Userspace Standalone CPU Time Jitter RNG (32-bit with Internal Timer) Entropy Source version 3.4.0

SP 800-90B Non-Proprietary Public Use Document

Document Version 1.1

Document Date: 2023-02-21

Prepared by:

atsec information security corporation

9130 Jollyville Road, Suite 260

Austin, TX 78759

www.atsec.com

Table of Contents

1 Description

The Userspace Standalone CPU Time Jitter RNG (32-bit with Internal Timer) version 3.4.0 is a non-physical entropy source. The noise generation of this entropy source is based on the tiny variations in the execution time of the same piece of code. The execution time of this piece of code is made unpredictable by the complexity of the different hardware components that comprise modern CPUs and the different internal states that the operating system can have at a certain point in time.

This entropy source uses an internal timer provided by the shared library that implements the entropy source. The shared library supports two choices of timers: the external CPU timer and an internal timer provided by the library itself.

The entropy source was tested on the operational environments listed in Table 1 using both possible timers. The noise source was tested under the assumption that its output is non-IID.

Table 1: Operational environment and version.

Manufacturer	Model	Operational Environment and Version	Processor
Supermicro	Super Server X11DDW-L	SUSE Linux Enterprise Server 15 SP4	Intel(R) Xeon(R) Silver 4215R
GIGABYTE	R181-Z90-00	SUSE Linux Enterprise Server 15 SP4	AMD EPYC(TM) 7371
GIGABYTE	G242-P32-QZ	SUSE Linux Enterprise Server 15 SP4	Ampere(R) Altra(R) Q80-30
IBM	z/15 Model T01	SUSE Linux Enterprise Server 15 SP4	IBM z15
IBM	IBM 9080-HEX	SUSE Linux Enterprise Server 15 SP4	IBM Power10

2 Security Boundary

The entropy source is provided as a standalone shared library. The security boundary for this non-physical, software-based entropy source is the shared library in which it resides.

Figure 1 depicts the overall design of the entropy source and its core operations.

The noise source is implemented by collecting and accumulating time variances of variable memory accesses and variances in the execution time of a defined set of instructions, which includes an implementation of SHA3-256. The time variances, in the form of time deltas, are accumulated and mapped down to 256-bits by the SHA3-256 vetted conditioning function which outputs 256 bits of full entropy.

If the Repetition Count Test (RCT) or the Adaptive Proportional Test (APT) health tests fail, the noise data is discarded, the entropy source halts without outputting any data, and a failure code is returned to the caller.

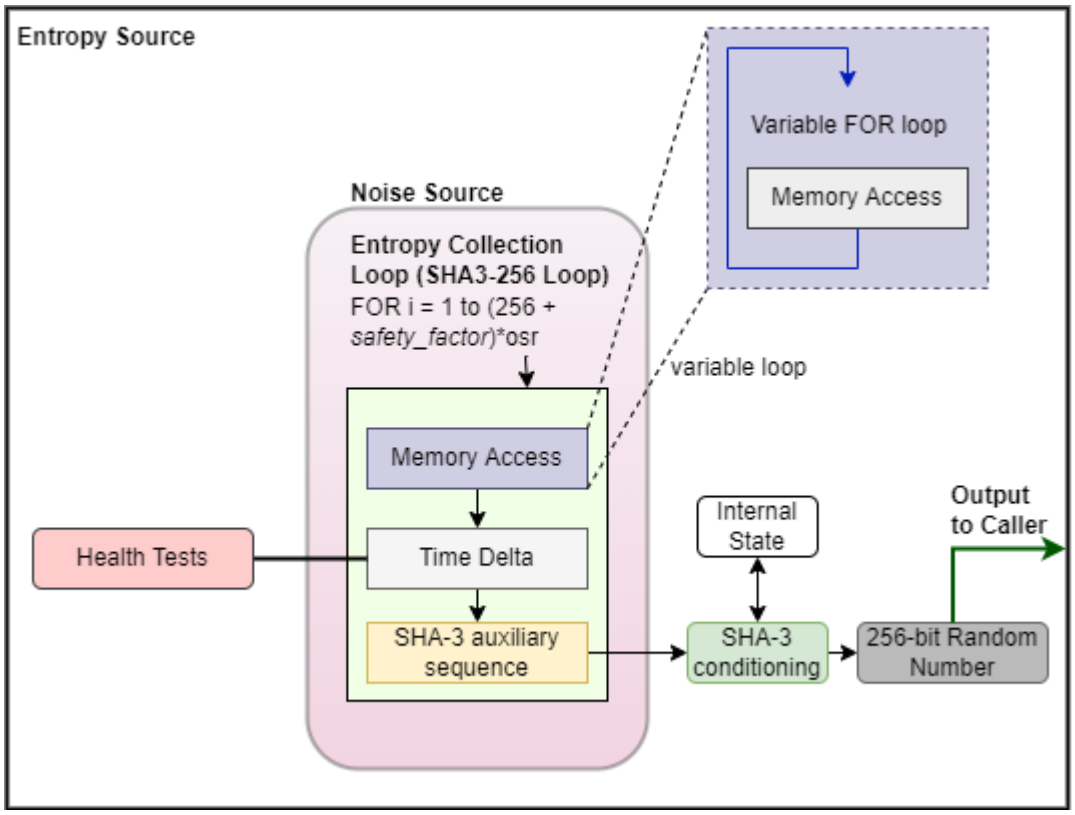

Figure 1: CPU Jitter 3.4.0 Design

3 Operating Conditions

The noise source is non-physical, and thus the operating conditions are inherited from the operational environment in which the entropy source is installed, as shown in Table 2 below.

Table 2: Operating Conditions for each Operational Environment

Manufacturer / Model	Temperature	Voltage	Humidity	Clock Speed	Cache Sizes
Supermicro Super Server X11DDW-L	10C° – 35C°	+12 V	8% - 90%	3.2 GHz	L1: 8x32 KB L2: 8x1 MB L3: 11 MB
GIGABYTE R181-Z90-00	10C° – 35C°	100-240 V	20% - 95%	3.1 GHz	L1: 16x64 KB or 16x32 KB L2: 16x512 KB L3:64 MB
GIGABYTE G242-P32-QZ	10C° – 35C°	100-240 V	8% - 80%	3.3 GHz	L1: 80x64 KB L2: 80x1 MB L3: 80x32 MB

Manufacturer / Model	Temperature	Voltage	Humidity	Clock Speed	Cache Sizes
IBM z/15 Model T01	5C - 40C	200-240 V	8% - 85%	5.2 GHz	L1: 128 KB L2: 4 MB L3: 256 MB
IBM 9080-HEX	4C – 40C	4.6k VA	8% – 85%	3.5 GHz	L1: 48+32 KB L2: 2 MB L3: 120 MB

4 Configuration Settings

The Userspace Standalone CPU Time Jitter RNG (32-bit with Internal Timer) version 3.4.0 obtains time jitter noise from the internal timer. The CPU timer takes precedence over the internal timer during the initialization of the entropy source. Only if the CPU does not support a high-resolution timer or if the library is explicitly configured to do so (JENT_FORCE_INTERNAL_TIMER flag), the internal timer is used.

In order to use this entropy source, the following flag must be provided during the initialization of the entropy source by calling the jent_entropy_init() API.

- JENT_FORCE_INTERNAL_TIMER

5 Physical Security Mechanisms

The noise source is non-physical. The physical security mechanisms only apply to the hardware component of the operational environment in which the entropy source is installed, and thus the entropy source inherits those mechanisms.

6 Conceptual Interfaces

The entropy source provides the following interfaces:

- jent_entropy_init(): Initializes the entropy source context, including the configuration of the CPU timer or the internal timer.
- jent_read_entropy(): Obtains conditioned entropy for the caller. This is the main function of the entropy source, the one that shall be used to request entropy data. This interface corresponds to the GetEntropy() conceptual interface from SP800-90B.
- jent_hash_time(): Obtains raw noise data for testing purposes. This interface corresponds to the GetNoise() conceptual interface from SP800-90B.
- jent_entropy_collector_free(): zeroizes and frees the given entropy collector instance.

7 Min-Entropy Rate

The noise source provides an entropy rate for each time delta of $H_{submitter}$ = 1/3 bits.

The entropy source collects 960 time deltas of 64 bits each from the noise source in order to input to the SHA3-256 vetted conditioning component at least 320 bits of entropy. In other words, the entropy source uses an oversampling rate of three samples in order to collect one bit of entropy; the 320 bits of input entropy (h_{in}) considers the expected output entropy of 256 bits (h_{out}) and a safety factor of 64 bits of additional entropy defined in SP800-90C.

This design ensures that the entropy source can provide full entropy, that is, 256 bits of entropy in its output.

The oversampling rate (osr) and safety factor are fixed values defined in the implementation and cannot be altered.

8 Health Tests

The entropy source implements the following continuous health tests:

- Repetition Count Test conforming to SP 800-90B section 4.4.1.
  - $H = 1$ bit of entropy per 8-bit sample.
  - alpha value of $\alpha = 2^{-30}$.
  - Cutoff value $C = 31$.
- Adaptive Proportion test conforming to SP 800-90B section 4.4.2.
  - $W = 512$
  - $H = 1$ bit of entropy per 8-bit sample
  - alpha value of $\alpha = 2^{-30}$.
  - Cutoff value $C = 325$.
- Stuck (Non-Permanent) Test: The stuck test computes the first, second and third discrete derivatives of the time value that will be processed by SHA3-256. If any of these derivatives are zero, then the received time delta is considered stuck. In this case the input state to SHA3-256 is not updated, and the entropy value is not counted. The stuck test then triggers the RCT for further processing. The second derivative is in fact the RCT itself.
- Lag Predictor Test: The goal of this test is to detect a failure mode in which the outputs may become mostly deterministic. In essence, this test constructs a scoreboard and tracks the number of times that a subpredictor was correct. The subpredictor that scored the most correct predictions is used to predict the next value of a series. The lag predictor test is configured in this entropy source with the following parameters:
  - $\alpha = 2^{-22}$
  - Window size: $window\_size = 131072$
  - Lag history size: $lag\_history\_size = 8$
  - Global cutoff = $InverseBinomialCDF = CRITBINOM(n = window_{size} - lag_{history_{size}}; p = 2^{-\frac{1}{osr}}; 1 - \alpha)$
  - Local cutoff = 111

The continuous-health tests are applied to each new sample obtained from the noise source. Whenever a failure is detected during the health testing specifically for the RCT and APT, entropy data is not returned to the caller; instead, a failure code is returned to enable the caller to acknowledge the failure. The entropy source then halts and will refuse new

requests for entropy. Upon return of the failure code, the caller shall attempt to reset or reboot the entropy source or return an error to its own operator. The stuck test is considered non-permanent, as positive stuck tests will be registered but will not immediately halt the entropy source.

Startup tests conduct the same set and parameters of the continuous health tests on 1024 samples of noise data. The data is discarded after the startup tests have completed successfully.

On-demand health tests of the noise source may be performed by rebooting the operational environment, which results in the immediate execution of the start-up tests. Typically, this entropy source designed for user space cannot be reloaded without restarting the executable. Similarly, the data used for the on-demand health tests are discarded after successful completion.

The following error codes are defined for jent_read_entropy():

- -1 entropy_collector is NULL
- -2 RCT failed
- -3 APT test failed
- -4 The timer cannot be initialized
- -5 LAG failure

9 Maintenance

There are no maintenance requirements as this is a software-based entropy source.

10 Required Testing

To test the entropy source, raw data samples must be collected using both timer options using a test harness that is capable of accessing the jent_hash_time() noise interface from the entropy source. The test harness and accessory tools must be supplied by the vendor.

Raw noise data samples consisting of at least 1,000,000 bits must be collected from the operational environment at its normal operating conditions and processed by the SP 800-90B entropy tool that is provided by NIST. The expected min-entropy rate must approach the one in Section 7.

Restart data must be collected at normal operating conditions through the jent_hash_time() interface following the restart procedure specified in SP 800-90B (i.e., 1,000 samples from 1,000 restarts each) and processed by the NIST SP 800-90B entropy tool. The minimum of the row-wise and column-wise entropy rate must be more than half that of the raw noise entropy rate.

In order to collect samples output from the vetted SHA3-256 conditioning component, a test harness that is capable of accessing the jent_read_entropy() interface is required.