Ctrl IQ, Inc.

Rocky Linux Kernel CPU Time Jitter RNG Entropy Source

v2.2.0

SP 800-90B Non-Proprietary Public Use Document Kernel CPU Time Jitter

Document Version: 0.5

Prepared for:	**Prepared by:**
Ctrl IQ, Inc.	**Corsec Security, Inc.**
1050 N. Hills Blvd	12600 Fair Lakes Circle
Unit 61180	Suite 210
Reno, NV 89506	Fairfax, VA 22033
United States of America	United States of America
Phone: +1 510 345 4730	Phone: +1 703 267 6050
www.ctrliq.com	www.corsec.com

Revision History

Version	Code Version	Modification Date	Modified By	Description of Changes
0.1	2.2.0	2023-06-06	Steele Myrick	Initial draft.
0.2	2.2.0	2023-10-11	Daniel Nguyen	CPU full names, CIQ entropy estimates, Rocky 9 CPU
0.3	2.2.0	2023-10-11	Daniel Nguyen	Added Rocky versions to OE
0.4	2.2.0	2023-10-24	Daniel Nguyen	Combined Entries of Table 1 & 2
0.5	2.2.0	2023-11-02	Daniel Nguyen	Removed full entropy claims

Table of Contents

List of Tables

List of Figures

1. Description

The Kernel CPU Time Jitter RNG Entropy Source is a non-physical (NP) entropy source that meets the standards of NIST SP 800-90B. It makes no IID claim and thus meets all the requirements for non-IID compliance. The report is for Kernel CPU Time Jitter version 2.2.0. The table summarizes the Ctrl IQ, Inc. Rocky Linux Kernel CPU Time Jitter RNG Entropy Source X11SSE-F and NX-8170-G8 platforms and their respective CPUs for which testing was performed.

Table 1 – CIQ and Nutanix CPUs

Model	CPU	Linux Version
SuperMicro SuperServer (X11SSE-F)	Intel(R) Xeon(R) CPU E3-1270 v6, Sandy Bridge	Rocky Linux 8 Rocky Linux 9
Nutanix NX-8170-G8	Intel(R) Xeon(R) CPU Gold 6326, Ice Lake	Rocky Linux 8

2. Security Boundary

The boundary of the Kernel CPU Time Jitter RNG Entropy Source implementation is shown in Figure 1. The basic concept that is implemented with the Kernel CPU Time Jitter RNG Entropy Source can be summarized as follows: The unpredictable phenomenon of variances in the execution time of a given set of instructions is collected and accumulated. This set of instructions is separated into the LFSR component and the memory access component. The measurement of the execution time jitter is performed over the post-processing logic of the LFSR and supportive functions, i.e., the CPU Time Jitter NPTRNG measures the execution time of the SHA3- 256 and the execution time of memory access. After which the execution time is injected into the SHA3-maintained entropy pool. The JENT output is used seed an SP 800-90A compliant DRBG (which is outside of the boundary of the entropy source). The security boundary of the JENT implementation contains the source code files provided as part of the software delivery. The source code contains API calls which are used by modules requesting entropy from JENT.

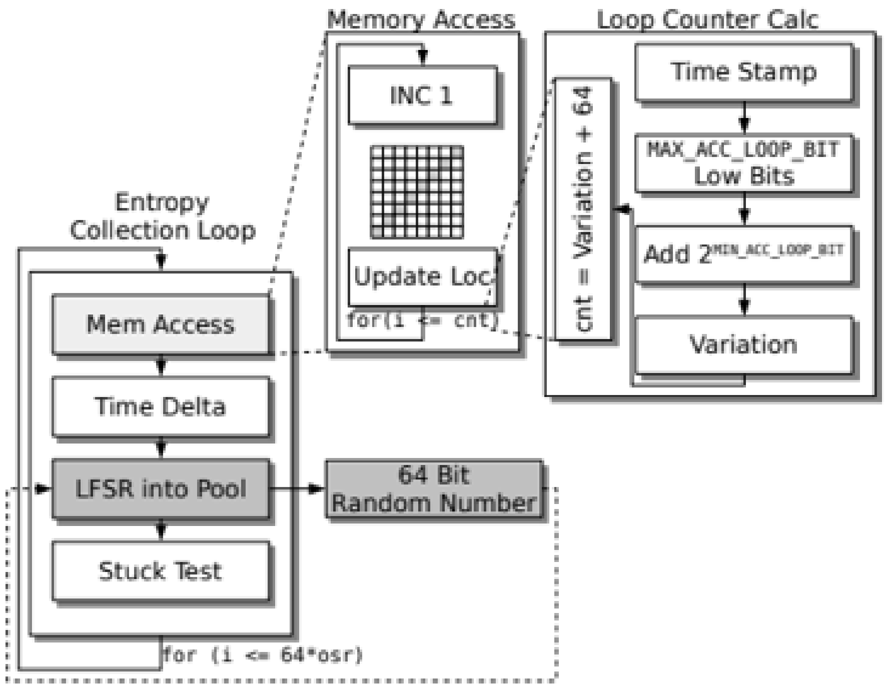

Figure 1 – Entropy collection operation

3. Operating Conditions

Table 2 summarizes the operating conditions for each of the tested CPUs.

Table 2 – Operating Conditions

CPU	CPU Family	CPU Clock Speed	Platform	Platform Temperature	Platform Voltage (AC)	Linux Version
Intel(R) Xeon(R) CPU E3-1270 v6, Sandy Bridge	Sandy Bridge	3.8 GHz	SuperMicro SuperServer (X11SSE-F)	10℃ – 35℃ 50℉ – 95℉	100 VAC – 240 VAC	Rocky Linux 8 Rocky Linux 9
Intel(R) Xeon(R) CPU Gold 6326, Ice Lake	Ice Lake	2.90 GHz	Nutanix NX-8170-G8	10℃ – 35℃ 50℉ – 95℉	180 VAC – 240VAC 100VAC – 210VAC	Rocky Linux 8

4. Configuration Settings

Table 3 summarizes the configuration settings used by the CIQ implementation of JENT. These settings must be preserved to comply with the JENT ESV certificate.

Table 3 – Configuration Settings

Setting	Value
JENT_MEMORY_BLOCKS	64
JENT_MEMORY_BLOCKSIZE	32
JENT_MEMORY_ACCESSLOOPS	128

5. Physical Security Mechanisms

The JENT implementation is a software module. As such, it does not include physical security mechanisms. The JENT module may be used on a variety of devices with differing security levels. Physical security requirements are, therefore, dependent on the modules that use JENT.

Conceptual Interfaces provided by JENT:

- GetEntropy: The call to jent_read_entropy serves as a GetEntropy interface according to the SP 800-90B definition. The jent_read_entropy function generates a random number in a way that is compliant with all SP 800-90B requirements. The caller specifies the size of the random value that it wants the function to return. The function sets a return code to indicate whether or not the request was fulfilled successfully. The return code also reports failures of the health tests.

- GetNoise: The JENT library does not have a GetNoise interface, but JENT comes with scripts that collect the raw noise data (jitterentropy-rng). The scripts may be used to compile specific executables that collect 1,000,000 consecutive samples or 1,000 sets of 1,000 samples of raw data.

- HealthTest: The JENT library does not have a HealthTest interface, but health tests can be run by allocating a new JENT handle. The health tests are compliant with SP 800-90B.

6. Min-Entropy Rate

The JENT implementation generates an output that has around 58 bits of entropy per 64 bit. A request for 256 bits of entropy results in 232 bits of entropy per output sample.

7. Health Tests

Per the SP 800-90B requirements, health tests run when JENT starts, and are then run continuously while it is operating. All tests check for persistent failures base on their respective "cutoff" values, which represent expected error thresholds.

JENT implements four types of health tests:

- Stuck Test
- Repetition Count Test (RCT)
- Adaptive Proportion Test (APT)
- Lag Predictor Test

The stuck test calculates the first, second, and third discrete derivative of the time to be processed by the hash. The received time delta is considered to be non-stuck only if all three values are non-zero.

The Lag Predictor Test is a vendor-defined conditional test that is designed to detect a known failure mode where the result becomes mostly deterministic. It is based on the Lag Prediction Test described in SP 800-90B, section 6.3.8.

The RCT and APT tests are implemented as described in SP 800-90B.

When any health test fails, the API call to generate random numbers (jent_read_entropy(3)) informs the caller about the failure with error codes and the Jitter RNG block causing the failure is not returned to the caller.

All health test failures are considered permanent failures. If one is triggered, the current instance of the Jitter RNG will always remain in error state. The documentation of the API call for jent_read_entropy(3) explains that the caller can only clear this error state by deallocating the Jitter RNG instance followed by an allocation of a new Jitter RNG instance to reset the noise source.

8. Maintenance

There are no maintenance requirements for the JENT library.

9. Required Testing

Raw data was collected by running the invoke_testing.sh script included in the JENT package. This script first gathers 1,000,000 consecutive samples of raw data. It then gathers 1,000,000 samples of raw data in groups of 1,000 samples and restarting JENT after each group of data for the restart tests.

For best compliance to SP 800-90B, the number of hashing operations is always one (and is the default setting for JENT 2.2.0).

Raw Data Analysis

Each raw data sample consists of one timestamp delta, which is 64 bits long. It is assumed that only the least significant 4 bits of each timestamp delta contains any true entropy. The JENT design states that the Jitter RNG can deliver around 58 bits of entropy per 64 bit if and only if the min-entropy is at least $\frac{1}{osr}$ bit of entropy per timestamp. The Jitter RNG implementation uses an oversampling rate (OSR) of 3.

Table 4 below shows that the min-entropy for the raw data samples ranges from 2.929234 to 4.416493 bits of entropy per 8-bit sample.

Table 4 – Final Raw Min-Entropy Estimates

Entropy Result	Intel(R) Xeon(R) CPU E3-1270 v6, Sandy Bridge Rocky 8	Intel(R) Xeon(R) CPU Gold 6326, Ice Lake	Intel(R) Xeon(R) CPU E3-1270 v6, Sandy Bridge Rocky 9
H_original	4.596798	5.306244	4.416493
H_bitstring	0.366154	0.510450	0.405227
min(H_original, 8 X H_bitstring)	2.929234	4.083602	3.241812

Restart Data Analysis

The CPU Jitter RNG does not have a stochastic model, but an $H_{submitter}$ estimate of 1 was determined using analysis as described in SP 800-90B Section 3.2.2, Requirement 3. The Jitter RNG implementation uses an oversampling rate of 3. For that reason, $H_{submitter}$ is set to 0.333 ($\frac{1}{3}$) for the purpose of running the NIST restart tests.

For the restart tests, the raw entropy data is collected for 1,000 Jitter RNG instances allocated sequentially. That means, for one collection of raw entropy, one Jitter RNG instance is allocated. After the conclusion of the data gathering it is deallocated and a new Jitter RNG instance is allocated for the next restart test round.

Table 5 – Final Restart Min-Entropy Estimates

Entropy Result	Intel(R) Xeon(R) CPU E3-1270 v6, Sandy Bridge Rocky 8	Intel(R) Xeon(R) CPU Gold 6326, Ice Lake	Intel(R) Xeon(R) CPU E3-1270 v6, Sandy Bridge Rocky 9
H_r	4.640932	5.477366	4.562895
H_c	4.887493	5.306244	4.618356

Entropy Result	Intel(R) Xeon(R) CPU E3-1270 v6, Sandy Bridge Rocky 8	Intel(R) Xeon(R) CPU Gold 6326, Ice Lake	Intel(R) Xeon(R) CPU E3-1270 v6, Sandy Bridge Rocky 9
H_I	1.000000	1.000000	1.000000
min(H_r, H_c, H_I):	1.000000	1.000000	1.000000