A Statistical Benchmarking of Keystroke Entropy Pools Versus State-of-the-Art Software Random Number Generators

Koushani Roy
South Point High School
Kolkata, India
koushaniroy21@gmail.com

***Abstract*—Randomness lies at the foundation of computational modeling, simulation, and cryptography. While keystroke dynamics have been explored as a supplementary source of entropy for cryptographic systems, rigorous, modern benchmarking of their standalone statistical quality is rarely reported. This project provides a empirical evaluation of a keystroke-derived entropy pool in contrast with two state-of-arts pseudorandom generators: Python's built-in `random` library and NumPy's `PCGC64`. Using a suite of established tests (Shannon entropy test, Wald-Wolfowitz runs test, autocorrelation state), a comparitive analysis has been performed to prove the viability of this data suite. Additionally, the pool has been tested with respect to a Quantum Entanglement simulation loosely based on Monte-Carlo Acceptance Rejection method to demonstrate an application.**

***Index Terms*—keystroke dynamics, entropy extraction, pseudorandom number generator (PRNG), Shannon entropy, Wald–Wolfowitz runs test, autocorrelation, Monte Carlo simulation, quantum entanglement simulation, randomness benchmarking, statistical physics**

I. Introduction

Human behavioral entropy has been studied primarily for biometrics. In cryptographic scenario, most sources use it as supplement for the main data-pool. While keystroke dynamics have seen application in authentication, few efforts have examined them as standalone entropy sources.

Speaking of the PRNGs used for comparison, Python's built-in `random` uses the Mersenne Twister algorithm (MT19937), while NumPy's `PCG64` is a modern generator based on Permuted Congruential Generator (PCG) family. Of the two, NumPy's `PCG64` is generally considered as more robust of the two.

More accurately, the first one is the default in Python and is commonly used in simulations, games, randomized algorithms, and educational tools. Though not cryptographically secure, it is known for its speed while the latter is increasingly used in scientific computing, machine learning pipelines, and data-intensive simulations. It offers better equidistribution and statistical quality compared to Mersenne Twister, while remaining efficient.

To give a fair picture of the random number generation from the keystroke pool, both have been used to show the limits of this data pool.

Classical PRNGs (Mersenne Twister algorithm, PCG, Xoshiro) and cryptographic generators (e.g., system-level CSPRNGs) undergo stringent testing using suites like NIST SP 800-22, Diehard, and TestU01. When coming to keystroke dynamics as a viable entropy generation resource, most explanations stop at "good enough in practice".

This work fills the gap by directly comparing physical keystroke entropy pools (normalized as floats for compatibility with standard libraries) to trusted generators. Both Python's `random` and NumPy's `PCG64` output floats in $[0, 1)$, and this normalization enables fair, direct benchmarking.

II. Methodology

A. Harvesting Keystroke Interval Data

Python pynput library has been used to create a Python logger to measure the time-interval between successive keypresses using the system clock. The process can be approximately summarised as:

- Each keypress was recorded correct to 6 decimal place precision (the last digits are the most random, highest entropy).
- The last four digit of the fractional part was extracted
- For every i in the extraction, $i/9999$ was performed to normalize into a float pool [0,1).
- The choice of extraction of last 4 digit was done on a hit and trial basis as a sweet spot between length and degree of randomness.

The resulting float pool obtained was about 49000 data points in length, with the frequency of distribution was observed as in Fig. 1.

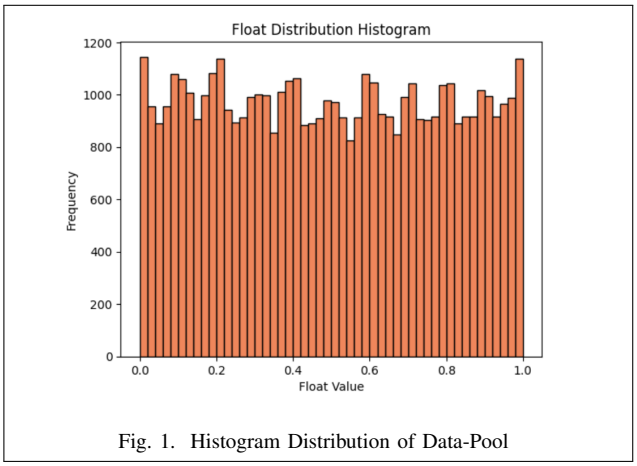

Fig. 1. Histogram Distribution of Data-Pool

B. Benchmark Tests for grading randomness

To grade randomness, three statistical evaluations were conducted for each of the three sources (Keystroke Interval Entropy Pool v/s the PRNGs)

Test	Purpose	Measures
Shannon Entropy	Quantifies uniformity and effective unpredictability	Closeness to ideal randomness
Wald-Wolfowitz Runs Test	Evaluates independence and alternation in sequence order	Sequence independence
Autocorrelation	Measures correlation between values at various lag distances	Sequence memory dependency

TABLE I
TESTS RUN AND THEIR RELEVANCE

a) Shannon Entropy Test: – for a set of values is calculated using the formula:

$$H = -\sum_{i=1}^{n} p_i \log_2 p_i$$

where p_i is the probability of the i-th character occurring in the dataset. In relevance to the experiment at hand, it gives a quantitative value to how close the data pool is to being random or free from bias based on the closeness of the yielded value to the theoretical value as calculated from the formula.

b) Wald-Wolfowitz Runs Test: non-parametric (does not assume any particular distribution in dataset) statistical test employed to test if the sequence of values in data pool occur in random order. For this experiment, the generated p-value (two-sided measurements taken so that both clustering and over-alternation from the expected number of runs are accounted for) from this test is relevant. A p-value > 0.05 indicates randomness.

Since this test specifically tests sequential ordering of values, the algorithm was implemented so as to choose different index starting point each time with circular wrapping so that the a distinct segment is represented every time. This change was made on an empirical basis since this yielded the maximum efficiency of data-set.

c) Autocorrelation test: assesses whether the values in a sequence exhibit dependence across timelags.

C. Entanglement-inspired correlation simulation as an application

From Quantum mechanics, in a standard quantum entanglement experiment, two particles with sufficient wave character (like electrons) are used. If we represent them with bit notation, entangled such that there is two entangled states possible – same spin entangled, 00 or 11 or opposite spin entangled, 01 or 10. One detector is assigned to each particle and when both measure the spin along same axis, result comes out to be as predicted by entanglement state assigned.

However, if we increase the angle difference gradually from 0 to 180 degrees, the probabilistic correlations decrease due to drifting away of the measurement axes from each other: from 100% to 0% theoretically. This behavior can be modeled using the formula:

$$P_{\text{same}} = \cos^2\left(\frac{\theta_1 - \theta_2}{2}\right)$$

In our classical setup for this simulation, the algorithm created loosely resembles the Monte-Carlo Acceptance Rejection method.

The steps can be approximated as:

- Inputs: angle_difference, number_of_runs_per_trial (n)
- Expected correlation (_exp) was calculated using the mentioned formula.
- A trial suite of 10 tests were run for each angle. For a particular trial:
  - A random start index was selected from the float list to preserve the dynamic nature of data-extraction and n floats were consumed sequentially (most deterministic way to traverse through a list).
  - Each run consumed one float, x:
    - if $x <$ *_exp*: interprets as a match since then point falls within the probability area and a parameter m, set to 0 at the start of each run, is incremented by 1.
  - Run is performed n times (each run consumed one float). $\frac{m}{n}$ is calculated as the probability for this particular trial.
- After 10 repetitions per angle_difference, the average (_avg_per_trial) is calculated for that particular angle_difference.
- To plot the graph: This is repeated for each angle from 0 to 180 degrees (step size = 5), and the graph showing _exp vs. _avg_per_trial is tallied.

It must be noted that this is merely a simulation and in no way does it illustrate actual quantum phenomena. The point of this is to show direct application for physics simulations that require the usage of random numbers (especially in statistical physics) and demonstrates the randomness pool's effectiveness in real scientific modeling.

On a final note, the data collection and benchmarking methods described are easily reproducible using standard Python libraries, and the programs used can be adapted for independent verification or further analysis. The code will be shared upon request for replication purposes.

III. Results

A. Test Results:

The results presented here are that of the median value over 100 trials (30 for autocorrelation because the process was too intensive on the system memory, otherwise) because the outcomes here are "stochastic" in nature, hence they change every run and it is fair to present a median value. Mean is not presented since that would give importance to the outliers.

Pool	Shannon Entropy (2-digit bins)
Keystroke	6.6360 bits
Python Random	6.6424 bits
NumPy PCG64	6.6424 bits

TABLE II
TEST RESULTS FOR SHANNON ENTROPY

1) Shannon Entropy : All tested pools achieve nearly the same Shannon entropy and is close to the theoretical maximum of 6.6439 bits for 2 digit binning. Binning over 2-digits reveals how evenly the distribution is maintained over 100 bins. This is standard convention. Binning for 4-digit proved to be inadequate due to the relatively small size of keystroke entropy pool. Hence from here we could conclude that there is no evidence of bias or concentration in any region of the value range for any source.

Pool	Two-Sided P-values
Keystroke	0.4594
Python Random	0.4715
NumPy PCG64	0.5157

TABLE III
TEST RESULTS FOR RUNS TEST

2) Wald-Wolfowitz Runs Test: For the second test, in all three cases, the p-value crosses the min. threshold of 0.05 and the performance of the keystroke data-pool was comparable to that of the standard PRNGs. This suggests the behavior is comparably well in terms of sequence independence and randomness of order.

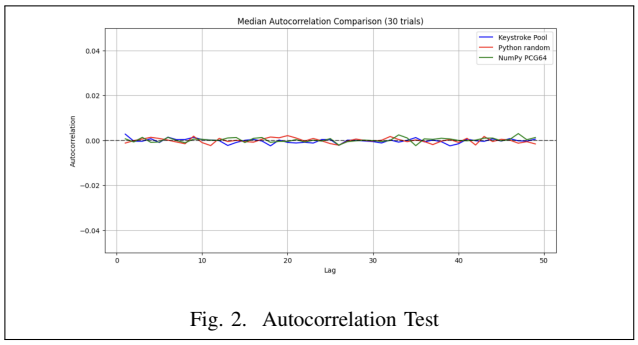

Fig. 2. Autocorrelation Test

3) Autocorrelation Graph Test:

- For better comparison, the results has been plotted in the same space.
- Lag 0 yields correlation = 1 trivially, hence omitted from plots for better visibility.
- All three lines remain tightly clustered around 0 which is expected behavior for random sequences with no visible correlation with previous value.

B. Quantum Simulation Outcome

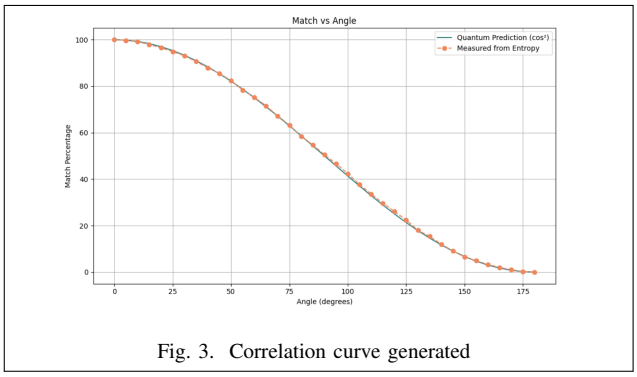

Fig. 3. Correlation curve generated

From the graph, it is evident that the plotted curve reaches the expected curve very accurately, hence showing the usefulness of entropy-pool for simulation purposes. It is again relevant to mention that the purpose of this classical simulation is **not to recreate quantum mechanics**, but to test whether a given entropy source can **statistically reproduce** known correlation patterns, which are sensitive to random input quality. Since quantum simulations often rely on Monte Carlo methods for probabilistic modeling, this one has also made an attempt to use such a strategy. A randomness pool that replicates expected distributions in this setting demonstrates its practical **applicability in statistical physics and computational modeling.**

IV. Conclusion

From the results above, the keystroke entropy pool exhibited statistical characteristics closely matching those of well-established pseudorandom generators such as Python's random and NumPy's `PCG64` across Shannon entropy, Wald–Wolfowitz runs, and autocorrelation tests, with results nearing theoretical ideals.

This supports its practical viability for applications in statistical physics simulations (the Quantum Entanglement simulation being a direct proof), informal password generation, and low-level encryption (low budget personal projects with lightweight security concerns).

Moreover, keystroke entropy is inherently personal to each human and the data can easily be harvested in personal spaces and offices. Notably, the entire system is implemented using lightweight Python tooling and does not depend on cryptographic PRNGs, system entropy daemons, or secure hardware. This makes it especially viable in air-gapped, educational, or constrained environments.

However, more robust applications are discouraged since the current pool is limited to a four-digit precision float

representation. This caps the maximum entropy since we can have not more than 10000 unique floats.

Much larger datasets with enhanced precision (more number of digits) are necessary for future testing against more rigorous cryptographic test suites.

One important point to note is that while the applied statistical benchmarks strongly indicate high-quality randomness, no amount of tests can guarantee absolute randomness although comparison with state-of-art PRNGs has been provided for some comparison.

This work establishes a reproducible baseline for evaluating physical entropy sources and highlights the pathway for refinement and extended validation in future research.

V. Acknowledgment

I would like to thank South Point High School for providing the opportunity to carry out such a project. I am also grateful to my school teachers who provided valuable feedback during development. This project has been carried out with personal infrastructure and no external funding/affliation has been maintained.

References

1) L'Ecuyer, P. (2011). Random number generation. In *Handbook of computational statistics: concepts and methods* (pp. 35–71). Berlin, Heidelberg: Springer Berlin Heidelberg.
2) Metropolis, N., & Ulam, S. (1949). The Monte Carlo Method. *Journal of the American Statistical Association*, *44*(247), 335–341. https://doi.org/10.1080/01621459.1949.10483310
3) Killourhy, K. S., & Maxion, R. A. (2009, June). Comparing anomaly-detection algorithms for keystroke dynamics. In *2009 IEEE/IFIP international conference on dependable systems & networks* (pp. 125–134). IEEE.