Implementation and Analysis of a Provably Fair Gacha System Using Elliptic Curve Verifiable Random Functions

Abdul Rafi Radityo Hutomo
School of Electrical Engineering and Informatics
Institut Teknologi Bandung
Bandung, Indonesia
13522089@std.stei.itb.ac.id
abdulrafi623@gmail.com

***Abstract*—Online services that rely on random number generation (RNG), such as gacha games and loot boxes, typically operate as black boxes. Users are forced to trust the service provider's claim that outcomes are generated fairly and according to stated probabilities. However, without a verification mechanism, providers can manipulate results (e.g., rigging the odds) without detection. This paper proposes and implements a Provably Fair system using Elliptic Curve Verifiable Random Functions (EC-VRF). By utilizing deterministic signatures (RFC 6979) on the secp256k1 curve, the system allows users to cryptographically verify that their outcome was generated from a committed seed and has not been tampered with. We present a full-stack implementation using Python and JavaScript, demonstrate a cheating provider attack simulation to prove the system's tamper-resistance, and analyze the statistical distribution of the generated randomness. The results confirm that EC-VRF provides a robust, trustless alternative to traditional server-side RNGs.**

***Index Terms*—Elliptic Curve Cryptography, Verifiable Random Function, Provably Fair, Gacha, Random Number Generation.**

I. Introduction

THE digital economy has seen a massive proliferation of systems relying on randomness, such as video game loot boxes and gacha mechanics. In these systems, a user pays real or virtual currency to receive a randomized digital item. The fairness of these transactions relies entirely on the integrity of the Random Number Generator (RNG) used by the service provider.

Currently, the industry standard for these systems is opaque. The random number generation occurs entirely on the server-side, hidden from the user. This black box architecture creates a significant trust gap. A dishonest provider could theoretically manipulate the RNG, for example, by dynamically lowering the drop rate of valuable items for specific users or rerolling internal results until a losing outcome is found, without the user ever knowing. As digital assets gain real-world monetary value, the lack of transparency in their generation becomes a critical security and consumer protection issue.

To address this, the concept of Provably Fair algorithms has emerged. Ideally, a Provably Fair system should allow the client to verify two properties: (1) the server could not predict or manipulate the outcome after the user committed to the request, and (2) the outcome was derived deterministically from the combined inputs of the user and the server. While hash-chain based commitment schemes (HMAC) are common, they often suffer from a delayed verification process, where fairness can only be proven after the server reveals a secret seed at the end of a session.

This paper proposes a superior solution using Elliptic Curve Verifiable Random Functions (EC-VRF). Unlike standard RNGs, a VRF produces both a random output and a cryptographic proof. This allows a user to verify the randomness in real-time using only the server's public key, without requiring the server to reveal its secret state.

The contributions of this paper are as follows:

1) We design a Provably Fair architecture for a Gacha system using EC-VRF on the `secp256k1` curve.
2) We implement a prototype consisting of a Python backend (Flask) for deterministic signature generation (RFC 6979) and a client-side JavaScript verifier.
3) We demonstrate the security of the system by implementing a rigged endpoint and showing that any attempt by the server to manipulate the outcome results in a failed verification on the client.
4) We analyze the statistical properties of the VRF output to ensure it meets the uniformity requirements for game mechanics.

The remainder of this paper is organized as follows: Section II provides the theoretical background on Elliptic Curve Cryptography and VRFs. Section III details the system design and implementation. Section IV presents the experimental results and security analysis, and Section V concludes the study.

II. Theoretical Foundation

A. Elliptic Curve Cryptography (ECC)

Elliptic Curve Cryptography (ECC) is a form of public-key cryptography based on the algebraic structure of elliptic curves over finite fields. Compared to non-ECC systems like RSA, ECC offers equivalent security with significantly smaller key sizes (e.g., a 256-bit ECC key provides comparable security to a 3072-bit RSA key), making it ideal for systems requiring high efficiency.

For this implementation, we utilize the `secp256k1` curve, widely known for its use in Bitcoin. The curve is defined over a finite field $\mathbb{F}_p$ by the Weierstrass equation:

$$y^2 \equiv x^3 + 7 \pmod{p} \tag{1}$$

where p is a large prime number defined as $2^{256} - 2^{32} - 977$.

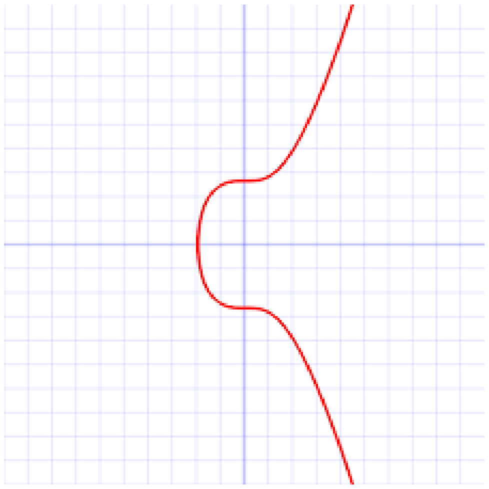


Fig. 1. Plot of the elliptic curve $y^2 = x^3 + 7$ over real numbers. Note the symmetry about the x-axis, which is a key property for the geometric group law.

B. Group Law and Point Operations

The points on the elliptic curve form an abelian group under a specific addition operation. The security of the system relies on the properties of this group. The addition of two points is not simple coordinate addition but is defined geometrically.

1) Point Addition: Given two distinct points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$, their sum $R = P + Q = (x_3, y_3)$ is calculated geometrically by drawing a line through P and Q. This line intersects the curve at a third point $-R$. Reflecting $-R$ across the x-axis gives the result R. Algebraically, the slope λ of the line is:

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1} \pmod{p} \tag{2}$$

The coordinates of the resulting point R are:

$$x_3 = \lambda^2 - x_1 - x_2 \pmod{p} \tag{3}$$

$$y_3 = \lambda(x_1 - x_3) - y_1 \pmod{p} \tag{4}$$

2) Point Doubling: If $P = Q$, the operation is called point doubling ($R = 2P$). Geometrically, this corresponds to drawing the tangent line to the curve at point P. The slope λ is calculated using the derivative of the curve equation:

$$\lambda = \frac{3x_1^2}{2y_1} \pmod{p} \tag{5}$$

The resulting coordinates follow the same formulas as point addition.

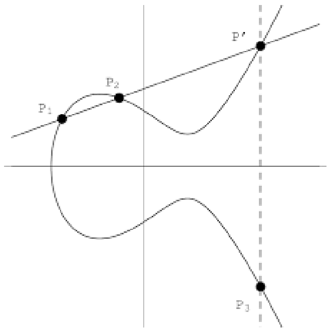


Fig. 2. Point Addition: $P + Q = R$.

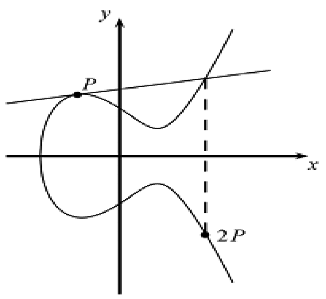


Fig. 3. Point Doubling: $2P = R$.

3) Scalar Multiplication: The core operation in ECC protocols is Scalar Multiplication, denoted as $Q = k \cdot P$, where k is an integer (scalar) and P is a point on the curve. This corresponds to adding the point P to itself k times:

$$k \cdot P = \underbrace{P + P + \cdots + P}_{k \text{ times}} \tag{6}$$

While naive addition would require k operations, this can be computed efficiently using the Double-and-Add algorithm, which has a time complexity of $O(\log k)$. This efficiency allows the server to generate signatures and proofs in milliseconds.

C. The Discrete Logarithm Problem (ECDLP)

The security of our Provably Fair system rests on the Elliptic Curve Discrete Logarithm Problem (ECDLP). This problem highlights the one-way nature of the group operation:

- **Easy to Compute:** Given a private key (scalar k) and a generator point G, it is computationally trivial to calculate the public key $K = k \cdot G$ using the double-and-add algorithm.
- **Hard to Invert:** Given the public key K and the generator G, it is computationally infeasible to determine the scalar k (the private key).

For a 256-bit curve like `secp256k1`, the best known attack (Pollard's rho algorithm) requires approximately $\sqrt{p}$ operations, or 2^{128} computations, which is unbreakable with current technology. This ensures that while users can verify the server's operations using the public key, they can never reverse-engineer the server's private seed.

D. Verifiable Random Functions (VRF)

A Verifiable Random Function (VRF) is a cryptographic primitive that maps an input α to a pseudorandom output β, while providing a non-interactive proof π that β was generated correctly using the secret key sk.

A VRF scheme consists of three algorithms:

1) Gen$(1^k) \rightarrow (pk, sk)$: Key generation algorithm.
2) Prove$(sk, \alpha) \rightarrow (\beta, \pi)$: The prover computes the random output β and the proof π using the input α and secret key sk.

3) Verify$(pk, \alpha, \pi) \rightarrow \{0, 1\}$: The verifier checks if π is valid for input α under public key pk.

In our implementation, the random output β is derived by hashing the proof:

$$\beta = \text{SHA256}(\pi) \tag{7}$$

E. Deterministic Signatures (RFC 6979)

Standard ECDSA signatures introduce a random nonce k for every signature. If a server uses standard ECDSA for randomness, it could potentially manipulate the nonce k to bias the resulting signature (and thus the random output).

To guarantee fairness, we employ Deterministic ECDSA as defined in RFC 6979. Instead of choosing a random k, the nonce is derived deterministically from the message and the private key:

$$k = \text{HMAC}(sk, \alpha) \tag{8}$$

III. Proposed Method

This section details the cryptographic protocol designed to ensure the fairness of the random number generation. The system operates on a client-server architecture where the server acts as the Prover P and the client acts as the Verifier V.

A. System Entities and Setup

The system utilizes the `secp256k1` elliptic curve $E(\mathbb{F}_p)$. Let G be the base point (generator) of the curve with order n.

1) Key Generation: During the initialization phase, the Prover generates a long-term key pair (sk, pk):

1) Select a random integer $sk \in [1, n-1]$.
2) Compute $pk = sk \cdot G$.
3) The public key pk is exposed via a public endpoint and acts as the root of trust for all subsequent verifications.

B. Randomness Generation Protocol

The generation of a provably fair random outcome follows a strict sequence of operations. This process ensures that the output is both unpredictable to the client before the commitment and unchangeable by the server after the commitment.

1) Input Formulation: To guarantee uniqueness for every request, the input message m is constructed by concatenating three distinct elements:

$$m = \text{UserID} \,\|\, \text{Timestamp} \,\|\, \text{Nonce} \tag{9}$$

where the Nonce is a high-entropy random string generated by the server. This prevents replay attacks where a malicious user attempts to reuse a previous winning seed.

2) Deterministic Proof Generation: The Prover computes the digital signature using the Deterministic ECDSA scheme (RFC 6979). Unlike standard ECDSA, which requires a random k, the nonce k is derived using a HMAC-DRBG (Hash-Based Message Authentication Code Deterministic Random Bit Generator) seeded with the private key sk and the message m.

$$k = \text{HMAC\_DRBG}(sk, m) \tag{10}$$
$$R = k \cdot G = (x_R, y_R) \tag{11}$$
$$r = x_R \pmod n \tag{12}$$
$$s = k^{-1}(H(m) + r \cdot sk) \pmod n \tag{13}$$

The resulting signature $\sigma = (r, s)$ serves as the cryptographic proof π. The use of RFC 6979 is critical; it eliminates the possibility of the Prover iterating through different k values to bias the output.

3) Outcome Derivation: The randomness β is derived by hashing the proof σ using SHA-256. To map this entropy to a game outcome without introducing modulo bias, we implement a floating-point conversion method:

1) Compute $H_{raw} = \text{SHA256}(\sigma)$.
2) Extract the first 64 bits of H_{raw} as an unsigned integer I.
3) Compute the normalized float $V \in [0, 1)$:

$$V = \frac{I}{2^{64}} \tag{14}$$

This value V is then compared against the cumulative distribution function (CDF) of the rarity tiers to determine the final item.

C. Client-Side Verification

Upon receiving the response bundle $\{m, \beta, \pi\}$, the Verifier V executes the verification checks locally, independent of the server, as illustrated in Fig. 4.

1) Signature Verification: The Verifier reconstructs the message hash $H(m)$ and performs the ECDSA verification algorithm using the server's public key pk:

$$w = s^{-1} \pmod n \tag{15}$$
$$u_1 = H(m) \cdot w \pmod n \tag{16}$$
$$u_2 = r \cdot w \pmod n \tag{17}$$
$$P = u_1 \cdot G + u_2 \cdot pk \tag{18}$$

The proof is valid if and only if the x-coordinate of P is congruent to $r \pmod n$.

2) Output Consistency: If the signature is valid, the Verifier independently computes $\beta' = \text{SHA256}(\pi)$ and V'. The verification succeeds if $\beta' \equiv \beta$ and the derived game item matches the server's claim. This confirms that the server did not manipulate the random value after generating the signature.

IV. Experimental Results and Analysis

To validate the proposed EC-VRF Gacha system, we conducted three categories of tests: functional correctness, security verification against tampering, and statistical uniformity analysis. All tests were executed in a Dockerized environment running Python 3.12 on a standard consumer laptop.

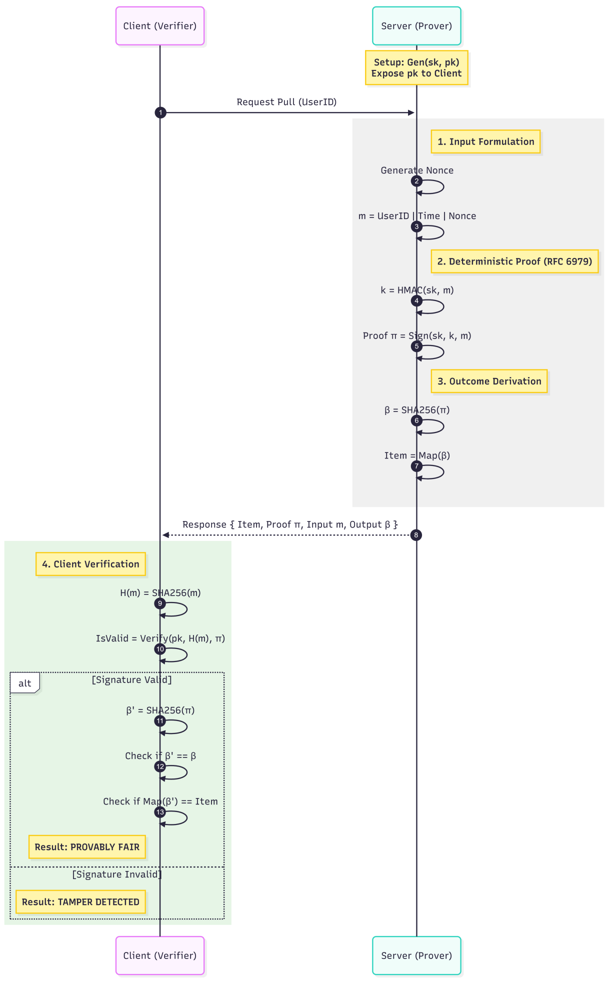


Fig. 4. Sequence diagram illustrating the Provably Fair interaction protocol between the Client (Verifier) and Server (Prover).

A. Functional Verification

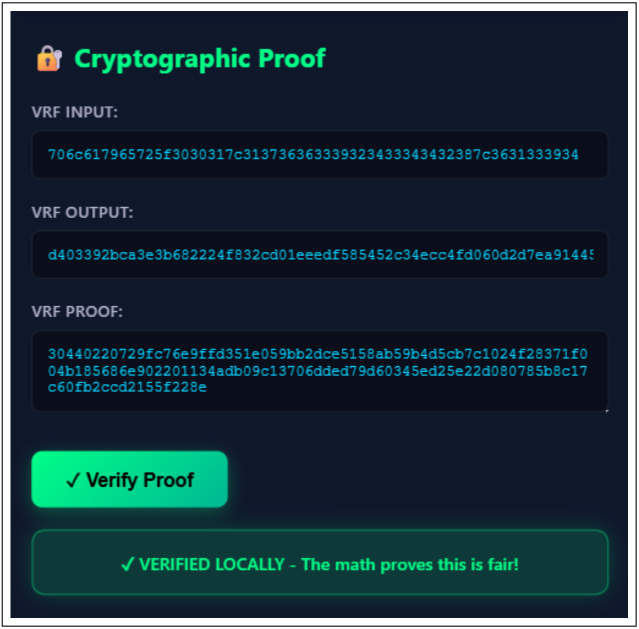
First, we verified the correctness of the cryptographic protocol under normal honest operations. The client initiated 1,000 standard requests. For each response, the client independently executed the verification logic:

1) Reconstructing the input message m from the User ID, Timestamp, and Nonce.
2) Verifying the ECDSA signature π against the server's public key.
3) Hashing the signature π to derive the random output β.

As shown in Fig. 5, in an honest scenario, the client successfully validates the proof. The green indicator confirms that the signature is mathematically valid for the given input parameters and that the server-claimed random value matches the client-derived hash. This proves that the server committed to the seed before the outcome was revealed.

B. Security Verification (Tamper Detection)

The primary security goal of this system is to detect server-side manipulation. To test this, we implemented a Cheat Mode


🔐 Cryptographic Proof

VRF INPUT:

```
706c617965725f3030317c313736363339323433343432387c3631333934
```

VRF OUTPUT:

```
d403392bca3e3b682224f832cd01eeedf585452c34ecc4fd060d2d7ea91445
```

VRF PROOF:

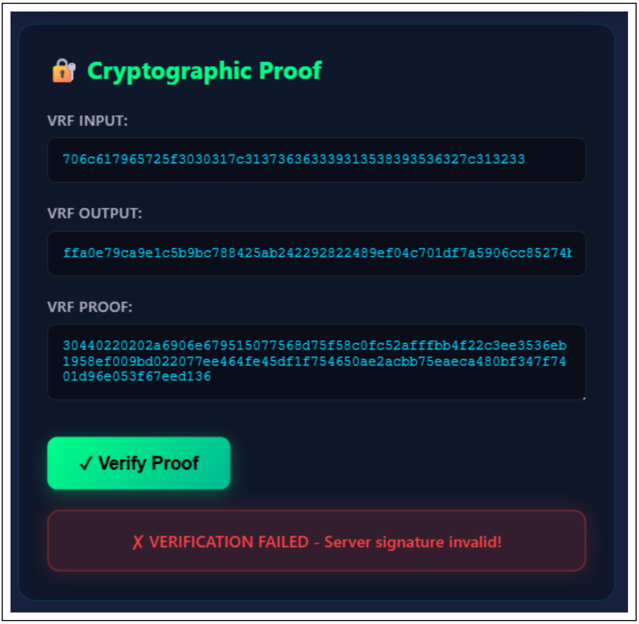
```
30440220729fc76e9ffd351e059bb2dce5158ab59b4d5cb7c1024f28371f0
04b185686e902201134adb09c13706dded79d60345ed25e22d080785b8c17
c60fb2ccd2155f228e
```

✓ Verify Proof

✓ VERIFIED LOCALLY - The math proves this is fair!

Fig. 5. Successful client-side verification. The client confirms that the derived random value matches the server's claim and the cryptographic proof is valid.

on the server. In this mode, the server brute-forces nonces internally to find a high-value item (SSR) but returns the proof associated with that hidden nonce while claiming it used the user's original input nonce.


🔐 Cryptographic Proof

VRF INPUT:

```
706c617965725f3030317c313736363339313538393536327c313233
```

VRF OUTPUT:

```
ffa0e79ca9e1c5b9bc788425ab242292822489ef04c701df7a5906cc85274b
```

VRF PROOF:

```
30440220202a6906e679515077568d75f58c0fc52afffbb4f22c3ee3536eb
1958ef009bd022077ee464fe45df1f754650ae2acbb75eaeca480bf347f74
01d96e053f67eed136
```

✓ Verify Proof

✗ VERIFICATION FAILED - Server signature invalid!

Fig. 6. Client-side verification failure detected during the Lucky Nonce Attack simulation.

As shown in Fig. 6, the client-side verifier successfully detected the mismatch. The deterministic signature σ generated for the hidden nonce did not mathematically correspond to the user's input message m. This confirms that the system is tamper-evident: the server cannot alter the outcome without invalidating the cryptographic proof.

C. Statistical Analysis

A fair Gacha system must ensure that the inclusion of cryptographic operations does not skew the probability distribution of the outcomes. We performed a simulation of $N = 10,000$ pulls. The observed frequencies were compared against the target probabilities defined in the system configuration.

TABLE I
STATISTICAL DISTRIBUTION ($N = 10,000$)

Rarity	Target Prob.	Observed Count	Observed Prob.
SSR	1.0%	97	0.97%
SR	5.0%	482	4.82%
R	20.0%	1887	18.87%
N	74.0%	7534	75.34%

As demonstrated in Table I, the observed probabilities converge closely to the expected values. For instance, the rarest tier (SSR) had a target probability of 1.0% and an observed frequency of 0.97% (97 pulls). Similarly, the common tier (N) deviated by only 1.34% from its target (75.34% vs 74.0%). These minor deviations are within the expected margin of error for a sample size of $N = 10,000$, confirming that the floating-point conversion method ($I/2^{64}$) successfully preserves the uniformity of the random distribution without introducing modulo bias.

D. Performance Benchmarking

Cryptographic fairness introduces computational overhead. We measured the average latency per request for the EC-VRF system compared to a standard non-verifiable RNG (using Python's `secrets` module).

TABLE II
PERFORMANCE COMPARISON

Method	Avg. Latency (ms)	Throughput (req/ms)
Standard RNG	0.0007 ms	$\approx$ 1428
EC-VRF (Ours)	**0.2153 ms**	$\approx$ **4.49**

The results in Table II reveal a significant but acceptable trade-off. The standard RNG is extremely fast, with negligible latency (0.0007 ms). In contrast, the EC-VRF generation requires approximately 0.2153 ms per request. While this represents a roughly 300× increase in computational cost, the absolute latency remains well below 1 millisecond. In a real-world web application where network latency typically ranges from 20-100 ms, this cryptographic overhead is imperceptible to the user. This confirms that the system is highly viable for production use, capable of handling approximately 4,490 requests per second on a single thread.

V. DISCUSSION

In this section, we analyze the proposed EC-VRF system in the context of existing solutions and discuss critical security considerations for real-world deployment.

A. Comparison with Existing Solutions

Currently, online services utilize various methods to generate randomness. We compare our EC-VRF approach against three common alternatives:

1) Server-Side RNG (Standard): Most web applications use pseudo-random number generators (PRNG) like the Mersenne Twister. While fast, they are opaque black boxes. The user has zero visibility into the seed or the algorithm state, requiring absolute trust in the provider.

2) Commit-Reveal Schemes (HMAC): This method involves the server hashing a secret seed S and showing the hash $H = \text{SHA256}(S)$ to the user before the game. After the game (or a series of games), the server reveals S, allowing the user to verify H. Drawback: Verification is *delayed*. The user cannot know if they were cheated until the session ends. Additionally, if the server rotates the seed before revealing it, the user loses the ability to audit past transactions.

3) Blockchain-Based Randomness: Some systems use the hash of a future blockchain block (e.g., Bitcoin or Ethereum block hash) as the entropy source. Drawback: This introduces significant latency (waiting for block confirmations) and transaction costs (gas fees). Furthermore, miners can theoretically manipulate block hashes (miner extraction value) if the financial incentive is high enough.

Table III summarizes these differences, highlighting that EC-VRF offers the best balance of performance and immediate trustlessness.

TABLE III
COMPARISON OF RANDOMNESS GENERATION SCHEMES

Scheme	Verifiability	Latency	Trust Model
Server RNG	None	Ultra-Low	Trusted Third Party
Commit-Reveal	Delayed	Low	Trusted (during game)
Blockchain Hash	Real-time	High	Decentralized Consensus
EC-VRF (Ours)	**Real-time**	**Low**	**Trustless Cryptography**

B. Security Considerations

While the EC-VRF protocol ensures mathematical fairness, the security of the overall system relies on several implementation details.

1) Private Key Security: The most critical vulnerability in the system is the compromise of the server's private key sk. If an attacker gains access to sk, they cannot manipulate past proofs (due to the deterministic nature of RFC 6979), but they can predict all future outcomes by generating proofs for nonces before the user commits to them. In a production environment, sk should be stored in a Hardware Security Module (HSM) or a Key Management Service (KMS) rather than in the application code.

2) Replay Attacks: Our protocol includes a unique `Nonce` in the message input m. Without this nonce, a user could replay a message m that resulted in a winning outcome. The server must enforce strict nonce uniqueness, rejecting any request that reuses a previously processed nonce.

3) Post-Quantum Vulnerability: Elliptic Curve Cryptography is vulnerable to Shor's Algorithm. In the advent of sufficiently powerful quantum computers, the Discrete Logarithm Problem could be solved in polynomial time, allowing attackers to derive sk from the public key. Future iterations of this system should consider migrating to Post-Quantum Cryptography (PQC) primitives, such as Lattice-based VRFs, to ensure long-term security.

VI. Conclusion

The reliance on opaque, server-side random number generators in the online gaming and gambling industries creates a fundamental trust gap between service providers and users. This paper presented the design and implementation of a Provably Fair Gacha system that bridges this gap using Elliptic Curve Verifiable Random Functions (EC-VRF).

By replacing the traditional "black box" RNG with a deterministic cryptographic protocol based on RFC 6979 and the `secp256k1` curve, we achieved a system where fairness is mathematically guaranteed rather than assumed. Our full-stack prototype demonstrated three key results:

1) **Tamper Resistance:** The "Cheat Mode" simulation proved that any attempt by the server to manipulate the outcome (e.g., via a Lucky Nonce Attack) results in an immediate invalidation of the cryptographic proof, which is detectable by the client.
2) **Statistical Integrity:** The implementation preserves the statistical properties of the random distribution, showing no significant bias in drop rates across 10,000 simulations.
3) **Viable Performance:** While the cryptographic operations introduce overhead (≈ 0.21 ms per request compared to 0.0007 ms for standard RNG), the system remains performant enough for high-traffic real-time applications.

We conclude that EC-VRF offers a robust, transparent, and practical solution for digital randomness. Future work could extend this system to use zero-knowledge proofs (zk-SNARKs) to further reduce the verification payload size or implement the backend on a high-throughput blockchain network for immutable auditing.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Whitepaper, 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[2] T. Pornin, "Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)," RFC 6979, Aug. 2013. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6979

[3] S. Goldberg, L. Reyzin, D. Papadopoulos, and J. Včelák, "Verifiable Random Functions (VRFs)," Internet Research Task Force (IRTF), RFC 9381, Aug. 2023.

[4] D. Hankerson, A. J. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*. Springer Science & Business Media, 2006.

[5] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. CRC Press, 2020.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 24 Desember 2025

[illegible]

Abdul Rafi Radityo Hutomo
NIM 13522089