The Architecture and Design of FARE Protocol

Jay McCarthy (jay@fareprotocol.io)
FARE

1 Introduction

This white-paper describes the architecture of FARE protocol, a Web3 probabilities protocol. FARE is designed to facilitate decentralized random trials that produce and consume resources, specified via the FARE token. In this document, we imagine FARE being used to build a decentralized casino to elucidate the design of FARE, although it can be used to build much money.

This document is structured as follows: First, we discuss the essence of what a casino is and the concept of the "bankroll". Second, we describe the problem casinos must solve to stay solvent and make a profit by ensuring a "house edge". Third, we discuss how a token with probabilistic minting and burning operates like a casino with an infinite bankroll. Fourth, we relate the infinite bankroll casino to the necessity of edge and the requirement to support arbitrary bets safely. Fifth, we discuss a general method for deriving bet payout tables. Sixth, we present our extension to the basic structure to provide for fee distribution. Eighth, we describe some details of practical deployment. Finally, we show an extended example of a typical gambling scenario, a slot machine.

2 What is a casino? A miserable pool of capital

The essence of a casino is a pool of capital that is used to take the opposite side in bets initiated by players.[1] This pool of capital is called the *bankroll* and those who pool it are called the *house*. For example, some investors may come together and pool \$1 million dollars of capital and offer 2 to 1 odds on predicting the result of coin flips: players come; put up one dollar; state whether they expect the coin to be Heads or Tails; the casino operator flips the coin; if they were right, they get two dollars, and if they were wrong, they get nothing.

This basic example suggests some questions:

1. What is the maximum number of dollars the casino should allow to be bet at once?
2. After many bets, how much should we expect to be in the bankroll?

The first question is about the bet limit, while the second is about the expected outcomes of the bets. These two questions are intricately linked. We will focus on the second question first: what is the expected result of many bets?

We can derive the expectation of many bets from the expectation of a single bet. If the coin is fair, then half of the time it will be Heads and half of the time it will be Tails. This means that with probability 50% the casino will lose \$1 and with probability 50% the casino will gain \$1. Since these two values and probabilities are equal and opposite, the expected value of the bet is \$0. This means that neither the casino, nor the player, should expect to gain anything from such a bet.

Therefore, if there are only single dollar bets allowed, then we would expect that the total value of the bankroll is always equal to \$1 million dollars, because it is extremely unlikely for there to be many repeated casino-losses in a row. For example, for the casino to be down \$100, players would need to be correct 100 times in a row without fail. The likelihood of this is $\frac{1}{2}^{100} \approx 1^{-30}$. This is extremely unlikely.

[1] We are deliberately ignoring the bookmaking business of many casinos, where the casino facilitates the matching of two players on opposite sides of a bet and charges a spread fee.

On the other hand, if the casino allows a \$500,000 bet, then it only needs to lose one time (50%) for the entire bankroll to be wiped out. This is why the first question, on bet limit, is so important for practical casinos. The details of computing such limits is beyond the scope of this paper, but the interested reader should investigate Kelly limits. The essential idea is that we want to compare the expected rate of growth of a strategy (accepting the bet) compared to other strategies (e.g., buying US Treasury bills). The Kelly limit equations will compute the proportion of our wealth that we should put in each strategy. The proportion dedicated towards the bet, multiplied by our bankroll, is the bet limit. (The Kelly limit system can be used to produce different limits for each different game, based on their different expectations).

In the case of our even-odds coin flip bets, the Kelly limit computation will instruct us not to accept bets of any size, even \$1, because there's no expected rate of growth and the null strategy (e.g. buying US Treasuries) is better.

3 Edge: Why the house always wins

The even-odds coin flip is a bad investment for the house, because the expected value is 0. For a single outcome bet, we can compute the expected value as:

$$\begin{align}
EV &= P(\text{Win}) \times V(\text{Win}) + P(\text{Loss}) \times V(\text{Loss}) \tag{1}\\
&= P(\text{Win}) \times V(\text{Win}) + (1 - P(\text{Win})) \times V(\text{Loss}) \tag{2}\\
& \text{...in the case of even-odds coin flip...} \tag{3}\\
&= 0.5 \times 1 + (1 - 0.5) \times (-1) \tag{4}\\
&= 0.5 \times 1 + 0.5 \times (-1) \tag{5}\\
&= 0.5 - 0.5 \tag{6}\\
&= 0 \tag{7}
\end{align}$$

In this equation $P(X)$ represents the probability of outcome X, while $V(X)$ represents the value of that outcome. ($V(\text{Loss})$ is -1, not -2, because the house pays \$2, using the player's dollar and one of its own.) Since this value is zero, there is no value in accepting the bet.

How might a casino modify the bet to make it profitable? Given that there are three variables ($P(\text{Win})$, $V(\text{Win})$, and $V(\text{Loss})$), there are three options:

1. Increase $P(\text{Win})$, making it more likely for the casino to win.
2. Increase $V(\text{Win})$, i.e. the cost of making the bet, so the casino gains more per win.
3. Decrease $V(\text{Loss})$, i.e. the amount the player wins, so the casino loses less money per loss.

In principle, all of these options are possible. In practice, the first is most commonly done. But the third is the easiest to explain in the coin-flip case.

Consider that rather than accepting \$1 and paying out \$2 on a predicted coin-flip, instead we accept \$1 and pay out \$1.98. In this case the computation is as follows:

$$\begin{align}
EV &= P(\text{Win}) \times V(\text{Win}) + P(\text{Loss}) \times V(\text{Loss}) \tag{8}\\
&= P(\text{Win}) \times V(\text{Win}) + (1 - P(\text{Win})) \times V(\text{Loss}) \tag{9}\\
&= 0.5 \times 1 + (1 - 0.5) \times (-0.98) \tag{10}\\
&= 0.5 \times 1 + 0.5 \times (-0.98) \tag{11}\\
&= 0.5 - 0.49 \tag{12}\\
&= 0.01 \tag{13}
\end{align}$$

Thus, over many bets, we expect the casino to make 1% returns. This is quite bad compared to US Treasuries, but it is positive!

The (positive) difference between the expected value and zero is called the *house edge*. (Actually, people normally think of the house edge as being the difference between $P(\text{Win})$ and 0.5 (in a single outcome bet), because that represents the higher probability of the house winning, as compared to the player.) With unit bets, the house edge is essentially the rate of return of the capital in the bankroll, assuming an inexhaustible supply of bettors.[2]

4 Life without a Bankroll: Probabilistic Minting

With this foundation, we can begin to address FARE and its goals. FARE enables developers to create decentralized, permission-less, and censorship-proof casinos, in addition to other value-laden probability consuming computations. One of the essential problems in realizing that goal is dealing with the necessity of assembling a bankroll of sufficient size. It is simply not feasible for decentralized parties to assemble such capital, nor would it satisfy the other goals, because the capital providers would be easily identifiable.

FARE's innovation is based on a simple insight: the market cap of a token is analogous to the value of a bankroll, where inflation (or rather, deflation) is a measure of its growth rate. In typical cryptocurrency tokens, the inflation is tied to a preplanned schedule, but this is not a necessary part of how tokens must work. Instead, we could tie the token supply rate directly to the value of the bankroll. A bankroll increases in value when players lose and decreases in value when players win. A token increases in value when the supply shrinks and decreases when the supply grows. This implies that a token-based betting system with no bankroll should tie player loss to supply shrinking and player wins to supply growth. In other words, when players lose, they burn tokens and when players win, they mint tokens. Moreover there is no other way to either burn or mint tokens: it is completely controlled by players and their gambling behavior.

It is typical for ERC-20 tokens on Ethereum to support operations like `mint` and `burn` that are available to special "administrator" accounts. Instead, imagine that we have a single function called `mintOrBurn`. This function has one argument, n, and it randomly either burns n from the caller's account or it mints n to the caller's account. This is like the aforementioned equal-odds coin flip: half of the time, the bettor wins and receives (newly minted) assets; and half of the time, the bettor loses and some of their assets are burned. A token with this function would have the same value proposition as the simple example casino bankroll, except with a crucial difference: this token could support arbitrarily sized bets on day one, because winnings can always be minted.

Of course, if we want a profitable bankroll, or a deflationary token, then we need to ensure that the burns (player losses / casino wins) are likely to outnumber the mints (player wins / casino losses). This is done by creating a house edge by either decreasing the player's gain on a win or by increasing the likelihood of loss. We could imagine the token having a global configuration that controls the bias on the coin flip inside of `mintOrBurn` as a mechanism for increasing the likelihood of player loss. This configuration is a "knob" that controls the token's deflation. For the rest of this white-paper, we assume that there is some house edge and thus the token is overall deflationary.

There is an interesting side-effect of this system: when players win, they receive assets that appreciate in value. Players bet FARE and receive FARE when they win, but FARE is deflationary, so the 100 FARE they win will be worth more tomorrow, based on how much the FARE protocol ecosystem wins over the next day. It is as if you played at a physical casino in Las Vegas where instead of chips, you played with shares of the casino itself: winning means receiving an asset that increases in value when other players lose. This creates a remarkable situation where players can always be happy about activity in the FARE ecosystem:

- When players win, the individual player is happy to receive an asset of value, but holders (past winners) are unhappy, because their token is inflated.
- When players lose, the individual player is unhappy to lose an asset, but holders (past winners) are happy, because their token is deflated.
- The house has an edge, so player losses dominate, meaning that holders (past winners) expect to be happy in the long run.
- Thus, there is an expectation of value in holding the token, which increases demand for the token and the value possible from betting with the token, which encourages betting.

[2]Why do the bettors take these bets? They are losing money in expectation, but they are presumably getting something in exchange, like fun and excitement.

Of course, the `mintOrBurn` function only allows a trivial and boring kind of bet. It is more interesting to support more bets from different kinds of games.

5 Holding on without losing the edge: Generalized Safe Bets

In real casinos, players are making bets on things other than coin flips and getting payouts other than 1.98 on a 1 bet. For example, consider a bet on 14 in Roulette that pays 35x or 0. Or, consider a "Rock-Paper-Scissors" (RPS) slot machine that returns either 2 (win), 0.5 (tie), or 0 (loss). Neither of these bets can be represented by the aforementioned `mintOrBurn` function. In the case of Roulette, the payout is not equal to the cost; while in the case of RPS, there are three outcomes, not two. In this section, we describe a general way to represent any such bet.

The key to representing arbitrary bets is embedded in the expected value formula:

$$EV = P(\text{Win}) \times V(\text{Win}) + P(\text{Loss}) \times V(\text{Loss}) \tag{14}$$

The defining things about a probabilistic process are:

1. The outcomes (Win, Loss)
2. The probabilities of each outcome (which add to one)
3. The payoff of each outcome

If we extend the EV computation for three outcomes it is:

$$\begin{aligned}
EV = {} & P(\text{Outcome 1}) \times V(\text{Outcome 1}) + P(\text{Outcome 2}) \times V(\text{Outcome 2}) && (15)\\
& + P(\text{Outcome 3}) \times V(\text{Outcome 3}) - \text{Cost} && (16)\\
& \text{...in the case of the RPS bet...} && (17)\\
& = P(\text{Win}) \times V(\text{Win}) + P(\text{Tie}) \times V(\text{Tie}) + P(\text{Lose}) \times V(\text{Lose}) - \text{Cost} && (18)\\
& = 1/3 \times 2 + 1/3 \times 1/2 + 1/3 \times 0 - 1 && (19)\\
& = 2/3 + 1/6 - 1 && (20)\\
& = 4/6 + 1/6 - 6/6 && (21)\\
& = 5/6 - 6/6 && (22)\\
& = -1/6 && (23)
\end{aligned}$$

(Given that the player's expected value is negative, this is a profitable bet for the casino.)

We refer to the sequence of probabilities (P(Outcome 1) through P(Outcome N)) as the Q vector and the sequence of payouts (V(Outcome 1) through V(Outcome N)) as the K vector. (We further simplify by dropping the outcome with a zero payout, because its probability is always eliminated anyways, after being multiplied by zero.) Furthermore, going forward, we will always assume that the cost of a bet is 1 and that players can scale their bet by multiplying the cost (and every payout) by a fixed factor.

With these simplifications in place, we observe that the expected value is simply the dot-product of the QK vectors, minus the cost.

$$EV = Q \cdot K - 1 \tag{24}$$

Therefore, if $Q \cdot K$ is less than 1, then there is house edge. We call the target value for the house edge the *EV threshold*.

Therefore, rather than a token offering a bland `mintOrBurn` that always flips a slightly biased coin and pays out equal values; the token could offer a version that accepts the QK vectors, validates them, and evaluates the corresponding bet. In this case, validate means:

1. Ensure that Q sums to less than one (it does not need to equal one, because the final zero entry can be omitted).
2. Ensure that $Q \cdot K$ is less than the EV threshold.

A token that works this way seamlessly supports any bet of any size without any constraining bankroll and without any need to pre-approve or validate bettors or game interfaces: all valid bets are automatically supported without approval, while guaranteeing that the deflationary constraint on the token supply is maintained, by virtue of the EV threshold constraint.

6 Finding the α: Deriving QK Vectors

Given this mathematical model, how do we map human level gaming experiences to particular QK vectors? For example, what is the QK vector for the Roulette game? What is the QK vector for a typical slot machine with six different possible payouts for different combinations of fruit?

Most people tend to think of bets as being about observations of some probabilistic process: the Roulette wheel spins, the ball lands in a spot, we observe the spot, and that determines the outcome of the bet. In this model, we have a process with outcomes having some probability, and we may want to know what the "appropriate" payout for that particular outcome is. This is like knowing Q (the probability vector) but not knowing K (the payout vector).

Alternatively, you can think about the bet as being fundamentally about certain possible payouts: you are either going to get 200x, 50x, 15x, 1x, or 0x. The process that derives why you get these payouts is irrelevant: what matters are the payouts themselves. But, we are curious about what the "appropriate" probability is for each payout. This is like knowing K (the payout vector) but not knowing Q (the probability vector).

Mathematically, these situations are identical: we have one equation $EV = Q \cdot K - 1$ where there are two knowns (EV and either Q or K) and one unknown (K or Q). How might we determine the unknown from the known?

Consider the case of a single outcome, where Q is known and K is unknown:

$$\begin{aligned} EV &= Q \cdot K - 1 && (25)\\ &= Q_0 \times K_0 - 1 && (26)\\ EV + 1 &= Q_0 \times K_0 && (27)\\ Q_0 &= \frac{EV+1}{K_0} && (28) \end{aligned}$$

We can do some basic algebraic balancing and derive an equation for the unknown. Consider the case of the 35x payout of the Roulette bet with a 1% house edge, the probability should be:

$$\begin{aligned} Q_0 &= \frac{EV+1}{K_0} && (29)\\ &= \frac{-0.01+1}{35} && (30)\\ &= \frac{0.99}{35} && (31)\\ &\approx 0.028 && (32) \end{aligned}$$

But, we cannot determine the probabilities as soon as the QK vectors have more than one element, because we have two unknowns (Q_0 and Q_1) but only one equation: the system is under-constrained.

One way to think about this problem is that there is a pre-determined EV threshold and we can "distribute" that EV to different outcomes. Let α be a value in $[0, 1]$, which represents the share of the EV allocated to the first outcome. Then the equation for two unknowns is:

$$\begin{aligned} EV &= Q \cdot K - 1 && (33)\\ &= Q_0 \times K_0 + Q_1 \times K_1 - 1 && (34)\\ EV + 1 &= Q_0 \times K_0 + Q_1 \times K_1 && (35)\\ (\alpha + (1-\alpha)) \times (EV+1) &= Q_0 \times K_0 + Q_1 \times K_1 && (36)\\ \alpha \times (EV+1) + (1-\alpha) \times (EV+1) &= Q_0 \times K_0 + Q_1 \times K_1 && (37)\\ \alpha \times (EV+1) &= Q_0 \times K_0 && (38)\\ Q_0 &= \frac{\alpha \times (EV+1)}{K_0} && (39)\\ Q_1 &= \frac{(1-\alpha) \times (EV+1)}{K_1} && (40) \end{aligned}$$

So, if we apply this to a game with a 5x and 10x payout and a 1% house edge, we get:

$$Q_0 = \frac{\alpha \times (EV + 1)}{K_0} \tag{41}$$
$$= \frac{\alpha \times 0.99}{5} \tag{42}$$
$$Q_1 = \frac{(1 - \alpha) \times (EV + 1)}{K_1} \tag{43}$$
$$= \frac{(1 - \alpha) \times 0.99}{10} \tag{44}$$

We can then apply these formulae to $\alpha = 0.5$:

$$Q_0 = \frac{0.5 \times 0.99}{5} \tag{45}$$
$$= 0.0990 \tag{46}$$
$$Q_1 = \frac{0.5 \times 0.99}{10} \tag{47}$$
$$= 0.0495 \tag{48}$$

But we could just as well choose $\alpha = 0.1$:

$$Q_0 = \frac{0.1 \times 0.99}{5} \tag{49}$$
$$= 0.0198 \tag{50}$$
$$Q_1 = \frac{0.9 \times 0.99}{10} \tag{51}$$
$$= 0.0891 \tag{52}$$

This is a drastic difference! In one case, the probability of winning 10x is 5% and in the other it is 9%! Yet, the expected value (or house edge) is the same in these two different schemes.

This α-based way of dividing the EV among many outcomes generalizes to any number of outcomes: we simply need a sequence of α-coefficients that sum to 1, one for each of the outcomes.

Given the symmetric relationship between Q and K, we can use this same scheme for deriving K from Q as we just demonstrated we can use it to derive Q from K. This means that we can translate any probabilistic process with certain probabilities of observable outcomes into safe payouts for bets on those outcomes; *and*, we can translate any payout table into safe probabilities of winning those payouts. Moreover, we can do this in a wildly flexible way that essentially allows game designers (or players!) to choose which outcomes they'd like to dedicate more EV to: imagine a slot machine that allows you to increase the probability of certain payouts based on your own preferences!

7 Remember to tip the dealer: Supporting fees

Real casinos do not post placards of QK vectors next to each game; and real players care about more than just the expected value of each bet. Real gamblers are getting something from the experience of gambling beyond the pure probabilistic payout potentials. This is why famous casinos and games have exciting lights, drinks, entertainment, ambiance, and why each game has its own unique flavor and history that makes it special. In theory, the only thing that separates a game of Blackjack at the Bellagio and the Wild West Casino behind Days' Inn is just the size of the bankroll and the bet limit; but in practice, there is a gigantic difference.

In the FARE ecosystem, we want to create a system that can reward the Bellagio for attracting players differentially from the Wild West, even though they end up making the same smart contract calls on the FARE contract. (Here "the Bellagio" means some frontend site that attracts more players, perhaps by being more enjoyable, creating a better community, or being better advertised.) We call the category of actor that runs a frontend and facilitates bets the "host" or "dealer".

The most obvious way to do this is to charge a fee to the user independent of their bet. This means that when a player makes a unit bet on some QK vector, they pay an additional ϕ amount which would be sent to the dealer. If we were to write down this idea in pseudo-code it would look like this:

When a player makes a bet on a QK spec with a given `host`:

1. Transfer ϕ from player to `host`
2. Burn 1 of player's tokens
3. Determine outcome, i, based on Q
   (a) If $K_i > 0$, mint K_i tokens to player.
   (b) If $K_i == 0$, do nothing.

Although simple, there are some subtle consequences of this fee setup. We can get at these with the following questions:

1. How does this change the player's expected values?
2. Why would a player designate a host?

Player Expected Value.

Assume that network-wide EV threshold is 0.99. This means that when a player puts in a unit bet of 1.00, they expect to receive 0.99 in return. If the fee, ϕ, is 0.01, then they pay 1.01 to get 0.99 in expected value. If we normalize this back to 1.00, then the EV threshold is approximately 0.98. This means that the ϕ value effectively increases the house edge, with the increase going to the host, rather than the house.

However, player psychology suggests that players find fees like this distasteful, because they are more overt than obscuring the decreased expected value behind a lower probability of success. We can address this by paying the host from either an actual unit bet or via the winnings, and adjust the probabilities (or EV threshold) to compensate for the different payout values.

Incentivized Hosts.

In the above description, the player "designates" a host; what does this mean in practice? Essentially the user will make a smart contract call (constructed by a frontend) that includes a field for `host`. It would be trivial to monkey-patch any frontend to remove this field (or set it to be the player themselves). Essentially, there is no benefit to the player for setting a host, except the abstract benefit of consuming the additional glitz of the Bellagio. It would essentially act as a "tip": an optional payout that a sophisticated user could bypass, but that the average user might not because of the inconvenience of avoiding it.

We solve this lack of incentivization by splitting off a percentage of the fee into a "jackpot". Each host designates a ϕ_j percentage of the entire fee that is transferred not to the host, but to a special jackpot register. Any time a player bets, in addition to winning the bet associated with the given QK vectors, they have a chance to win the jackpot. Jackpots grow as hosts are effective in drawing players to them, which creates a positive feedback loop as the expected value of bets to particular hosts increases with the size of the jackpot.

Further Extensions.

We can extend this distinction between the percentage of a fee that goes to the host and to the jackpot to include more fee recipients. For example, we can require that all fees give some portion to a network treasury to pay for active maintenance operations and ecosystem support. We can require that there be some floor of house edge that is a mandated burn amount, so that fees can consume all of the deflationary power of the FARE token.

Our actual system incorporates all of these extensions and allows each host to set their own fee schedules and percentages, subject to a network-wide constraint on the minimum value burned and the minimum value given to the network treasury. This allows hosts to experiment with different fee structures and determine which are most appealing to users.

8 Playing safely with trusted RNG

Gambling is appealing because of its unexpected positive outcomes. The logic of house edge is only valid if there is no systemic bias to the random outcomes. In other words, a house is wise to invest in a bank roll if the house edge is actually 1%, but not if it is actually -10%. Similarly, players intuitively understand that there is an edge, but they must trust that the edge is 1% and not 50%. Each one of these considerations is connected with the trustworthiness of the random sampling mechanism that determines the outcome of a bet.

In this section, we discuss the implementation details of how to evaluate bets based on QK vectors, with particular emphasis on how to deploy such an implementation on actual blockchain networks.

Evaluation algorithm.

A QK spec contains two parallel vectors of probabilities (Q) and payouts (K). For example, the vectors $Q = [0.125, 0.0625, 0.03125, 0.5]$ and $K = [2, 4, 8, 0.5]$. This means there is a 12.5% chance of winning double and a 50% chance of losing only half. What is an algorithm to randomly determine which payout to return?

An incorrect algorithm is to consider four random numbers in the range $[0, 1]$, one for each outcome, checking if the generated value is below Q_i, because those random values are new trials. If you did that in order, then, for example, the probability of winning 8x would be $(1-0.125)\times(1-0.0625)\times 0.03125$, because you'd have to fail the previous trials to get to attempt the 8x outcome.

Instead, we have to consider only one random number and use that single source of randomness to determine which outcome is used. One algorithm to do this is as follows: Given random number $r \in [0, 1]$, check if $r <= Q_i$; if so, then the payout is Q_k; otherwise, set $r = r - Q_i$ and increment i. Thus, a value that would result in an 8x payout in our example is 0.203125, because this value is greater than 0.125 and we set r to 0.078125, which is greater than 0.0625, so we set r to 0.015625, which is less than 0.03125, so we pay out $K_2 = 8$.

Thus, if we can acquire one random number in the range $[0, 1]$, then we can evaluate any QK vector.

Random number generation (RNG).

The algorithm discussion assumes that random numbers are available, but it is notoriously difficult to characterize and generate truly random numbers, especially on blockchain networks. The current state of the art is to use an on-chain beacon attached to an off-chain verifiable source of randomness, like a verifiable random function (VRF), as deployed by companies like Chainlink.

The methods use an asynchronous pattern whereby one smart contract registers a request for a random value and passes a unique identifier to the VRF provider, which then, at some future time, returns the random value and the identifier, so the computation can continue.

This structure introduces multiple complexities: first, the VRF provider must be paid for provisioning the random values; and second, the asynchronous pattern introduces delays and the potential for timeouts or dropped requests.

The cost of random number generation, in the case of Chainlink, is paid either in the network currency (e.g. ETH on Ethereum) or in a unique VRF currency (e.g. LINK). The FARE system must collect these fees from users and pass them to Chainlink; or, the FARE ecosystem has to socialize the cost through generalized betting fees that are regularly used to acquire VRF fee tokens.

Although Chainlink rarely drops requests, we have to ensure that bets cannot become locked or dead. In order to prevent this problem, we include a "timeout" provision where if a RNG request fails, then an automated FARE operator posts a random value, which is combined with low-quality network entropy (like block hashes), for carrying out the outcome selection.

RNG risk and analysis.

The status quo in the world of digital gambling is centralized, untrustworthy, and non-transparent RNG. Video poker and slots machines are regulated by government boards that check the RNG algorithms, but such regulation does not use cryptographic standards like VRF. Worse, centralized online gaming sites make vague to non-existent statements about their RNG algorithms.[3] In contrast, the Chainlink VRF is transparent and verifiable. This protects both players and FARE token holders (the house).

In the FARE system, the biggest risk is that Chainlink RNG fulfillment will systematically timeout and the FARE timeout handler will be able to use low-quality, gameable entropy for random generation. In practice, we almost never see these timeouts; and the cost of controlling block entropy is very large, although not infinite. We believe this is a reasonable trade-off of risk and performance.

Other technical details.

The evaluation algorithm assumes that a random value is a number in $[0, 1]$, but Chainlink provides 256-bit integers in $[0, 2^{256} - 1]$. Thus, we need to map such values into the $[0, 1]$ space.

One way to do this is to translate probability vectors like $Q = [0.125, 0.0625, 0.03125, 0.5]$ by multiplying by $2^{256} - 1$, so $Q_0 = 1.4474011155 \times 10^{76}$. But, this is impractical for computing expected values, which the FARE contract must for determining if a bet is safe.

Instead, we could consider the 256-bit integer as a fixed point number, such as by using the first 128 bits for the integer component and the second 128 bits for the fractional component. If we did that, almost no numbers would be in the $[0, 1]$ range, but we could ignore the integer component and arrive a 128 bit random value in the $[0, 1]$ range that could be used for all other computations.

Unfortunately, there is no good open-source, trustworthy library for such fixed-point numbers on Ethereum. Instead, most fixed-point number libraries are decimal fixed-point, meaning that they use a significantly smaller number of bits for their fractional component. In our first version of FARE, we use PRBMath, which provides 18 decimal fractional components, or about 60 bits. (We considered using ABDK Math, which provides 64-bits of fixed fractional precision, but found that library to be less efficient and featureful.)

Given that we only use the fractional component of the random value, this means that Chainlink's 256-bit random value is reduce to just 60 bits of randomness in FARE. This means that outcomes with a less than 10^{-18} probability cannot ever occur in FARE. For comparison, the probability of winning the hardest lottery in the world, the United States Powerball, is $3.42x10^{-9}$; this means that on FARE, the lowest probability event is less likely than winning Powerball twice in a row. From another perspective, if the house edge is 1%, then the EV threshold is 0.99, and the payout on this low probability event would be $0.99/10^{-18} = 99 \times 10^{1}6$; this number is about 10,000 times the GDP of Planet Earth. Unfortunately, FARE cannot handle bets with payouts so large.

This choice also produces a subtle problem in that some fractional values are more common than others (they occur once more), because the number of bits used for the fractional component is not even. Thus, we have to reject random numbers with a maximum integer value, because not all fractional values are represented with that integer value. This rejection is a source of possible VRF timeouts.

9 Example: A Slot Play

In this section, we consider a real-world gambling scenario, represent it as a QK vector, and describe one evaluation scenario.

Consider a slot machine with the following payout table:

Three Bells	800
Three Strawberries	200
Three Oranges	40
Three Plums	15
Any Three Fruits	6
Any Two Bells	6
Any One Bell	2

[3]One online casino has a "Fairness and RNG Testing" page that says nothing more than "[Casino] certainly adheres to the industry's best practices by utilizing a Random Number Generator (RNG) to ensure that the outcomes of their games are fair and not foreseen - guaranteeing the unpredictability of each game. The RNG uses a computational algorithm to produce a sequence of numbers and/or symbols which cannot be mathematically determined." There is literally nothing more available about this site.

This slot machine can be modeled with the vector $K = [800, 200, 40, 15, 6, 2]$. Even though there are two outcomes with a payout of 6, it only appears one time in K, because the graphical depiction of that outcome is independent of the payout amount. Whatever outcome the evaluation mechanism selects, the user interface can then generate a display that shows that particular selection of images showing up. (In other words, the reel display is a *result* of the outcome selection, rather than the reels being computed (and displayed), then determining the outcome from those reel values.)

This K vector can be paired with multiple Q vectors for a given EV threshold. Suppose the EV threshold is 0.99 and then EV is split evenly across all outcomes. Then, the vector $Q = [0.00020625, 0.000825, 0.004125, 0.011, 0.0275, 0.0825]$. (The probability of no return is 0.87384375). The expected value (dot product) of these vectors is 0.99, as required.

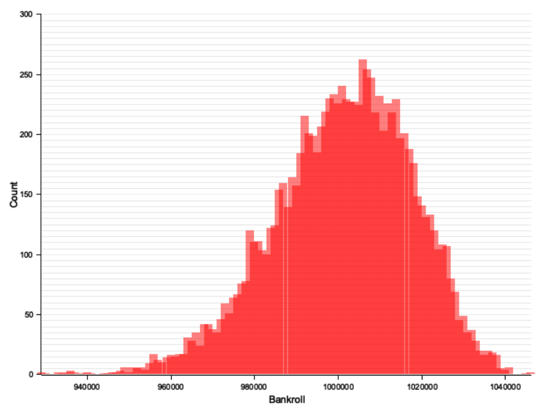
Suppose a user places a 5 FARE bet on this machine. They authorize the FARE contract to burn 5 FARE, which it does, as well as transferring some small fee to the host, jackpot, and network treasury. Then, the FARE contract registers a VRF request to Chainlink with the identifier 1337.

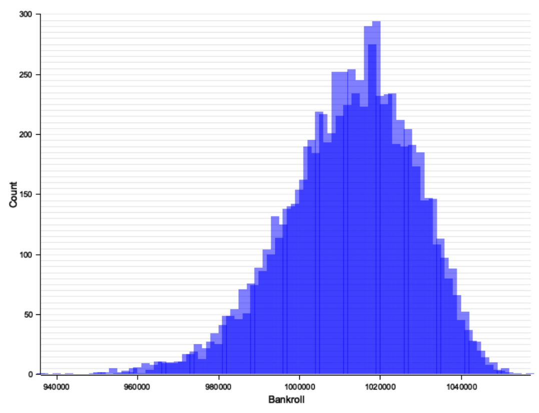
While the animation of a spinning slot machine runs, Chainlink posts a random number to FARE with the id 1337. In this case, it posts a number interpreted as 5231.01615625. FARE ignores the integer components of sets $r = 0.01615625$. This value is greater than 0.00020625, so the payout is not 800x. r is set to 0.01595, which is greater than 0.000825, so the payout is not 200x. r is set to 0.015125, which is greater than 0.004125, so the payout is not 40x. r is set to 0.011, which is less than or equal to 0.011, so the payout is 15x.

The FARE contract mints $15 \times 5 = 75$ FARE and transfers it to the player and emits an event. The front-end observes the event and completes the animation with three plums showing up and an extremely exciting explosion of coins.

This outcome is very unlikely, so those 75 FARE tokens were minted only after a larger number of tokens were burned in the past.

In our simulation environment, we ran a trial of 10,000 runs of 10,000 bets, with an initial bankroll of 1 million, where every bet's maximum payout was 1% of the total bankroll, and found that the average ending bankroll was $1,001,214.75$. A histogram of the final bankrolls (averaged to the nearest $1,000$) is shown on the left in red):





The right-side weight of this graph shows that this game is likely to produce returns for the house, but only in expectation. The house could use a more aggressive edge, such as 0.90, and receive a histogram like the one on the right in blue.

10 Conclusion

This paper describes the architecture of FARE protocol, a Web3 probabilities protocol that facilitates a generic and powerful gambling mechanism. We've demonstrated how this mechanism supports arbitrary bets, fee distribution to multiple parties, and can be deployed safely in the modern Ethereum ecosystem. For more information about FARE, please visit our site at https://www.fareprotocol.io.