003, pp. 1–17.

[43] T. Jager, "Verifiable random functions from weaker assumptions," in *TCC*, 2015, pp. 121–143.

[44] D. Hofheinz and T. Jager, "Verifiable random functions from standard assumptions," in *TCC*, 2016, pp. 336–362.

[45] S. Azouvi, P. McCorry, and S. Meiklejohn, "Winning the caucus race: Continuous leader election via public randomness," *CoRR*, vol. abs/1801.07965, 2018.

[46] B. Bünz, S. Goldfeder, and J. Bonneau, "Proofs-of-delay and randomness beacons in ethereum," *S&B*, 2017.

[47] 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.

[48] I. Cascudo and B. David, "SCRAPE: scalable randomness attested by public entities," in *ACNS*, vol. 10355, 2017, pp. 537–556.

[49] B. M. David, P. Gazi, A. Kiayias, and A. Russell, "Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake protocol," *IACR Cryptol. ePrint Arch.*, p. 573, 2017.

[50] T. Hanke, M. Movahedi, and D. Williams, "DFINITY technology overview series, consensus system," *CoRR*, vol. abs/1805.04548, 2018.

[51] V. P. Abidha, T. Barakbayeva, Z. Cai, and A. K. Goharshady, "Gas-efficient decentralized random beacons," in *ICBC*, 2024, pp. 205–209.