Network Attached QRNG Appliance

User Guide

Crypta Labs

Contents

1 Introduction

This document describes the usage of Crypta Labs Network Attached QRNG Appliance. Random Numbers are delivered via TCP/IP interface.

2 User Guide

2.1 Technical Speciication

- RNG Transmit speed: 492Kbit/s
- Internal Processor: ARM Cortex M4; STmicroelectronics Nucleo-F439ZI.
- Random number format: RAW / NIST Approved Post-Processed

2.2 Requirements

2.2.1 Hardware and Network

- Crypta Labs Network Attached QRNG Appliance
- Ethernet cable(s)
- LAN with DHCP server
- Allow traffic to TCP port **54936**

2.2.2 Software

For operation of the API:

- Python 3
- Python script `qrngappliancetool.py` available for download from cryptalabs.com/qrng-driver-downloads

2.3 The Appliance - Overview

The standard 1U rack mounted Appliance contains 2 independent QRNGs, accessible via separate front-mounted Ethernet ports.

Front

Back

2.4 How to set up

2.4.1 Connections

- Connect the appliance to the main power (220V).
- Connect each QRNG to the desired Ethernet LAN using the RJ45 ports on the front panel.
- Power on the Appliance using the power button.

2.4.2 Identify the devices on the network

The QRNG requires DHCP to obtain the IP configuration (IP address, default gateway etc), and presents itself with the hostname `qrng-<serial number>` on the network (e.g. qrng-3972326031375119 or qrng-3972326031375119.local); it can be reached with that name if the network allows the mDNS protocol.

The device can also be found using mDNS queries (e.g. avahi-browse, bonjour discovery) with the service: `_qrng._tcp`. This feature is available only if the network allows mDNS traffic. Each device has a unique MAC address that can be configured on the DHCP server to get reserve a fixed IP address (if required).

Names and addresses for the Appliance are found on a sticker on the side of the device.

WARNING

Communication is NON secure "plain" TCP/IP so it is advisable to use the QRNG appliance in a secure environment. Location in the same locked rack as the host that requires random numbers is advised.

2.5 Indicator panel

There are 3 LEDs on the appliance that are used to give a quick visual indication of the status of the test bench firmware.

- **Power button**
  Toggle power ON/OFF.
- **Reset button**
  Reset both QRNGs.
- **Activity LED**
  - Off: Ready
  - Blink briefly: request in progress
  - Slow constant blink (toggle every second): Error, try to reset/restart the Appliance

3 Appliance command line tool

The Appliance command line tool script `qrngappliancetool.py` is provided to interface with the QRNG Appliance. It implements the QRNG communication protocols and allows control of the Appliance, as well as to read random data and store it in a file. It shall run on a host that can reach the QRNG (e.g. on the same LAN). It can also be imported by another Python script and used as a library.

NOTE: If a host is accessing a QRNG, that QRNG will reject any other connection request until the active session is closed, i.e. ONLY ONE CLIENT can access a QRNG at a time.

3.1 Basic Usage

3.1.1 Get Post-Processed random data

Post Processed Random Numbers are required when using an RNG for Cryptographic operations, as well as any operation requiring a random number. The option -r(--read) can be used to get post-processed

random data from the QRNG. The post-processing configuration can be set with the --postprocess option as a standalone command or in the same invocation of --read.

In order to read Quantum Random Numbers to a file, perform the following command:

```
$ python qrngappliancetool.py -d <IP Address or hostname> -r <bytes to read> -o
↪  <filename>
```

The numbers are written in binary format to the specified output file. The maximum number of bytes that can be extract with a single command depends on the current status of the QRNG. Use the `--get-status` (3.1.3) option to check how many bytes are available.

It is possible to append the extracted random data to an existing file with the –a option:

```
$ python qrngappliancetool.py -d <IP Address or hostname> -r <bytes to read> -o
↪  <existing_filename> -a
```

3.1.2 Get QRNG device info

Check QRNG device serial number and firmware version.

Command

```
$ python qrngappliancetool.py -d <IP Address or hostname> --info
```

Output

```
### GET INFO DONE
 Core version: 1.2
 SW version: 1.5
 Serial: 3972326031375119
 HW info: NUCLEO-F439zi
```

3.1.3 Get QRNG status

Check the QRNG status, number of bytes available for reading and error flags.

Command

```
$ python qrngappliancetool.py -d <IP Address or hostname> --get-status
```

Output

```
### STATUS:
 started: 1
 startup_test_in_progress: 0
 voltage_low: 0
 voltage_high: 0
 voltage_undefined: 0
 bitcount: 0
 repetition_count: 0
 adaptive_proportion: 0
 Ready bytes: 33600
```

3.1.4 Post-process configuration

By default, the QRNG will operate with NIST approved SHA256 post-processing enabled. It is possible to reconfigure the QRNG to output the RAW noise using the 'set configuration' command:

```
$ python qrngappliancetool.py -d <IP Address or hostname> --postprocess RAW_NOISE
```

To re-enable SHA256 post-processing:

```
$ python qrngappliancetool.py -d <IP Address or hostname> --postprocess SHA256
```

To check the current post-processing configuration, use the command:

```
$ python qrngappliancetool.py -d <IP Address or hostname> --get-config
```

The configuration is volatile and the QRNG starts with the firmware default (SHA256 for FW V.1.x).

3.2 qrngappliancetool.py Operating Parameters

- **-h, --help**
  Show the help message and exit
- **-d DEVICE, --device DEVICE**
  Device hostname or IP address
- **-s, --get-status**
  Get current QRNG configuration
- **-g, --get-config**
  Get current QRNG post-processing configuration
- **-t, --get-statistics**
  Get QRNG statistics
- **-r {READ_SIZE}, --read {READ_SIZE}**
  One shot read. READ$_SIZEinBytes$

- **-o {OUTPUT_FILE},--output-file {OUTPUT_FILE}**
  Output file (used with -r, –read)
- **-a, --append**
  Append to output file (used with -o, –output-file)
- **--reset**
  Reset error statistics and repeat start-up test
- **--info**
  Get device information
- **--postprocess SHA256,RAW_NOISE**
  Set output type to RAW or post-processed
- **-v, --verbose**
  Print debug information and random data

4 Examples

This section provides several examples of qrngappliancetool.py usage. In all the examples the IP Address of the QRNG device is 192.168.137.5. In practice, it must be changed to the actual hostname or IP address assigned by your DHCP server.

4.1 Read RAW noise

By default the QRNG is configured to condition the data with the NIST approved hashing algorithm SHA256. It is possible to read the raw noise data before condtioning by setting the configuration RAW_NOISE with the --postprocess option before a --read.

Command

```
$ python qrngappliancetool.py -d 192.168.137.5 --postprocess RAW_NOISE
```

Output

```
### SET CONFIGURATION DONE
```

Check the current post-processing configuration:

Command

```
$ python qrngappliancetool.py -d 192.168.137.5 -g
```

Output

```
### Post process Configuration:RAW NOISE
```

Read 1024 bytes of raw noise:

Command

```
$ python qrngappliancetool.py -d 192.168.137.5 -r 1024 -o raw_noise.bin
```

Output

```
One shot read success
```

4.2 Randomness quality test

There are several tools available for Linux that can be used to check the quality of randomness of the data generated by the QRNG. Here we will look at some examples.

4.2.1 ENT tool

Ent performs a variety of tests on the stream of bytes in file and produces output on terminal. A file must be generated first following the basic usage guide (see 3.1), in this case we use the append option to read 20000 bytes at a time for 5 times (to get a random datafile of 100000 bytes). The file is then passed to the ent tool for analysis:

Commands

```
$ python qrngappliancetool.py -d 192.168.137.5 -r 20000 -o random_data.bin -a
$ python qrngappliancetool.py -d 192.168.137.5 -r 20000 -o random_data.bin -a
$ python qrngappliancetool.py -d 192.168.137.5 -r 20000 -o random_data.bin -a
$ python qrngappliancetool.py -d 192.168.137.5 -r 20000 -o random_data.bin -a
$ python qrngappliancetool.py -d 192.168.137.5 -r 20000 -o random_data.bin -a
```

Command

```
$ ent random_data.bin
```

Output

```
Entropy = 7.998001 bits per byte.

Optimum compression would reduce the size
of this 100000 byte file by 0 percent.

Chi square distribution for 100000 samples is 278.42, and randomly
would exceed this value 15.01 percent of the times.

Arithmetic mean value of data bytes is 127.1995 (127.5 = random).
Monte Carlo value for Pi is 3.147245890 (error 0.18 percent).
Serial correlation coefficient is 0.002949 (totally uncorrelated = 0.0).
```

4.2.2 FIPS 140-2 tests

The Linux tool **rngtest** can be used to test the random numbers generated by the QRNG according to the standard FIPS 140-2 test (see http://csrc.nist.gov/cryptval/140-2.htm).

Example: Configure the post-processing as required and generate a random data file as the previous example then pass the file to rngtest:

Command

```
$ rngtest < random_data.bin
```

Output

```
rngtest 5
Copyright (c) 2004 by Henrique de Moraes Holschuh
This is free software; see the source for copying conditions.  There is NO warranty; not
↪  even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

rngtest: starting FIPS tests...
rngtest: entropy source drained
rngtest: bits received from input: 800000
rngtest: FIPS 140-2 successes: 39
rngtest: FIPS 140-2 failures: 0
rngtest: FIPS 140-2(2001-10-10) Monobit: 0
rngtest: FIPS 140-2(2001-10-10) Poker: 0
rngtest: FIPS 140-2(2001-10-10) Runs: 0
rngtest: FIPS 140-2(2001-10-10) Long run: 0
rngtest: FIPS 140-2(2001-10-10) Continuous run: 0
rngtest: input channel speed: (min=6666666666.667; avg=17727272727.273; max=0.000)bits/s
rngtest: FIPS tests speed: (min=139.223; avg=208.424; max=216.744)Mibits/s
rngtest: Program run time: 3953 microseconds
```

4.2.3 Dieharder tests

Dieharder is a testing and benchmarking tool for random number generators that implements the dieharder test suite (see https://en.wikipedia.org/wiki/Diehard_tests). It takes a binary file as input, see the following example where dieharder performs all the test (-a option) on a file containing random data acquired with the appliance tool:

Command

```
$ dieharder -f random_data.bin  -a
```

Output

```
#=============================================================================#
#            dieharder version 3.31.1 Copyright 2003 Robert G. Brown          #
#=============================================================================#
   rng_name    |           filename             |rands/second|
        mt19937|                 random_data.bin|  1.27e+08  |
#=============================================================================#
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
#=============================================================================#
   diehard_birthdays|   0|       100|     100|0.20520191|  PASSED
      diehard_operm5|   0|   1000000|     100|0.07143619|  PASSED
  diehard_rank_32x32|   0|     40000|     100|0.17916889|  PASSED
```

```
    diehard_rank_6x8|   0|    100000|     100|0.06068535|  PASSED
   diehard_bitstream|   0|   2097152|     100|0.65992924|  PASSED
        diehard_opso|   0|   2097152|     100|0.40558091|  PASSED
        diehard_oqso|   0|   2097152|     100|0.09068025|  PASSED
         diehard_dna|   0|   2097152|     100|0.76942639|  PASSED
diehard_count_1s_str|   0|    256000|     100|0.98338052|  PASSED
diehard_count_1s_byt|   0|    256000|     100|0.71167368|  PASSED
 diehard_parking_lot|   0|     12000|     100|0.60921267|  PASSED
    diehard_2dsphere|   2|      8000|     100|0.90275483|  PASSED
    diehard_3dsphere|   3|      4000|     100|0.91490512|  PASSED
     diehard_squeeze|   0|    100000|     100|0.96414388|  PASSED
        diehard_sums|   0|       100|     100|0.01469459|  PASSED
        diehard_runs|   0|    100000|     100|0.95815388|  PASSED
        diehard_runs|   0|    100000|     100|0.03510837|  PASSED
       diehard_craps|   0|    200000|     100|0.53225310|  PASSED
       diehard_craps|   0|    200000|     100|0.72374167|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.95711679|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.54363687|  PASSED
         sts_monobit|   1|    100000|     100|0.68486675|  PASSED
            sts_runs|   2|    100000|     100|0.55119226|  PASSED
          sts_serial|   1|    100000|     100|0.44307686|  PASSED
          sts_serial|   2|    100000|     100|0.76314686|  PASSED
          sts_serial|   3|    100000|     100|0.98540422|  PASSED
          sts_serial|   3|    100000|     100|0.37450409|  PASSED
          sts_serial|   4|    100000|     100|0.45600606|  PASSED
          sts_serial|   4|    100000|     100|0.16186904|  PASSED
          sts_serial|   5|    100000|     100|0.32918255|  PASSED
          sts_serial|   5|    100000|     100|0.86359472|  PASSED
          sts_serial|   6|    100000|     100|0.91619297|  PASSED
          sts_serial|   6|    100000|     100|0.19457935|  PASSED
          sts_serial|   7|    100000|     100|0.48190974|  PASSED
          sts_serial|   7|    100000|     100|0.38694001|  PASSED
          sts_serial|   8|    100000|     100|0.03035239|  PASSED
          sts_serial|   8|    100000|     100|0.17244589|  PASSED
          sts_serial|   9|    100000|     100|0.02096218|  PASSED
          sts_serial|   9|    100000|     100|0.11640641|  PASSED
          sts_serial|  10|    100000|     100|0.20486413|  PASSED
          sts_serial|  10|    100000|     100|0.85495091|  PASSED
          sts_serial|  11|    100000|     100|0.10751553|  PASSED
          sts_serial|  11|    100000|     100|0.59565344|  PASSED
          sts_serial|  12|    100000|     100|0.15995577|  PASSED
          sts_serial|  12|    100000|     100|0.73393350|  PASSED
          sts_serial|  13|    100000|     100|0.93322062|  PASSED
          sts_serial|  13|    100000|     100|0.90484373|  PASSED
          sts_serial|  14|    100000|     100|0.64573464|  PASSED
          sts_serial|  14|    100000|     100|0.47560009|  PASSED
          sts_serial|  15|    100000|     100|0.95282361|  PASSED
          sts_serial|  15|    100000|     100|0.07098736|  PASSED
          sts_serial|  16|    100000|     100|0.54614810|  PASSED
          sts_serial|  16|    100000|     100|0.86597839|  PASSED
         rgb_bitdist|   1|    100000|     100|0.80120980|  PASSED
         rgb_bitdist|   2|    100000|     100|0.33186853|  PASSED
         rgb_bitdist|   3|    100000|     100|0.28207120|  PASSED
```

```
        rgb_bitdist|   4|    100000|     100|0.96677579|  PASSED
        rgb_bitdist|   5|    100000|     100|0.83416727|  PASSED
        rgb_bitdist|   6|    100000|     100|0.98353834|  PASSED
        rgb_bitdist|   7|    100000|     100|0.91319192|  PASSED
        rgb_bitdist|   8|    100000|     100|0.80437914|  PASSED
        rgb_bitdist|   9|    100000|     100|0.97169278|  PASSED
        rgb_bitdist|  10|    100000|     100|0.99958223|   WEAK
        rgb_bitdist|  11|    100000|     100|0.32359194|  PASSED
        rgb_bitdist|  12|    100000|     100|0.25326094|  PASSED
rgb_minimum_distance|   2|     10000|    1000|0.78225574|  PASSED
rgb_minimum_distance|   3|     10000|    1000|0.18493280|  PASSED
rgb_minimum_distance|   4|     10000|    1000|0.93658380|  PASSED
rgb_minimum_distance|   5|     10000|    1000|0.59430072|  PASSED
    rgb_permutations|   2|    100000|     100|0.20386863|  PASSED
    rgb_permutations|   3|    100000|     100|0.60678863|  PASSED
    rgb_permutations|   4|    100000|     100|0.78815324|  PASSED
    rgb_permutations|   5|    100000|     100|0.55815493|  PASSED
      rgb_lagged_sum|   0|   1000000|     100|0.83777492|  PASSED
      rgb_lagged_sum|   1|   1000000|     100|0.68799508|  PASSED
      rgb_lagged_sum|   2|   1000000|     100|0.74013041|  PASSED
      rgb_lagged_sum|   3|   1000000|     100|0.95881177|  PASSED
      rgb_lagged_sum|   4|   1000000|     100|0.42821003|  PASSED
      rgb_lagged_sum|   5|   1000000|     100|0.48243836|  PASSED
      rgb_lagged_sum|   6|   1000000|     100|0.75218489|  PASSED
      rgb_lagged_sum|   7|   1000000|     100|0.98037223|  PASSED
      rgb_lagged_sum|   8|   1000000|     100|0.44197233|  PASSED
      rgb_lagged_sum|   9|   1000000|     100|0.36654044|  PASSED
      rgb_lagged_sum|  10|   1000000|     100|0.81731950|  PASSED
      rgb_lagged_sum|  11|   1000000|     100|0.93515683|  PASSED
      rgb_lagged_sum|  12|   1000000|     100|0.90936569|  PASSED
      rgb_lagged_sum|  13|   1000000|     100|0.10019861|  PASSED
      rgb_lagged_sum|  14|   1000000|     100|0.90134486|  PASSED
      rgb_lagged_sum|  15|   1000000|     100|0.48642885|  PASSED
      rgb_lagged_sum|  16|   1000000|     100|0.98634579|  PASSED
      rgb_lagged_sum|  17|   1000000|     100|0.21781613|  PASSED
      rgb_lagged_sum|  18|   1000000|     100|0.12102322|  PASSED
      rgb_lagged_sum|  19|   1000000|     100|0.41823422|  PASSED
      rgb_lagged_sum|  20|   1000000|     100|0.73335150|  PASSED
      rgb_lagged_sum|  21|   1000000|     100|0.40956982|  PASSED
      rgb_lagged_sum|  22|   1000000|     100|0.29099940|  PASSED
      rgb_lagged_sum|  23|   1000000|     100|0.81700238|  PASSED
      rgb_lagged_sum|  24|   1000000|     100|0.32492036|  PASSED
      rgb_lagged_sum|  25|   1000000|     100|0.48324813|  PASSED
      rgb_lagged_sum|  26|   1000000|     100|0.65786102|  PASSED
      rgb_lagged_sum|  27|   1000000|     100|0.22258597|  PASSED
      rgb_lagged_sum|  28|   1000000|     100|0.33944816|  PASSED
      rgb_lagged_sum|  29|   1000000|     100|0.56701377|  PASSED
      rgb_lagged_sum|  30|   1000000|     100|0.78989726|  PASSED
      rgb_lagged_sum|  31|   1000000|     100|0.90601059|  PASSED
      rgb_lagged_sum|  32|   1000000|     100|0.29979095|  PASSED
     rgb_kstest_test|   0|     10000|    1000|0.10677483|  PASSED
     dab_bytedistrib|   0|  51200000|       1|0.87344353|  PASSED
             dab_dct| 256|     50000|       1|0.91348851|  PASSED
```

```
Preparing to run test 207.  ntuple = 0
        dab_filltree|  32|  15000000|       1|0.98202684|  PASSED
        dab_filltree|  32|  15000000|       1|0.81157277|  PASSED
Preparing to run test 208.  ntuple = 0
       dab_filltree2|   0|   5000000|       1|0.37235767|  PASSED
       dab_filltree2|   1|   5000000|       1|0.66626998|  PASSED
Preparing to run test 209.  ntuple = 0
        dab_monobit2|  12|  65000000|       1|0.46153353|  PASSED
```

4.3 Read random data in Python

The qrngappliancetool.py can be imported as a module in a custom Python script, the following example is a simple script that reads 1024 random bytes from a QRNG with IP 192.168.137.5 and prints them in hexadecimal format:

```
from qrngappliancetool import qrng_cmdctrl

# Open the connection with the QRNG
device_comm = qrng_cmdctrl("192.168.137.5")

# Execute the read one shot command for 1024 bytes
random_data = device_comm.start_one_shot(1024)

# the results is a bytearray with the requested random bytes or null in case of error
if(random_data):
        print(random_data.hex())
else:
        print("Error reading")

# Close the connection
device_comm.close()
```