IEEE ICBC'24 1570982060

Gas-Efficient Decentralized Random Beacons

***Abstract*—Decentralized random number generation is a widely-studied problem in the blockchain community and much attention has been paid to the so-called on-chain random beacons, i.e. smart contracts that generate randomness which can in turn be used in other contracts. Following the classical methodology of RANDAO, most on-chain beacons receive inputs from a large number n of participants and then aggregate them to compute a final random output. The aggregation is done in a manner that ensures the final output is uniformly random as long as at least one of the participants acts honestly. While being highly successful in providing security guarantees such as unpredictability and tamper-resistance, a major downside of these beacons is their cost. Since every participant has to call a function in the smart contract to provide their input, the total gas usage to generate a single random number is at least $\Omega(n)$.**

In this work, we propose a novel protocol that offloads most of the on-chain communication between the participants and the smart contract to an alternative off-chain communication with a dealer. This leads to a gas-efficient on-chain random beacon with only $O(1)$ gas usage per generated output. Crucially, our protocol is trustless and the dealer is unable to predict or tamper with the result. We maintain the same security guarantees as previous on-chain beacons, while significantly reducing the gas usage. We also show that our protocol is secure even if all but one of the participants, potentially including the dealer, are malicious.

***Index Terms*—Blockchain, Random Number Generation, Off-chain Computations**

I. Introduction

RNG. A Random Number Generator (RNG), or random beacon, is an important component in many distributed protocols. Its applications range from efficient Byzantine consensus [1] to Proof-of-Stake (PoS) [2]–[4] and trusted setup of cryptographic protocols [5]. Along with the rapid growth of blockchains and decentralized finance applications, many on-chain RNG protocols have been proposed. Examples include [6]–[11].

Smart Contracts and Gas. Ideally, DeFi applications which are usually implemented as smart contracts would want to access fresh random numbers as simply as calling a library function. There are many smart contracts that provide this functionality, most notably RANDAO [10]. These RNG smart contracts require a number of participants in the network to jointly contribute to new random numbers to achieve decentralization. They use different cryptographic techniques to ensure bias-resistance. This includes commitment schemes, publicly verifiable secret sharing (PVSS) [2], verifiable delay functions (VDF) [12]–[14], and homomorphic encryption (HE) [15]. Irrespective of the underlying cryptographic protocols, in all these contracts the participants join the RNG process by making transactions for which they are charged transaction fees in the form of gas. As a result, either the participants have to pay the transaction fees themselves and thus few users would want to participate, undermining the decentralization, or the RNG service has to cover the transaction fees and thus the generation of each new random output becomes highly expensive and cost $\Omega(n)$ gas where n is the number of participants. In this work, we alleviate this issue by a novel protocol inspired by layer-2 solutions.

Off-chain/Layer-2. Many popular blockchains, such as Bitcoin and Ethereum, have a small throughput and high transaction fees. Therefore, there have been many proposals to use off-chain communication to replace on-chain transactions [16]. A quintessential example is that of payment channels such as the lightning network [17] in which a pair of users can use off-chain communication to make a large number of transactions between themselves. An on-chain transaction is required only to open or close the channel. There are many layer-2 solutions for Ethereum, as well, moving the contracts off the main chain and avoiding high gas fees [18]–[20].

Optimistic rollups. Optimistic rollups are a widely-adopted layer-2 solution to increase the throughput of layer-1 blockchains and reduce/avoid gas fees [21]. To process a batch of m transactions, each updating the blockchain state, an optimistic rollup operator aggregates these transactions and publishes the final blockchain state after adding all m transactions. Since the $m-1$ intermediate states are not published, this reduces the workload of layer-1 blockchain from m transactions to only one. However, the operator might publish an invalid final blockchain state. In contrast to ZK-rollups where the operator should also provide a succinct proof of computation, optimistic rollups do not require any such proof of validity. Instead, their correctness relies on other participants to challenge the claims of rollup operators. After the operator publishes the updated blockchain state, other nodes are allowed to challenge its correctness within a specified period. If a challenge succeeds, the update is discarded and the operator is economically punished. If the rollup operator is honest then there will be no disputes and the updates become confirmed. In practice, the batch of transactions is sent to the operator off-chain, while there is an on-chain smart contract to publish the updates and resolve disputes.

Purely Off-Chain RNG. It is possible to implement a random beacon based on an independent distributed system, without an underlying blockchain [11]. However, in order to use the result of this kind of beacon in a smart contract, one would need to create a bridge between the beacon and the blockchain, usually in the form of a centralized oracle. Using an oracle would violate both trustlessness and decentralization.

Motivation for Gas-efficient RNG. Truly decentralized RNG is possible only if we can incentivize many participants to join.

With current methods, generating one fresh random number takes $\Omega(n)$ units of gas, where n is the number of participants. To incentivize participation, we will have to at least cover the participants' gas fees. Thus, current methods have a cost of $\Omega(n)$ per generated random number. A gas-efficient RNG protocol that moves most of the communication off-chain can avoid this unnecessary gas usage and thus make decentralized RNG much more affordable for the end-users.

Our contribution. Inspired by off-chain and layer-2 techniques such as optimistic rollups, we propose to offload most of the transactions of a classical RNG smart contract to off-chain communication. Our contributions are as follows:

- We present a novel RNG protocol that, while remaining trustless, moves most of the messaging off-chain.
- We show that our protocol satisfies all the desired properties of an RNG smart contract, such as tamper-resistance, unpredictability and liveness.
- As in previous random beacon smart contracts, our approach's output is guaranteed to be uniformly distributed as long as there is at least one honest participant. Thus, we are secure against any coalition of all but one of the participants, even if the coalition includes the dealer.
- In contrast to previous approaches, to generate a fresh random number our protocol only consumes constant $O(1)$ gas, whereas previous methods require $\Omega(n)$ units of gas, where n is the number of participants.

II. Preliminaries

Problem Setting. In this work, we present a smart contract R that serves as a decentralized random beacon and performs RNG. We denote the number of participants, i.e. nodes willing to take part in the RNG, by n. We consider a multi-round protocol consisting of many sessions/rounds. Each session generates a fresh random number. Each participant registers in R and can contribute to every session until she withdraws. Participants should also pay a deposit at registration. If a participant honestly follows the protocol until she withdraws, she can claim the deposit back. Otherwise, her deposit is confiscated in the round in which she is caught cheating and she would not be able to contribute to future rounds unless she deposits again. Registration and withdrawal transactions are each sent only once for each participant. Therefore, their gas cost is negligible when amortized over the many sessions to which this participant contributes. Thus, we only consider the cost of one session in our analysis.

RNG with Commitment Schemes and VDFs. Following many previous protocols, for each session we adopt the common approach of RNG based on commit-reveal and VDFs, to make sure the distributed RNG protocol is tamper-resistant. In the first commit stage, each participant i should choose a random value x_i and send the commitment $c_i = hash(x_i, n_i)$ to the smart contract R, where $hash$ is a cryptographic hash function and n_i is a random nonce to hide x_i. In the second reveal stage, i should send x_i and n_i to R. Then R can compute $r = Delay(Combine(x_1, x_2, \ldots, x_n))$, where $Delay$ is a pre-defined verifiable delay function (VDF), ensuring that the result is unpredictable, and $Combine$ can be any operation that combines the participant's inputs, such as xor or hash chaining. In the former case, we will have $r = Delay(x_1 \oplus x_2 \oplus \ldots \oplus x_n)$ and in the latter $r = Delay(hash(x_1, hash(x_2, \ldots, hash(x_n))))$. Similar approaches have been widely used in previous works such as RANDAO [10] and Unicorn [22]. As we will see, our approach uses a Merkle tree [23] to combine the values.

Insufficiency of Rollups. We note that decentralized RNG protocols have stronger security requirements than optimistic rollups. Specifically, it is important that every participant's value x_i must be included in the aggregation function $Combine$, otherwise the RNG result is unreliable and might have been tampered with by the operator. In optimistic rollups, ignoring transactions is not a major issue because the victimized user can reach out to another rollup operator, much in the same way that if a miner refuses to include a transaction in a layer-1 block, another miner will eventually pick it up. However, in our setting, there should be strong guarantees that the operator is not excluding any contributions from the participants. Therefore, even in the presence of optimistic rollups, designing gas-efficient RNG protocols leveraging off-chain communication is an interesting problem.

III. Our Protocol

Our protocol considers two types of users: a dealer and n participants. We assume that the participants have a secure and authenticated channel that can be used to send messages to the dealer. This is a standard assumption in many blockchain protocols that combine on-chain and off-chain communication and can easily be realized in the real-world by using any standard secure internet-based communication method. We also assume that the dealer can announce messages to every participant, either using the same channel, or on a public bulletin board which is visible to everyone. In practice, one might like to add a new type of user, a client, who pays for the costs of the random number generation and the rewards that are provided to the participants. Below, we assume these costs are borne by the dealer, but it is easy to assign them to a separate entity as needed. We first start by explaining our aggregation method and registration procedure and then present the main protocol.

Aggregation Method. Suppose each participant i has provided the input x_i to the protocol. We create a complete binary Merkle tree T with n leaves, all at the same depth, where the i-th leaf contains x_i. We then define $Combine(x_1, \ldots, x_n) = root(T)$ to be the root hash of this Merkle tree. Since T is a complete binary tree, the path from its root to the i-th leaf is uniquely determined by the binary representation of i and has length $O(\log n)$. Finally, as is standard, we apply a fixed verifiable delay function $Delay$ to obtain our final random number $r := Delay(Combine(x_1, \ldots, x_n))$.

Initialization. Our protocol is implemented as a smart contract that supports many rounds of RNG. The dealer deploys the contract on the blockchain and also sets the following values:

- The deposit d that each participant should put down to take part in RNG. This deposit is used to penalize the participant in case of dishonest behavior.
- The cost d^* of challenging a commitment. The use of d^* will become apparent further below.
- The time limits $t_1, t_2, \ldots, t_{14}$ for each of the steps below. Specifically, each step i can start only after time t_{i-1} and must end by time t_i. The smart contract functions mentioned in each step below enforce these time requirements. So, one step's functions are not callable when the contract is in another step. In practice, each t_i can be a timestamp or a block number.

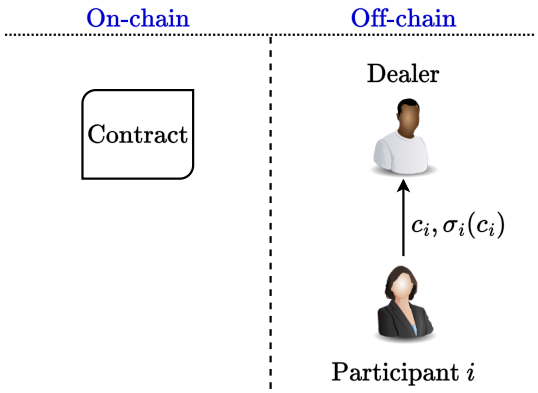
Registration. Our smart contract is an open protocol that allows anyone on the blockchain to sign up as a participant by calling its `register()` function and paying a deposit of at least d. The participant can choose to pay a higher deposit. She remains active and can take part in the RNG sessions as long as her remaining deposit is at least d. A participant can withdraw from the smart contract in between sessions. To do so, she can call the `withdraw()` function. The smart contract records her intention to withdraw and allows her to receive her deposit and any rewards she has accumulated at the end of the current session, or immediately if no session is in progress.

Session Creation. If there is no active session in progress, the dealer can create a new RNG session by calling `new_session()` and paying an amount ρ to the contract. ρ is the reward of the current session and will be divided among participants who take part in the session. Additionally, it is possible to set a fixed/minimum amount for ρ in the initialization phase. The dealer also pays an additional deposit d' to the contract at this point.

Overview of our Protocol. The main idea behind our protocol is quite simple. We take the classical commit-reveal-VDF approach of RNG smart contracts and devise a method to move most of its gas-consuming steps off-chain. In each session, the participants commit their values off-chain to the dealer, instead of providing them on-chain and having to pay for gas. Since off-chain communication is essentially free, this leads to huge gas savings. The dealer then creates a Merkle tree of these commitments, publishes it off-chain to all the participants and only announces its root hash on-chain on the smart contract. Then, we repeat the same procedure for revealing the actual inputs x_i. The participants reveal these to the dealer off-chain and the dealer creates a Merkle tree of x_i's and publishes its root on the smart contract. Finally, we apply a verifiable delay function (VDF) to this root value to get our desired random output. While this idea sounds straightforward, we have to implement it in a trustless manner. This becomes complicated due to the fact that the contract does not have access to the off-chain communication between the participants and the dealer. Thus, we have to implement accountability mechanisms that ensure any dishonest party is caught and punished on-chain. We also need game-theoretic incentives to guarantee that any rational participant/dealer will act honestly. In the presence of rational/honest parties, our approach incurs a tiny amount of gas usage per round and thus generates a fresh random number with a total gas cost of only $O(1)$.

Details of the Protocol. We are now ready to provide a complete description of our protocol. In each session, our protocol has the following steps:

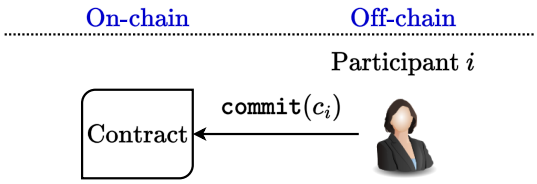
Step 1: Off-chain Commitment. Each participant i chooses a value x_i that is her contribution to the RNG. She also chooses a random nonce n_i. She then computes $c_i = hash(x_i, n_i)$ and sends c_i to the dealer *off-chain*. We re-emphasize that this is using a secure and authenticated off-chain channel, meaning that the dealer not only receives c_i but also the participant's signature $\sigma_i(c_i)$ on c_i[1].



Step 2: On-chain Commitment Request. After the end of the previous step at time t_1, the dealer is expected to have received commitment messages from all participants. If he has not yet received the commitment from a participant i, he calls the function `request_commit`(i) of the smart contract on-chain. This freezes participant i's deposit and locks it in the smart contract until she provides her commitment on-chain. If the dealer does not call `request_commit`(i) in this step, this is seen as an implicit admission that he has received c_i off-chain.

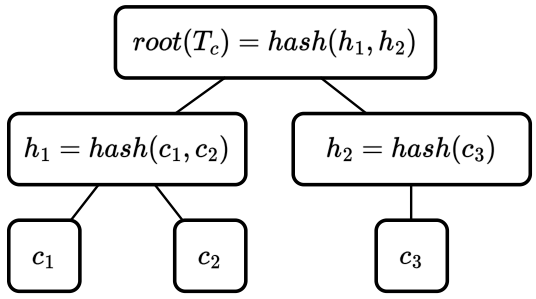
Step 3: On-chain Commitment. Every participant i who was requested to commit on-chain in the previous step has to call the smart contract function `commit`(c_i) by the end of this step and provide her commitment value c_i. This call unfreezes her deposit, meaning that she can receive it back when she withdraws from the contract. If she fails to provide c_i on-chain by the time limit t_3, it is assumed that $c_i = x_i = 0$ and her deposit will remain locked in the contract forever. She would have to pay a new deposit if she wishes to take part in future sessions.

[1]To guard against signature reuse attacks, we assume that the signatures contain a timestamp as well and is $\sigma_i(c_i, \texttt{timestamp})$.

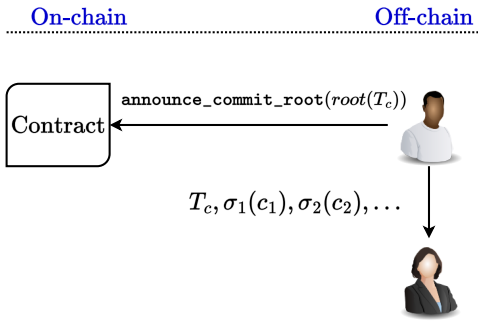


Incentives. We note that the steps above strongly incentivize the dealer and all participants to be honest and handle the commitments off-chain. Each participant i is incentivized to send her commitment in Step 1 since otherwise the dealer will require it on-chain in Step 2, causing i to have to pay a gas fee to provide it on-chain. The dealer is also incentivized to be honest and require an on-chain commitment only if he has not received the value off-chain. This is because calling `request_commit()` is not free and uses gas. Thus, a rational dealer prefers not to call this function as far as possible. Finally, if c_i is not received in Step 1, the dealer is incentivized to require it on-chain in Step 2 since it is otherwise assumed that he has received c_i and he will fail in the following Steps 4-6 and lose his deposit if he does not actually know c_i.

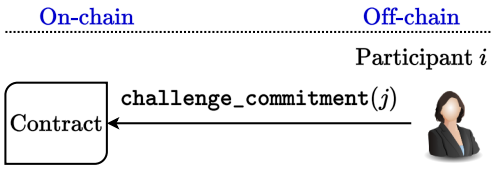
Step 4: Merkle Tree Announcement. Using the c_i values, the dealer creates a complete binary Merkle tree T_c with n leaves, all at the same depth, where the i-th leaf contains c_i. For example, the figure below shows the tree T_c when there are $n = 3$ participants:



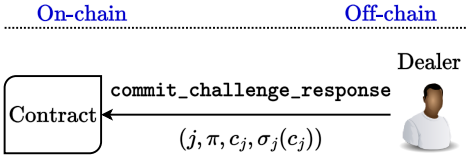
He publishes the Merkle tree off-chain to every participant. For every c_i that was communicated to him off-chain, he also publishes participant i's signature on c_i, i.e. $\sigma_i(c_i)$. Finally, he calls the smart contract function `announce_commit_root`$(root(T_c))$ where $root(T_c)$ is the root hash of the Merkle tree T_c. The contract records this value.



Step 5: On-chain Commitment Challenges. Any participant i who believes the dealer is acting dishonestly can issue a challenge by calling the smart contract function `challenge_commitment`(j), indicating that she believes the commitment of participant j is not correctly included in the tree T_c. For reasons that will become apparent soon, this function call freezes a part d^* of the deposit of the challenging participant[2].



Step 6: On-chain Responses. In response to each challenge, the dealer has to call the smart contract function `commitment_challenge_response`$(j, \pi, c_j, \sigma_j(c_j))$ providing the proof π of the path from the root to the j-th leaf of the Merkle tree. Specifically, π is the standard Merkle proof and contains the hash of every vertex on this path, as well as every child of such vertices. See [23] for details. Additionally, c_j is the commitment of participant j, which is also included as leaf j, and $\sigma_j(c_j)$ is j's signature on this commitment. Thus, π (together with the previously saved root) proves that c_j is included in the tree T_c and $\sigma_j(c_j)$ proves that it came from participant j and was not forged by the dealer.



The contract checks the validity of the proof and signature. If the checks fail, or if the dealer fails to provide the necessary information by the deadline of this step, then his deposit is confiscated and divided among the participants and the protocol ends. In practice, one can divide this unequally, with more funds allocated to the participant who caught the dealer by challenging. Each participant can call `withdraw()` to receive her money.

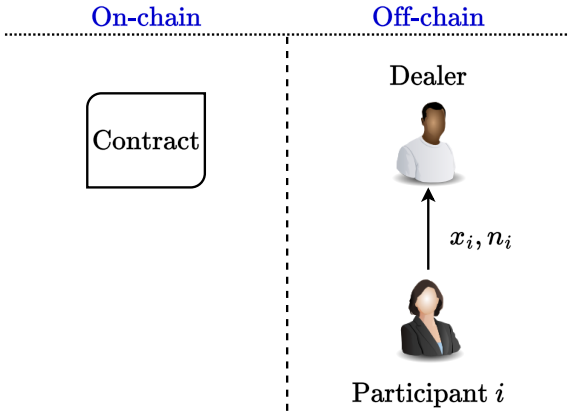
On the other hand, if the dealer successfully responds to the challenge, the contract computes the amount of gas fees g he had to pay for the call to `commitment_challenge_response()` and deducts $\min\{d^*, g/2\}$ from the challenging participant i's deposit, paying it to the dealer, and unfreezes the rest.

Incentives. Let us look at the steps above from the point-of-view of a participant i. If the dealer has published a T_c that excludes a commitment or includes a tampered value, then it is in i's best interest to challenge the dealer, hence receiving

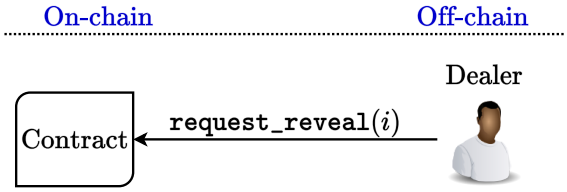
[2]We usually have $i = j$ and one challenges if she sees her own commitment is not included, or if the tree is not published off-chain in the previous step, but our protocol allows challenges with $i \neq j$.

a share of his deposit. If T_c does not contain i's commitment, or if the dealer has not published T_c and i cannot be sure whether it contains her commitment, she is again incentivized to challenge the dealer since she would otherwise lose her own deposit in the following parts of the protocol where she has to reveal. Thus, in the presence of rational participants, any dishonest action by the dealer, be it exclusion, addition or modification of a commitment, will be challenged. On the other hand, there is a disincentive for spurious challenges, since they will ultimately fail and the challenger has to pay part of the gas fees. Now, let us switch to the dealer's point-of-view. It is in his best interest to publish T_c and announce its root honestly to avoid being challenged. A challenge is costly for the dealer in any case, even if he succeeds in responding, since he has to pay part of the gas fees. So, he would prefer to avoid it as far as possible. Note that this is a necessary aspect of our mechanism design. If we charge the entire cost of a failed challenge to the challenger, our protocol will be open to attack by a malicious dealer who does not publish T_c off-chain but publishes a valid root hash on-chain and can thus respond to challenges. With the current design, being honest is every participant's/dealer's optimal strategy.

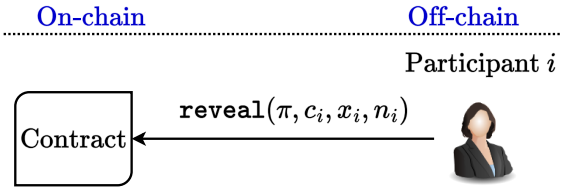
Step 7: Off-chain Revealing. Each participant i sends x_i and n_i to the dealer off-chain. This is similar to Step 1. The dealer checks to ensure that $hash(x_i, n_i) = c_i$. Otherwise, he rejects the message.



Step 8: On-chain Revealing Request. If the dealer has not received the valid (x_i, n_i) of participant i, he calls the smart contract function `request_reveal`(i). This is similar to Step 2. It freezes i's deposit until she responds in the following step. It also freezes d^* units of the dealer's deposit. If the dealer does not call `request_reveal`(i), this is interpreted as his implicit agreement that he has received valid values of x_i and n_i off-chain.

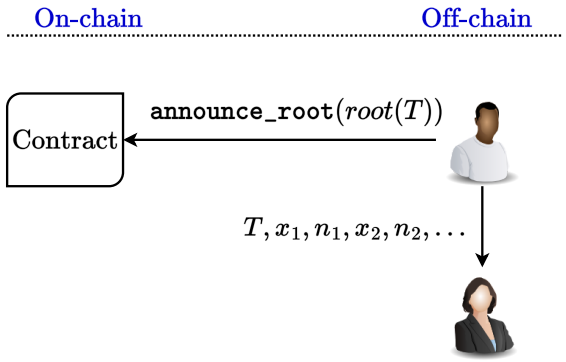


Step 9: On-chain Revealing. Each participant i who was asked to reveal on-chain in the previous step has to call the smart contract function `reveal`(π, c_i, x_i, n_i). Here, π is a Merkle proof showing that the i-th leaf of the tree T_c contains the value c_i. This can be verified by the smart contract since it already knows T_c's root. The contract also verifies that $hash(x_i, n_i) = c_i$. If any of these checks fail, or if i fails to call the `reveal`() function by the deadline of this step, the protocol assumes $x_i = 0$ in the future steps and participant i's deposit is burned, while the dealer's deposit is unfrozen. Otherwise, the contract computes the total gas g that participant i has had to pay for the call to `reveal`() and deducts $\min\{d^*, g/2\}$ from the dealer's deposit and pays it to i. In other words, the participant and dealer share the gas costs.

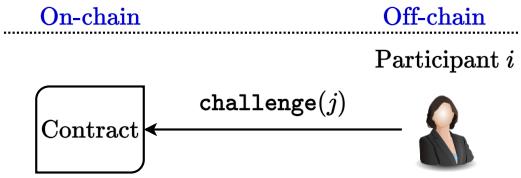


Incentives. The incentive analysis of Steps 7-9 are similar to the previous steps. It is in every participant's best interest to reveal off-chain, hence avoiding a costly (in terms of gas) on-chain revealing. It is in the dealer's best interest to require on-chain revealing for any participant who has failed to reveal off-chain since he would otherwise be unable to follow the next steps of the protocol and loses his own deposit. Moreover, there is no incentive for spurious on-chain reveal requests since the dealer has to pay part of the gas fees.

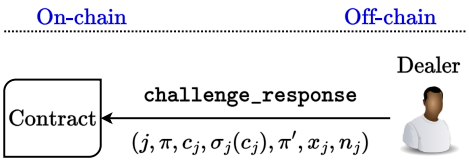
Step 10: Final Merkle Tree Announcement. The dealer creates a new Merkle tree T with n leaves with the i-th leaf containing x_i, i.e. the contribution of participant i. He publishes T and all (x_i, n_i) values off-chain to all participants. This ensures that everyone can check the validity of T. Finally, he calls the smart contract function `announce_root`$(root(T))$ where $root(T)$ is the root hash of T, and will be recorded by the contract.



Step 11: On-chain Challenges. Similar to Step 5, any participant i can challenge the dealer to prove the validity the entry in the j-th leaf of T by calling `challenge`(j). This freezes a portion d^* of the challenging participant's deposit.



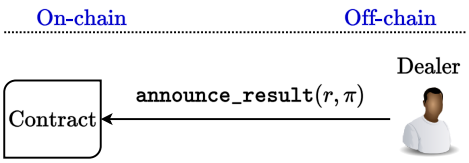
Step 12: On-chain Responses. Similar to Step 6, the dealer has to respond to every challenge issued in Step 11 by calling the smart contract function `challenge_response`$(j, \pi, c_j, \sigma_j(c_j), \pi', x_j, n_j)$. Here, j is the leaf index, π is a Merkle proof showing that c_j was the value at the j-th leaf of our first Merkle tree T_c, $\sigma_j(c_j)$ is participant j's signature proving that he had committed to c_j, and π' is a Merkle proof showing that x_j is the value at the j-the leaf of our second Merkle tree T. The contract checks both Merkle proofs, the signature, and the fact that $hash(x_j, n_j) = c_j$.



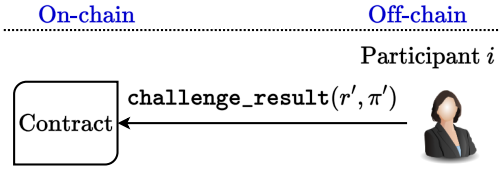
As in Step 6, if the dealer fails to call this function in time or to provide valid values that pass the contract's checks, his deposit is confiscated and divided among the participants and the protocol ends. Otherwise, if he successfully handles the challenge, the contract computes the amount g of gas fees used in responding to the challenge and deducts $\min\{d^*, g/2\}$ from the challenging participant's deposit, paying it to the dealer.

Incentives. The incentive analysis for Steps 10–12 is exactly the same as Steps 4–6 and omitted for brevity.

Step 13: Verifiable Delay Function. The dealer computes $Delay(root(T)) = Delay(Combine(x_1, \ldots, x_n))$. The evaluation of a VDF leads to a result r and a proof of evaluation π. The dealer calls the function `announce_result`(r, π) of the smart contract. r is the generated random number and our RNG output. However, it is not yet finalized. The smart contract only records r and π but does *not* verify them at this stage.



Step 14: VDF Challenge. Given that $root(T)$ is publicly known, every participant can evaluate the VDF and compute (r, π) on her own machine. If a participant realizes that the dealer has cheated in the previous step and announced the wrong value of r, the participant can then call the smart contract function `challenge_result`(r', π'), providing the correct result r' and VDF evaluation proof π' to the contract. At this point, the contract verifies both claims by running the VDF verification algorithm on both (r, π) and (r', π'). The dishonest party, be it the participant or the dealer, is punished by having their deposit confiscated and paid to the other party. If no challenge is made in this step, or if all challenges are unsuccessful, the dealer can receive his deposit d' from the contract.



Incentives. If the dealer cheats in Step 13, every rational participant has an incentive to challenge him in Step 14 and win his deposit. Thus, the rational dealer has no incentive to cheat in the first place.

Participation Rewards. Any participant who completes a session (until the end of Step 14) and whose deposit is not confiscated, would be entitled to an equal share of the reward ρ of the session. All rewards will be paid to her when she later withdraws from the contract by calling `withdraw`$()$.

IV. Analysis

In this section, we first analyze the security of our protocol and then discuss its gas usage and communication and computation complexity. We also discuss the required deposit amounts.

Desired Properties. Much like previous methods such as RANDAO [10], our protocol ensures the following desired properties:

- **Game-theoretic Guarantees of Honesty:** In our protocol, every party is strictly incentivized to act honestly in following the steps and to keep the communication off-chain. The incentives were already covered in the previous section. Thus, rational parties will follow the protocol.
- **Bias-resistance**: In each session, no party can manipulate the random output if he controls at most $n - 1$ participants. The Dealer cannot manipulate the random output either, even if he registers as participants or bribes other participants, as long as there exists one participant acting honestly. This is because all values x_i contribute to $root(T)$ which in turn decides $r := Delay(root(T))$. Every participant is committing to her value x_i before knowing any other participant's x_j. This is due to the commitment scheme. Moreover, every participant is incentivized to reveal since not revealing would cause them to lose their deposit. Finally, even if a participant decides not to reveal their value, they are doing this before

knowing r, which is obtained after applying a VDF. Thus, they cannot strategically manipulate the output.

- **Liveness**: In each session, no party that controls at most $n-1$ participants can prevent the protocol from proceeding through each step and outputting a tamper-proof random number. The only case where the protocol ends without an output is when the dealer is caught cheating. However, this does not happen in practice if the dealer is honest since his cheating would cause him his deposit.
- **Unpredictability**: In each session, no party can know the random output r until after the x_i values are revealed in Step 7 and the VDF is evaluated. As long as the VDF security parameter and the time limits t_i are chosen suitably, the VDF evaluation cannot be completed before t_{12}. Thus, the VDF results are only known in Step 13.
- **Profitability**: Honest participants are guaranteed to make profits in each session. The dealer can also make a profit from customers by providing the RNG as a service.

Times and Deposits. In the initialization phase, the dealer has to decide on values for the time limits $t_1, \ldots, t_{14}$ and the deposits d and d^*. When starting a round, he also puts down his own deposit d'. It is possible to hard-code constraints on the d' already at the initialization phase. In setting these values, the dealer has to ensure the following:

- The deadline of each step should allow for sufficient time for all participants to make the function calls that are potentially needed in that step.
- Evaluating the verifiable delay function $Delay()$ should take strictly more than $|t_{12} - t_6|$ time. In other words, when the participants start revealing their values at the beginning of Step 7, no one should be able to compute the VDF until Step 13. On the other hand, t_{13} should be large enough to allow the dealer and participants to evaluate the VDF in Step 13.
- The deposits should be large enough to (i) cover the gas fees that have to be reimbursed in case of unsuccessful challenges, and (ii) provide practical deterrence. Specifically, since answering the challenges in Steps 6, 9 and 12 require the verification of a Merkle proof that consumes $\Theta(\lg n)$ gas, we must have $d^* \in \Omega(\lg n)$ and $d' \in \Omega(n \cdot \lg n)$.

Gas Usage. The major selling point and contribution of our protocol is that it significantly reduces the gas usage required for one round of RNG. As long as all parties are rational, they will follow the protocol honestly. The incentives for this were outlined in the previous section. This means that in each round the participants only communicate with the dealer off-chain and have to pay no gas fees. The dealer also pays only $O(1)$ in gas fees due to his calls to the contract functions. Specifically, we have the following:

- Session creation requires $O(1)$ gas which is paid by the dealer.
- Steps 1 and 7 and the VDF evaluation in Step 13 are off-chain and incur no gas usage.
- Steps 2, 3, 5, 6, 8, 9, 11, 12 and 14 cause on-chain function calls only if the parties have not acted honestly in the previous off-chain steps. This will severely penalize the parties and would not happen in the presence of our incentives and rational parties. Thus, in practice, the functionality of these steps exists only as a threat to deter dishonest behavior and is never invoked. Thus, they do not consume any gas, either.
- The on-chain part of Step 4 is announcing a single hash to the smart contract and requires $O(1)$ gas, paid by the dealer. The same applies to the on-chain function calls in Steps 10 and 13.

Communication Complexity. Assuming there is a public bulletin board visible to everyone, our protocol has a overall off-chain communication complexity of $O(n)$ which is inevitable because RNG requires fresh inputs x_i from all of the n participants. However, the cost of sending n messages off-chain is negligible compared to the gas costs on programmable blockchains such as Ethereum and can even be considered free. Our contribution is not to decrease the overall communication complexity, but rather to move all but $O(1)$ of the communication off-chain, ensuring that it remains free and does not incur gas costs.

Without the public bulletin board, i.e. in a setting where only messages between pairs of parties are allowed, our communication complexity increases to $O(n^2)$ since the dealer has to send trees of size $O(n)$ to each of the n participants. However, we can further optimize Steps 4 and 10 so that for every player i, the dealer only sends the proof of the i-th leaf. This will reduce the communication complexity, even in the absence of a bulletin board, to $O(n \cdot \log n)$.

V. Conclusion

In this work, we proposed a novel RNG protocol and decentralized random beacon which is implemented as a smart contract and reduces the gas usage for generating a fresh random output from $\Omega(n)$ to $O(1)$ by a clever and trustless mixture of on-chain and off-chain communication between the parties. Secured by this careful design of off-chain/on-chain interoperation, our protocol provides the same security guarantees as previous state-of-the-art RNG smart contracts while being extremely gas-efficient.

References

[1] M. J. Fischer, N. A. Lynch, and M. Paterson, "Impossibility of distributed consensus with one faulty process," *J. ACM*, pp. 374–382, 1985.

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

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

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

[5] A. Kate, G. M. Zaverucha, and I. Goldberg, "Constant-size commitments to polynomials and their applications," in *ASIACRYPT*, 2010, pp. 177–194.

[6] M. Raikwar and D. Gligoroski, "SoK: Decentralized randomness beacon protocols," in *ACISP*, 2022, pp. 420–446.

[7] K. Choi, A. Manoj, and J. Bonneau, "SoK: Distributed randomness beacons," in *S&P*, 2023, pp. 75–92.
[8] G. Wang and M. Nixon, "RandChain: Practical scalable decentralized randomness attested by blockchain," in *IEEE Blockchain*, 2020, pp. 442–449.
[9] M. Krasnoselskii, G. Melnikov, and Y. Yanovich, "No-Dealer: Byzantine fault-tolerant random number generator," in *INFOCOM*, 2020, pp. 568–573.
[10] "RANDAO: A DAO working as RNG of Ethereum," 2019. [Online]. Available: https://github.com/randao/randao
[11] P. Schindler, A. Judmayer, N. Stifter, and E. R. Weippl, "HydRand: Efficient continuous distributed randomness," in *S&P*, 2020, pp. 73–89.
[12] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, "Verifiable delay functions," in *CRYPTO*, 2018, pp. 757–788.
[13] N. Ephraim, C. Freitag, I. Komargodski, and R. Pass, "Continuous verifiable delay functions," in *EUROCRYPT*, 2020, pp. 125–154.
[14] B. Wesolowski, "Efficient verifiable delay functions," in *EUROCRYPT*, 2019, pp. 379–407.
[15] 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 *IEEE Blockchain*, 2019, pp. 572–579.
[16] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais, "Sok: Off the chain transactions," p. 360, 2019.
[17] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2016.
[18] C. Sguanci, R. Spatafora, and A. M. Vergani, "Layer 2 blockchain scaling: A survey," *arXiv preprint arXiv:2107.10881*, 2021.
[19] L. Bousfield, R. Bousfield, C. Buckland, B. Burgess, J. Colvin, E. W. Felten, S. Goldfeder, D. Goldman, B. Huddleston, H. Kalodner *et al.*, "Arbitrum nitro: A second-generation optimistic rollup," 2018.
[20] L. T. Thibault, T. Sarry, and A. S. Hafid, "Blockchain scaling using rollups: A comprehensive survey," *IEEE Access*, 2022.
[21] A. Asgaonkar, *Scaling Blockchains and the Case for Ethereum*, 2022, pp. 197–213.
[22] A. K. Lenstra and B. Wesolowski, "A random zoo: sloth, unicorn, and trx," *IACR Cryptol. ePrint Arch.*, p. 366, 2015.
[23] R. C. Merkle, "Method of providing digital signatures," 1982, US Patent 4,309,569.