Adaptive Self-Healing Architecture for Failure Detection and Automatic Recovery in Hardware TRNG Systems

Lutfi Hamdan
Independent Researcher, Texas, USA
ORCID: 0009-0003-8117-4263 lutfinazam4@gmail.com

Abstract— **Hardware TRNGs can fail under drift, noise collapse, and ADC clipping. Many designs detect failure offline, and they react late. This paper presents a self-healing control layer for embedded TRNG systems. The layer monitors raw bits in real time and scores entropy health. We define an Entropy Confidence Index, ECI, from 0 to 1. ECI combines min-entropy, bias magnitude, and serial correlation. When ECI falls below a set limit, the controller triggers recovery actions. Actions include source switching, source mixing, and automatic conditioning activation. The design targets low-cost microcontrollers and supports multiple analog entropy sources. The paper defines failure modes, detection rules, and recovery policies. It also maps rules to health test goals in NIST SP 800-90B. The paper describes a stress model for EMI and fault injection. It also discusses bitrate loss and detection latency.**

Keywords—TRNG, entropy source, health tests, failure detection, self-healing, embedded security, NIST SP 800-90B

I. Introduction (*Heading 1*)

True random number generators support key generation and secure protocols. Embedded systems often rely on hardware entropy sources. Common sources include Zener diode noise, ring oscillator jitter, and thermal noise. These sources feed an analog front-end and an ADC. The system then extracts raw bits for cryptographic use.

Real deployments face harsh conditions. Temperature changes shift noise amplitude. Supply ripple distorts analog signals. Electromagnetic interference injects structured patterns. Hardware aging alters device behavior over time. These effects reduce min-entropy and increase bias. Weak randomness threatens key strength and protocol security.

Many TRNG designs focus on statistical validation in laboratory settings. Designers report min-entropy values between 0.85 and 0.99 per bit under controlled conditions. Field behavior often receives less attention. Runtime degradation may pass unnoticed for long periods. Startup health tests detect catastrophic faults only. Gradual entropy loss can escape simple checks.

NIST SP 800-90B defines health tests for entropy sources. It specifies repetition count and adaptive proportion tests. These tests target severe failures. They do not provide a continuous quality score. They do not link detection to automatic output control.

This paper addresses that gap. It proposes a self-healing architecture for hardware TRNG systems. The design introduces a runtime entropy health metric named Entropy Confidence Index, ECI. ECI combines min-entropy, bias magnitude, and serial correlation into a single score from 0 to 1. The monitoring layer evaluates ECI over sliding windows of 4096 bits. The recovery engine blocks output when ECI falls below a defined threshold.

The proposed model integrates detection and control in one framework. It defines a clear failure taxonomy. It assigns numeric thresholds to each failure mode. It bounds weak entropy exposure time to less than 24 ms at 500 kbit/s. The architecture targets low-cost microcontrollers and embedded devices.

The main contributions of this work are threefold. It defines a unified runtime entropy score. It introduces a control policy that links entropy degradation to automatic recovery. It provides a structured security analysis under bias drift, entropy collapse, and fault injection.

The following sections describe the system model, the ECI metric, the recovery engine, and the security evaluation.

II. Related Work

Research on TRNG health tests focuses on statistical validation.
Most designs apply NIST SP 800-22 after data collection.
Some systems implement startup tests only. Few systems monitor entropy continuously at runtime.

NIST SP 800-90B defines health tests for entropy sources.
It specifies repetition count and adaptive proportion tests. These tests detect catastrophic failures. They do not measure gradual entropy degradation well. Several studies analyze post-processing methods.
Hash-based conditioning improves statistical quality. XOR mixing reduces bias in multi-source systems. These methods act after entropy is generated. They do not prevent weak raw output during failure.

Embedded TRNG implementations often use: Zener diode noise, ring oscillators, thermal noise, or MEMS sensors.
Design papers report min-entropy values between 0.85 and 0.99 per bit. Most evaluations occur in controlled lab settings. Field behavior under stress receives less coverage.

This paper differs in one key aspect. It proposes a runtime entropy health score. It links detection directly to automatic recovery logic. The design blocks weak output before key generation.

III. System Model and Failure Taxonomy

A. *System Model*

The target system is a hardware TRNG inside an embedded device. The device uses one or more analog entropy sources. Examples include Zener diode noise, ring oscillator jitter, and MEMS sensors.

Each source feeds an analog front-end. The front-end applies amplification and filtering. An ADC samples the conditioned signal at a fixed rate. A bit extractor converts samples into raw bits. Raw bits pass to a monitoring layer before release.

The monitoring layer runs on the same microcontroller. It processes sliding windows of 4096 bits. It computes min-entropy, bias, and serial correlation. It then computes the Entropy Confidence Index, ECI.

An output gate controls final bit release. If ECI stays above threshold, output flows normally. If ECI drops below threshold, the gate blocks output. The recovery engine then activates corrective actions.

The model assumes a 32-bit microcontroller at 100 MHz.
It assumes ADC resolution between 10 and 12 bits. It assumes sampling rates from 100 kS/s to 1 MS/s.

B. *Failure Taxonomy*

We classify failures into five categories.

Bias Drift - Temperature rise changes noise amplitude. Power ripple skews the distribution of samples. The ratio of ones to zeros shifts from 0.5. Bias magnitude exceeds 0.02 in repeated windows.

Entropy Collapse - The entropy source stalls or saturates. Noise amplitude drops near zero. Min-entropy per bit falls below 0.8.Output becomes predictable.

ADC Clipping- Front-end gain is too high. ADC hits rail values frequently. More than 1 percent of samples equal 0 or full scale. Distribution becomes distorted.

Stuck-at-Fault - Hardware failure forces constant logic level. Longest run length exceeds 64 in a 4096-bit window. Min-entropy approaches zero.

Active Stress or Injection - An attacker injects EMI near the analog path. Periodic patterns appear in output bits. Serial correlation exceeds 0.01 consistently.

Each failure mode affects entropy differently. Some cause gradual degradation. Others cause sudden collapse. The monitoring layer must detect both types.

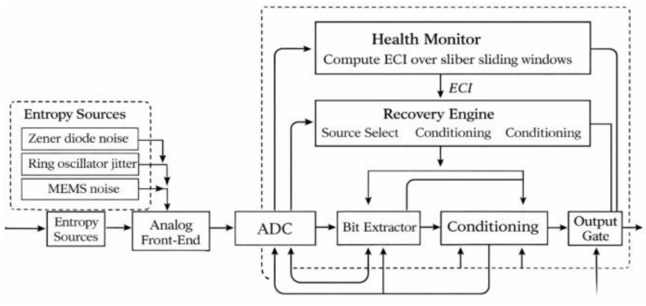

FIGURE 1. SELF-HEALING TRNG SYSTEM ARCHITECTURE

IV. ECI Metric and Detection Algorithms

The monitoring layer evaluates entropy in real time. It processes fixed sliding windows of raw bits. Each window contains 4096 bits. The system updates metrics every window shift.

A. *Min-Entropy Estimation*

Let $p\ max$ be the maximum probability of a bit value. For binary data, compute $p1$ as the ratio of ones. Then compute:

$$H_{min} = -\log_2(p_{max})$$

If $p_1 > 0.5$, then $p_{max} = p_1$.
If $p_1 \leq 0.5$, then $p_{max} = 1 - p_1$.

Healthy raw TRNG output should maintain $Hmin >$ 0.9.

B. *Bias Estimation*

Bias magnitude measures deviation from equal probability. Compute: $b = | 2p1 - 1 |$
Ideal output gives $b = 0b$.
The monitor raises a warning if b> $0.02b$

C. *Serial Correlation*

$$r = \frac{\sum(x_i - \mu)(x_{i+1} - \mu)}{\sum(x_i - \mu)^2}$$

Here $xi \in \{0,1\}$. A healthy stream keeps | r |< 0.01.

D. *Entropy Confidence Index*

$$ECI = w_1 H_{min} + w_2(1 - b) + w_3(1 - |r|)$$

Weights satisfy $w1 + w2 + w3 = 1$

Use $w1 = 0.5, w1 = 0.5, w2 = 0.3, w3 = 0.2$

ECI ranges from 0 to 1. A value near 1 indicates strong entropy health. Set trip threshold $T = 0.85$ .Trigger recovery if $ECI < T$ for three consecutive windows.

E. *Detection Logic*

The monitor runs every 4096 bits. It updates $H\ m\ i\ n\ b$ and r. It computes ECI. It increments a failure counter if ECI falls below threshold. It resets the counter if ECI recovers. It activates the recovery engine after three failures. This structure detects gradual degradation. It also detects sudden entropy collapse.

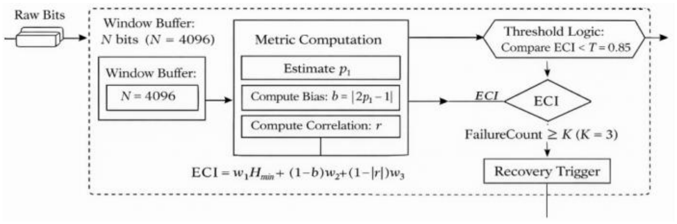

FIGURE 2. ECI COMPUTATION AND THRESHOLD LOGIC PIPELINE

V. Recovery Engine and Control Policy

The recovery engine activates when ECI remains below threshold. The system uses a failure counter over three windows. The output gate blocks bit release during recovery.

VI. CONTROL STATES

The controller operates in five states.

A. *Normal*

ECI remains above 0.85. Output flows without restriction.

B. *Warning*

ECI drops below threshold for one window.

The system logs the event. Output still flows.

C. *Blocked*

ECI remains low for three consecutive windows.

The output gate closes immediately. No bits reach cryptographic modules.

D. *Recovering*

The engine applies corrective actions. t resets the analog front-end. It reconfigures ADC gain and offset. It clears internal buffers.

E. *Fallback*

The engine switches to a secondary entropy source.

If multiple sources exist, it mixes them using XOR.

If only one source exists, it activates conditioning.

VII. RECOVERY ACTIONS

A. *Soft Reset*

The controller resets the entropy acquisition chain.
It reinitializes ADC registers.

B. *Source Switching*

The system selects a second entropy input.
It verifies stability before reopening output.

C. *Source Mixing*

Two entropy streams combine with XOR.
This reduces impact of single source degradation.

D. *Conditioning Activation*

The system hashes raw blocks with SHA-256.
Block size equals 512 raw bits. Output uses 256-bit digest.

E. *Output Release Rule*

The gate reopens only after ECI exceeds 0.9.
The system requires three clean windows before release.

VIII. DETECTION LATENCY

Window size equals 4096 bits.
At 500 kbit/s raw rate, detection occurs in 8 ms.
Three-window confirmation takes 24 ms.
This bounds exposure time to weak entropy.

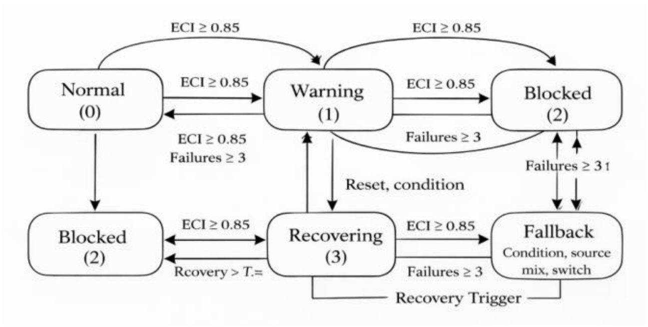

FIGURE 3. RECOVERY STATE MACHINE FOR SELF-HEALING TRNG

IX. Security Analysis and Threat Evaluation

The proposed architecture limits weak entropy exposure time. The system blocks output after three failing windows. At 500 kbit/s, detection completes within 24 ms. This reduces risk during entropy degradation.

X. Bias Drift Scenario

Temperature rise shifts noise amplitude. Bias magnitude increases gradually above 0.02. ECI decreases over consecutive windows. The monitor detects deviation before collapse. The gate blocks output before key generation.

XI. Entropy Collapse Scenario

The entropy source stalls. Min-entropy drops below 0.8. ECI falls sharply in one window. The system enters Blocked state immediately. Recovery resets the analog chain.

XII. EMI Injection Scenario

An attacker injects periodic interference. Serial correlation increases above 0.01. The correlation detector triggers Warning state. Repeated windows push system to Blocked state. Fallback activates source mixing.

XIII. Stuck-at Fault Scenario

Hardware failure forces constant logic level. Run length exceeds 64 bits. Min-entropy approaches zero. ECI collapses below threshold. Output remains blocked until recovery completes.

XIV. Exposure Time Bound

Maximum weak output duration equals detection latency.
With 4096-bit windows at 500 kbit/s, one window equals 8 ms. Three-window confirmation equals 24 ms.
This sets a strict upper bound on exposure.

XV. Comparison with Traditional Designs

Many TRNGs rely on startup tests only. Some rely on offline statistical validation. Few designs block output automatically at runtime. The proposed model integrates detection and control.

XVI. Discussion and Implementation Considerations

The proposed monitor runs on a low-cost microcontroller.
All calculations use integer or fixed-point arithmetic. A 32-bit MCU at 100 MHz handles the load easily.

XVII. Computational Cost

Each 4096-bit window requires:

Counting ones for bias and min-entropy. Computing lag-1 correlation. Updating a weighted sum for ECI.

Counting ones requires 4096 operations. Correlation requires 4095 comparisons. Total operations remain below 10,000 per window.

At 500 kbit/s, one window arrives every 8 ms. A 100 MHz MCU executes 800,000 cycles in 8 ms. The monitoring task consumes a small fraction of CPU time.

XVIII. Memory Requirements

The window buffer stores 4096 bits. This equals 512 bytes. Additional variables require less than 100 bytes. Total RAM usage remains under 1 KB.

XIX. Bitrate Impact

Normal mode introduces no bitrate loss. Blocked mode stops output temporarily. Conditioning reduces output rate by half when SHA-256 processes 512-bit blocks into 256-bit output.

The system trades short pauses for improved security control.

XX. Parameter Selection

Window size affects detection latency. Smaller windows detect faster but increase false alarms.
Larger windows reduce noise but delay detection.

Threshold T equals 0.85 in this design. Raising T increases sensitivity. Lowering T reduces false triggers.

Three consecutive failing windows balance stability and speed.

XXI. Practical Integration

The architecture fits FPGA or MCU designs. It requires one control state machine. It requires simple arithmetic blocks. No external hardware modules are needed.

The model supports single-source systems. It scales to multi-source designs with XOR mixing.

XXII. Conclusion

This paper presented a self-healing architecture for hardware TRNG systems. The design introduces a runtime entropy health score named ECI.
ECI combines min-entropy, bias, and serial correlation.
The monitoring layer detects both gradual degradation and sudden failure.

The recovery engine blocks weak output automatically.
It applies reset, source switching, mixing, or conditioning.
Detection latency remains under 24 ms at 500 kbit/s. Memory and CPU costs remain low.

The architecture strengthens embedded cryptographic systems.
It reduces exposure to weak randomness during field operation.
The model aligns with health test goals in NIST SP 800-90B.

Patents: No patents result from the work reported in this manuscript. Supplementary Materials: Supplementary materials are available with this manuscript. Figure S1: Extended histograms for all entropy sources. Table S1: Full statistical metrics for each dataset. Author Contributions: Conceptualization, L.H.; methodology, L.H.; software, L.H.; validation, L.H.; formal analysis, L.H.; investigation, L.H.; resources, L.H.; data curation, L.H.; writing—original draft preparation, L.H.; writing—review and editing, L.H.; visualization, L.H.; supervision, L.H.; project administration, L.H. Funding: This research received no external funding. The APC was not funded by any agency. Data Availability Statement: The datasets generated and analyzed during this study are available from the corresponding author upon reasonable request. Acknowledgments: During the preparation of this manuscript, the author used ChatGPT (GPT-5.2, OpenAI) for language editing and formatting assistance. The author reviewed and edited the output and takes full responsibility for the content of this publication.
Funding: This research received no external funding. The APC was not funded by any agency. Data Availability Statement: The datasets generated and analyzed during this study are available from the corresponding author upon reasonable request.

Acknowledgments: During the preparation of this manuscript, the author used ChatGPT (GPT-5.2, OpenAI) for language editing and formatting assistance. The author reviewed and edited the output and takes full responsibility for the content of this publication.

Conflicts of Interest: The author declares no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results

References

[1] National Institute of Standards and Technology. NIST SP 800-90B: Recommendation for the Entropy Sources Used for Random Bit Generation. NIST, 2018.

[2] National Institute of Standards and Technology. NIST SP 800-22 Rev.1a: A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. NIST, 2010.

[3] B. Jun and P. Kocher. The Intel Random Number Generator. Cryptography Research Inc., 1999.

[4] M. Bucci and R. Luzzi. Design of Testable Random Bit Generators. IEEE Transactions on Computers, vol. 52, no. 4, pp. 403–415, 2003.

[5] C. Petrie and J. Connelly. A Noise-Based IC Random Number Generator for Applications in Cryptography. IEEE Transactions on Circuits and Systems I, vol. 47, no. 5, pp. 615–621, 2000.

[6] B. Sunar, W. Martin, and D. Stinson. A Provably Secure True Random Number Generator with Built-In Tolerance to Active Attacks. IEEE Transactions on Computers, vol. 56, no. 1, pp. 109–119, 2007.

[7] M. Holcomb, W. Burleson, and K. Fu. Power-Up SRAM State as an Identifying Fingerprint and Source of True Random Numbers. IEEE Transactions on Computers, vol. 58, no. 9, pp. 1198–1210, 2009.

[8] A. Rukhin et al. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. NIST Technical Report, 2010.