# Dynamic User Entropy

Secure Random Number Generation in
Distributed Adversarial Card Games

Boris Radulov
@VAPORGRIPS
sc20br@leeds.ac.uk

Nikolay Mitev
@T_SYMMETRY
nikolay.mitev@yale.edu

José Betancourt
@TRIDENTITY_
jose.betancourt@yale.edu

22 June, 2022

**Abstract**

The following paper outlines a decentralized protocol for generating verifiably random numbers. Furthermore, we show different methods for dealing with duplicate values (i.e. collision detection) for use in dealing cards. The protocol requires no central authority for shuffling the decks or mediating communication. Specifically, the outlined implementation is intended to be used for card games but can easily be adapted for any multiplayer RNG game.

# 1 Introduction

This paper outlines a decentralized protocol for generating random numbers that focuses on Agreement and Fault Tolerance. Besides generating random numbers, the protocol is well geared towards card games as it also has built in collision detection that can be used to prevent the duplication of cards. Despite this, it can easily be adapted to non-card games as well, as some dApps in web3 currently don't require collision detection. This paper is a continuation of previous efforts in the field such as Golle's[1] and Rivest-Shamir-Adleman's[2] work. It satisfies the following properties:

- secure up to only one fair player;
- no central authority for shuffling or mediating communication;
- implements collision detection (i.e. it is impossible for players to deal the same card twice);
- can detect when and which players attempt to cheat.

To achieve this, we use a combination of Pedersen DKG[3], ElGamal Homomorphic Encryption[4], Schnorr Singatures[5], and DISPEP[6]. The general outline is as follows:

1. Players use DKG to create a shared public key for an ElGamal cryptosystem where they each have equally-important pieces of the private key.
2. Players chose random numbers and provide cryptographic commitments and encryptions to those numbers.
3. Players then use the homomorphic properties of the encryption to sum the numbers without revealing them.
4. Players do the necessary verification and collision checking.
5. If everything so far has been successful, players can selectively reveal their initial random seeds to others depending on if the randomness needs to be disclosed publicly or to a subset of players.

The protocol also assumes players have a channel to communicate individually in a secure way. In the case of web3, this is true because they all have private / public key pairs associated with their wallets that can be used for communication.

## 2 ElGamal Implementation

We briefly outline the ElGamal asymetric encryption necessary for this protocol. The properties of the keys are as follows: Let $p$ be a 1024-bit prime and $q$ be a 160-bit prime such that $q \mid (p-1)$. Let $g \in \mathbb{Z}_p^*$ be our generator. For any private key $x \in \mathbb{Z}_q^*$, the public key can be obtained as follows: $y = g^x \mod p$.

To encrypt a message $m$, pick a random entropy coefficient $e \in \mathbb{Z}_q^*$ and let $E(m) = (g^e, my^e)$. To decrypt a ciphertext $(a, b)$, compute $\frac{b}{a^x} = m$. From this, it is trivial to also show that if $(a_1, b_1)$ and $(a_2, b_2)$ and the ciphertexts of $m_1$ and $m_2$, $E(m_1 m_2) = (a_1 a_2, b_1 b_2)$, meaning that ElGamal is homorphic, allowing us to perform mathematical operations on data without revealing it.

## 3 Distributed Key Generation

For the distributed key generation, we use a modified version the method shown by Pedersen[3]. It goes as follows:

1. Every player $i$ generates a random polynomial $f_i(x) = b_{i,0} + b_{i,1}x + ... + b_{i,k}x^k$ where $k = \#\text{number of players} - 1$ and $b_{i,0} = s_i$.
2. Then, every player $i$ computes $B_{i,j} = g^{b_{i,j}}$ for $j \in 0, ..., k$ as well as $f_i(j) \mod q$ for $j \in \{0, ..., k\}$.
3. $B_{i,j}$ and $f_i(j)$ are then sent to players $j \in \{0, ..., k\}$.
4. Each player $j$ verifies $g^{f_i(j)} = \prod_{m=0}^{k} (B_{i,m})^{j^m} \mod p$. Failure to verify for values sent by user $i$ means they're cheating, and a new instance of the protocol can be established without them.
5. After verification, the public key $y = \prod g^{s_i} \mod p$ for all players $i$.
6. The value $s_i$ is player $i$'s private key were the full private key is $x = \sum s_i \mod q$, although that is never assembled in the protocol.

## 4 Generating Randomness

To generate the distributed randomness necessary we first define $\mathbb{N}_D$ which is the set of values our randomness can be in (card games - $D = 52$, roulette - $D = 36$, coin toss - $D = 2$, etc.). We then proceed as follows:

1. Each player $i$ chooses their random seed $c_i \in \mathbb{N}_D$.
2. They compute the ElGamal ciphertext $E(g^{c_i})$ and send a cryptographic commitment $C(E(g^{c_i}))$ to all other players.
3. After the commitments have been verified, the ciphertexts are revealed and players compute $\prod E(g^{c_i}) = E(g^R)$ where $R = \sum c_i \mod D$.

Every player now has access to the encrypted version of the final randomness value $R$.

## 5 Revealing Public Randomness

In the case where $R$ is meant to be a publicly accessed value such as a face-up card or a roulette spin result, the players all reveal their seeds $c_i$ and their randomness coefficients $e_i$ used in computing $E(g^{c_i})$. Players can then verify $c_i$ is in the correct domain (although if it is not, it is trivial to show that cheating of this sort is relatively harmless) and that the ciphertexts provided are in fact correct. If all the encryptions are valid, the players can compute $R = \sum c_i \mod D$. If any of the verification steps fail, the fair players run a new DKG and establish a new group. The punishment for the cheating player depends on where this protocol is implemented and is outside of the scope of this paper. For games that don't require collision detection and private randomness such as coin toss or roulette, the protocol as described up to this point is sufficient. The following sections deal with matters regarding collision detection and private random generation (e.g. face-down cards).

## 6 Conclusion

The outlined procedure has all the necessary functionalities to be used in any online game for decentralized RNG, including games with private randomness and collision detection of random values. The protocol can easily be adapted for the needs of any service requiring decentralized trustless randomness and fulfills the properties and safety guarantees described in the introduction. Because the protocol happens between players, it can be executed off chain in web3 environments for increased speed and significant cost deductions. Combined with the appropriate roll-up, it can transfer the processing responsibility for an online game entirely to the player's machine.

# References


[1] Golle, P. "Dealing Cards in Poker Games." International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II, 2005, https://doi.org/10.1109/itcc.2005.119.

[2] Shamir, A., Rivest, R.L., Adleman, L.M. (1981). Mental Poker. In: Klarner, D.A. (eds) The Mathematical Gardner. Springer, Boston, MA. https://doi.org/10.1007/978-1-4684-6686-7_5

[3] Pedersen, Torben Pryds. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing." Advances in Cryptology — CRYPTO '91, 991AD, pp. 129–140., https://doi.org/10.1007/3-540-46766-1_9.

[4] Elgamal, T. "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms." IEEE Transactions on Information Theory, vol. 31, no. 4, July 1985, pp. 469–472., https://doi.org/10.1109/tit.1985.1057074.

[5] Schnorr, C.P. (1990). Efficient Identification and Signatures for Smart Cards. In: Brassard, G. (eds) Advances in Cryptology — CRYPTO' 89 Proceedings. CRYPTO 1989. Lecture Notes in Computer Science, vol 435. Springer, New York, NY. https://doi.org/10.1007/0-387-34805-0_22

[6] Jakobsson, M. and Juels, A. Millimix: Mixing in Small Batches. DIMACS RUTGERS, 1999, 99-33. http://www.arijuels.com/wp-content/uploads/2013/09/JJ99b.pdf