SP 800-90B Non-Proprietary Public Use Document

Infinera Groove CPU Time Jitter RNG Entropy Source

Infinera Corporation
August 16, 2024

Version 1.1

Revision History

Version	Change
0.1	Draft
0.2	Update the operation conditions and tests details
0.3	Update the configuration and tests details
0.4	Polishing and simplification
1.0	Update with latest test result
1.1	Update sections of Min-Entropy Rate & Health Tests

Table of Contents

1. Description 4
2. Security Boundary 4
3. Operating Conditions 4
4. Configuration Settings 4
5. Physical Security Mechanisms 4
6. Conceptual Interfaces 5
7. Min-Entropy Rate 5
8. Health Tests 5
9. Maintenance 5
10. Required Testing 5
11. References 6

1. Description

The entropy source, *Infinera Groove CPU Time Jitter RNG Entropy Source*, is a software-based entropy source running as part of an EGD (entropy gathering daemon) on Linux, in a network node controller.

The entropy source is cloned from SmuellerDD jitterentropy-library *[1]*, which comes with a full design/reuse document *[2]*.

The controller has a CPU of following model:
NXP p2040, em500c powerpc core, 1GHZ.

2. Security Boundary

The boundary of this entropy source is exactly the jitterentropy-library souce tree, which contains a group of C and Asm codes, and linked statically into the EGD executable.
The executable contains this entropy source, plus an EGD service part, which provides TRNG produced by the entropy source to user applications through EGD protocol.

3. Operating Conditions

The software entropy source operates in the same conditions as the network nodes, which are listed below:

Chassis Model	Operating Temp	Relative Humidity	Operating Voltage
AC G30	5~40° C (at sea level)	8-85%	100~127V 200~240V
DC G30	5~40° C (at sea level) -5~45° C (short term)	5-93%	-40.5~-72V

4. Configuration Settings

jitterentropy-library has some key configurations which are passed in during initialization, and most of which are not changed. The default guarantees standard compliant already.

Please refer to the jitterentropy-library man-page for the configurations.

There is one exception: JENT_FORCE_FIPS is applied in both the egd, and the tester. The reason is that we do not use kernel parameter “fips=1”.

The build process is kept the same as all in-house NOS applications. The only difference is a forced “-O0” compilation option in the jitterentropy-lib core, to meet the jitterentropy-library requirement.

5. Physical Security Mechanisms

N/A.

6. Conceptual Interfaces

- GetEntropy: jent_read_entropy_safe()
- GetNoise : during the normal software running, we do not get noise. To pass the SP800-90B validation, however, Jitterentrpy-library provides tools (see below ch. "required testing") to collect raw noises. The tools use "jent_measure_jitter()" as interface to collect noises.
- HealthTest : HealthTest is built in the initialization and GetEntropy logic. The jitterentropy-library internal logic uses "jent_health_failure()" to get the health status from the context data.

7. Min-Entropy Rate

$H_{submitter}$ = 1/3 bit per 64 bits of time delta. The original noise sample size has a length of 64 bits, and it is further reduced to 8 bits for assessment and health testing purposes. The 64-bit time deltas are concatenated to gather sufficient entropy for the SHA3-256 vetted conditioning function.
The entropy source provides an output of 256 bits. This output provides 256 bits of entropy.

8. Health Tests

The Jitter RNG implements the following continuous health tests:

- Stuck Test
- Repetition Count Test
- Adaptive Proportion Test

The Jitter RNG applies a startup health test of 1,024 noise source samples. Additional tests are applied. The collected noise source samples are not re-used for the generation of random numbers.
The noise source supports on-demand testing in the sense that the caller is allowed to deallocate and reallocate a new Jitter RNG handle. During the reallocation, the startup health tests are re-executed.
When a health test failure occurs, the Jitter RNG block causing the failure is not returned to the caller. In the failed case and before the Jitter RNG instance restart, the consumer of the entropy (DRBG in consumer applications) will be stalled waiting.

9. Maintenance

N/A.

10. Required Testing

Basically the raw data (time jitter) produced by the core of the implementation (running on the G30 NE) need to meet the minimum entropy requirements. The processing part is identical on any platform so we can reuse the original SP800-90B Analysis.

In practice, following steps were done to gather and test the raw data:

- Use the raw data collecting application from the jitterentropy-library:

    - ./jitterentropy-hashtime 1000000 1 <path> 0 fips
    - ./jitterentropy-hashtime 1000 1000 <path> 0 fips
- Run analysis tool provided by jitterentropy-library, on a x86-64 Linux PC, to analyze the results:
    - Continous test: min entropy per sample of lower boundary data is 2.287259, which exceeds the 1 entropy per sample requirement.
    - Restart test, lower boundary data: H_r=2.287259 and H_c=3.200384, both of which exceed the "1.0" H_I.

11. References

[1] GitHub - smuellerDD/jitterentropy-library: Jitterentropy Library, https://github.com/smuellerDD/jitterentropy-library

[2] CPU Time Jitter Based Non-Physical True Random Number Generator, Stephan Müller, July 1, 2022, http://www.chronox.de/jent/doc/CPU-Jitter-NPTRNG.pdf

[3] jitterentropy-library man-page, jitterentropy-library/jitterentropy.3 at master · smuellerDD/jitterentropy-library · GitHub