QCicada QRNG

User Guide

Version 1.3

51 St John's Sq
London
EC1V 4JL

Contact details:
Email: support@cryptalabs.com
Internet: https://www.cryptalabs.com/

Contents

Introduction

This document contains the user's guide for the Crypta Labs USB Quantum Random Number Generator (QRNG) – also known as **QCicada QRNG**.

System Requirements

Operating System	Linux, Windows Tested on Ubuntu 18.x/20.x/22.x/24.x, Centos 7, Windows 10/11
Interface	USB 2.0

Crypta Labs supply the following executables/scripts to support the QCicada QRNG:

libqcc.so	QCC Library for Linux, used to access the device
qcc.dll	QCC Windows driver, used to access the device
qcc-cli	QCC Interface tool for RNG generation/status query/firmware update.
pyqcc-x.y.z-py3-none-any.whl	Python implementation of QCC Library and interface tool.
pyqcicada-x.y.z-py3-none-any.whl	Python extension library supporting Secure Communications mode.

Crypta Labs supply the following source code to support development with the QCicada QRNG:

qcc-source	Source code for the above Linux/Windows executables

All of the above files are available via https://cryptalabs.com/support/releases

Device Specification

Performance	0.5Mbps (~1,950 Random Numbers per second*)
NIST Compliant output	Yes
Quality Independently verified	Yes
Entropy Source	1 x Quantum Optics Module
Size	2cm x 7cm
Power	
Single Input Voltage	5V
Normal operation	48mW
Idle Mode	12mW
Startup from Idle	<1ms
Recommended operating Temperature	-30°C ~ +85°C
Max operating Temperature	-40°C ~ +125°C
Interface	USB - 1Mbps

*Assumes random number of 256bits

Installation Guide

For details on how to install the Python version of the QCC Control Library, please see the later section: Developing with QCicada \ Python.

Linux

Preparation for Installation

In order to test QCicada operation, we will use the QCC Control Library and `qcc-cli` executable. All examples shown are tested on Ubuntu 18.x and above.

Setup local directories

1. Download the Crypta Labs assets from the links within the System Requirements section above.

You need to create a suitable directory in which to place the `qcc-cli` and `libqcc.so`. In this example, and used elsewhere in this guide, we are creating a directory called `/opt/QCicada`

2. Enter the following commands to create this new directory. In the `chown` command, specify the user and the owning group as regards the directory ownership and permissions. Execute these commands with `sudo` permissions if appropriate:

```
cd /opt
mkdir QCicada
chown {user}:{owning_group} QCicada
```

3. Now extract and copy the contents relevant to your OS to the `/opt/QCicada` directory. (E.g. If you use Ubuntu, move files from the `Ubuntu` folder within the archive).
Next set the permissions for files to be executable:

```
chmod 744 ./qcc-cli libqcc.so
```

4. Move `libqcc.so` to the `/usr/lib` folder:

```
sudo mv libqcc.so /usr/lib
```

You are now ready to validate that the QRNG is working correctly.

5. Plug the Qcicada device into a spare USB port on your computer.

6. The device ID is required on the command line to identify which QCicada is being addressed. To find this, Enter the command:

   ```
   dmesg
   ```

   This will produce output on the console like the below. QCicada will be detected as `FTDI USB Serial Device converter.`

   In the case of this example, the ID of QCicada is `ttyUSB0`: and this ID will be used across all examples in this document. **Ensure you use the ID relevant to your system.**

   ```
   [    52.856736] usb 2-1: Detected FT-X
   [    52.868211] usb 2-1: FTDI USB Serial Device converter now
   attached to ttyUSB0
   ```

7. Enter the command:

   ```
   ./qcc-cli -d /dev/ttyUSB0 -f -v
   ```

   You should see results similar to the following:

   ```
   #### QCC command-line tool ####
   #### Copyright (C) Crypta Labs 2023 ####

   Initialize QCC

   ## QRNG INFO:
    core version: 0x1000A
    FW version:   0x5000A
    Serial:       QC0000000101
    HW info:      CICADA-QRNG
   ```

If you see output like the above, the QCicada is functioning correctly.

NOTE:

To run the code without root privileges, usually the QCicada Serial device is presented on Linux as `/dev/ttyUSB0` (if one device is present) and usually that device is assigned to the "dialout" group.

The user accessing the QCicada device must be added to the group "dialout", in this way the process will be able to use the device without needing to be a super user.

For example, the file permissions for the `/dev/ttyUSB0` device are as follows:

```
crw-rw----  1 root dialout 188,   0 Jul 11 11:49 ttyUSB0
```

To be able to use the device without root privileges, add the non-root user to the dialout group with the following command::

```
sudo usermod -a -G dialout <non-root-username>
```

Update the users groups by restarting the ssh session, or rebooting the system, after that you should be able to use the binary as a non-root user.

Windows

Preparation for Installation

In order to test QCicada operation, we will use the supplied `qcc-cli` executable. All examples shown are tested on Windows 10.

Setup local directories

1. Download the Crypta Labs assets from the links within the System Requirements section above.

You need to create a suitable directory in which to place the `qcc-cli.exe` and `libqcc.dll`. In this example, and used elsewhere in this guide, we are creating a directory called `C:\QCicada`

2. Either use Windows Explorer to create a directory named QCicada in the root of C:\ or enter the following commands in Command Prompt:

```
cd\
md QCicada
```

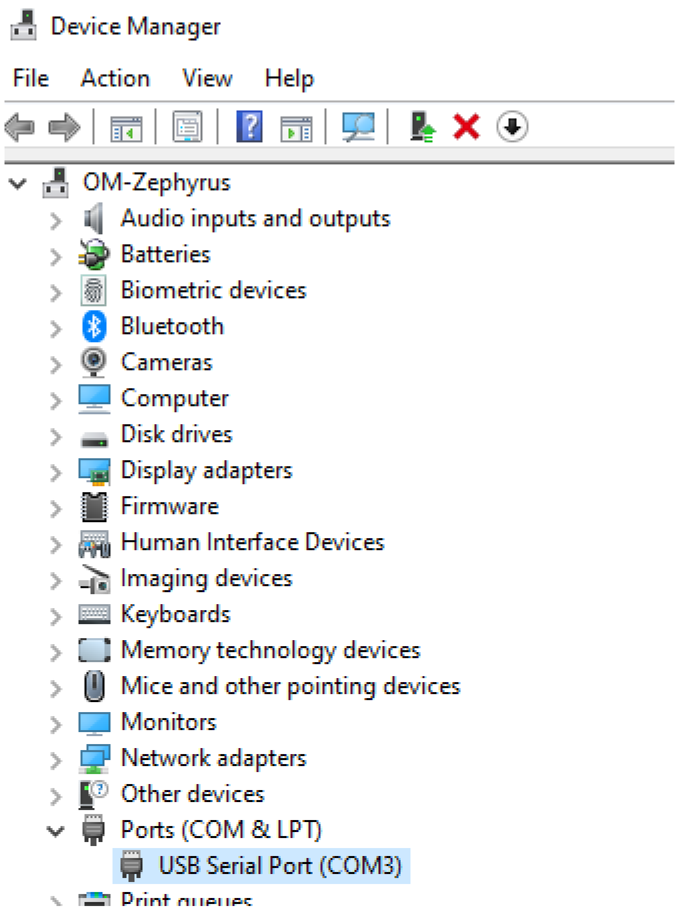
3. Now extract and copy the contents from the `\Windows.x64\` folder within the downloaded archive to the newly created directory.

You are now ready to validate that the QRNG is working correctly.

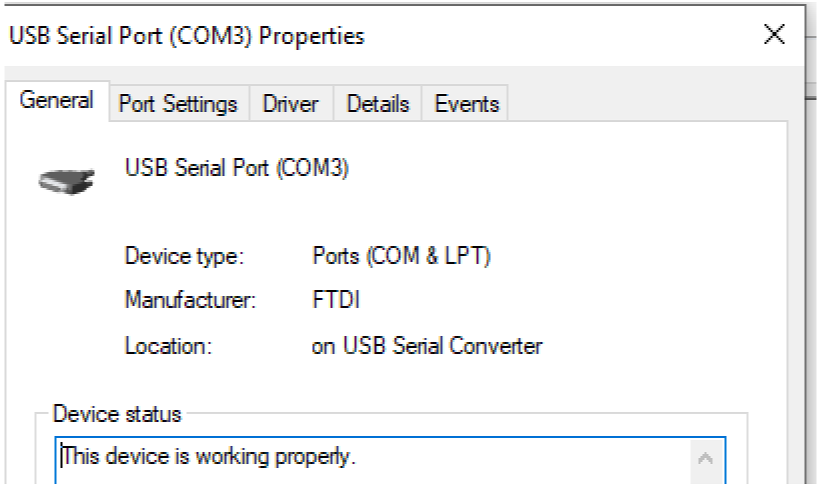
Validating the install

4. Plug the Qcicada device into a spare USB port on your computer.

5. The COM port number is required on the command line to identify which QCicada is being addressed. To find this, use Device Manager and navigate to ‘Ports (COM & LPT)’. Expand the list to see connected devices as illustrated in the image on the next page.



If you are unsure on which device is which, right click on a detected 'USB Serial Port (COMx)' device and choose 'Properties'. QCicada will display as 'FTDI' as the Manufacturer.



8. In this example we are using `COM3` as the relevant port. Enter the command in Command Prompt:

```
qcc-cli -d COM3 -f -v
```

You should see results similar to the following:

```
#### QCC command-line tool ####
#### Copyright (C) Crypta Labs 2023 ####

Initialize QCC

## QRNG INFO:
 core version: 0x1000A
 FW version:   0x5000A
 Serial:       QC0000000101
 HW info:      CICADA-QRNG
```

If you see output like the above, the QCicada is functioning correctly.

NOTE:
All of the Usage Examples in the following pages are demonstrated on a Linux System.

Windows users must substitute the serial port identifier as below:

Linux = `/dev/ttyUSB0`
Windows = `COM3`

Also, there is no need to precede calls to `qcc-cli` with `./`

Example:

The following illustrates the difference in the syntax for the same command executed on windows and Linux:

Linux Command:

```
./qcc-cli -d /dev/ttyUSB0 -f -v
```

Windows Command:

```
qcc-cli -d COM3 -f -v
```

NOTE:
If the detected COM port is above COM9 (COM10 upwards), the port number must be preceded with `\\.\`

Example:

```
qcc-cli -d \\.\COM10 -f -v
```

Product Usage

The supplied `qcc-cli` tool can be used to access many of QCicada's features. In the below use cases we will document commands used and the expected output.

All examples shown use Linux syntax. Windows users, please see the note verso.

Note: in all examples we will be using the `-v` switch to enable Verbose mode and enhance the console output.

Some Crypta Labs QRNG devices support communication over UDP. **QCicada only supports Serial communication** and this is selected by default in the examples.

All of the below commands are summarised and displayed to the console with the command:

```
./qcc-cli -d /dev/ttyUSB0 --help
```

Generate a Random Number

One Shot mode

One Shot mode will read a requested amount of random data up to 13,440 bytes in size.
To generate 1000 bytes of random data and save to a file named Randgen.bin, use the command:

```
./qcc-cli -d /dev/ttyUSB0 -r 1000 -o Randgen.bin -v
```

The console output should look like the below. The last field named `Data` will display the first and last bit from the generated random data:

```
#### QCC command-line tool ####
#### Copyright (C) Crypta Labs 2023 ####

Initialize QCC
Read 1000 bytes
1000 bytes of data written to file Randgen.bin
Data= 3b ... 6a
```

Continuous mode (Streaming)

Continuous mode is used to generate data continuously until a `Stop` command is issued. It is used when data capture is required in greater amounts than 13,400 bytes. To generate 15,000 bytes of data and save to a file named Randgen.bin, use the following commands:

```
./qcc-cli -d /dev/ttyUSB0 --start -v
```

The above command begins a continuous stream of data being generated on QCicada and passed to the host. You will see the output:

```
#### QCC command-line tool ####
#### Copyright (C) Crypta Labs 2023 ####

Initialize QCC
Continuous mode started!
```

The next command will begin a capture of the data:

```
./qcc-cli -d /dev/ttyUSB0 -c -r 15000 -o Randgen.bin -v
```

Capture time is dependent on the speed of the device, which operates at ~0.5Mbps. A large capture will result in longer wait times to complete. You will see this result on the console:

```
#### QCC command-line tool ####
#### Copyright (C) Crypta Labs 2023 ####

Initialize QCC
Read 15000 bytes
15000 bytes of data written to file Randgen.bin
Data= 28 ... 1b
```

The device remains in continuous mode until it is stopped. To issue a Stop command, use the following:

```
./qcc-cli -d /dev/ttyUSB0 --stop -v
```

Successful response will return the following to the console:

```
#### QCC command-line tool ####
#### Copyright (C) Crypta Labs 2023 ####

Initialize QCC
Continuous mode STOP!
```

Appending captured data to a file

In the previous examples, when a filename is specified to save the captured random data, that file is created or overwritten if it already exists. In order to Append data to an existing file, use the switch `-a` within the command. The following command shows One Shot mode appending 1000 bytes of data to a file named Randgen.bin:

```
./qcc-cli -d /dev/ttyUSB0 -r 1000 -o Randgen.bin -a -v
```

```
Successful console output looks like this:
#### QCC command-line tool ####
#### Copyright (C) Crypta Labs 2023 ####

Initialize QCC
Read 1000 bytes
1000 bytes of data written to file Randgen.bin
Data= 11 ... e8
```

Disable/Enable Post Processing

Post Processing is a function performed on the RAW capture form the onboard Quantum Optics Module (QOM) to ensure the output is compliant with NIST Standards.

By default post processing is enabled on QCicada upon startup.

It may be desirable to output data without Post Processing, this is known as Raw data. QCicada supports 3 methods of output:

- (`0`) Post Processing enabled (NIST compliant output using SHA256)
- (`1`) Raw Noise (Data from the QOM with some conditioning in accordance with the health tests)
- (`2`) Raw Samples (Data directly from the QOM with no conditioning)

To disable Post Processing and enable Raw Noise capture, use the following command:

```
./qcc-cli -d /dev/ttyUSB0 -P 1 -v
```

To disable Post Processing and enable Raw Samples capture, use the following command:

```
./qcc-cli -d /dev/ttyUSB0 -P 2 -v
```

To enable Post Processing, use the following command:

```
./qcc-cli -d /dev/ttyUSB0 -P 0 -v
```

Display QRNG Status

Statistics are available from QCicada as to the status of the device, the readily available data in the buffer and the results of the Active Health Tests which run during operation to ensure quality is maintained.

To view the status, run the command:

```
./qcc-cli -d /dev/ttyUSB0 -s -v
```

You will see output similar to the following on the console:

```
#### QCC command-line tool ####
#### Copyright (C) Crypta Labs 2023 ####

Initialize QCC

## QRNG STATUS:
 Initialized:               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:               13440
```

Note: It is normal to see some errors logged within these statistics. QCicada is constantly monitoring itself and will adjust the QOMs operational parameters accordingly to maintain quality output.

Display Device Information

To display device specific information such as Serial number and firmware revision, enter the command:

```
./qcc-cli -d /dev/ttyUSB0 -f -v
```

You should see results similar to the following:

```
#### QCC command-line tool ####
#### Copyright (C) Crypta Labs 2023 ####

Initialize QCC

## QRNG INFO:
 core version: 0x1000A
 FW version:   0x5000A
 Serial:       QC0000000101
 HW info:      CICADA-QRNG
```

Using Multiple QCicada Devices

It is possible to use multiple QCicada's on the same host. We have seen in previous examples the need to specify the device you are communicating with. This is achieved via the `-d` switch. In order to send a command to a different QCicada, specify the full identifier of the device within the command.

In this example, we will read 1000 bytes of data from a device with ID `ttyUSB1` to a file named `Randgen.bin`:

```
./qcc-cli -d /dev/ttyUSB1 -r 1000 -o Randgen.bin -v
```

Firmware Update

QCicada supports the ability for a user to update firmware. **Only official Firmware releases from Crypta Labs can be used on QCicada.**

In the following case we are updating a QCicada to version 5.10 of the firmware:

```
./qcc-cli -d /dev/ttyUSB0 -U qcicada-ota-5.10-bx.bin -v
```

If the firmware has been updated, output like the following will be displayed on the console:

```
#### QCC command-line tool ####
#### Copyright (C) Crypta Labs 2023 ####

Initialize QCC
Initialize update, FW image size 66944 bytes
Chunk len 1024 written, Update status 1, remaining 65920
Chunk len 1024 written, Update status 1, remaining 64896
[..]
Chunk len 1024 written, Update status 1, remaining 384
Chunk len 384 written, Update status 2, remaining 0
FW UPDATED!
```

Product Usage - Signed Read mode

NOTE:
This feature is only available via the Python library `pyqcicada`. For details on how to install the Python version of the QCC Control Library, please see the later section: Developing with QCicada \ Python.

This function allows the user to verify that the random data is coming from a known and trusted Crypta Labs device.

The data is ECDSA signed using a private key stored within the secure One Time Programmable (OTP) area of the device (NIST P256 curve).

The device Private Key is generated internally in the device and saved during device Certificate Provisioning.

Getting Started - Certificate Provisioning

Before Signed Read mode can be used, the QRNG must be provisioned with a Certificate. Certificate Provisioning **is performed one time only** per QCicada device.

Certificate Provisioning is the process of creating and signing a Certificate with a key pair unique to the user/company using/distributing the QCicada device, and then writing it to the device's secure OTP area. The Certificate is generated using the following device specific data:

- HW ID
- Device serial
- Device public key

The company key pair used to sign the certificate is also known as the 'Certificate Authority key'. **Extreme care should be taken to store the Private part of the CA key**.

How to perform Certificate Provisioning

1. Generate a CA key pair
2. Write it to the QCicada

   Follow the instructions in the online reference guide:
   https://cryptalabs.com/support/docs/pyqcicada/#certificate-authority

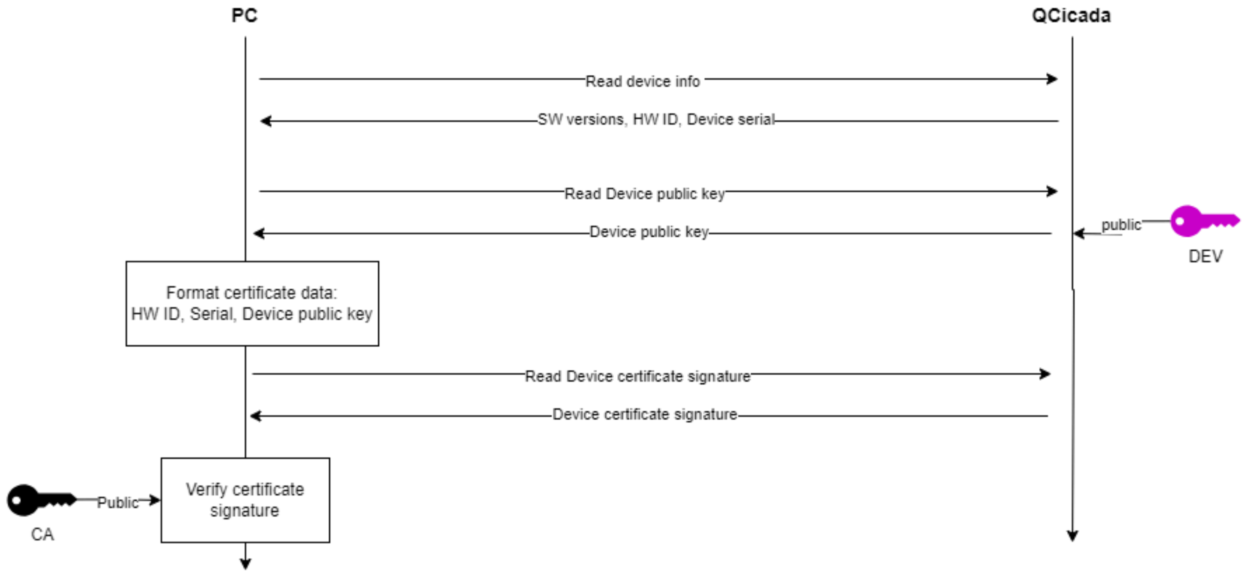
Generating and Reading Signed Data

A user can verify if the device is genuine by reading the device specific information to re-create the full certificate and verify the signature.

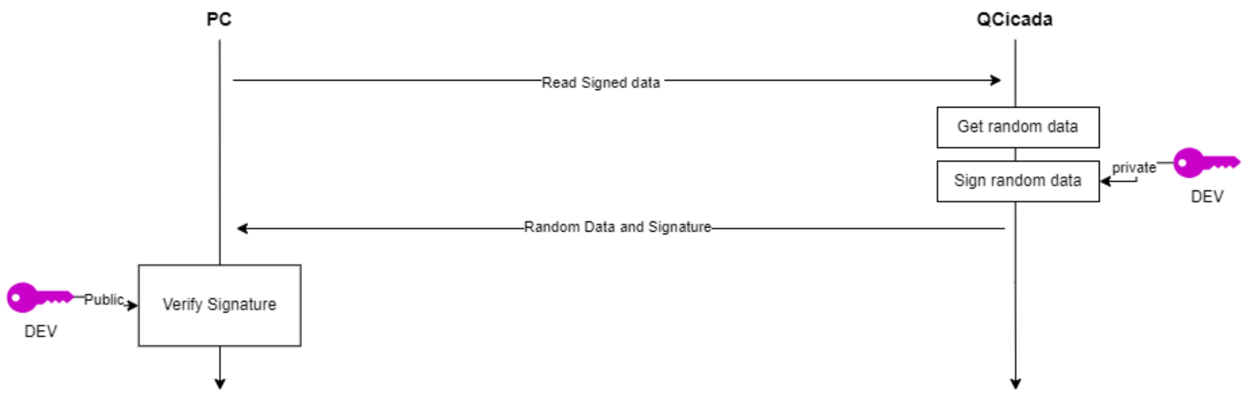
When the user asks for Signed data (1 to 13400 bytes per read) the device will reply with the data bytes and 64 bytes ECDSA signature. The user can then verify the signature of the received data with the device public key (that can be read and verified before reading the data) The signature verification allows the user to verify the authenticity and integrity of the random data.

This process is illustrated as follows:

Step1: The device public key is extracted and device is verified:



Step 2: The data is verified upon receipt from the device:



See the below section ‘Developing with QCcicada’ for examples on how to read and verify data.

Product Features

Secure Boot

Firmware images are encrypted and digitally signed. When the firmware image is loaded into QCicada it is decrypted and verified. The private key used to sign the image is retained and kept secure by Crypta Labs. The encryption uses AES-256 in CBC mode.

Real time Health Tests

Health tests run in conjunction with random number generation to ensure the data provided by QCicada is of the highest quality. Tests implemented include those approved and specified within the NIST 800-90b Standard.

Environmental factors such as heat can have an impact on the optics within the QRNG, and so tests are conducted on the QOM and AutoCalibration functions are in place to adjust the operational parameters as required.

Developing with QCicada

NOTE:

The QRNG sends data in little-endian format. Depending on your host machine, it might be necessary to account for that.

Stopping the Continuous mode can take some time, this depends on the timeout used.

Examples

Python

Python 3.9 or higher is required

Installation

1. Download the latest version of pyqcc from the link in the System Requirements section
2. Use the following command to install pyqcc:

```
pip install pyqcc-x.y.z-py3-none-any.whl
```

Within `pyqcc` we have produced a standalone implementation of both the communication library/driver and the CLI example. It can be used independently, or as a module which is imported into your own projects.

The command line tool `pyqcc-cli` is installed during the above installation, and can be used to communicate with any Crypta Labs QRNG devices supporting USB-CDC and TCP/UDP communication interface. **QCicada only supports USB-CDC as an interface.**

Reading Quantum Random Numbers from a command line

In the following example, 1000 bytes of random data is read from a QCicada device and saved to a local file and named Randgen.bin:

```
pyqcc-cli -d /dev/ttyUSB0 -r 1000 -o randgen.bin
```

Importing pyqcc as a module

In this example, data is read from the QCicada with ID `ttyUSB0`, which it then prints the first and last bytes to the screen.

```
#Example script that reads random data from a QCicada device

import pyqcc
qccdev = pyqcc.cmdctrl.device("serial","/dev/ttyUSB0")

if(qccdev.start_continuous()):
        print("Continuous mode started!")
        # Read 1MB random data
        rnd_bytes = qccdev.read_continuous(1000000)
        if(rnd_bytes):
                #print first and last random bytes
                print("Continuous read success!
{:02X}..{:02X}".format(rnd_bytes[0], rnd_bytes[-1]))
        else:
                print("Error continuous read")
        # Stop continuous mode
        if(qccdev.stop()):
                print("Continuous mode Stopped!")
        else:
                print("Stop error!")
else:
        print("Error starting continuous mode")
```

Reading signed data

In this example we will read and verify signed data from the QCicada.

1. Ensure Certificate Provisioning has been performed on the QCicada.
2. Install the `pyqcicada` package to enable the verification of signed read functionality:

   Download the latest ver of `pyqcicada` from https://cryptalabs.com/support/releases

   Use the following command to install `pyqcicada`:

   ```
   pip install pyqcicada-x.y.z-py3-none-any.whl
   ```

3. See the example script below:

```
import pyqcicada

# Open the device
qcicadadev = pyqcicada.cmdctrl.device("/dev/ttyACM0")

# Read the raw device public key, verified with the CA public key

dev_pubkey_verified =
qcicadadev.get_dev_pub_key_verified("path/to/ca_pubkey.pem")

if(dev_pubkey_verified):

   print("Device public key valid: " + dev_pubkey_verified.hex())

   data = cc.read_signed_and_verify(args.read_verify,
          dev_pubkey_verified)

   if(data):
      print("OK: Data signature is valid!")
      print("Data: {:02X} ... {:02X}".format(data[0],
           data[args.read_verify-1]))

   else:
      print("Error reading signed data")

else:
   print("Device key not valid")
```

API Reference - Python

Full documentation on the Python `pyqcc` and `pyqcicada` APIs is available online:

https://cryptalabs.com/support/docs/pyqcc/
https://cryptalabs.com/support/docs/pyqcicada

C

You can include the QCC library source code and compile it together with your application, or build the library and statically/dynamically link it with your application.

Structure packing

When using a compiler different to gcc or MSVC, take care of the correct packing of the data structure defined in cmdctrl_types.h. If your compiler doesn't support `#pragma pack(1)` modify the file to be compatible with your compiler and perform the correct packing.

Building

Requirements Linux

- make
- cmake (min version 3.16.3)

Requirements Windows

- Visual Studio, MSVC compiler
- cmake

Cmake is used to build both the library (libqcc.so or qcc.dll) and the cli executable under Linux and Windows.

Download the source code package from the link in the System Requirements section and extract to a working folder. Navigate to the folder and into the `/qcc` directory and run the following commands:

```
cmake -S . -B build

cmake --build build
```

It will generate the toolchain files and the final compiled artifacts within the build directory.
The whole build configuration is defined in the file `CMakeLists.txt`
NOTE: On some systems cmake 3.xx command is `cmake3`

Example

Read data from the QRNG

The following program will check the status of the QRNG and then read 1024 bytes of data in One Shot mode, before printing it to the console.

```
#include <stdint.h>
#include <stdio.h>
#include "qcc.h"

qcc_hdl_t qcc;
uint8_t random_data[1024];

int main(void) {
  int ret;
  ret = qcc_init(&qcc, QCC_SERIAL, (char *)"/dev/ttyUSB0", 100,
2048);

  if(ret != QCC_OK){
    printf("ERROR: initializing QRNG device, return=%d\n", ret);
    return -1;
  }

  ret = qcc_cmd_start(&qcc, CMDCTRL_START_ONE_SHOT, random_data,
1024);

  if(ret != QCC_OK){
    printf("ERROR: Reading data, return=%d\n", ret);
     qcc_close(&qcc);
    return -1;
  }

  qcc_close(&qcc);

  for(int i = 0; i < 1024; i++)
       printf("%02X ", random_data[i]);

  printf("\r\n");
  return 0;
}
```

API Reference - C

Return codes

Defined in `qcc_errno.h`, all the APIs return one of the following integer codes:

- `QCC_OK`
- `QCC_ERROR`
- `QCC_TIMEOUT`
- `QCC_INVALID_ARGUMENT`
- `QCC_MALLOC_ERROR`
- `QCC_NACK`

QCC types

- `qcc_hdl_t`: main handle, all the device APIs need this, initialized by the `qcc_init()` function
- `qcc_comm_type_t`: communication interface type

Cmd&Ctrl types

They represent the requests and responses of the Cmd&Ctrl protocol:

- `cmdctrl_start_mode_t`: one shot or continuous mode
- `cmdctrl_status_t`: QRNG status, error conditions and number of ready bytes
- `cmdctrl_config_t`: QRNG configuration
- `cmdctrl_statistics_t`: QRNG Statistics
- `cmdctrl_info_t`: Serial number, software and hardware version

Functions

`int qcc_version(void);`
Description Get QCC library version.
Parameters N/A

`int qcc_init(qcc_hdl_t *hdl, qcc_comm_type_t comm, char *dev_id, int timeout_ms, int read_size)`
Description Initialize QCC device and link to a specific communication handler
Parameters • `hdl` : qcc handle to initialize • `comm` : communication type = `QCC_SERIAL` • `dev_id` : device identification, e.g "`COM1`", "`/dev/ttyUSB0`" • `timeout_ms`: timeout used in the communication with the device • `read_size` : maximum number of bytes to read in one go, depends on the communication interface used and the overall speed of the link.

`int qcc_cmd_get_status(qcc_hdl_t *hdl, cmdctrl_status_t *sts)`
Description Read QRNG status
Parameters • `hdl` : qcc handle • `sts` : pointer to a `cmdctrl_status_t` structure filled with the status received

```
int qcc_cmd_start(qcc_hdl_t *hdl, cmdctrl_start_mode_t mode,
uint8_t *buf, uint16_t len)
```

Description
Start QRNG Generation

Parameters

- `hdl` : qcc handle
- `mode`: start mode, `QCC_ONE_SHOT` to read a chunk of data straight away or `QCC_CONTINUOUS` to start the continuous mode
- `buf` : pointer to a pre-allocated buffer to fill with the data received (only if `QCC_ONE_SHOT`)
- `len` : How many bytes to read (only if `QCC_ONE_SHOT`), limited by the currently available bytes

```
int qcc_read_continuous(qcc_hdl_t *hdl, uint8_t *buf, int len)
```

Description
Read data from a QRNG device currently in continuous mode.
Start continuous mode before using this function

Parameters

- `hdl` : qcc handle
- `buf` : pointer to a pre-allocated buffer to fill with the data received
- `len` : How many bytes to read, limited to 2GB

```
int qcc_cmd_stop(qcc_hdl_t *hdl)
```

Description
Stop continuous mode

Parameters

- `hdl` : qcc handle

```
int qcc_cmd_reset(qcc_hdl_t *hdl)
```

Description
Reset Random number generation

Parameters

- `hdl` : qcc handle

```
int qcc_cmd_set_config(qcc_hdl_t *hdl, cmdctrl_config_t *cfg)
```

Description
Set QRNG configuration

Parameters

- `hdl` : qcc handle
- `cfg` : pointer to a valid `cmdctrl_config_t` structure

```
int qcc_cmd_get_config(qcc_hdl_t *hdl, cmdctrl_config_t *cfg)
```

Description
Get QRNG configuration

Parameters

- `hdl` : qcc handle
- `cfg` : pointer to `cmdctrl_config_t` structure to fill with the configuration received

```
int qcc_cmd_get_statistics(qcc_hdl_t *hdl, cmdctrl_statistics_t
*stats)
```

Description
Get QRNG statistics

Parameters

- `hdl` : qcc handle
- `stats` : pointer to `cmdctrl_statistics_t` structure to fill with the statistics received

```
int qcc_cmd_get_info(qcc_hdl_t *hdl, cmdctrl_info_t *info)
```

Description
Get QRNG device information

Parameters

- `hdl` : qcc handle
- `info` : pointer to `cmdctrl_info_t` structure to fill with the information received

`int qcc_close(qcc_hdl_t *hdl)`
Description Close QCC handle
Parameters ● `hdl` : qcc handle