Probabilistic Smart Contracts: Secure Randomness on the Blockchain

Krishnendu Chatterjee*, Amir Goharshady*, Arash Pourdamghani**

*IST Austria, **Sharif University of Technology
Random Numbers on the Blockchain

- Current programmable blockchains do not allow probabilistic behavior.
- Probabilistic programs are much more general than non-probabilistic programs.
- Many financial contracts (e.g. lotteries and gambling) are inherently probabilistic.
- Random number generation can be used for proof-of-stake mining.
- Many distributed algorithms and protocols rely on randomness.
All names, characters, businesses, places, events and incidents portrayed in this talk are either the products of the author's imagination or used in a fictitious manner. Any resemblance to actual persons, living or dead, or actual events is purely coincidental. I am a poor graduate student who cannot afford legal fees. Do not sue me,
The Lottery Story

- Ed, a well-known celebrity and billionaire, is rolling the raffle drum 4 times to find a winner.
- When the number 8 comes out in the 2nd draw, Ed says he hates this number, puts it back in the drum, and rolls it again.
- Has Ed cheated?
The Lottery Story

- Turns out Ed had bought half of all the tickets.
- He did not buy any tickets with 8 in them.
- By this trick, he increased his chance of winning the lottery.

**No-redraw rule:**
Redrawing is cheating!
*Ed should not be able to change the results.*
The Lottery Story

- Next year, the organizers ban redraws.
- Ed is rolling the drum again.
- The number 8 never appears in the rolls.
- Turns out Ed has bribed the drum maker.

**No-centralization rule:**
Centralization is cheating!

No central authority (including the lottery organizers) should make or roll the drums.
The Lottery Story

- Next year, the organizers invite 4 celebrities.
- They each bring their own drum.
- Each celebrity draws a number and announces it. Ed is last.
- Ed wins again!

Concurrency rule:
Everyone should draw at the same time!
(or at least before knowing other draws)
• Next year, the organizers enforce concurrency.
• Ed does not announce his number.
• He just walks away.
• The organizers have to invite another celebrity for the 4th draw.
• Ed wins.

**Penalty rule:**
There should be a penalty for not announcing the draw. The penalty should be at least as big as the lottery prize itself.
Next year, the organizers enforce penalties using deposits.
Ed wins.

Rule of 1:
Even if one participant is generating uniformly random draws, the whole result should be uniformly random.
The Lottery Story

- Next year, each participant draws 4 times.
- The result is determined by XORing.
- Ed wins.
- Turns out he bribed every participant.

**Openness:**

*Drawing should be open to everyone.*

*Let's do it on the blockchain!*
Next year, anyone who can pay the deposit can participate.
The result is determined by XORing.
Ed wins.
Turns out no one is willing to participate and deposit money without being paid.

Incentivization:
Each participant should be paid for their input.
• Next year, anyone who can pay the deposit can participate. Each participant receives a reward for providing random numbers.
• Ed wins.
• Turns out participants did not buy drums. They just reported 0s as the result.

_Incentivization:_
Each participant should be paid for their input. It should also be in their best interest to provide uniformly random inputs.
More on Incentives

• Consider a classic one-shot game with n players.
• Nash Equilibrium: No player is willing to change strategies.

What if the players can collude?

• Strong Nash Equilibrium: No set of players can collude to change strategies so that all of them profit.

What if the players can share rewards?
• Quasi-strong Nash Equilibrium: No set of players can collude to change strategies so that their total payoff increases.
Previous Approaches

- Relying on block hash/timestamp *(Ed is the miner)*
- Using an oracle *(Ed is the oracle owner)*
- Using commitment schemes *(No incentivization for random inputs)*
  - In the registration phase:
    - Each participant pays a deposit
    - They commit to a bit $b$, by submitting $\text{hash}(b, \text{nonce}, \text{id})$.
  - In the revealing phase:
    - Each participant reveals their nonce

The generated random bit is the XOR of all submitted bits.

Rewards for each participant who reveals the correct nonce. Confiscation of deposit for others.
Our Approach

• Use commitment schemes
  • but *let the reward depend on the submitted random bits*

• Make it a game where submitting *uniformly random* bits is the only quasi-strong equilibrium
The Game

- n players.
- An even-numbered player can play either 0 or 2.
- An odd-numbered player can play either 1 or 3.
- Let’s say that player $i$ plays $s_i$. Then the utility for player $i$ is:

$$u_i(s_1, \ldots, s_n) := \sum_{j \neq i} f(s_i, s_j)$$

where

$$f(s_i, s_j) := \begin{cases} 
1 & s_i \equiv s_j + 1 \pmod{4} \\
-1 & s_i \equiv s_j - 1 \pmod{4} \\
0 & \text{otherwise}
\end{cases}$$
The Overall Protocol

• Implemented as a solidity smart contract that can be called by other contracts for generating random bits.
• Consists of 6 steps:

1. Another contract/node requests a random bit and sets the penalty and the reward.
2. Participants can register in a given timeframe. To register, they should provide:
   • A deposit equal to the penalty
   • hash(b, nonce, id)
3. In a given timeframe after the registration, each participant has to reveal their nonce.
4. The deposits are paid back.
5. The game is played and the rewards are calculated. \( r_p := \alpha \cdot \left(1 + u_p(s) / n'\right) \)
6. The output is the xor of the submitted bits.
Guarantees

Secure Randomness on the Blockchain

- No-redraw rule (by design)
- No-centralization rule (by design)
- Concurrency rule (commitment schemes)
- Penalty rule (by design)
- Rule of 1 (due to XOR)
- Openness (anyone can register)
- Incentivization (due to the game)
- Safety against malicious miners (block withholding, DoS)