Ribbon Communications, Inc.

Ribbon CPU Jitter Entropy Source

v3.3.0

SP 800-90B Non-Proprietary Public Use Document CPU Jitter (JENT) v3.3.0

Document Version: 0.6

Prepared for:

Ribbon Communications, Inc.
4 Technology Park Drive
Westford, MA 01886
United States of America

Phone: +1 855 467 6687
www.ribboncommunications.com

Prepared by:

Corsec Security, Inc.
12600 Fair Lakes Circle, Suite 210
Fairfax, VA 22033
United States of America

Phone: +1 703 267 6050
www.corsec.com

Revision History

Version	Code Version	Modification Date	Modified By	Description of Changes
0.1	3.3.0	2023-06-30	Steele Myrick	Initial draft.
0.2	3.3.0	2023-08-16	Steele Myrick	Added additional information about both CPU's, Operating Conditions, Configuration Settings, and cache calculations to show how that the OE processors will force a memory access operation in main memory.
0.3	3.3.0	2023-08-16	Steele Myrick	Added full software name
0.4	3.3.0	2023-10-02	Steele Myrick	Modified section 7 for clarity, updated the operating conditions table and updated processor information.
0.5	3.3.0	2023-10-13	Steele Myrick	Changed entropy in 64-bit time stamp to be the 8 lsb instead of 4 lsb
0.6	3.3.0	2023-11-17	Steele Myrick	Provided additional configuration settings in section 4

Table of Contents

List of Tables

List of Figures

1. Description

CPU Jitter Entropy (JENT) is a non-physical true random number generator source of entropy. It makes no IID claim and thus meets all the requirements for non-IID compliance. The report is for JENT version 3.3.0. The table summarizes the Ribbon Communications, Inc. Ribbon CPU Jitter Entropy Source for the software module SBC SWe Session Border Controller (SBC SWe 10.1.3) (software version 10.1.3) and the hardware module SBC 5400 Session Border Controller (SBC 5400) and their respective CPUs for which testing was performed.

Table 1 – CPUs

Model	**CPU**	CPU Clock Speed	CPU Cache L1d size	**CPU Cache L1i size**	CPU Cache L2 size	CPU Cache L3 size
SBC SWe 10.1.3	Intel(R) Xeon(R) CPU E5-2630 v3, Haswell	2.40 GHz	32 KiB	32 KiB	256 KiB	2.5 MiB
SBC 5400	Intel(R) Xeon(R) CPU E5-2648L v2, Ivy Bridge	1.90 GHz	32 KiB	32 KiB	256 KiB	2.5 MiB

2. Security Boundary

The boundary of the JENT implementation is shown in Figure 1. The basic concept that is implemented with the CPU Time Jitter NPTRNG 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 SHA3-256 component and the memory access component. The measurement of the execution time jitter is performed over the post-processing logic of the SHA3-256 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.

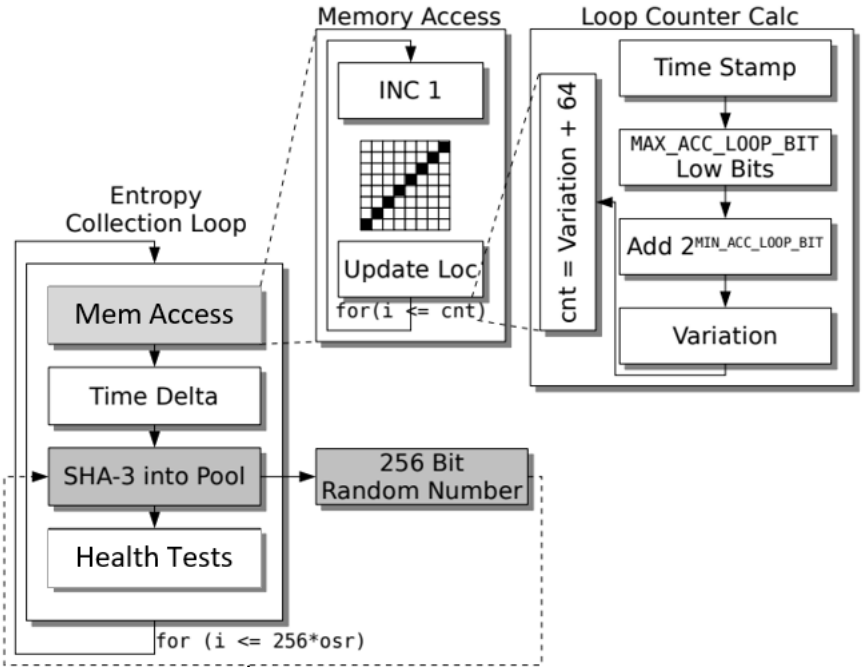

Figure 1 – Entropy collection operation

3. Operating Conditions

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

Table 2 – Operating Conditions

CPU	CPU Clock Speed	CPU Cache L1d size	CPU Cache L1i size	CPU Cache L2 size	CPU Cache L3 size
Intel(R) Xeon(R) CPU E5-2630 v3, Haswell	2.40 GHz	32 KiB	32 KiB	256 KiB	2.5 MiB
Intel(R) Xeon(R) CPU E5-2648L v2, Ivy Bridge	1.90 GHz	32 KiB	32 KiB	256 KiB	2.5 MiB

CPU Family	Operating System	Platform	Platform Temperature	Platform Voltage (AC)
Haswell	Ribbon ConnexIP OS 10 on VMware ESXi 6.5	SBC SWe Session Border Controller(software)	NA	NA
Ivy Bridge	Ribbon ConnexIP OS 10	SBC 5400 Session Border Controller	Nominal: 5℃ - 40℃ Short Term Operation: -5℃ - 55℃	90V – 264VAC

4. Configuration Settings

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

Table 3 – Configuration Settings

Setting	Value	Description
JENT_RANDOM_MEMACCESS	Enabled	Determines the method of memory access
JENT_CONF_ENABLE_INTERNAL_TIMER	Disabled	Disables the use of internal timers
JENT_CONF_DISABLE_LOOP_SHUFFLE	Enabled	Disables the random number of hash loops performed for each call (only one hash operation is performed with this setting)
JENT_MIN_OSR	3	Oversampling rate
JENT_MEMORY_BITS	17	Default setting
JENT_MEMORY_SIZE	128KiB	Default setting
JENT_MEMORY_ACCESSLOOPS	128	Default setting

For FIPS validated modules running as virtual machines (VMs), the vendor recommends that the VM snapshot configuration does not select the memory snapshot option e.g., do not select "Snapshot the virtual machine's memory". This will take a cold snapshot with no memory state save to ensure that the VM is rebooted after resuming on the same or another host computer. If memory was snapshotted, the VM must be rebooted after a restore operation.

The SWe, with the Haswell CPU, was configured with 1 socket, 6 cores, and 24GB of RAM for our statistical testing.

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 is considered to have full entropy. A request for 256 bits of entropy results in 256 bits of entropy per output sample, or full entropy.

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 allowed by SP 800-90B and associated IGs.

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 3.3.0).

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 8 bits of each timestamp delta contains any true entropy. The JENT design states that the Jitter RNG can deliver full entropy 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.

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.

Cache Calculations

Based on the memory block size, number of access loops and cache size, here is the calculation to show that the memory access operation will surpass the L1 cache, L2 cache, and L3 cache then enter main memory to provide the desired timing jitter. The 128 memory access loops will ALWAYS exceed the available L1 cache, L2 cache, and L3 cache using the following constants:

- JENT_MEMORY_BITS as 17
- JENT_MEMORY_SIZE as 128KiB
- JENT_MEMORY_ACCESSLOOPS as 128

Intel(R) Xeon(R) CPU E5-2630 v3, Haswell:

- L1 cache is 64 KiB
- L2 cache is 256 KiB
- L3 cache is 2.5 MiB

Assuming the memory access loop stays on the same core, then 128KiB x 128 = 16MiB will always exceed available L1 cache, L2 cache, and L3 cache (sum of cache sizes 2.82 MiB) on the OE processor and force a memory access operation in main memory.

Intel(R) Xeon(R) CPU E5-2648L v2, Ivy Bridge:

- L1 cache is 64 KiB
- L2 cache is 256 KiB
- L3 cache is 2.5 MiB

Assuming the memory access loop stays on the same core, then 128KiB x 128 = 16MiB will always exceed available L1 cache, L2 cache, and L3 cache (sum of cache sizes 2.82 MiB) on the OE processor and force a memory access operation in main memory.