Verifying Loot-box Probability Without Source-code Disclosure

Jing-Jie Wang
National Taiwan University
r10944074@csie.ntu.edu.tw

An-Jie Li
National Taiwan University
b09902017@csie.ntu.edu.tw

Ting-Yu Fang
National Taiwan University
r12922036@csie.ntu.edu.tw

Hsu-Chun Hsiao
National Taiwan University
Academia Sinica
hchsiao@csie.ntu.edu.tw

***Abstract*—Loot boxes, a common revenue model in contemporary mobile games, offer players the opportunity to acquire random rewards. However, their fairness has been the subject of numerous disputes worldwide, in part because game players cannot directly observe the logic of the loot-box mechanism. Apple App Store has required app providers to disclose the odds associated with their loot boxes to customers since 2017, and Google Play followed suit beginning in 2019. However, a practical method for allowing the public to verify whether a game complies with its probability statements has not previously been devised. Existing methods, such as source-code audits and statistical examination of player-reported samples, are misaligned with the game companies' interests, and/or may encounter biased samples. Therefore, this paper proposes a verifiable loot-box process without disclosing their source codes. We utilize two cryptographic components, functional commitment and public randomness beacon, to devise a verifiable loot-box process comprising a verifiable loot-box function and a verifiable random source. In particular, we propose two protocols: one for probability verification and the other for loot-box opening. The former allows players to verify the winning probability of loot boxes using publicly verifiable random sources. The latter establishes a mechanism whereby game servers and players can agree on a random input, ensuring that neither party can manipulate the outcome. Our implementations of both these protocols, along with experiments to evaluate their performance, demonstrate that they are practical.**

***Index Terms*—security, cryptography, loot box, verification.**

1. Introduction

Loot boxes are virtual game items commonly utilized to allow players to spend in-game currency and receive a random in-game item in return. Currently, most games incorporating loot-box mechanisms are mobile free-to-play (F2P) games, which allow players to use at no cost but charge fees if the players require supplementary services such as obtaining rare items. The loot-box business model has gained considerable popularity recently and is currently dominant in the gaming industry. In 2023, F2P games were responsible for generating 80% of the total revenue in the gaming industry [1].

Psychological and social science research has demonstrated a strong correlation between individual engagement with loot boxes and the severity of problem gambling [2], [3], [4], [5]. This correlation is even more pronounced among adolescent players [6]. Some studies have suggested that, to disentangle loot boxes from problem gambling, new regulations should mandate that game companies disclose the probability of obtaining rare items in loot boxes. Several countries and platforms have already published regulations regarding loot boxes through legal power or self-regulation (for details, see Appendix A).

Nevertheless, there is currently no practical means of assuring that published probabilities are accurately reflected in the actual game implementations. Numerous disputes have arisen from players' suspicions that the winning probabilities did not match those declared by the game companies [7], [8], [9]. This situation has prompted some players to invest extensively in loot-box purchases and conduct statistical testing to present a proof of misconduct. Unfortunately, employing this method to validate the probability of rare items would be prohibitively expensive for most.

To mitigate the financial burden on individual players, a recent development introduces a third-party verification platform to collect crowd-reported samples for statistical testing [10]. Players can submit loot-box opening records, supported by screen-recorded videos for authentication. The platform then evaluates the accuracy of companies' probability statements within various confidence intervals. However, such third-party verification platforms exhibit three notable drawbacks. Firstly, they depend on a trusted third party for verification, lacking a mechanism for the public to authenticate the data. Secondly, these platforms may be incapable of detecting hidden inputs—factors influencing loot-box outputs that players are unaware of. Given the doubts surrounding the treatment of different players with varying winning probabilities (e.g., [11]), it is crucial to confirm that there are no hidden inputs in the loot-box algorithm. Lastly, these platforms are susceptible to selective data uploading; game companies could choose and submit a disproportionate number of winning records from a larger sample of opening records, resulting in biased probability outcomes.

Another frequently proposed approach is mandating game companies to release their source code for verification. However, this approach presents three significant challenges. Firstly, it introduces the need to verify that the code running on the servers is identical to the open-sourced version. Secondly, even with access to the loot-box implementation,

biased randomness algorithms could skew the outcomes, undermining fairness. Finally, this approach misaligns with the current gaming industry's interests, insofar as loot-box algorithms are considered trade secrets by game companies, which go to great lengths to protect and conceal them from the public. Some studies [12] have partially addressed the first challenge by leveraging blockchain and smart contracts to ensure the loot-box implementation is identical to the open-source version, yet leaving the other two challenges unaddressed. Moreover, such smart-contract-based solutions are impractical for most mobile games today, which are operated from centralized servers controlled by game companies. Consequently, merely disclosing source code is inadequate to guarantee the fair operation of loot boxes.

Therefore, our primary goal in this work is to propose a loot-box method suitable for the current gaming industry, allowing players to verify the probability of loot boxes while minimizing disclosure of information about how the loot-box algorithm works. This ensures public trust in the loot-box results while protecting the interests of gaming companies.

Our core insight to achieve the goal is that the loot-box mechanism in virtual games is analogous to the lottery system in the real world. Just as lotteries involve a lottery drum and a shuffling process, loot boxes require a loot-box function and a random source. Consequently, to achieve a verifiable loot-box process, we need a verifiable function and a verifiable random source. To this end, we leverage advanced cryptographic components, *Functional Commitment (FC)* and *Public Randomness Beacon (PRB)* to construct two novel protocols: **probability verification** and **loot-box opening**. The probability verification protocol allows players to verify the advertised winning probabilities of loot boxes using publicly verifiable random sources. The loot-box opening protocol allows game servers and players to agree on random input, ensuring neither party can manipulate the outcome.

This paper's contributions can be summed up as follows.

1) It examines the problem of loot-box probability disclosure and provides an up-to-date picture of loot-box regulation.
2) It proposes novel probability-verification and loot-box opening protocols as means of verifying loot-box probability statements without disclosing the source code.
3) Its implementation and evaluation of the two novel protocols mentioned above show that they are efficient and practical.

2. Problem Formulation

This section presents our formulation of loot-box functions, followed by the assumptions, threat model, and protocol objectives.

2.1. Loot-box Function

To accurately characterize the functionality of loot boxes, we consulted with gaming companies in the industry. Based on their insights, we identified some essential requirements of loot-box functions. Our findings indicate that current loot-box implementations typically involve a base claim probability, the lower bound probability of receiving a rewarding item with each draw. Gaming companies may dynamically adjust the probabilities slightly higher than the base rate to incentivize continued gameplay. These adjustments are based on factors such as a player's status or the number of prior loot-box attempts. By increasing the chances of obtaining valuable items, companies aim to motivate players to participate in loot-box drawing. In practice, the implementation of loot-box functions involves even more complex designs. For instance, in recent regulations announced in South Korea [13], besides the aforementioned adjustment of winning probabilities based on the number of draw attempts, various other game mechanisms are also addressed.

To clearly express our protocol, we denote a general loot-box function as $f(r, \texttt{others})$, where r represents the random source and `others` denotes the other input parameters, such as the number of attempts made by the player in opening loot boxes.

Formally speaking, let R be the random space, $\mathbb{O}$ the domain space for `others`, and $\{0,1\}$ the event of a player winning the loot-box. The loot-box function is defined as $f : R \times \mathbb{O} \rightarrow \{0, 1\}$.

When a game company states that its winning probability is p_0, we deem the probability statement valid if the actual probability is **greater than or equal to** p_0 for both the domain R and $\mathbb{O}$.

$$\Pr\left[f(r, \texttt{others}) = 1 \middle| \begin{array}{l} r \in R \\ \texttt{others} \in \mathbb{O} \end{array} \right] \geq p_0$$

Specific loot-box mechanisms' probabilities may vary across different `others` input parameters. In such cases, separate testing for each input parameter should be conducted. For example, some games with a "guarantee mechanism" use `count` as an input parameter representing the number of attempts made by the player. As `count` gets higher, the probability increases linearly to 100% when it reaches the "guaranteed count". For such a loot-box design, we can perform probability testing for each `count` to verify whether the actual probability is equal to or higher than the claimed probability.

Below, we will refer to each game player as a *client* and to each game company as a *server*.

2.2. Assumptions

In devising our protocols, we made the following assumptions:

1) The server does not abort, even though it knows the outcome before responding to the client.
2) There is a publicly accessible append-only bulletin board that allows everyone to read and write data. In practice, we can use a blockchain as our bulletin board.

3) The communication channel is secured using TLS connection, and we additionally introduce digital signatures to ensure non-repudiation of messages.
4) Following the common setting of public randomness beacon (PRB) [14], there is at least one honest contributor to the PRB. (PRB is defined in Section 3.1).

2.3. Objectives

To overcome the limitations inherent in prior proposals, we aim to design protocols that achieve the following five objectives:

1) **Correctness and Soundness**: The probability-verification protocol succeeds if and only if the actual winning probability of the loot-box function f is greater than or equal to the stated probability p_0.
2) **Public Verifiability**: The probability-verification protocol allows anyone to verify the correctness of a given probability statement.
3) **Individual Verifiability**: The loot-box opening protocol allows the client to verify that the winning probability is not biased by the server.
4) **Input Transparency**: The loot-box function f should use only transparent input parameters; i.e., no hidden inputs should be involved.
5) **Algorithmic Hiding**: Neither protocol should reveal the evaluation points of f and should not disclose any other information about f.

2.4. Threat Model

In the probability-verification protocol, there are two types of adversaries:

1) The adversarial game server aims to deceive the client by announcing a false probability higher than the actual probability.
2) The adversarial competitor acts as a rival to the game server and seeks to deceive the client by proving a false probability lower than the actual probability.

Both types of adversaries can contribute to the PRB, but they cannot manipulate or predict the randomness contributions of every client. Additionally, the adversarial competitor does not have access to the game server's loot-box function.

After the verification process in the probability-verification protocol, we can ensure that the loot-box function operates with the correct probability. However, the adversarial game server may subsequently attempt to use another fake function to create a misalignment with the verified function.

In the loot-box opening protocol, we consider both server or client may try to predict the outcome for their own benefit, so it is important that no party should be able to bias the result for the sake of fairness. So there are two types of adversaries in the loot-box opening protocol:

1) The adversarial server may want to bias the probability by either not using committed loot-box function or control the randomness input to the function.
2) The dishonest client may want to predict or control the randomness input to the function to affect the outcome.

3. Preliminaries

In this section, we introduce the cryptographic primitives utilized in our protocols. We offer formal definitions and security requirements for public randomness beacon and functional commitment.

3.1. Public Randomness Beacon

A public randomness beacon (PRB) is a service that generates and publishes unpredictable, bias-resistant, and publicly verifiable random values at regular intervals. The main goal of PRB is to provide a reliable source of randomness that cannot be predicted or manipulated by any party, including the service provider.

The PRB we refer to in this work is modeled as follows:

- `PRB.Setup`$(1^\lambda, I)$: Given the security parameter λ and beacon interval I, outputs an implicit public parameters. This should be a randomized algorithm.
- `PRB.Contribute`(x): Inserts a contributed randomness x to the PRB.
- `PRB.Eval`$(\{x_1, ..., x_n\}) \rightarrow r$: Given a set of randomness contributions $\{x_1, ..., x_n\}$, outputs r as the outcome of this beacon interval.
- `PRB.Verify`$(x, r) \rightarrow \{0, 1\}$: Given a randomness contribution x and randomness outcome r, outputs a decision bit $\{0, 1\}$ representing if x is involved when generating r.

In addition, a PRB should possess the following properties:

1) **Unpredictability**: Before `PRB.Eval` returns, it is computationally infeasible to predict the output of `PRB.Eval`.
2) **Bias-Resistance**: There is no adversary that can manipulate the output of `PRB.Eval` to its own advantage.
3) **Verifiability**: Honest contributors can verify that the PRB's output is both unpredictable and bias-resistant.

We surveyed several publicly-verifiable randomness beacons constructions, including those based on public verifiable secret sharing (PVSS) [15], [16], [17], verifiable random function (VRF) [18], Byzantine fault-tolerant (BFT) state machine replication [19], homomorphic encryption [20], and verifiable delay function (VDF) [14], [21]. Each of the various existing PRB constructions has distinctive trade-offs among factors such as evaluation complexity, and verification complexity, and their trust models also differ (see Section 8.2 for more details).

Specifically, we opted to use the HeadStart beacon [14], as it enables contributors to verify unpredictability

and bias-resistance without relying on assumptions like an honest majority. Additionally, we selected HeadStart due to its low verification complexity, which is expressed as $O(L \times \text{polylog}(T) + \log C)$, where L represents the number of beacon outcomes after a user's contribution, T is the period of each such outcome, and C is the number of contributions.

3.2. Functional Commitment

As defined by Boneh et al. in BNO21 [22], a functional commitment (FC) scheme allows a committer to make a commitment to a secret function f, and subsequently prove that $y = f(x)$ for public x and y while keeping all other details about f concealed.

Here, we slightly modified BNO21's notation, and transformed the evaluation protocol into a non-interactive one using the Fiat-Shamir heuristic for ease of explanation. A non-interactive FC has the following four algorithms.

- $\texttt{FC.Setup}(1^\lambda, N)$: Given the security parameter λ and upper limit of the number of gates N, outputs an implicit public parameters. This should be a randomized algorithm.
- $\texttt{FC.Commit}(f, r) \to c$: Given a secret function f and a randomness r, outputs a commitment c to f. This should be a deterministic algorithm.
- $\texttt{FC.Eval}(f, r, x, y) \to \pi$: Given a function f, another randomness r, an evaluation point x, and a claimed evaluation value y, outputs a proof π that convinces the verifier that $f(x) = y$.
- $\texttt{FC.Verify}(c, x, y, \pi) \to \{0, 1\}$: Given a commitment c of function f, an evaluation point x, a claimed evaluation value y, and a proof π, outputs a decision bit $\{0, 1\}$ denoting whether $f(x) \stackrel{?}{=} y$.

A **secure** non-interactive FC have the following properties:

1) **Binding**: It is computationally infeasible to find distinct functions f_1, f_2 such that $\texttt{FC.Commit}(f_1, r_1) = \texttt{FC.Commit}(f_2, r_2)$ for some r_1 and r_2.
2) **Hiding**: For commitments c_1, c_2 derived from two distinct functions, c_1 and c_2 are computationally indistinguishable.
3) **Completeness**: For all commitments c generated by `FC.Eval`, the verification `FC.Verify` always return 1.
4) **Evaluation Zero-knowledge**: The proof π reveals nothing other than the evaluation $f(x) = y$.
5) **Knowledge Soundness**: A valid evaluation proof can only be produced by provers possessing knowledge of the secret function f.
6) **Evaluation Binding**: It is computationally infeasible to construct valid evaluation proofs for different evaluation values $y_1 \neq y_2$ on the same input x.

In our scenario, the **hiding** property is the most important, because it ensures that the functional-commitment scheme does not disclose any information about the secret function.

We use both BNO21 [22] and KZG10 [23] as a FC scheme in our implementation for evaluating performance. Please refer to Section 7 for implementation details.

4. Proposed Solution

In this section, we begin by providing an overview of our proposed solution and emphasizing its core ideas. We then delve into the detailed explanation of two protocols: probability-verification protocol and loot-box opening protocol. Their alignment with our objectives will be discussed further in Section 6.

4.1. Overview

As illustrated in Figure 1, the main problem with the current loot-box mechanism is that the game server may provide a function with worse probabilities, while the user has almost no way to verify.

First, to address this, our core idea is to use a PRB to generate publicly verifiable test data for verification (step ❶). Based on the secure randomness generated by the PRB, we can perform statistical analyses to determine whether the server's provided probabilities are correct. However, an adverserial server might still substitute the verified function with another one to cheat.

Next, we apply functional commitment to ensure that the server cannot modify the function after it has been verified (Step ❷). This also prevents attacks from adversarial competitors, as described in Section 2.4, who might generate fake results to make the function appear less good. Since only the game company can provide results with valid functional commitment proof, such attempts by competitors would not succeed. Nevertheless, the server could still introduce bias into the final result by controlling the randomness during the function's execution.

Finally, we introduce a trusted randomness source during the loot-box execution (step ❸). Unlike loot-box verification, loot-box openings require real-time service and cannot use mechanisms like PRB that have a waiting period. Therefore, for the random source input required in loot-box openings, we design a collaborative approach between the server and client. It prevents both parties from predicting each other's provided random source.

Given that users often request loot-box openings multiple times, we further enhance our design by using a hash chain as the server's random source. The server provides the user with the last node of the chain, and subsequently, all the nodes of the chain can serve as the server's random source in reverse order without requiring additional proof.

4.2. Probability-verification Protocol

The most challenging aspects of designing our probability-verification protocol were 1) ensuring that the implementation details of the opening function f remain confidential (Algorithmic Hiding), and 2) verifying that sample

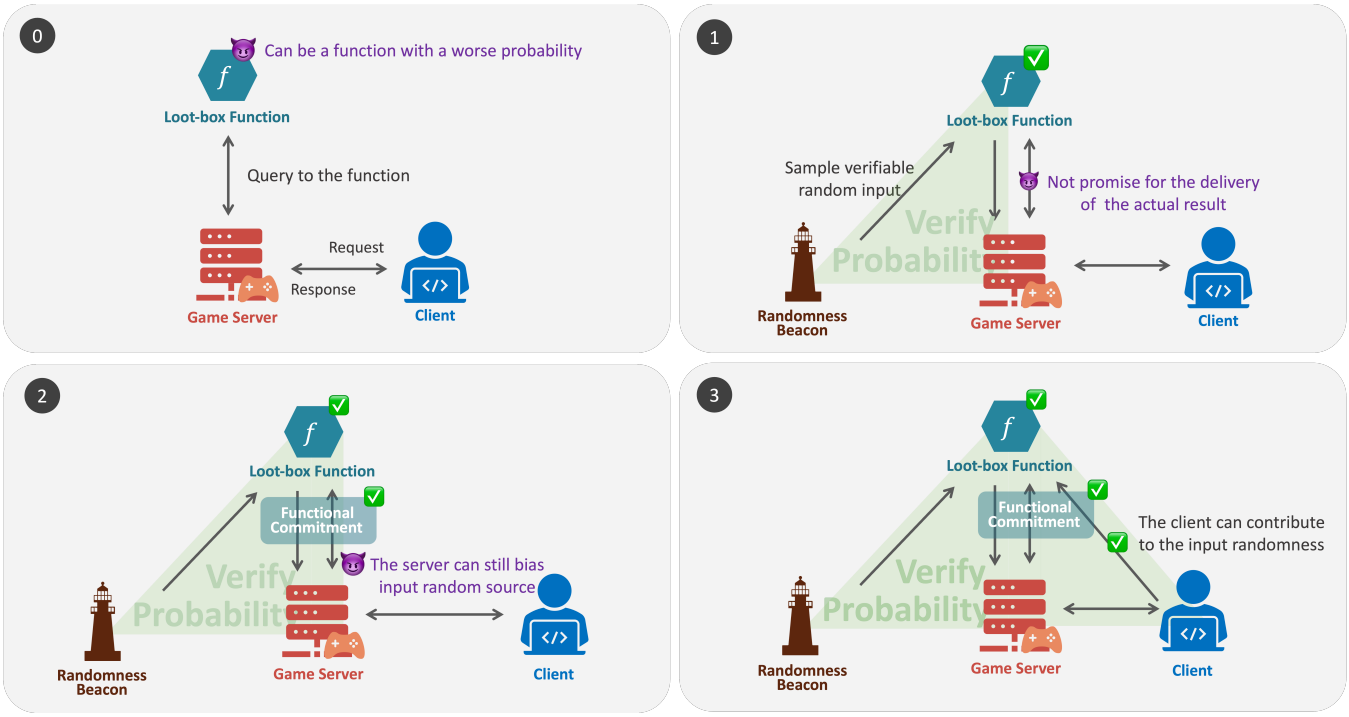


Figure 1: A high-level overview of our proposed verifiable loot box process. It illustrates the evolution of our solution system as each core component is introduced.

input parameters are genuinely generated from randomness. To address the first of these challenges, we employed FC, and to address the second, we leveraged a PRB.

The probability-verification protocol can be broken down into four distinct phases: setup, randomness contribution, evaluation, and verification. Each is discussed in turn in its own subsection below. Additionally, a flow chart of it can be found in Figure 2a.

Phase 1: Setup. In this phase, the server first runs the setup algorithm for each cryptographic primitive, including `PRB.Setup` and `FC.Setup`. Secondly, the server designates the number of beacon intervals, denoted by n, and indicates via the bulletin board that it will utilize the random output generated after n intervals from the moment it publishes the commitment. Third, the server designates a *mapping function* f_M, i.e., how randomness is mapped onto test data. More specifically, this mapping function takes public randomness r as its input and generates the corresponding test data as output. Formally, it can be expressed as

$$f_M(r) \rightarrow \{(r_1, o_1), ..., (r_m, o_m)\}$$

where $r_i \in R$ and $o_i \in \mathbb{O}$ for $1 \leq i \leq m$.

The mapping function must generate test data that reflects the real-world distribution of the underlying loot box. For instance, if the parameters of $\mathbb{O}$ include game levels 1-100, the function should uniformly sample from these levels. Since $f_M(r)$ is public, players can verify whether the distribution aligns with their actual gameplay experience. By ensuring that the mapping function is consistent with the real-world distribution, the scheme can produce sufficient test data without needing to simulate gameplay. However, for players who prefer not to rely solely on the mapping function, our method also supports using data from actual gameplay as an independent, secondary source. For further details, please refer to Section 5.3.

Finally, the server commits the loot-box opening function f to a commitment c by $\texttt{FC.Commit}(f, r) \rightarrow c$, where r is sampled from randomness. Subsequently, the server publishes the public parameters, the interval number n, the mapping function f_M, and the commitment c on the bulletin board.

Phase 2: Randomness Contribution. In this phase, the clients examine the validity of the setup data published by the server, including public parameters, n, f_M, and c. Then, interested clients can sample their local randomness and contribute through `PRB.Contribute`. Here, it is important to note that clients who have already contributed their local randomness can skip this step, as the PRB protocol ensures unpredictability and bias-resistance for such clients.

Phase 3: Evaluation. In this phase, the server first obtains the randomness outcome r of the n^{th} PRB result counted from the setup phase, then uses the mapping function f_M to obtain test data, i.e.,

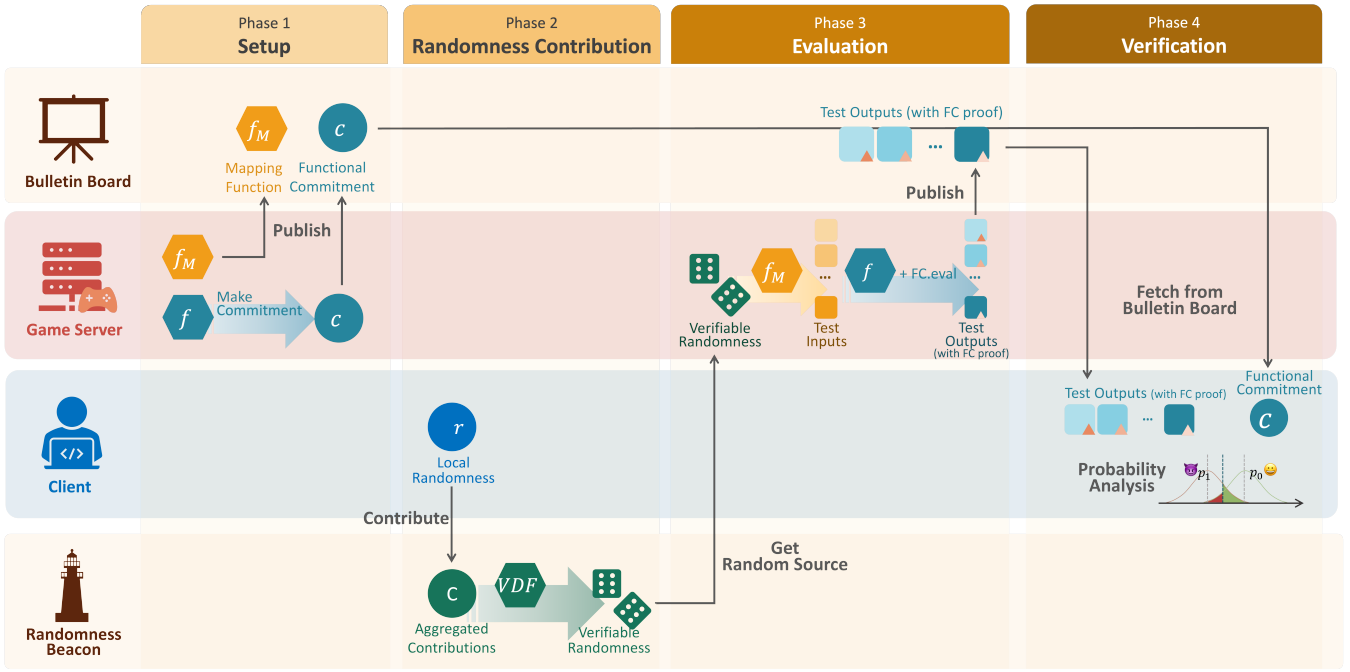
$$f_M(r) \rightarrow \{(r_1, o_1), ..., (r_m, o_m)\}$$

where $r_i \in R$ and $o_i \in \mathbb{O}$ for $1 \leq i \leq m$.

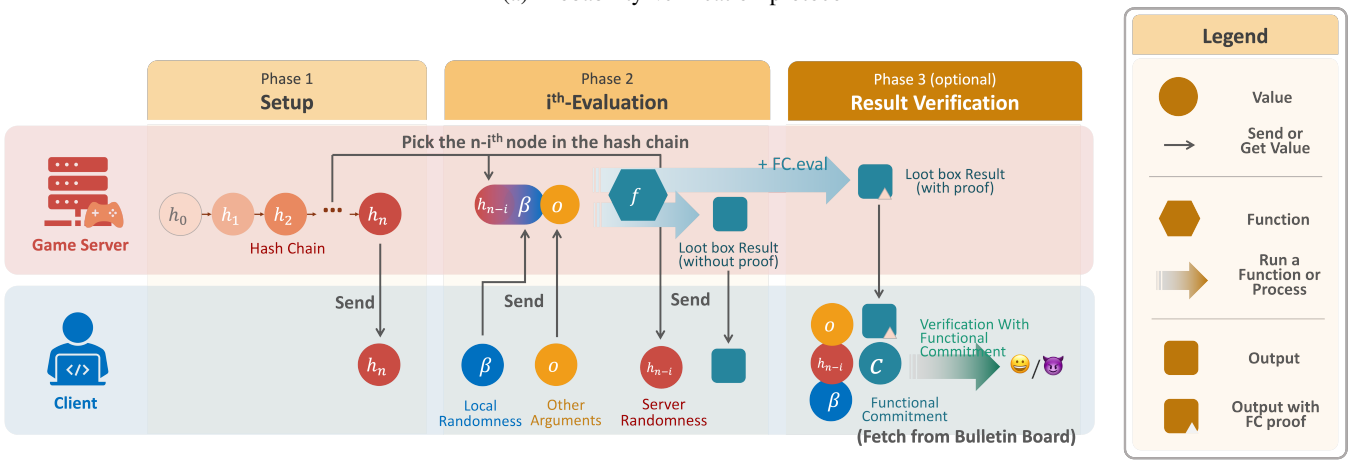
Then, the server computes the outputs of the function f and evaluation proofs

$$f(r_i, o_i) \rightarrow y_i$$

$$\texttt{FC.Eval}(f, (r_i, o_i), y_i) \rightarrow \pi_i$$



(a) Probability verification protocol



(b) Loot-box opening protocol

Figure 2: The flowchart of protocols

for $1 \le i \le m$.

Lastly, the server publishes the evaluations along with their proofs $(y_1, \pi_1), ..., (y_m, \pi_m)$ on the bulletin board.

Phase 4: Verification. In this phase, the client first retrieves the randomness r from the PRB and invokes `PRB.Verify` to verify the validity of r. If the verification fails, the client aborts. Next, the client maps r onto test data using the mapping function f_M and obtain $(r_1, o_1), ..., (r_m, o_m)$. Then, the client retrieves the evaluations from the bulletin board and verifies them via

$$\texttt{FC.Verify}(c, (r_i, o_i), y_i, \pi_i) \stackrel{?}{=} 1$$

for $1 \le i \le m$.

Finally, the client can use those verified evaluations to ascertain whether the probability statement is accurate.

The outcome of the probability verification protocol consists of randomly sampled data. Clients can utilize this data to verify the accuracy of the probability statement. We delve into the statistical tools facilitating this verification in Section 5.

4.3. Loot-box Opening Protocol

The key purpose of our loot-box opening protocol is to provide individual verifiability: i.e., to allow each client, after the result-verification phase, to confirm that the server has not biased the odds of winning in any way.

This protocol comprises three phases: setup, evaluation, and result verification. Each is described in detail in the following paragraphs. Additionally, a flow chart of it can be found in Figure 2b.

Phase 1: Setup. In the setup phase, the server samples a randomness α_0 and generates a hash chain of length N, i.e.,

$$\{\alpha_i | \alpha_i \leftarrow H^i(\alpha_0), 1 \leq i \leq N\}$$

where H is a cryptographic hash function. The server then sends the client the last element α_N of the chain.

Phase 2: Evaluation. In the evaluation phase, the client first samples a local randomness β and prepares the input parameters `others`, which may include metadata about the player such as their game level, number of loot-box openings, and so on. The client then sends $(\beta, \texttt{others})$ to the server.

When the last-opened hash chain position is α_i, the server uses $\alpha_{i-1} \parallel \beta$ as the random source. Formally, this can be expressed as

$$f(\alpha_{i-1} \parallel \beta, \texttt{others}) \rightarrow y.$$

The server then generates the evaluation proof:

$$\texttt{FC.Eval}((\alpha_{i-1} \parallel \beta, \texttt{others}), y) \rightarrow \pi.$$

Finally, the server sends back (α_{i-1}, y, π) to the client.

Phase 3: Result Verification (Optional). The result verification phase is optional for clients, as verification is only necessary when they have doubts or concerns regarding the fairness of the loot box results. After receiving (α_{i-1}, y, π) from the server, the client first verifies the validity of α_{i-1} using

$$H(\alpha_{i-1}) \stackrel{?}{=} \alpha_i$$

then verifies the validity of the evaluation proof:

$$\texttt{FC.Verify}(c, (\alpha_{i-1} \parallel \beta, \texttt{others}), y, \pi) \stackrel{?}{=} 1.$$

If both verification succeed, the client can be confident that the server has not biased the odds of loot-box opening.

This protocol uses a hash chain to ensure that the server cannot manipulate the randomness after the setup phase, and it is computationally infeasible to find α_{i-1} given α_i, which prevents dishonest clients from biasing the outcome.

A practical consideration is the server may need to repeat the setup phase if the hash chain is exhausted. While increasing N can reduce the likelihood of this happening, it also leads to higher space usage. To address this, the server can choose to store only the initial element α_1, and compute $\alpha_i \leftarrow H^i(\alpha_0)$ on demand when a client requests opening a loot-box. Another approach would be saving a subset of breakpoints $\{\alpha_1, \alpha_{101}, \alpha_{201}, \dots\}$ to strike a balance between computational and storage requirements.

5. Applying the protocols

This section delves into the practical applications of our protocol, covering how to validate the accuracy of disclosed probabilities (§5.1), determine the appropriate sample size (§5.2), and devise alternative approaches to probability verification that address other demands in real-world scenarios (§5.3).

5.1. Hypothesis Testing

To rigorously assess whether a probability statement holds true, we first specify a null hypothesis, H_0, and an alternative hypothesis, H_1. In our scenario, the null hypothesis is defined as the condition in which the claimed probability holds true, i.e., $H_0 : p \geq p_0$. The alternative hypothesis is defined as the condition in which the claimed probability does not hold true, i.e., $H_1 : p < p_0$.

With a predetermined significance level α, we aim to perform a hypothesis test on the sample data to calculate the probability of observing the test results, assuming the null hypothesis is true. We reject the null hypothesis if this probability is less than the chosen α. This rejection implies that the sample data provides strong evidence supporting the alternative hypothesis, which suggests that the actual probability of winning is likely lower than the claimed probability advertised by the game company.

From the central-limit theorem, the sample mean of independent and identically distributed (i.i.d.) random variables converge to a normal distribution. Let us assume that the loot-box results are i.i.d. Binomial distribution. We can approximate the sample mean as

$$X = \frac{X_1 + \dots + X_n}{n} \sim N(p_0, \sqrt{\frac{p_0(1-p_0)}{n}})$$

where X_i is the random variable of each sample result for each $1 \leq i \leq n$. To calculate the critical value when we reject the null hypothesis, we can use the z-score table to find z_α, such that

$$Pr(\hat{p} \leq p_0 - z_\alpha \sqrt{\frac{p_0(1-p_0)}{n}}) < \alpha$$

where $\hat{p}$ is the winning probability in the sample data. The critical value for this hypothesis test is computed as $p_L = p_0 - z_\alpha \sqrt{\frac{p_0(1-p_0)}{n}}$.

For example, when the claimed probability p_0 is 0.3, the sample size n is 200, significance level $\alpha = 5\%$, then $z_\alpha = 1.645$ by looking up to the z-score table. We calculate the critical value

$$p_L = p_0 - z_\alpha \sqrt{\frac{p_0(1-p_0)}{n}} = 0.3 - 1.645 \sqrt{\frac{0.3 * 0.7}{200}} = 0.247$$

In this case, we reject the null hypothesis if the sample's winning probability $\hat{p} \leq p_L$ is 0.247, i.e., the claimed probability is deemed wrong.

5.2. Determining the Number of Test Inputs

To determine the proper size of test dataset, we utilized statistical power analysis. This involved, first, estimating the false-negative probability for an instance of the alternative hypothesis. Assuming the real winning probability is $p_1 < p_0$, we calculated the probability of rejecting the null hypothesis, called statistical power $(1 - \beta)$. Usually, we expect the statistical power to be greater than 80%, i.e., $\beta < 20\%$. To achieve this, we used the z-score table again to find z_β

corresponding to $1-\beta$. Specifically, we found an n that satisfying the inequality:

$$p_1 + z_\beta \sqrt{\frac{p_1(1-p_1)}{n}} \leq p_L = p_0 - z_\alpha \sqrt{\frac{p_0(1-p_0)}{n}},$$

where L.H.S is the accumulative probability $1-\beta$ of the sample mean random variable having the Bernoulli distribution of probability p_1.

For example, let us assume that the claimed probability p_0 is 0.03, that the actual probability p_1 is 0.015, that α is 0.05, and that β is 0.2. We first use the z-score table to establish that $z_\alpha = 1.645$, $z_\beta = 0.845$. The above inequality then becomes:

$$0.015 + 0.845\sqrt{\frac{0.015 * 0.985}{n}} \leq 0.03 - 1.645\sqrt{\frac{0.03 * 0.97}{n}}$$

$$\rightarrow n \geq 653.068$$

In other words, if our sample size is larger than about 653, there will be more than an 80% probability of rejecting the null hypothesis, given the actual probability $p_1 = 0.015$ and the claimed probability 0.03. In real-world circumstances, self-regulation associations, government, and/or the player community could reach a consensus on the selection of a specific value for p_1, such as $p_1 = 0.9p_0$.

5.3. Alternative Approaches to Probability Verification

Dry-run API Verification. In real-world scenarios, the server may take advantage of setting special-case conditions, causing a much lower probability for specific clients. For example, game companies may design a function that exhibits a different probability distribution for a specific subset of users, which is unlikely to be detected when the supplementary data $\mathbb{O}$ are randomly generated. To address such situations, we can utilize a dry-run API provided by the server, allowing verifiers to test the system without incurring actual payments. Substituting this dry-run API for the mapping function in the Probability-verification Protocol, we can sample the distribution for specific clients and use the statistical testing discussed above to verify an individual client probability. In practice, it is essential to set an upper limit on the invocation times of the dry-run, as unlimited use could potentially allow verifiers to reverse-engineer the private function. Relying on FC, this method ensures precise winning probability verification of each client but imposes an additional burden on the server. Therefore, we consider it as an optional extension, requiring no additional preprocess from our proposed protocols but providing a robust arbitration basis in case of disputes.

Third Party Verification. Although our method can operate without the need for additional third-party involvement, in practice, it may be beneficial to consider the introduction of third-party certification bodies to assist in verifying the Probability Verification Protocol or employing the Dry Run Verification mentioned earlier.

Under this framework, gaming service providers would provide Functional Commitment to third-party organizations and execute the Probability Verification Protocol, rather than directly sharing their most confidential source code with third-party entities. After verification, the third-party organization can digitally sign the Functional Commitment, indicating to users that the function complies with its probability claims.

This approach alleviates concerns for gaming companies regarding the potential disclosure of too many input-output pairs in the Probability Verification Protocol, which might enable users to bypass functional commitment with approximate loot-box functions through mathematical methods. Moreover, it simplifies the process for users, relieving them from handling complex cryptographic and probability calculations. This not only facilitates easier understanding for users but also mitigates the risk of users misinterpreting the implications of probabilities.

Public Bulletin Board Verification. The method we propose requires running the probability verification protocol before the loot-box opening protocol. If the server prefers to avoid running two protocols, it can combine the two by slightly modifying the loot-box opening protocol. To do so, when designing the loot-box function, the server replaces the random sources submitted by both the server and client with the use of PRB as the random source input, and additionally introduces a global counter to keep track of the current loot-box attempt. This way, the server can upload the loot-box records to the Public Bulletin Board for all clients to verify. Verifiers with access to the public bulletin board can then confirm that these counters reflect a continuous series, ensuring that all relevant data have been uploaded, leaving no room for the server to cherry-pick. While this method relies on continuous access to PRB resources, we acknowledge that its practical execution efficiency may not match the implementation of two separate protocols. Nevertheless, it provides the server with an additional option.

6. Security Analysis

In this section, we provide a brief explanation of how our proposed method satisfies the five objectives defined in Section 2.3.

Correctness and Soundness. Assuming the server provides an adequate amount of test data, as explained in Section 5.2, the correctness of the protocol is highly probable, as indicated by the parameter β. On the other hand, the properties of the PRB guarantees such test inputs are sampled from unpredictable and bias-resistant random; and the *evaluation-binding* property of FC ensures that it is infeasible for the server to find multiple evaluation outcomes for a single input. As a result, soundness is satisfied after the client successfully rejects the null hypothesis discussed in Section 5.1.

Public Verifiability. Every player can contribute their randomness to the PRB and verify the unpredictability and bias-resistance of the randomness outcome after n beacon intervals. Additionally, every player can verify the correctness of FC evaluations. Thus, the probability-verification protocol satisfies our requirement.

Individual Verifiability. Using a hash chain as a random source can prevent the server from manipulating the

randomness to its own advantage. This is because the hash function is collision-resistant. On the other hand, because we assume 1) that the server cannot predict or bias the randomness sampled by the client and 2) that the client generates its randomness β every time it performs an evaluation, we can ensure that the randomness input $\alpha_{i-1} \parallel \beta$ is unpredictable and unbiased by the server. As a result, once the loot-box opening protocol has been verified, it is guaranteed that the server has not biased the probability of winning.

Input Transparency. Input transparency means that there is no hidden input to the loot-box opening function. This property occurs after FC's *evaluation binding* property, whereby it is guaranteed that, given input x, it is infeasible to find distinct evaluation values $y_1 \neq y_2$ that satisfy both $f(x) = y_1$ and $f(x) = y_2$. To verify those evaluation proofs, the client should knows all the input parameters of the loot-box opening function. This ensures transparency regarding all the input parameters involved in the process.

Algorithmic Hiding. Algorithmic hiding is achieved by FC's *hiding* and *evaluation zero-knowledge* properties. In Section 3.2, we provided the full definition of these two properties, but to recap, *hiding* guarantees the indistinguishability of commitment c, and *evaluation zero-knowledge* guarantees that the evaluation reveals nothing other than an evaluation point. In our probability-verification protocol, we only reveal m evaluations, where m is the number of test inputs, as specified by the server. Therefore, it is not feasible to restore the opening function using commitments or evaluations.

7. Implementation and Evaluation

In this section, we first sketch our implementation of PRB and FC. Then, we analyze the computational complexity of our implementation and a comparison of execution times across various setups.

7.1. Implementation of PRB

A crucial adjustment we made in the HeadStart protocol involved the introduction of a new parameter, denoted as W, which governs the window size of the aggregated VDF proof. In this modification, the proof at stage L represents an aggregation of VDF evaluations from stage $\max(0, L - W)$ to stage L, instead of incorporating every stage from 1 to L. The primary goal of this modification was to alleviate the workload of the PRB at each stage while minimizing the verification overhead for clients.

Our PRB implementation [24] in Python follows the specifications outlined in HeadStart [14] with the adjustment described above. For the verifiable delay function (VDF) proof aggregation, we forked [25] Chia VDF [26] and combine it with the implementation from HeadStart [14], [27].

7.2. Implementation of FC

Functional Commitment (FC) is a new area with sparse representation in operational open-source projects. In our proof-of-concept implementation, we adopted two existing FC frameworks [22], [23], each exhibiting certain limitations.

In the first framework BNO21 [22], we supported non-polynomial loot-box functions. However, the implementation [28] of this FC remained incomplete due to constraints on the usable functions and some incomplete aspects of its function-hiding capability.

The second framework, utilizing KZG10 [23] as a polynomial commitment scheme, restricts us to supporting polynomial loot-box functions. KZG10, a submodule for the FC framework mentioned above BNO21, has undergone more comprehensive development. While it adheres to the definitions outlined in Section 3.2, it should be noted that it exclusively supports f as a polynomial.

To overcome the limitation of current functional commitment schemes, we introduce *Pre-processing function* and *Post-processing function* to simplify the process of implementing loot-box functions using functional commitment. Pre-processing function can be used to perform tasks like converting randomness inputs into bit-vectors, which is hard to achieve in BNO21. Post-processing functions is for mapping function outputs to concrete loot-box reward items, which is useful for KZG10 because its output is an element of underlying field $\mathbb{F}_q$. Both functions should be published to public along with the commitment of the hidden function so that verifying functional commitment is still possible.

Our implementation [29] consists of the code needed to evaluate the performance of our protocol under different types of FC, and an interactive GUI for demonstrating our how our *Probability Verification Protocol* works step by step.

7.3. Complexity Analysis

Because our protocols rely on the complex cryptographic operations of PRB and FC, we discuss the computational overhead they incur in this analysis.

PRB is used in the probability-verification protocol for generating test data, with the computation delegated to a third-party PRB server. As described above, we optimized the PRB implementation for scenarios where many players seek to verify their randomness contributions. This optimization ensures that the PRB server efficiently generates aggregated VDF evaluation proofs. For clients verifying PRB's randomness, the number of VDF verification operations required depends on d, the number of stages between a client's last contribution stage and the stage they wish to verify. Although our modification results in a complexity of $O(d)$ instead of $O(1)$ as in HeadStart [14], the verification process remains efficient, provided that the selection of W, the window size for VDF aggregation, is appropriate.

In loot-box opening, the game server generates one functional commitment per loot-box function. It generates an evaluation proof whenever a test input is evaluated, or an in-game loot-box is opened. Taking KZG10 FC scheme as an example, the complexity of generating a commitment is $O(t)$, where t is the degree of the hidden polynomial. The complexity of generating an evaluation proof is also $O(t)$. The client verifies each input-output pair with the

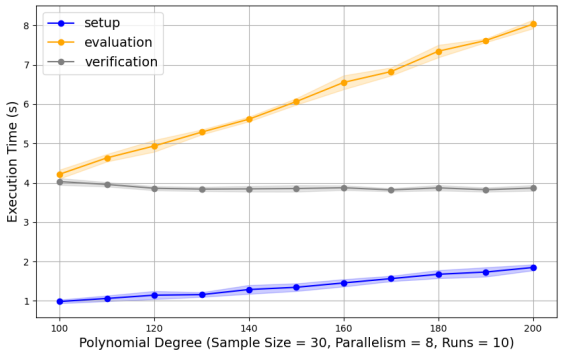


Figure 3: Variation in the probability-verification protocol's execution times across various polynomial degrees

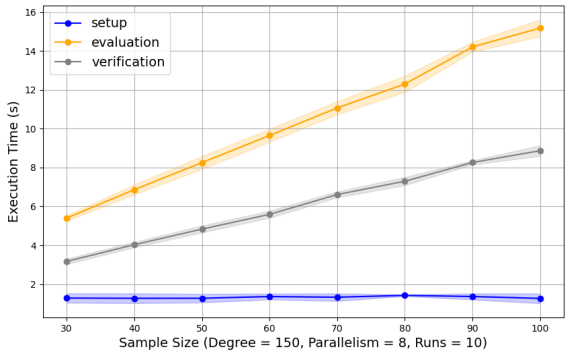


Figure 4: Variation in the probability-verification protocol's execution times across various sample sizes

functional commitment. The verification complexity of the KZG10 commitment is $O(1)$, comprising two elliptic curve pairing operations.

Since result verification is optional, the game server can evaluate using existing non-FC Loot-box implementation with the input obtained from hash chain and client randomness. Players should record the resulting input-output pair in this case. The only cost would be the generation of hash chain and client-server communication overhead, which is negligible compared to `FC.Eval` and `FC.Verify`. Upon receiving verification request, the server should use `FC.Eval` to evaluate on the same input again, so players can verify the if the result matches the record and passes `FC.Verify`.

7.4. Execution-time Analysis

In addition to conducting theoretical complexity analysis, we observed the results our implementation obtained under various parameter settings. Our experiments were conducted on a server with an Intel Xeon Platinum 8352V processor.

We first analyzed the execution times of our probability verification protocol's setup, evaluation, and verification phases over various polynomial degrees. We set the sample size at 30, meaning there were 30 test instances in this experiment, the results of which are shown in Figure 3. As expected, setup and evaluation phase execution times grew linearly with the polynomial degree, but verification time remained constant.

Typically, the polynomial degrees were approximately three times the number of gates used in general circuit construction [30]. Our experiments demonstrated that evaluation times were practical for polynomials with degrees between 100 and 200, corresponding to 30-60 gates.

We then measured the execution times at different sample sizes, and the results we obtained with a fixed polynomial degree of 150 are shown in Figure 4. As expected, execution times for both evaluation and verification grew linearly with sample size.

Additionally, we conducted an experiment using an FC implementation of BNO21 with a 3-bit XOR function, to

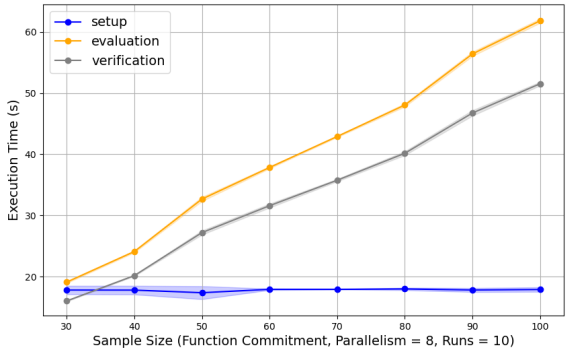


Figure 5: Variation in the probability-verification protocol's execution times across various sample sizes with FC from BNO21

measure execution times across various sample sizes. These results are shown in Figure 5. Similar to our polynomial commitment, execution times for both evaluation and verification grew linearly with sample size. Notably, these times were approximately 4 to 5 times higher than those measured in polynomial commitment.

Conclusively, our algorithm shows practical execution times for polynomial degrees between 100 and 200, suitable for 30-60 gate general circuits. For larger degrees, it may still be considered acceptable depending on the specific requirements and constraints of the application.

7.5. Detecting Adversarial Loot Boxes

To verify the effectiveness of our protocol in detecting adversarial loot boxes that falsely claim higher probabilities, we conducted experiments using our implementation and recorded the results.

Following our discussion in §5.2, assuming that the server claims a winning probability of 0.03, we aim to distinguish an adversary with a real win probability of 0.015 with an 80% confidence level. To achieve this, we would need 653 samples.

Therefore, we repeated the experiment 100 times, with each experiment consisting of 653 loot-box draws, and used the hypothesis testing discussed in §5.1 to check if the results matched the claimed probability.

We conducted experiments on three adversarial loot boxes with real probabilities of 0.01, 0.015, and 0.02. The number of successful detections out of 100 trials was 96, 85, and 51, respectively. These results once again confirm that the required number of tests is related to the expected accuracy of the probability testing. They also illustrate how to use statistical power analysis to determine the proper size of the test dataset.

7.6. Potential Implementation Obstacles for Game Developers

In currently available FC implementations [28], creating FC for every new loot-box function requires a manual translation and recompilation in Rust. Such a process is inconvenient for game developers. Fortunately, new tools like Circom [31], [32] are capable of converting arithmetic circuits into the R1CS form required for zkSnarks. Following BNO21, all we need is to convert the function into t-FT form, i.e., there is no need for manual translation and recompilation of all source code. This means that game developers only need to implement their loot-box functions in a high-level language like Circom and that such functions' FCs can be automatically completed. This strongly supports the feasibility of our method.

While we modeled a loot box as a binary function for simplicity (§2.1), our solution can be extended to support multi-choice outcomes easily. As demonstrated in our prototype implementation, a three-output loot-box (e.g., 1, 2, and 3-star cards) can be modeled using three binary functions, each verifying the probability of obtaining an X-star card ($X = 1, 2, 3$).

8. Related Work

To the best of our knowledge, the majority of academic research on loot boxes has predominantly focused on examining their connections to problem gambling [2], [3], [4], [5], primarily within the realms of psychological and social science research. However, only a few studies delve into the technical aspects to ensure the fairness and verifiability of loot boxes. This section explores two distinct types of proposals designed to facilitate the verification of loot-box probabilities. One approach involves utilizing a trusted third party, while the other leverages decentralized blockchain technology. Furthermore, we have surveyed relevant work on our core cryptographic components, i.e., PRB and FC. Finally, we highlight the significant differences between the loot-box protocol and the e-lottery protocol to clarify the positioning of our approaches in relation to prior work on e-lottery verification.

8.1. Loot-box Probability Verification Methods

Trusted Third-party Probability Verification. One possibility is to delegate the verification to a third-party verification website [10]. Players can upload their loot-box opening records to this platform along with screen-recording videos that provide proof of authenticity. Once a sufficient number of such records has been collected, the platform can publish its assessments of the validity of companies' probability statements, with various confidence intervals.

However, using such third-party verification platforms has several drawbacks. First, as well as being costly, its reliance on screen-recording videos to authenticate records does not guarantee 100% accuracy of the data. Also, the platform does not offer a means for the public to authenticate data. While the platform claims to have a dedicated team responsible for data verification, its lack of transparency leaves us with no choice but to simply trust it, which is not ideal. Moreover, given the doubts surrounding the treatment of different players with varying winning probabilities (e.g., [11]), it is crucial to confirm that there are no hidden inputs in the loot-box algorithm. But the detection of hidden inputs by the platform is not currently possible. Finally, the platform does not sufficiently guard against selective data uploading. For instance, game companies could select and upload a disproportionate number of winning records from among a larger sample of opening records, leading to biased probability.

Transparent Loot-box on Blockchain. As an alternative, Carvalho [12] proposed a transparent loot-box scheme that utilizes blockchain. However, that approach necessitates game companies to implement their loot-box functions using smart contracts, resulting in the disclosure of the function's source code once it is uploaded to the blockchain. In addition, its random source is the timestamp, despite this being predictable to the players.

We have thoroughly compared these two methods with our proposed method in a table in Section C.

8.2. Constructions of PRB

To date, numerous constructions of PRB have emerged, relying on diverse sources of randomness and cryptographic techniques. Some constructions extract randomness from the blockchain [33] and [34], while others utilize financial data for the same purpose [35]. These protocols prove unsuitable for public engagement, as they fail to guard against malicious participants who may withhold the random output strategically to gain an advantage [21], [17].

Another type of construction is based on publicly-verifiable secret sharing (PVSS), including Ouroboros [36], RandHound and RandHerd [37], Scrape [15], and HydRand [17]. Although those constructions provide uniform randomness, they require most contributors to act honestly and remain online throughout the process.

Verifiable delay function (VDF) is the building block of HeadStart [14], which guarantees a lower bound of execution time, ensuring no one can obtain the result, thereby achieving bias resistance. One notable advantage of VDF-based PRB

is its independence from the assumption of honest majority, coupled with efficient verification.

8.3. Constructions of FC

The term "functional commitment" was initially introduced in [38]. However, this definition primarily emphasizes the *input-hiding" property*, rather than *function-hiding*. as highlighted in [22]. The definition of input-hiding entails that for a given public function f and a value y, the committer provides proof that a private x exists such that $f(x) = y$. In contrast, function-hiding functional commitment (FC) involves revealing public values x and y along with the private function f. Several cryptographic primitives can be viewed as specific instances of function-hiding FC, such as polynomial commitment [23]. However, it is noteworthy that, as of now, the only construction available for general circuit FC is presented exclusively in [22].

8.4. E-lottery Protocols

E-lottery is a type of multi-party lottery where users purchase tickets to participate, and a winner is selected through a verifiable randomness. The mainstream approach to e-lottery currently involves using a Verifiable Delay Function (VDF) instead of relying on a trusted third party, as first proposed by Chow et al. [39]. Subsequent developments [40], [41] have mainly focused on improving the efficiency of this protocol, particularly in the design of the VDF. E-lottery protocols share some similarities with loot-box protocols, as both rely on verifiable randomness. However, loot-box protocols differ significantly from e-lottery protocols in two key aspects. First, loot-box protocols often involve proprietary algorithms that companies wish to protect, unlike the typically open-source nature of e-lottery protocols. Second, while e-lottery protocols primarily focus on verifying the VRF contributed by all participants, our work centers on verifying loot-box results and safeguarding the integrity of the overall protocol.

9. Conclusion

In conclusion, this paper reconciles the dilemma between verifiability and intellectual property surrounding the loot-box mechanisms, proposing a novel approach to verify loot-box probability without source-code disclosure. By leveraging two advanced cryptographic primitives, our probability-verification protocol safeguards loot-box probability statements, and the loot-box opening protocol prevents manipulation by game servers or players. We address real-world deployment issues and introduce rigorous hypothesis testing and statistical power analysis. These statistical tools allow for determining the truthfulness of probability statements and guide the selection of an optimal sample size.

This work aligns well with the incentives of both players and game providers. We anticipate its positive impact on the game ecosystem.

Acknowledgment

This research was supported in part by the National Science and Technology Council (NSTC) of Taiwan under grants 112- 2223-E-002-010-MY4 and 113-2634-F-002-001-MBK, and National Taiwan University under grants 113L7871 and 113L900901/113L900902/113L900903.

References

[1] J. Clement, "Free-to-play (f2p) pc games market revenue worldwide from 2018 to 2023," https://www.statista.com/statistics/346552/f2p-pc-gaming-revenue/, 2023.

[2] G. A. Brooks and L. Clark, "Associations between loot box use, problematic gaming and gambling, and gambling-related cognitions," *Addictive behaviors*, vol. 96, pp. 26–34, 2019.

[3] A. Drummond and J. D. Sauer, "Video game loot boxes are psychologically akin to gambling," *Nature human behaviour*, vol. 2, no. 8, pp. 530–532, 2018.

[4] W. Li, D. Mills, and L. Nower, "The relationship of loot box purchases to problem video gaming and problem gambling," *Addictive behaviors*, vol. 97, pp. 27–34, 2019.

[5] M. von Meduna, F. Steinmetz, L. Ante, J. Reynolds, and I. Fiedler, "Loot boxes are gambling-like elements in video games with harmful potential: Results from a large-scale population survey," *Technology in Society*, vol. 63, p. 101395, 2020.

[6] S. Kristiansen and M. C. Severin, "Loot box engagement and problem gambling among adolescent gamers: Findings from a national survey," *Addictive Behaviors*, vol. 103, p. 106254, 2020.

[7] Anny, "Understand in one article the legal battle between dinter, gamania, and the enactment of gacha regulation." https://www.inside.com.tw/article/30725-DinTer-Gamania-Lineage-M-Lawfare, 2023.

[8] M. Handrahan, "Nexon and netmarble fined for loot box practices," https://www.gamesindustry.biz/nexon-and-netmarble-fined-for-loot-box-practices, 2018.

[9] M. J. Koeder, E. Tanaka, and H. Mitomo, "" lootboxes" in digital games-a gamble with consumers in need of regulation? an evaluation based on learnings from japan," 2018.

[10] Gamelab, National Taiwan University of Science and Technology, "Third-party probability verification platform," https://gacha.gamelab.com.tw/, 2023.

[11] LiLi0719, "Internal news - the real situation about gacha probability."," https://forum.gamer.com.tw/C.php?bsn=30518&snA=47974, 2021.

[12] A. Carvalho, "Bringing transparency and trustworthiness to loot boxes with blockchain and smart contracts," *Decision Support Systems*, vol. 144, p. 113508, 2021.

[13] M. of Culture Sports and Tourism, "Guideline on the disclosure of probabilistic item for probabilistic item," https://www.mcst.go.kr/kor/s_notice/notice/noticeView.jsp?pSeq=17910, 2024.

[14] H. Lee, Y. Hsu, J.-J. Wang, H. C. Yang, Y.-H. Chen, Y.-C. Hu, and H.-C. Hsiao, "Headstart: Efficiently verifiable and low-latency participatory randomness generation at scale," *Network and Distributed System Security Symposium (NDSS)*, 2022.

[15] I. Cascudo and B. David, "Scrape: Scalable randomness attested by public entities," in *Applied Cryptography and Network Security: 15th International Conference, ACNS 2017, Kanazawa, Japan, July 10-12, 2017, Proceedings 15*. Springer, 2017, pp. 537–556.

[16] I. Cascudo, B. David, O. Shlomovits, and D. Varlakov, "Mt. random: Multi-tiered randomness beacons," *Cryptology ePrint Archive*, 2021.

[17] P. Schindler, A. Judmayer, N. Stifter, and E. Weippl, "Hydrand: Efficient continuous distributed randomness," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 73–89.

[18] D. Galindo, J. Liu, M. Ordean, and J.-M. Wong, "Fully distributed verifiable random functions and their application to decentralised random beacons," in *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2021, pp. 88–102.

[19] A. Bhat, N. Shrestha, Z. Luo, A. Kate, and K. Nayak, "Randpiper–reconfiguration-friendly random beacons with quadratic communication," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 3502–3524.

[20] A. Cherniaeva, I. Shirobokov, and O. Shlomovits, "Homomorphic encryption random beacon," *Cryptology ePrint Archive*, 2019.

[21] P. Schindler, A. Judmayer, M. Hittmeir, N. Stifter, and E. Weippl, "Randrunner: Distributed randomness from trapdoor vdfs with strong uniqueness," 2021.

[22] D. Boneh, W. Nguyen, and A. Ozdemir, "Efficient functional commitments: How to commit to a private function," *Cryptology ePrint Archive*, 2021.

[23] A. Kate, G. M. Zaverucha, and I. Goldberg, "Constant-size commitments to polynomials and their applications," in *Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16*. Springer, 2010, pp. 177–194.

[24] "Toy headstart implementation," https://anonymous.4open.science/r/toy-headstart-py-49F5, 2024.

[25] "Fork of chia vdf," https://anonymous.4open.science/r/chaivdf-3666, 2024.

[26] "Chia vdf," https://github.com/Chia-Network/chiavdf, 2020.

[27] "Headstart aggvdf," https://github.com/csienslab/headstart, 2022.

[28] "Function-hiding functional commitment," https://github.com/geometryresearch/functional-commitment, 2022.

[29] "Lootboxprotocol implementation," https://anonymous.4open.science/r/loot-box-protocol-B3BD, 2024.

[30] A. Gabizon, Z. J. Williamson, and O. Ciobotaru, "Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge," *Cryptology ePrint Archive*, 2019.

[31] M. Bellés-Muñoz, M. Isabel, J. L. Muñoz-Tapia, A. Rubio, and J. Baylina, "Circom: A circuit description language for building zero-knowledge applications," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 6, pp. 4733–4751, 2023.

[32] "circom: circuit compiler," https://github.com/iden3/circom, 2023.

[33] J. Bonneau, J. Clark, and S. Goldfeder, "On bitcoin as a public randomness source." *IACR Cryptol. ePrint Arch.*, vol. 2015, p. 1015, 2015.

[34] C. Pierrot and B. Wesolowski, "Malleability of the blockchain's entropy," *Cryptography and Communications*, vol. 10, no. 1, pp. 211–233, 2018.

[35] J. Clark and U. Hengartner, "On the use of financial data as a random beacon." *EVT/WOTE*, vol. 89, 2010.

[36] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Annual International Cryptology Conference*. Springer, 2017, pp. 357–388.

[37] E. Syta, P. Jovanovic, E. K. Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford, "Scalable bias-resistant distributed randomness," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 444–460.

[38] B. Libert, S. C. Ramanna *et al.*, "Functional commitment schemes: From polynomial commitments to pairing-based accumulators from simple assumptions," in *43rd International Colloquium on Automata, Languages and Programming (ICALP 2016)*, 2016.

[39] S. S. Chow, L. C. Hui, S.-M. Yiu, and K.-P. Chow, "An e-lottery scheme using verifiable random function," in *International Conference on Computational Science and its Applications*. Springer, 2005, pp. 651–660.

[40] Y.-N. Liu, H.-G. Liu, L. Hu, and J.-B. Tian, "A new efficient e-lottery scheme using multi-level hash chain," in *2006 international conference on communication technology*. IEEE, 2006, pp. 1–4.

[41] B. Liang, G. Banegas, and A. Mitrokotsa, "Statically aggregate verifiable random functions and application to e-lottery," *Cryptography*, vol. 4, no. 4, p. 37, 2020.

[42] L. Y. Xiao, L. L. Henderson, R. K. Nielsen, and P. W. Newall, "Regulating gambling-like video game loot boxes: A public health framework comparing industry self-regulation, existing national legal approaches, and other potential approaches," *Current Addiction Reports*, vol. 9, no. 3, pp. 163–178, 2022.

[43] L. Y. Xiao, "Breaking ban: Belgium's ineffective gambling law regulation of video game loot boxes," *Collabra: Psychology*, vol. 9, no. 1, 2023.

[44] L. Y. Xiao, L. L. Henderson, Y. Yang, and P. W. Newall, "Gaming the system: suboptimal compliance with loot box probability disclosure regulations in china," *Behavioural Public Policy*, pp. 1–27, 2021.

[45] A. Hiramatsu, "A research of social game users' attitude to" gacha" probability announcement," in *2019 8th International Congress on Advanced Applied Informatics (IIAI-AAI)*. IEEE, 2019, pp. 115–120.

[46] N. G. Authority, "Study into loot boxes: A treasure or a burden?" *Amsterdam, Netherlands*, vol. 10, 2018.

[47] E. Obedkov, "South korea passes new amendment on loot box probability disclosure," https://gameworldobserver.com/2023/02/28/south-korea-loot-boxes-probability-disclosure-law, 2023.

[48] Paul, "Promotion of taiwan's online game gacha regulation," https://join.gov.tw/idea/detail/ee5dd8b8-bdeb-4d5e-8315-bb0601169d68, 2021.

[49] Department of Consumer Protection, Taiwanese Executive Yuan, "Disclosing loot box odds to protect gamers' interests," https://cpc.ey.gov.tw/en/4212D8C5A29ACA5F/61e3c731-23e9-41af-abd0-d12f2912e31c, 2022.

[50] Apple, "App store review guidelines," https://developer.apple.com/app-store/review/guidelines/, 2023.

[51] Google, "Policy center," https://support.google.com/googleplay/android-developer/topic/9858052?hl=en, 2023.

Appendix

1. Loot-box regulation around the world

At the time of writing, several countries and platforms have promulgated regulations concerning loot boxes. However, these existing regulatory approaches differ considerably from one another. In part, this is because certain countries classify loot boxes as a form of gambling, while others do not [42]. Some of the most important regulatory frameworks are discussed in turn below.

Belgium. In 2018, the Belgian Gaming Commission ruled that loot-box mechanisms were gambling under existing legislation, and in effect prohibited loot boxes on that basis: i.e., game companies that implement paid loot boxes without a gambling license could face criminal prosecution. Nevertheless, a recent study showed that 82 of the 100 highest-grossing Belgian iPhone games had loot-box features [43].

China. In 2017, China became the first country to legally mandates that game companies disclose the probabilities of receiving randomized loot-box rewards [44].

Japan. In 2012, Japan's Consumer Affairs Agency (CAA) banned "complete gacha", which involve the player needing to collect a series of items before having the chance to obtain a specific rare item. However, disclosure of probabilities was achieved through self-regulation. In 2016, the Computer Entertainment Supplier's Association published guidelines for gacha probabilities and pricing and obligated its members to comply with them. A study on Japanese game players' attitudes towards loot-box probability statements [45] revealed that despite regulations and self-regulating guidelines, the majority of players did not trust such statements.

Netherlands. In 2018, a report by the Netherlands Gambling Authority [46] found that loot boxes offering tradable items were illegal and, hence, the underlying games could not be sold without an appropriate license.

South Korea. South Korea previously adopted a self-regulatory approach to loot-box games, led by the Korean Association of the Game Industry (K-GAMES). However, in February 2023, the National Assembly of South Korea collaborated with K-GAMES to pass a legislative amendment that mandates loot-box probability disclosure [47].

Taiwan. In June 2021, an online player started a petition on a public-policy participation platform, asking the Taiwanese government to formulate regulations on "gacha" mechanics. The petition received support from 6,560 people [48]. Following several incidents and extensive discussions among players, game companies, and the government, the Consumer Protection Committee announced revisions to its regulations in 2022 [49]. These included rules that:

- Game companies disclose the probability of obtaining randomized virtual item
- Such winning probabilities be clearly defined
- The scope of winning probability disclosure be clearly defined
- The manner of winning-probability disclosure should be clearly defined

Alongside national regulation of loot-box games, the Apple app store and Google play store have enacted some relevant policies.

Apple App Store. In 2017, Apple changed their developer guidelines to include the following: "Apps offering loot boxes or other mechanisms that provide randomized virtual items for purchase must disclose the odds of receiving each type of item to customers prior to purchase" [50].

Google Play Store. In 2019, Google announced a new policy for loot boxes, i.e., that "Apps and games offering mechanisms to receive randomized virtual items from a purchase including, but not limited to, loot boxes must clearly disclose the odds of receiving those items in advance of, and in close and timely proximity to, that purchase" [51].

2. Disputes of Loot-box Practices

Several incidents of unfair loot-box practices have been reported. Below, we briefly describe two such incidents.

In September 2021, a popular Taiwanese streamer accused the game company behind "Lineage M" of manipulating the probabilities of winning loot boxes in that online game [7]. Previously, the game company had stated that the probability of obtaining a particular item in the Taiwanese version was equivalent to that of the Korean version, i.e., 10%. However, despite spending more than NT$4 million (about US$144,000), the player only succeeded 11 times out of 475 attempts, a success rate of approximately 2.3%.

Upon reviewing the player's appeal, Taiwan's Fair Trade Commission launched an investigation. This revealed that, based on the game company's internal communication records, the actual probability of obtaining the item in question was 5%, which was clearly inconsistent with their previous claim. Consequently, the commission imposed a fine of NT$2 million on the game company for violation of the Fair Trade Act. This incident also sparked public discussion about the reform of loot-box regulation.

Later in 2021, an anonymous player of "Arena of Valor" claimed to be an employee of the company that produced it, and that the probability of obtaining items from loot boxes differed for each player, depending on how much they had spent in the game [11]. The same player also presented modified source code as evidence in support of this claim. This disclosure also prompted widespread discussion and further highlighted growing distrust between players and game companies.

3. Comparison of Verifying Loot-box with different methods

To further establish the positioning of our method, we compared the existing loot-box verification approaches mentioned in Introduction with our method, and organized the results into Table 1. We considered the five objectives proposed in 2.3, as well as "Reasonable Basic Assumption" (practical feasibility) and "Deployability" (the effort required to apply it to the current gaming industry).

Using a third-party verification platform to verify a loot-box has major issues. It assumes that a significant number of players are willing to provide their draw data, that these players are unbiased, and that game developers will not manipulate win rates using fake accounts. We believe these assumptions are unrealistic in practice, so it does not meet the "Reasonable Basic Assumption" criterion. Due to the likelihood of various deceptive practices, this solution does not fully meet the "Correctness and Soundness" and "Input Transparency" criteria.

The method of using smart contracts for loot-box draws can indeed satisfy most of our objectives. However, the main issue is that smart contracts are vastly different from the frameworks currently used in the gaming industry, making implementation difficult. Additionally, they are subject to delays caused by the execution of smart contracts or the PRB, making it challenging to meet the real-time requirements of loot-box services. They also need additional costs, such as the gas fees associated with blockchain transactions, making practical implementation even more problematic. Furthermore, smart

	Correctness & Soundness	Public Verifiability	Individual Verifiability	Input Transparency	Algorithmic Hiding	Reasonable Assumptions	Deployability
3rd-party platform	?	v	x	?	v	x	v
Smart Contract [12]	v	v	v	v	x	v	x
Our Method	v	v	v	v	v	v	v

TABLE 1: Comparison of Verifying Loot-box with different methods

contracts do not address the criterion of "Algorithmic Hiding". According to our interviews with actual game companies, revealing source code is seen as a significant leakage of commercial secrets and would have a considerable impact on the industry. Thus, current game companies are unlikely to adopt a design like smart contracts.

Our method, through cryptographic means, ensures that all five of our proposed objectives are met while considering the needs of current game companies. It relies solely on cryptographic assumptions, which we believe makes it more suitable for the current gaming industry compared to the other two methods.