I've been working on a protocol to generate publicly verifiable random numbers. My idea is to overcome the last actor attack [1] by using multiple entropy sources to feed a lightweight random number generator, which i have already implemented using inline assembly in Solidity [2].

Anyone that needs to generate a random number can generate a testimony, This testimony includes self selected "random numbers", expecting the counter part (the entropy providers) to fulfill the rest of the input of the PRNG. Best case scenario, is having all the entropy sources filled, so the number is generated in a combination of numbers delivered by the requester and the entropy providers. None of them knows how their inputs are going to be used until settlement,

so they don't have a way to selectively disclose information, which is the last actor attack (the last entropy provider to send data can decide the final output).

Entropy providers are incentivized to fulfill requirements by requesting some locked value, but, if they fail to fill this requirements, the user can generate the random number with the already delivered entropy sources (of course the most entropy sources delivered, the better).

My current implementation makes use of Chainlink, as Chainlink provides with the necessary architecture to align incentives between nodes operators (entropy providers) and requesters.

I'm attaching a little explanation here →https://drive.google.com/file/d/10sbXuKuFqSuQi5vxff_IQzoGhsZy77YB/view

Any feedback is appreciated. I'm halfway demonstrating that the "lightweight random number generator" is indeed cryptographically secure random by using a NIST statistical test suite [3]. Partial tests results can be found here in [4]. Best feature is that it is super easy to implement, so basically it can be considered a universal implementation since it only uses modulo, sum, adds and divisions. In [5] you can find a Python implementation. I'm planning on creating libraries for multiple languages (currently implemented in Solidity, Python, JavaScript and C).

Maybe there are still so many lose pieces, i'll be glad to elaborate more if requested.

Thanks!

[1] https://arxiv.org/pdf/1807.09225.pdf

[2] luckyseven-protocol-contracts/LuckySevenPRNG.sol at 1de0455b89a311ee7e536e5cdac070e51bb5b1eb · luckysevenio/luckyseven-protocol-contracts · GitHub

[3] https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf

[4] luckyseven/sts at master · matiasbn/luckyseven · GitHub

[3] luckyseven/luckyseven.py at 4d970e3063292587481d6d1ce5ffa8a952392315 · matiasbn/luckyseven · GitHub