Secure and Decentralized Generation of Secret Random Numbers on the Blockchain

Pouria Fatemi, Amir Goharshady

Secure and Decentralized Generation of Secret Random Numbers on the Blockchain

Pouria Fatemi and Amir Goharshady
Department of Computer Science and Engineering, Hong Kong University of Science and Technology
pouriafatemi@gmail.com, goharshady@cse.ust.hk

***Abstract*—We propose a trustless blockchain-based protocol for secure and decentralized generation of *secret* random numbers. In our protocol, which can be implemented as a smart contract, a client CASSIE can ask for a secret random number r. The protocol will then allow anyone on the blockchain network to contribute to the process of generating r, but crucially, only CASSIE would be able to see the final result. Additionally, our approach is auditable, i.e. if CASSIE later wishes to announce her secret random number r, she can prove that the output of the process was indeed the claimed value. Finally, we ensure no one else is able to deduce r before CASSIE discloses it.**

A primary use-case of this approach is to enable auditable online gambling and casinos with verifiable and tamper-proof randomness. If CASSIE is a casino, she can ask for a secret random number r at the beginning of the day, and then use it as a seed in a predefined pseudo-random number generator which is in turn used as the source of randomness for all bets during the day. At the end of the day, CASSIE can prove that the seed was indeed generated by our decentralized protocol. Thus, she can prove that the casino had no control over the outcomes of the bets. Moreover, this is a cost- and time-efficient approach as it does not require us to generate a new random number for every bet.

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

I. INTRODUCTION

Random Number Generation (RNG). Generating tamper-proof, unpredictable and verifiable random numbers is arguably one of the most fundamental technical challenges not only on the blockchain, but also in many other distributed systems [1]–[3]. Ideally, the random number should be generated in an open process which is not dominated by any particular node or group of nodes in the network. It should also be tamper-proof since the output is meant to be used in downstream applications which might incentivize dishonest behavior for some of the participants who stand to gain if certain values are chosen. Moreover, the entire process should be transparent and auditable, i.e. anyone should be able to verify that the protocol was executed correctly.

Motivation for RNG on Blockchain. In the context of blockchain, there are three primary motivating factors for an RNG process with the above security guarantees:

1) *Leadership Election*: A large number of vital protocols depend on the randomized selection of a leader or a committee for each round of their execution. The quintessential examples are randomized DAOs [4], [5] and, more importantly, proof of stake protocols which need to select a miner/validator for each block and in which each party's chance of being selected is proportional to their stake in the currency. Common proof of stake protocols, such as Algorand [6] and Ouroboros [7], not only choose the miner for the next block, but also one or more committees which help verify the validity of new blocks.
2) *Probabilistic Smart Contracts [3], [8], [9]*: Since probabilistic programs are much more expressive than their deterministic counterparts, it is theoretically interesting to enable built-in probabilistic behavior for smart contracts.
3) *Gambling and Online Casinos [10]–[12]*: Finally, an online casino which provides betting and gambling on the blockchain needs to prove to its customers and the regulators that the results of the bets are indeed random and not manipulated by the casino itself. The online gambling industry is worth billions of dollars. However, many blockchain-based solutions have failed due to their inability to stop dishonest behavior. Moreover, they have all remained illegal since they are unable to reach the required regulatory bar and provide assurances that the randomness is truly unpredictable and tamper-proof.

Our Setting. In this work, we mainly focus on the third motivation above and provide a secure, trustless, decentralized and tamper-proof protocol to generate ***secret*** random numbers on the blockchain. The main distinguishing factor of our setting is the secrecy, i.e. the output random number is only visible to the initiator of the protocol. Moreover, we provide auditability, meaning that the initiator can prove that a particular value was generated by the protocol. Our protocol is also easy to implement as a smart contract.

Motivation for Our Setting. Suppose a blockchain-based casino would like to prove to its customers and the regulators that the bets are not manipulated. To do this, the casino can generate a fresh random number for each bet using one of the previous RNG approaches. See Section V for a summary of these approaches. However, this would be both inefficient and costly. For example, using the Ethereum-based implementations of these protocols, each bet would take several minutes of time and would cost dozens of USD in gas. This is impractical at the scale of a real-world casino which processes hundreds of thousands of bets per day. It is also impractical for the bettors, who are effectively barred from small bets.

Instead, we propose a different approach: the casino CASSIE first requests a single random number r using an on-chain decentralized RNG at the beginning of the day. She then uses r as her seed for a predefined pseudo-random number generator `Rand`. Each bettor BETTY provides a nonce n_{BETTY} and then the casino computes $\texttt{Rand}(r, n_{\text{BETTY}})$ and uses it as the source of randomness for that particular bet. Note that this would work only if r is secret and exclusively known

to CASSIE. Otherwise, if r is known, then BETTY can cheat and choose an n_{BETTY} that is guaranteed to win her the bet. Unfortunately, previous blockchain-based RNG approaches disclose the random number r publicly. Thus, we need a protocol to generate *secret* random numbers. Finally, at the end of the day, CASSIE should disclose r and be able to prove to everyone that she did not tamper with it. This, together with a record of all bets, is sufficient information to verify that the casino has not cheated throughout the day[1].

Contributions. Using a novel combination of homomorphic encryption and secret sharing, we provide a trustless and decentralized protocol that generates a *secret* random number. Moreover, we show that the generated number is both auditable, i.e. the casino can prove it at the end of the day, and tamper-proof. We also show that it possesses several other desired security properties. To the best of our knowledge, this is the first known blockchain-based RNG protocol with secret outputs. As argued above, our approach enables an efficient and cheap implementation of a blockchain-based casino with provably tamper-proof randomness for the first time. Additionally, we use very lightweight cryptographic primitives and thus our approach's gas usage is negligible. We intentionally avoid using SNARKs and other gas-heavy building blocks. Note that minimizing gas usage is highly important and bounding or reducing it is a well-established research direction [13]–[18].

II. PRELIMINARIES

Our approach is a novel combination of homomorphic encryption and secret sharing. We provide a high-level introduction to these concepts here and refer to [19]–[21] for more details.

Homomorphic Encryption [20]. An encryption scheme is homomorphic if it allows computations to be performed on the encrypted form, without a need to decrypt it first. While homomorphic encryption is a deep subfield of cryptography with many recent advances, our use-case can be handled by the simplest type of partially homomorphic encryption, namely the Goldwasser-Micali Cryptosystem. Specifically, we would like to be able to encrypt a single bit $b \in \{0, 1\}$. Let $\texttt{Enc}(b)$ be the result of encrypting b. Note that this result is not unique since our encryption process is not necessarily deterministic. For any two bits b_1 and b_2, we would like to have the following homomorphic property:

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

where $\oplus$ is the exclusive or. In other words, given the encrypted version of two bits b_1 and b_2, we would like to be able to compute an encryption of their xor, $b_1 \oplus b_2$, without having to decrypt first, so that we can keep the original bits secret.

[1]During the day, when a bet takes place, CASSIE does not disclose r to BETTY, but only $\texttt{Rand}(r, n_{\text{BETTY}})$. If BETTY also has a signature from CASSIE on $(n_{\text{BETTY}}, \texttt{Rand}(r, n_{\text{BETTY}}))$, then any future wrongdoing by the casino can be proven. More specifically, CASSIE discloses and proves r at the end of the day. So, BETTY can check if $\texttt{Rand}(r, n_{\text{BETTY}})$ is tampered with. If so, she can publish the casino's signature and prove the tampering. Of course, we would expect a downstream smart contract to penalize CASSIE for such dishonest behavior, but this is an orthogonal issue and beyond the scope of the current paper.

Goldwasser–Micali Cryptosystem [27]. The Goldwasser-Micali Cryptosystem (GMC) was one of the first provably-secure asymmetric-key encryption schemes. Fortunately, it also has the desired homomorphic property above. Suppose ALICE intends to receive an encrypted message from BOB. The three parts of GMC are as follows:

- *Key Generation:* ALICE picks two large prime numbers p and q and sets $N = p \cdot q$. She then picks an integer x that is not a quadratic residue modulo either p or q. In other words, there is no y such that $y^2 = x \mod p$ or $y^2 = x \mod q^2$. This also ensures that x is not a quadratic residue modulo N. ALICE then publishes the public key (N, x) and keeps the private key (p, q) to herself.
- *Encryption:* To send a single bit b in encrypted form to ALICE, BOB first generates a uniform random integer y such that $\gcd(y, N) = 1$. He then computes

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

  and sends it to ALICE. Note that the encryption result is not unique and depends on BOB's choice of y. If the message is more than one bit long, BOB encrypts each bit separately, using different values of y, and sends the results to ALICE.
- *Decryption:* The encrypted value e is a quadratic residue modulo N if and only if $b = 0$, i.e. when x appears with an even exponent in e. Since $N = p \cdot q$, ALICE decrypts e by simply checking whether e is a quadratic residue modulo both p and q. Given a prime p, an integer e is a quadratic residue modulo p if and only if $e^{(p-1)/2} = 1 \mod p$. This is due to Fermat's Little Theorem. Thus, we have:

  $$\texttt{Dec}(e) := \begin{cases} 0 & \text{if } e^{(p-1)/2} \equiv_p 1 \quad \wedge \quad e^{(q-1)/2} \equiv_q 1 \\ 1 & \text{otherwise} \end{cases}$$

  ALICE can perform this check since she knows p and q. Without this knowledge, checking whether e is a quadratic residue modulo N is a notoriously hard unsolved problem in computational number theory [28].

We remark that GMC has our desired homomorphic property. Suppose BOB encrypts b_1 and b_2 using arbitrarily chosen integers y_1 and y_2, we have:

$$\texttt{Enc}(b_1) \cdot \texttt{Enc}(b_2) = y_1^2 \cdot x^{b_1} \cdot y_2^2 \cdot x^{b_2}$$

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

The next ingredient of our approach is secret sharing:

Shamir Secret Sharing [21]. Let n and t be positive integers and $t \leq n$. Suppose ALICE has a piece of secret information s that she wishes to distribute to n parties. She would like to send each party $i \in \{1, 2, \ldots, n\}$, a "share" s_i, such that any subset of t or more parties can come together and reconstruct the original value s, but any subset of $t - 1$ or fewer parties can find no information about s. Shamir Secret Sharing (SSS) achieves this using the following operations:

[2]Generating such numbers is easy. One can simply choose random values of x and check if they satisfy the requirements. If $p, q = 3 \mod 4$, then $x = N - 1$ would always work.

- *Share Creation:* ALICE chooses a large prime number p. She then generates a random polynomial g as follows:

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

  where s is the secret value and each a_i is a random integer. She then sends each party i the share $s_i := g(i)$.
- *Secret Reconstruction:* The secret is $s = g(0)$. Note that a polynomial of degree $t-1$ is uniquely determined by t points that it passes through. Moreover, for any t points with different x coordinates, there is exactly one polynomial of degree $t-1$ that passes through all of them. This is known as the Lagrange interpolation theorem and is a variant of the fundamental theorem of algebra. Thus, if t parties $i_1, i_2, \ldots, i_t$ come together, they have access to t points of the polynomial g, namely $(i_1, g(i_1)), (i_2, g(i_2)), \ldots, (i_t, g(i_t))$. Therefore, they can reconstruct g using Lagrange's formula:

  $$g(x) = \prod_{j=1}^{t} g(i_j) \cdot \ell_j(x)$$

  where

  $$\ell_j(x) := \frac{x - i_1}{i_j - i_1} \cdots \frac{x - i_{j-1}}{i_j - i_{j-1}} \cdot \frac{x - i_{j+1}}{i_j - i_{j+1}} \cdots \frac{x - i_t}{i_j - i_t}.$$

  The intuition here is that every ℓ_j is equal to 1 at i_j and 0 at every $i_{j'}$ with $j' \neq j$. Having g, they can simply compute $s = g(0)$. However, any set of $t-1$ or fewer parties can have no information about $g(0)$ since any value of $g(0)$ is possible given the information they have.

Our approach will combine GMC and SSS on top of an underlying blockchain to obtain the desired functionality. It is remarkable that the two ingredients above were invented in 1982 and 1979, respectively. Thus, in principle, we could design our protocol in the 1980s if only a shared ledger, like the one provided by the blockchain, existed back then.

III. OUR PROTOCOL

In this section, we first present the desired security properties of a secret RNG and then provide our main protocol.

Open Game and Xor-based Results. Many of the previous protocols for random number generation on the blockchain, such as [3], [29], share the same simple idea to ensure decentralization: Anyone on the blockchain network can take part in the protocol (becoming a "player") and submit a random number sampled from the uniform distribution. Let's say n players choose to take part and player i submits b_i. We let the result of the RNG be

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

and output r as our random number. This ensures that even if only one of the players was honest and chose their b_i uniformly at random, then the overall result r will also follow the uniform distribution.

Challenges. In our setting, suppose a casino CASSIE has asked for the generation of a random number. In addition to the standard requirements of decentralization and uniform distribution of the output, there are several additional challenges we must overcome. Formally, we have to ensure the following security properties:

1) *Secrecy.* Only CASSIE should be able to see r when it is generated. More specifically, even if all but one of the players collude, they should not be able to obtain r[3].
2) *Tamper-resistance.* No group of participants, including or not including CASSIE, should be able to leave the protocol prematurely or otherwise act dishonestly in a manner that affects the result r or its distribution. In other words, r should be tamper-proof.
3) *Auditability.* At a time of her choosing, CASSIE should be able to publicize and prove the r value she received.
4) *Accountability.* If a player i leaks her number b_i to anyone else, this dishonest behavior should be detectable and traced back to i. Moreover, no one other than player i should be able to know b_i. This includes CASSIE.

The reasons behind the accountability property are quite subtle:

- First, we would like to disincentivize the players from disclosing their numbers and thus avoid the catastrophic scenario where every player i is bribed by a bettor BETTY to disclose their b_i and then BETTY can compute r and win all her bets against the casino CASSIE. In practice, each player i would have to put down a deposit that is paid to anyone else who can uncover their bits b_i. Thus, if i discloses b_i to BETTY, then BETTY can cash i's deposit. So, even if only one other player is honest and refuses to collude with BETTY, then BETTY cannot find r, but can still obtain i's deposit – which she will most surely do if she is rational and trying to maximize her own payoff. In other words, a leaky player is taking a stupendous risk, which would lose them their deposit even if only one other player refuses to leak. Note that CASSIE herself can also take part as a player, so it is pretty likely that there would be at least one leak-tight player since the casino would lose a huge amount of money if r is leaked. This being said, our protocol simply ensures accountability, and the detailed amount of the penalty applied to a dishonest player is an orthogonal issue up to the developers.
- Second, we would like to make sure that if r is leaked, then only CASSIE should be blamed and if a b_i is leaked then only player i is blamed. If CASSIE can see the b_i's, she can perform the following attack: Suppose the resulting random value r is not in CASSIE's favor and has caused her to lose many bets throughout the day. She can intentionally leak all the b_i's, and thus r, and then claim that the players had leaked them and demand the generation of a new random number r and cancellation of all the bets performed using the leaked random value. Moreover, we would not be able to distinguish between the case that CASSIE herself leaked and the case where the players leaked.

[3]Of course, if all players collude then it is trivial to compute r since it is simply the xor of their bits b_i. However, since the system is decentralized and anyone can take part as a player, including CASSIE herself whose best interest is to keep r secret, it is safe to assume that there is at least one player who would not collude to leak r.

To simplify the presentation, we assume in the rest of this section that every submitted number is a single bit, but the same ideas work for any number of bits, e.g. we can require each person to submit 32 or 64 bits and all steps of the protocol below remain intact, just applied to each bit separately.

Intuition. In our protocol, each player chooses a bit b_i as is standard in previous approaches and the random result is the xor of all b_i's. However, to ensure secrecy, each b_i has to be encrypted using CASSIE's GMC public key. In other words, suppose that CASSIE does not receive $b_1, \ldots, b_n$, but instead $\texttt{Enc}(b_1), \texttt{Enc}(b_2), \ldots, \texttt{Enc}(b_n)$. She can then compute

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

and decrypt it to obtain r. Unfortunately, we cannot simply write a smart contract in which every player i announces $\texttt{Enc}(b_i)$. While this ensures secrecy and auditability, it does not achieve tamper-resistance. CASSIE herself might be controlling one or more of the players. Thus, she can decrypt the b_i values as they arrive and then choose the bits under her own control to tamper with the result and obtain a desired r. Thus, we have to ensure that every player commits to their own bit before seeing any of the other players' bits and that they cannot change their bit later. This property is obtained using SSS. Additionally, to ensure accountability, CASSIE should not obtain the $\texttt{Enc}(b_i)$'s but only $\texttt{Enc}(r)$.

Our Protocol. We are now ready to present our protocol, which can easily be implemented as a smart contract. Our protocol consists of the following six steps:

Step 1. Initialization. The contract starts with CASSIE calling an initialization function. This is equivalent to her asking for the generation of a random number.

- She should specify the amount of reward that she is willing to pay to the players and deposit it with the contract.
- To enable GMC, CASSIE generates two prime numbers p and q and lets $N = p \cdot q$. She also chooses a primitive root γ modulo N and an integer χ[4] such that $x := \gamma^{\chi} \mod N$ has all the required properties as in Section II, i.e. it is not a quadratic residue modulo either p or q. She then publicly announces N, γ, χ and x by recording them in the contract but keeps the private key (p, q) to herself. Basically (N, x) is the GMC public key and $\chi = \log_{\gamma} x \mod N$. The reason for using the logarithm has to do with the combination of homomorphic encryption and secret sharing and will become apparent in the next steps.
- CASSIE should also fix the parameter t for SSS, as well as a modulus p'. This modulus has to be a prime number that is significantly larger than N.[5] More formally, since CASSIE cannot foretell the number n of players, she actually sets a value for $n - t$. In practice, we would expect $n - t$ to be a small number so that we can be confident that no t of the players would be colluding with CASSIE. We will use this property later in our security analysis.
- Finally, CASSIE also sets time limits, in terms of number of blocks, for each of the following steps.

Step 2. Player Registration. In this step, anyone on the blockchain network can register as a player. Each player has to put down a deposit which will be confiscated if they act dishonestly. When the protocol successfully ends, each player will be paid a proportional share of the reward deposited by CASSIE in the previous step. Their deposits will also be reimbursed. As is standard in blockchain-based protocols, we assume each player's identity is their public key and is recorded in the contract. We use n to denote the number of players who sign up in this step.

Messaging. In the following steps, we assume that the players can have secure and authenticated communication with each other and that every message is delivered to its intended recipient. Thus, when we say player i sends a message m to player j, we assume that m was encrypted and only j could read it. We also assume that it was signed by i, so that j could verify the source of the message and later prove its existence if necessary. In theory, such communication can be implemented in the smart contract, i.e. using the contract as a message ledger and the players' public keys for encryption. However, this would be too costly as every message is added on-chain and incurs gas costs. The standard solution is to have an off-chain communication mechanism between the players and use the blockchain only to enforce the delivery of the messages. More specifically, if the protocol requires i to send a message to j and j does not receive the message in the allocated time off-chain, then j can log an on-chain request for the message with the smart contract and i is obliged to send the message on-chain before a specific deadline. If i fails to do so, they will be disqualified and removed from the set of players and their deposit confiscated and burnt.

Step 3. Encryption and Secret Sharing. Each player i samples a random bit b_i from the uniform distribution. They then choose a uniform random integer υ_i and compute $y_i := \gamma^{\upsilon_i} \mod N$ and encrypt b_i using CASSIE's GMC public key to obtain:

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

CASSIE should not see e_i as this would violate accountability. Moreover, we would like player i to commit to e_i. Thus, we will instead have them commit to $\lambda_i := \log_{\gamma} e_i$ using SSS. Specifically, player i computes $\lambda_i = 2 \cdot \upsilon_i + b_i \cdot \chi$. Note that $\lambda_i \leq 3 \cdot N$. Player i then generates $t - 1$ random numbers $a_{i,1}, a_{i,2}, \ldots, a_{i,t-1}$ and forms the following polynomial:

$$g_i(x) := \lambda_i + a_{i,1} \cdot x + a_{i,2} \cdot x^2 + \ldots + a_{i,t-1} \cdot x^{t-1} \mod p'$$

Basically, λ_i is the secret that is being shared. Player i sends each other player $j \neq i$ a share $g_i(j)$.

Step 4. Aggregation. In the previous step, every player has already received a share from every other player. Moreover,

[4] χ is chosen uniformly at random.

[5] To be more exact, a simple reverse-engineering of the next steps shows that it is sufficient to have $p' > 3 \cdot N \cdot n$ in order to avoid any overflow in our secret sharing and polynomial operations.

each player i has a secret polynomial g_i. Let us define a new polynomial g which is the sum of all the secret polynomials[6]:

$$g(x) := \sum_{j=1}^{n} g_j(x) \mod p'$$

In this step, every player i computes $g(i) = \sum_{j=1}^{n} g_j(i) \mod p'$. Note that she can do this since every $g_j(i)$ is given to her as a share by player j. Player i sends $g(i)$ to CASSIE by recording an encrypted version of $g(i)$, i.e. $\texttt{Enc}(g(i))$, in the smart contract. This particular encryption is done using CASSIE's public key, but it does not necessarily have to use GMC and can be implemented using any other encryption system, too.

Step 5. Obtaining r**.** CASSIE decrypts the messages received in the previous step and obtains all $g(i)$'s. Based on these $g(i)$'s, CASSIE can now compute the secret random number $r = \bigoplus_{i=1}^{n} b_i$. Note that the polynomial g goes through the points $(i, g(i))$. Moreover, it is of degree $t-1$ since every g_i was of degree $t-1$. Thus, as long as CASSIE has received the $g(i)$'s from at least t players, she can reconstruct g using Lagrange interpolation and then compute $g(0)$.

Here comes the magic! At this point, everything just lands in place and works beautifully together so that we have $\gamma^{g(0)} = \texttt{Enc}(r)$. This is because $g = \sum_{i=1}^{n} g_i$. Thus, we have:

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

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

All the equalities above are modulo N. The equalities in the first line are by definition chasing, whereas the second line uses the homomorphic property of GMC. Based on this equation, CASSIE computes $\texttt{Enc}(r) = \gamma^{g(0)} \mod N$ and then decrypts it using her GMC private key to obtain r.

Step 6. Audit. When it comes time for the audit, e.g. after the end of the working day of the casino, CASSIE publicly announces r by recording it in the smart contract. To enable anyone on the blockchain to verify that she did not fake the value of r, she also discloses all the $g(i)$'s and her GMC private key (p, q). Anyone with access to the blockchain can now simulate CASSIE's computation from the previous step and verify that it leads to r. This concludes our protocol.

IV. ANALYSIS OF THE PROTOCOL

We now briefly argue why our approach has the desired properties mentioned for a blockchain-based secret RNG.

Decentralization. Our protocol is open. Anyone on the blockchain network can sign up as a player in Step 2 and their contribution b_i is used in the computation of the final random value $r = \bigoplus_{j=1}^{n} b_j$. Thus, there is no central authority and all nodes on the network have the same permissions in our protocol.

[6]We know that every λ_i is at most $3 \cdot N$, thus their sum is no more than $3 \cdot N \cdot n$. Since $p' > 3 \cdot N \cdot n$, we are sure that $g(0) < p'$ and thus no information about $g(0)$ is lost in this sum.

Uniform Distribution. Since the random value is computed as $r = \bigoplus_{j=1}^{n} b_j$, even if one player, e.g. player i, is honest and provides a b_i that is truly sampled from the uniform distribution, the entire result r will also have the same distribution. Thus, a bettor BETTY who suspects CASSIE might intend to cheat can simply sign up as a player herself and play honestly to ensure r is uniformly distributed.

Secrecy. Every player i produces shares of $\lambda_i = \log_\gamma \texttt{Enc}(b_i)$ in Step 3. So, even if all the other players come together, they can only compute $\texttt{Enc}(b_i)$. However, only CASSIE can decrypt this and obtain b_i itself. Thus, all the other players together are unable to obtain b_i, which proves they cannot obtain r either.

Tamper-resistance. Each player i is committing to their b_i in Step 3. Moreover, their communication with the protocol ends at Step 4. At these points, it is impossible for player i to know the bit b_j submitted by any other player $j \neq i$ who is not colluding. This is true even if player i is CASSIE herself, since obtaining b_j requires a collusion of at least t players plus CASSIE to first get $\texttt{Enc}(b_j)$ from the SSS and then decrypt it using GMC. Thus, as long as there is no collusion of t players plus CASSIE, we have tamper-resistance.

More specifically, player i has no incentive to act dishonestly in Steps 3 and 4, since they have no information about r at this point and acting dishonestly or prematurely leaving the protocol will only cause them to lose their deposit. If a player leaves prematurely in Step 3, we can safely ignore them in the computation of the value since they did not contribute a b_i in the first place (without knowing anything about r). If they leave prematurely in Step 4, it will not affect the computation of r in Step 5 as long as there are at least t players who completed the protocol.

Failures. The only cases when the protocol can fail is if fewer than t players provide the required values in Step 4 or some players provide wrong values, leaving CASSIE unable to apply Lagrange interpolation in Step 5. This will not happen in practice as there is no benefit for the dishonest non-cooperating players and they are penalized by losing their deposits. In the implementation of the smart contract, we can have a fail-safe mechanism where CASSIE reports her inability to apply Lagrange interpolation. She will then unmask all the $g(i)$'s and the players unmask their secret numbers, thus identifying the dishonest parties. Their proof of dishonesty is then provided to the contract which penalizes them. This is similar to the accountability process below. We note again that such dishonest behavior is necessarily taking place before the dishonest parties have any information about r. Thus, we can simply penalize them and rerun the protocol anew.

Auditability. We have a built-in audit in Step 6. CASSIE publishes her GMC private key and anyone can perform the same computations she performed in Step 5 to obtain the same r.

Accountability. Consider a player i who has contributed an input number b_i consisting of k bits. In practice, k is at least 32. Our contract will have a leak report function in which anyone can accuse i of leaking, claim they know b_i and provide it. If such a report is received from ALICE, an accountability procedure will kick in after the end of Step 6, i.e. after r has

already been published and is no longer a secret. The role of this accountability procedure is to check if ALICE's claimed value indeed matches b_i and, if so, penalize player i.

In this process, player i has to provide all of their secret numbers generated throughout the protocol, i.e. $b_i, y_i, v_i, a_{i,1}, \ldots, a_{i,t-1}$, as well as the shares $g_i(j)$ which they have sent to all other players and the $g(i)$ which was sent to CASSIE. All these values will be recorded in the smart contract. If player i lies about any of these values sent to another player or CASSIE, the recipient can prove the lie and receive part of the reward. Otherwise, all messages issued by player i are recorded on the blockchain and now everyone, including the smart contract itself, can simulate player i's part of the protocol and verify whether b_i matches the value provided by the accuser ALICE. If and only if it does match, then player i will be penalized and their deposit will be paid to ALICE and any other recipient who helped uncover the dishonest behavior. Finally, since this process is a bit gas-consuming, an accuser ALICE has to also provide a deposit at the time of accusation which will be confiscated if the accusation were false and used to compensate player i for the gas they use to prove their innocence in the accountability procedure.

Note that the entire accountability procedure above is built upon the assumption that only player i can know and leak b_i and so if b_i was known to ALICE, player i must be guilty. We now prove this by considering every party involved. ALICE cannot simply guess b_i since it is supposed to be chosen uniformly at random by player i and a guess will only have a tiny 2^{-k} chance of being correct. The other players cannot obtain b_i as argued in our proof of secrecy above. Finally, CASSIE only receives shares of g, and not g_i. Thus, she cannot reconstruct b_i unless she colludes with t of the players who received shares of g_i, which we assumed is not possible based on the choice of t. Recall that t is close to n, so this kind of collusion requires CASSIE to control almost all of the players, which is in direct contrast to the openness and decentralization of the protocol which allows anyone on the blockchain network to join as a player.

Communication Complexity. Finally, we note that our protocol has a communication complexity of $O(n^2)$ since every player has to send shares to every other player. However, these communications can be performed off-chain and we need only $O(n)$ on-chain messages. Thus, the protocol is cost-effective in terms of gas usage. A total gas cost of $\Omega(n)$ is unavoidable since every player must at least sign up in the protocol.

V. RELATED WORKS

There is a wide variety of RNG solutions for blockchains in the literature. In this section, we provide an overview of some of these approaches. To the best of our knowledge, none of the previous approaches consider the problem of generating a secret random number and, in all cases below, the generated number is publicly visible to everyone on the blockchain network. While this helps with auditability, it certainly excludes the possibility of secrecy guarantees. We first consider approaches that can be implemented as a smart contract and do not require a change in the underlying blockchain protocol.

Relying on Hashes or Timestamps. To enable random numbers in a smart contract, one can use a pseudo-random number generator, then take one of the attributes of the block containing the transaction, such as block hash or timestamp, and use it as a seed. The idea here is to ensure the seed is not under the control of the function's caller. However, this is a highly vulnerable approach as it allows the miners to take control of the seed. Moreover, it does not guarantee that the seed is chosen from a uniform distribution. Despite many known attacks and vulnerabilities, this approach remains popular in real-world Ethereum smart contracts [30], [31].

Oracles [3], [32]. Another approach is to use oracles. Smart contracts can only access information that is stored on the blockchain. An oracle is simply a service that copies information from elsewhere in the real-world back on the blockchain. For example, one can use a service called Oraclize [32] to access random numbers generated by random.org on the blockchain. This approach is centralized and allows the developer/owner of the oracle to tamper with the output. Thus, it does not satisfy our security requirements.

Commitment-based Schemes [3], [29], [33]. A separate family of approaches, including RANDAO [29] which is currently one of the most widely-used RNGs on Ethereum, rely on commitment schemes and open participation to generate random numbers. Basically, each of the n players first commit to their random number b_i by choosing a nonce z_i and recording the hash of the random number and the nonce, i.e. $h(b_i, z_i)$, in the smart contract. Thus, commitments are done without any information about other players' inputs. Then, every player has to reveal their b_i and z_i and would be penalized for failing to reveal values that lead to the promised hash. The random number is simply the xor of everyone's inputs. If applied in our setting, aside from the output not being secret, these approaches are also vulnerable to tampering by CASSIE. She can wait for all other players to reveal their b_i's and then choose not to reveal hers in case the output is not favorable. This is effective tampering even if the protocol is restarted and a new r is generated, since by not revealing her input, CASSIE can still affect the distribution of r.

Aside from the methods above, there are also many techniques to generate random numbers at the level of the blockchain protocol itself, e.g. so that it can be used in proof-of-stake. Such approaches cannot be directly compared to ours, but we nevertheless provide a summary of some of the most prominent techniques in this direction.

Verifiable Delay Functions. To enable tamper-resistance, many protocols employ a verifiable delay function [34]–[36]. This is a function f whose computation requires a large number of sequential steps and cannot be sped up by parallelization. If such a function is given $b_1, \ldots, b_n$ as input, it takes a long time to compute $r := f(b_1, \ldots, b_n)$, which ensures no party could predict r and thus choose their b_i strategically to tamper with its value.

Verifiable Random Functions (VRFs). Many proof-of-stake protocols, such as Algorand [6] and Ouroboros [7], rely on verifiable random functions [37]–[40]. In their setting, a party can compute the result of the VRF locally. This result usually decides whether the current party is allowed to mine the next

block or be a member of a committee. Moreover, the VRF also provides a proof that can be checked by anyone to ensure its value was computed correctly. This provides a certain level of secrecy in the sense that a miner knows if they are allowed to mine the next block, but their identity remains secret until they provide the VRF output and proof. However, it cannot be used in our setting in which the secret value should specifically be in CASSIE's possession, i.e. the user who has access to the secret value is not chosen randomly.

Hydrand [2]. Hydrand is the closest previous work to our approach and uses publicly-verifiable secret sharing to continuously generate tamper-proof random numbers. It provides very similar guarantees to our approach and has the same communication complexity. However, the main difference is that Hydrand's output is publicly visible and verifiable, which makes it unsuitable for our secrecy requirements. In contrast, we construct a novel combination of secret sharing, which was already used in Hydrand, on the one hand and homomorphic encryption on the other hand to provide secrecy guarantees. To the best of our knowledge, our work is the first use-case of homomorphic encryption in blockchain-based RNG.

Other Approaches. Finally, it is noteworthy that there are many other approaches for the generation of publicly-verifiable random numbers, such as [41]–[46]. We refer to [2, Table I] for a detailed comparison of these works. Crucially, none of these approaches output *secret* random numbers.

ACKNOWLEDGMENTS. The research was partially supported by the Hong Kong Research Grants Council ECS Project 26208122, the HKUST-Kaisa Joint Research Institute Project Grant HKJRI3A-055 and the HKUST Startup Grant R9272.

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*. IEEE, 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] I. Barinov, V. Arasev, A. Fackler, V. Komendantskiy, A. Gross, A. Kolotov, and D. Isakova, "Posdao: Proof of stake decentralized autonomous organization," *SSRN 3368483*, 2019.

[5] Y.-Y. Hsieh, J.-P. Vergne, P. Anderson, K. Lakhani, and M. Reitzig, "Bitcoin and the rise of decentralized autonomous organizations," *Journal of Organization Design*, vol. 7, no. 1, pp. 1–16, 2018.

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

[7] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *CRYPTO*, vol. 10401, 2017, pp. 357–388.

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

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

[10] 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.

[11] 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.

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

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

[14] 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.

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

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

[17] K. Chatterjee, A. K. Goharshady, and Y. Velner, "Quantitative analysis of smart contracts," in *ESOP*, vol. 10801, 2018, pp. 739–767.

[18] 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*, vol. 118, 2018, pp. 11:1–11:17.

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

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

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

[22] A. K. Goharshady, "Irrationality, extortion, or trusted third-parties: Why it is impossible to buy and sell physical goods securely on the blockchain," in *Blockchain*, 2021, pp. 73–81.

[23] M. A. Meybodi, A. K. Goharshady, M. R. Hooshmandasl, and A. Shakiba, "Optimal mining: Maximizing bitcoin miners' revenues from transaction fees," in *Blockchain*, 2022, pp. 266–273.

[24] K. Chatterjee, A. K. Goharshady, and A. Pourdamghani, "Hybrid mining: exploiting blockchain's computational power for distributed problem solving," in *SAC*, 2019, pp. 374–381.

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

[26] K. Chatterjee, A. K. Goharshady, and E. K. Goharshady, "The treewidth of smart contracts," in *SAC*, 2019, pp. 400–408.

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

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

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

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

[33] 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.

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

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

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

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

[38] Y. Dodis, "Efficient construction of (distributed) verifiable random functions," in *PKC*, vol. 2567, 2003, pp. 1–17.

[39] T. Jager, "Verifiable random functions from weaker assumptions," in *TCC*, vol. 9015, 2015, pp. 121–143.

[40] D. Hofheinz and T. Jager, "Verifiable random functions from standard assumptions," in *TCC*, vol. 9562, 2016, pp. 336–362.

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

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

[43] 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.

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

[45] 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.

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