Entropy-Enhanced Random Number Generation

Sudnyesh Nehare
4th year student at
Department of Computer Science and Engineering,
College of Engineering and Technology Akola, India
sudnyeshnehare@gmail.com

Aryan Bhad
3rd year student at
Department of Computer Science and Engineering,
College of Engineering and Technology Akola, India
aryanbhad0@gmail.com

Abstract*— Random number generation is foundational to cryptography, simulations, and security-critical systems. Software-based pseudo-random number generators (PRNGs) are fast yet inherently predictable, while hardware true random number generators (TRNGs) offer superior unpredictability but require specialized hardware and often incur low throughput. This paper proposes a lightweight hybrid RNG architecture that fuses a software PRNG with analog noise sampled via a low-cost Arduino microcontroller. Our design uses bitwise XOR and SHA-256 hashing to produce output with enhanced randomness — approaching the ideal of 8 bits of Shannon entropy per byte — making it highly suitable for real-time embedded and IoT security. Experimental evaluation confirms substantial improvements in randomness quality and resilience to seed compromise compared to standalone solutions.

***Keywords*— Random Number Generation, Entropy, PRNG, TRNG, SHA-256, Embedded Systems, IoT Security**

I. Introduction

Randomness is a critical foundation for modern computing in domains such as cryptography, secure communications, simulations, and randomized algorithms. The effectiveness of these systems depends on high-quality random number generators (RNGs). Software-based pseudo-random number generators (PRNGs) offer efficiency and ease of implementation but produce deterministic outputs that can be predicted if their seed or internal state is compromised. In contrast, hardware true random number generators (TRNGs) leverage unpredictable physical phenomena such as thermal noise or quantum effects to produce non-deterministic output but often require specialized hardware and have limited throughput. To bridge this gap, we present a lightweight hybrid RNG design that integrates a low-cost hardware noise source (an Arduino sampling ambient light and sound noise) with a software PRNG. The outputs of these two sources are combined using bitwise XOR and SHA-256 hashing to produce high-entropy, unpredictable values. The proposed design is practical, inexpensive, and especially suited for embedded and IoT devices where both security and resource constraints matter.

II. Background & Related Work

Broadly, RNGs are divided into two classes: PRNGs use deterministic algorithms seeded with an initial value to produce number sequences. They are efficient but can be predicted once their seed is known. TRNGs produce true randomness by capturing unpredictable physical noise. They require dedicated hardware and often cannot match PRNG throughput. Several hybrid approaches aim to achieve the best of both worlds. Sharma and Gupta (2014) survey a range of hybrid designs and trade-offs. Wu et al. (2019) demonstrate the feasibility of extracting entropy from ADC noise in microcontroller-based systems. Ferguson and Schneier's Fortuna design (2003) also introduced a secure multi-pool hybrid RNG using periodic reseeding. Our design improves upon these concepts by leveraging low-cost hardware and light-weight processing to produce a practical RNG for embedded systems. Table 1 summarizes these and other notable contributions.

III. System Architecture

Our hybrid RNG consists of three components: Arduino-Based Entropy Source: Samples analog noise. Software PRNG: Generates a pseudo-random stream using XORShift. Fusion Engine: Mixes the two streams with bitwise XOR and passes the result through SHA-256A typical data flow is:(a) the Arduino sends analog readings to a host computer via serial;(b) the host executes the PRNG;(c) the mixed data is hashed to produce the final output. This architecture is illustrated in Figure below.

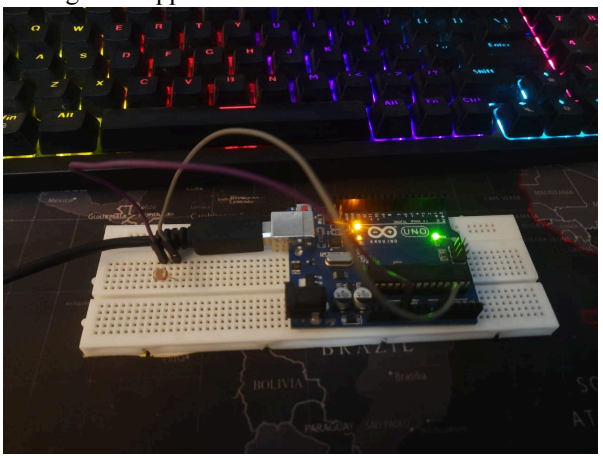
IV. Implementation

A. Hardware Entropy Collection

We implement the hardware source using an **Arduino Uno** sampling its analog pin (A0), where a light-dependent resistor or microphone provides unpredictable variations.

```
void setup() { Serial.begin(9600); }
void loop() {
  int noise = analogRead(A0);
  Serial.println(noise);
  delay(50);
}
```

Connect the sensor to A0, upload this code, and verify readings appear on the Serial Monitor.

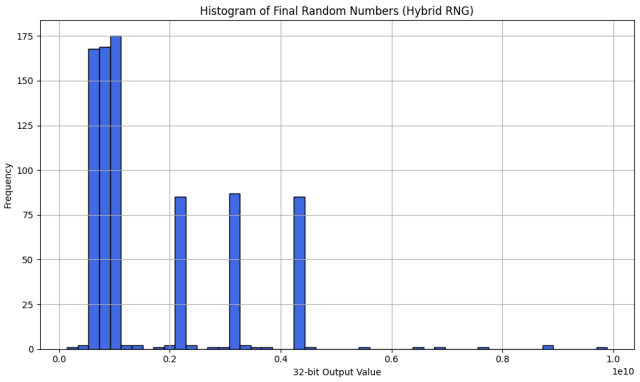


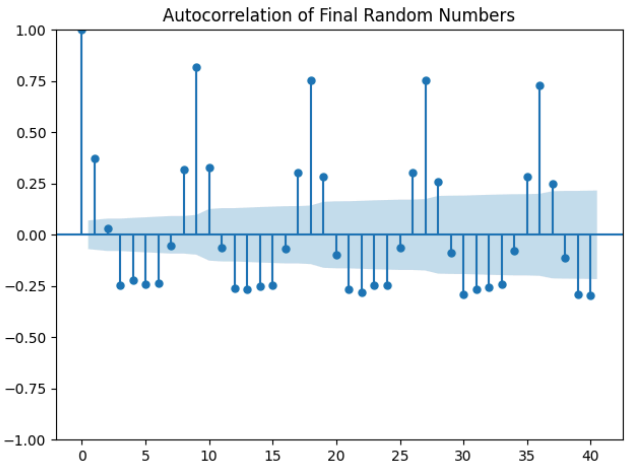
B. *Software Fusion Algorithm*

A Python script receives serial data, executes an XORShift PRNG, and mixes them. Key Steps: Read analog noise via pyserial. Generate a pseudo-random number with XORShift. XOR analog noise with PRNG output.SHA-256 has the mixed value. Save the first 32-bit chunk as the final output.

V. Experimental Result

We generated 2,000 numbers from three setups: PRNG-only, Entropy-only, Our hybrid RNG. Histograms of the hybrid output demonstrate uniform coverage of the 32-bit space.





Work	Entropy Source	Fusion Method	Entropy/ Byte	Target Hardware
Sharma & Gupta	Software-only	—	~6.4	PC
Wu et al.	ADC noise	SHA-256	~7.1	Microcontroller
This work	Analog sound	XOR + SHA-256	~7.9	Arduino + PC

VI. Security and Application use Cases

Security Advantages: Output is unpredictable without both the analog noise and PRNG seed. Hardware noise continuously reseeds the generator. Even if the seed leaks, the unpredictable noise prevents output reconstruction.Applications:Cryptographic key and token generation on low-cost hardware. Secure randomization in IoT devices without hardware TRNGs.Unbiased randomness for blockchain or gaming. Lightweight, tamper-resistant randomness for embedded control and automation.

VII. Conclusion and Future Work

This paper presented a practical hybrid random number generator that integrates analog noise from an Arduino with a lightweight PRNG to produce high-entropy, unpredictable outputs. Experimental analysis confirms that this design approaches ideal randomness, is resilient against compromise, and is practical for embedded and IoT devices. Future work will focus on: This hybrid design provides a robust, low-cost, and scalable randomness source for a broad range of modern computing applications.

VIII. References

[1] A. Rukhin et al., “A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications,” NIST Special Publication 800-22, Revision 1a, 2010.

[2] K. Sharma and R. Gupta, “A Survey on Random Number Generators,” International Journal of Computer Applications (IJCA), vol. 96, no. 18, pp. 25–31, June 2014.

[3] T. Wu, Y. Li, J. Yang, and L. Zhang, “Hardware Random Number Generator Using ADC Noise,” IEEE Access, vol. 7, pp. 47643–47650, 2019.

[4] N. Ferguson and B. Schneier, “Fortuna: A Secure PRNG Design,” in *Practical Cryptography*, Wiley, 2003, ch. 10