

SP 800-90B Non-Proprietary Public Use Document

Arista CloudVision Appliance Entropy Source for Java

Version 3.6.1

Arista Network, Inc.
5453 Great America Parkway
Santa Clara, CA 95054
USA

Document Version 1.3

September 2026

Document prepared by:



www.lightshipsec.com

Table of Contents

1 Description 3
2 Security Boundary 3
3 Operating Conditions 4
4 Configuration Settings 4
5 Physical Security Mechanisms 5
6 Conceptual Interfaces 5
7 Min-Entropy Rate 6
8 Health Tests 6
9 Maintenance 6
10 User Verification 6
10.1 Collecting samples 6
10.2 Raw Data Analysis 6
10.3 Entropy Source Version Verification 7
11 Vendor Permissions and Relationship 7

1 Description

This is the SP 800-90B non-proprietary Public Use Document for *Arista CloudVision Appliance Entropy Source for Java* (hereafter referred to as "the Entropy Source"). The Entropy Source is based on the jitterentropy-library v3.6.1.

The Entropy Source makes no IID claim and thus meets all the requirements for non-IID compliance. The tested platforms are described in Table 1.

The Entropy Source is compliant with NIST Special Publication 800-90B Recommendation for the Entropy Sources Used for Random Bit Generation (January 2018), IG D.J (Last Modified Date: November 5, 2021) and IG D.K (Last Modified Date: April 16, 2026).

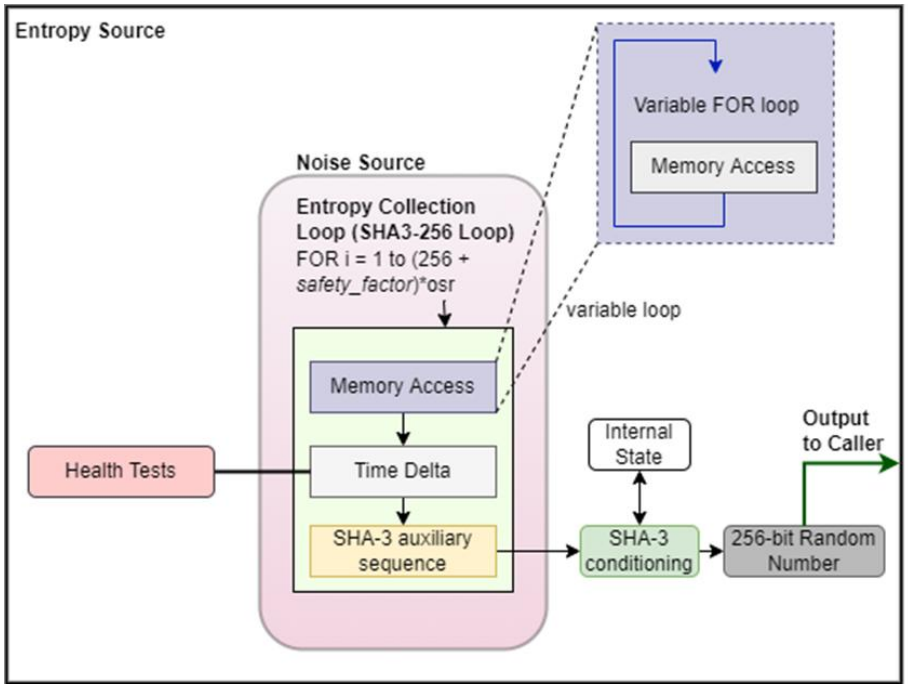
2 Security Boundary

The Entropy Source operates within a defined security boundary that includes both software and hardware components.
The software component is implemented as a software-only mechanism distributed within a Java archive (JAR) package. The software boundary encompasses:

- A collection of Java classes executed by the Java Virtual Machine (JVM) along with supporting classes provided by the Java Runtime Environment (JRE).
- A Linux-compatible shared object library implementing the CPU Time Jitter based non-physical random number generator (jitterentropy-library v3.6.1).
- An additional Linux-compatible shared object library that facilitates communication between the Java classes and the jitterentropy-library through the Java Native Interface (JNI).

The hardware component is the underlying computational environment required to execute the library, including the processor and system timer.
JEnt is based on the jitterentropy-library v3.6.1. Its design maps to the conceptual components described in NIST SP 800-90B and includes:

- A primary noise source (based on memory access and conditioning timing),
- Health tests to ensure the proper functioning of the entropy source
- A conditioning function implemented using SHA3-256.



3 Operating Conditions

The Entropy Source was tested in the following operational environments:

Table 1. Tested Operational Environments

Operating System and Software CPU Jitter RNG version 3.6.1 Oversampling rate: osr = 13 Health Testing reporting and use of safety factor: fips_enabled		
Label	**CPU**	**Operating System**
OE1	Intel® Xeon® Silver 4316 (Sunny Cove) CPU Clock Speed: 2.30 GHz L1 Cache: 48 KB L2 Cache: 1.25 MB L3 Cache: 30 MB	CloudVision Appliance v8 on VMware vSphere ESXi 9.0

4 Configuration Settings

The tables below list the Entropy Source's configuration parameters. These settings must be preserved to comply with the CPU Jitter ESV certificate.

Table 2. CPU Jitter Configuration Parameters

Parameter	Default Value	Location
JENT_CONF_DISABLE_LOOP_SHUFFLE	Defined	jitterentropy.h
JENT_HEALTH_LAG_PREDICTOR	Defined	jitterentropy.h
JENT_RANDOM_MEMACCESS	Defined	jitterentropy.h
ENTROPY_SAFETY_FACTOR	64	jitterentropy.h
JENT_MIN_OSR	13	jitterentropy.h
JENT_MEMORY_SIZE	2^{17}	jitterentropy.h
JENT_MEMORY_BITS	17	jitterentropy.h
JENT_MEMORY_BLOCKS	512	jitterentropy.h
JENT_MEMORY_BLOCKSIZE	128	jitterentropy.h
JENT_MEMORY_ACCESSLOOPS	128	jitterentropy.h
JENT_CONF_ENABLE_INTERNAL_TIMER	Not Defined	Makefile
JENT_APT_WINDOW_SIZE	512	jitterentropy.h
JENT_LAG_WINDOW_SIZE	2^{17}	jitterentropy.h
JENT_LAG_HISTORY_SIZE	8	jitterentropy.h
JENT_POWERUP_TESTLOOPCOUNT	1024	jitterentropy-base.c
CLEARCACHE	100	jitterentropy-base.c
MIN_HASH_LOOP	0	jitterentropy-noise.c
MIN_ACC_LOOP_BIT	0	jitterentropy-noise.c

Table 3. CPU Jitter Configuration Flags (for jent_entropy_init_ex and jent_entropy_collector_alloc)

Parameter	Argument Name	Meaning
JENT_DISABLE_MEMORY_ACCESS	flags	Disable the memory access timing.
JENT_FORCE_INTERNAL_TIMER	flags	Force use of pthreads-based internal timer, rather than the underlying hardware timer.
JENT_DISABLE_INTERNAL_TIMER	flags	Prevent the use of the pthreads-based internal timer.
JENT_FORCE_FIPS	flags	Force fips_enabled to be set (enabling health test reporting and use of the safety factor).
JENT_MAX_MEMSIZE_*	flags	Allows for run-time selection of the memory region size.
osr	osr	Sets the internal osr rate to max(osr, JENT_MIN_OSR).

5 Physical Security Mechanisms

N/A – The Entropy Source is non-physical.

6 Conceptual Interfaces

The entropy source provides the following interfaces:

Table 4. Interfaces

Interface	API Command	Description
GetEntropy	generateSeed()	Generates a random bit stream and returns it to the caller. This is an indirect call to the Entropy Source via Java SecureRandom API.

GetNoise	-	Not supported
HealthTest	generateSeed()	Initializes the Entropy Source and performs statistical tests to verify that the underlying system offers the properties needed for measuring and collecting entropy. In addition, calls to generateSeed() will also perform the necessary continuous tests as given in NIST SP 800-90A.

7 Min-Entropy Rate

The Entropy Source provides 256 bits of assessed min-entropy per 256-bit output sample, or full entropy.

8 Health Tests

The Entropy Source implements the following health tests:

- Stuck Test
- Repetition Count Test
- Adaptive Proportion Test
- Lag Predictor Test

These tests are run when the Entropy Source starts and continuously while it is operating. All tests check for persistent failures based on their respective "cutoff" values, which represent expected error thresholds. All these tests are performed on raw data before any conditioning is done.

All health test failures are fatal. Once triggered, the Entropy Source will transition into an error state and will no longer provide entropy. To clear the error state, the calling application must unload the failed Entropy Source and instantiate a new instance.

9 Maintenance

N/A – The Entropy Source does not require maintenance.

10 User Verification

To confirm that the Entropy Source has been configured properly, the developer shall run the 90B Entropy Assessment tools on the noise source outputs in the same manner that was done during the Entropy Source Validation exercise. The steps involved are described below.

10.1 Collecting samples

The developer shall run the jitterentropy-hashtime tool included in the Entropy Source package. This script gathers 1,000,000 continuous samples of raw noise data. It also gathers 1,000,000 samples of restart data which comprised of 1000 files each containing the first 1000 samples generated after the Entropy Source has been restarted.

10.2 Raw Data Analysis

The developer shall run the noise source samples through NIST's SP 800-90B Entropy Assessment tools. The Entropy Source is configured properly if the assessment yields a min entropy rate of at least 1/OSR bits per sample.

10.3 Entropy Source Version Verification

For administrative inspection and environment verification at the OS level, an administrator can verify the active jitterentropy-library version compiled into the system by executing the following command in bash mode:

- openssl list -providers

The expected output includes the active status, name, and version of the provider as follows:

- jitterentropy_fips
    - name: Arista CloudVision Appliance Entropy Source for Java
    - version: v3.6.1-1
    - status: active

The version format is v{X.Y.Z}-{A}, where X.Y.Z represents the underlying jitterentropy-library major, minor, and patch version obtained using the jent_version() API, and A represents the Arista entropy provider revision.

11 Vendor Permissions and Relationship

The reuse of this ESV requires written and signed permission from Arista Networks, Inc.