Random Thoughts

Last Generated: September 26, 2023

Contents

Legal

1 Introduction

2 Requirements

In this section we detail the requirements of the Random Thoughts generator. These requirements are used to drive the architecture and design of the system.

Random Thoughts consists of two parts: a hardware generator that attaches to the computer using a standard USB version 2.0 connection, and a command interface for communicating with the generator hardware. The generator's direct I/O facilities shall consist of three LEDs as follows.

1. *Green*: The generator is functioning correctly and has available entropy.
2. *Yellow*: The generator is functioning correctly but has limited or no entropy. This condition should correct itself automatically after the generator has accumulated entropy from its random source. Using the generator at this time is permitted but the quality of the random numbers produced may be poor.
3. *Red*: The generator is not functioning properly. It should not be used in this condition. Automatic correction of this condition is not expected.

Setting up the generator shall only require connecting it to the computer. The generator draws all its power from the USB connection. No batteries or other adjustments are required.

In addition the generator has a command language that details how the driver can interact with the hardware. This command language shall be usable by third parties for building drivers on systems not otherwise supported.

At the level of the command language only two requests need be supported. The first is a non-blocking request for a specific number of bytes of random data that must be "immediately" delivered, and a blocking request for a specific number of bytes of random data that must be delivered eventually. Immediate delivery means that the data is taken from a buffer in the generator. The request does not have to wait for the generator to produce all of the requested data. If the generator does not have all of the requested data on hand, it should return what it does have along with an appropriate count (which might be zero).

When a blocking request is issued, the generator attempts to return all of the data requested, perhaps after a delay to generate that data. The data does not have to be returned all at once. The generator may return the data in several bursts depending on the size of the request and the size of the generator's on-board buffer.

Note that the detailed format of the messages used in the command language is not specified here. In addition the command language must allow the generator hardware to signal an error condition to the calling software. This error condition is intended to indicate a hardware failure or bad statistics but the specific kinds of errors that are to be detected are not specified here. In particular the generator may or may not be capable of doing a statistical analysis of the numbers it produces. However, error codes for statistical problems should at least be reserved to streamline the development of future generators.

2.1 Core Requirements

The following requirements apply:

- **Core.GenSPARK**. The code on the generator hardware itself shall be in SPARK 2014 and proved free of the possibility of runtime error.
- **Core.Ravenscar**. The code on the generator hardware itself shall be written to conform with the restrictions of the Ravenscar profile.[1].
- **Core.TestsAda**. The statistical testing programs that run on the host shall be written in Ada.
- **Core.Monitor**. The generator shall have a monitoring function that alerts the user if the statistics of the generated random numbers is poor.

The details of **Core.Monitor** are left unspecified. However, the monitoring function must have access to enough generator data to compute reasonable conclusions about the quality of the generator's output. It is unspecified if the statistical monitoring function is implemented in the generator hardware or in the driver or someplace else.

The statistical monitoring discussed here is distinct from the entropy monitoring indicated by the LEDs on the generator hardware. It is possible for the generator to believe it has sufficient entropy to produce random numbers and yet produce such numbers with low quality. The statistical monitoring function serves as a software check against biases that might be inherent in the generation process.

2.2 Non-Functional Requirements

In this section we describe the non-functional requirements of Random Thoughts. The following requirements apply:

[1]This is likely implied by **Core.GenSPARK** anyway

- **NonFun.Review**. All software shall be subjected to a careful security analysis using both human review and automated tools where appropriate. The precise tools used are not specified here but they should be selected based availability and the current state of the art.

- **NonFun.Platform**. Random Thoughts shall support both 64 bit Ubuntu Linux 20.04 LTS. The low level interface shall be sufficiently well documented so that third parties could reasonably create drivers or interface libraries for other systems.

- **NonFun.Deployment**. Software installers, with appropriate digital signatures, shall be provided for all supported systems in a manner that is standard for each system. All software shall be provided in source form with full documentation to enable (and encourage) review.

- **NonFun.Performance**. The hardware shall be capable of producing random 256 bit session keys at the rate of one key per minute or greater. The system shall cache random data in such a way that it can satisfy a request for up to eight 256 bit random keys "immediately" provided there is sufficient lead time.

  Memory consumption and CPU utilization of the system shall be "negligible." No network bandwidth shall be used by the system when it is in normal operation. The system shall consume one USB slave position on the host.

- **NonFun.Security**. The standard assumption shall be made that an attacker has full knowledge of all algorithms, software, and hardware that is used. It shall also be assumed that the attacker has no physical access to the hardware during its operation (specifically during the production of the n bits mentioned below), nor has any administrative access on the machine to which Random Thoughts is connected.

  The security characteristic of Random Thoughts is as follows: Given knowledge of n bits of random output, where n is in the range $0 \leq n \leq 2^{32+3}$, an attacker can predict the next bit with probability of no more than 0.5001.

- **NonFun.Scale**. The system shall be able to service multiple users of the host machine simultaneously, but note that it's overall performance as detailed above need not affected by the number of users. For example, two users may see an average rate of random number production of one 256 bit key every two minutes, etc. The generator only needs to service a single computer at a time.

- **NonFun.Documentation**. The following documents shall be made available to users for review:

  - *Design Documentation.* There shall be a document containing a detailed design of Random Thoughts describing how the various components of the system work. It shall include block diagrams, schematics, or code snippets, as appropriate.

  - *Test Documentation.* There shall be a document describing how Random Thoughts and its associated software was tested, along with mathematical justification of the techniques used.

  - *User Documentation.* There shall be a document describing how to install, verify, and use Random Thoughts. In addition this document shall describe the low level command language understood by the system so third parties can build their own interfaces if desired.

  - *Web Site.* There shall be a web site that describes Random Thoughts and that provides software updates and other up-to-the-minute information about the system.

- **NonFun.Formats**. The communication protocol between the hardware and software components of the system shall be proprietary but also documented.

- **NonFun.Internationalization**. All user interface elements and documentation shall be in English and nationalized for the United States. As an extension the system may make use of localization information provided by the host operating system.

- **NonFun.Environment**. Random Thoughts shall be able to operate in typical office conditions. No special environmental hardening is required. However, the addition of shielding from electromagnetic snooping is a likely extension. The hardware component of the system shall be no larger than 8" x 4" x 4" and ideally as small as feasible.

3 Architecture

In this section we describe the high level architecture of Random Thoughts. This includes both the architecture of the system and the architecture of the development environment.

3.1 System Architecture

The system architecture describes the structure of the deployed system. There are three aspects of the system to consider:

1. The hardware controller and noise generating hardware used as the source of randomness.
2. The command language used between the host computer and the hardware controller.
3. The driver and library software on the host computer.

Each of these aspects are described in separate subsections below.

3.1.1 Hardware Architecture

The hardware controller consists of a low cost, single board microcontroller based on an ARM architecture processor. Specifically, we use the STM32F4DISCOVERY with the STM32F407VG microcontroller (32 bit ARM Cortex M4). In normal operation the controller is connected to a host computer as a USB slave via a USB 2.0 (or higher) interface. A virtual serial port is created on the USB interface so the controller appears as a serially connected device from the point of view of the host computer.

Also connected to the controller, via it's general purpose I/O interface, is a supplementary board containing a white noise generator together with an appropriate analog to digital converter (ADC) to convert the generated noise into a stream of digital values. The word size of the ADC is 8 bits; if more bits are available, only the lower 8 bits are used.

A software task on the controller reads the ADC ten times per second and appends the 8 bit value read to a buffer of random data 2 KiB in size called the "entropy pool." The part of the pool that is filled with data is called the “active pool.” In normal operation, the pool is completely filled, and the active pool is the same as the whole pool. Initially, while the pool is filling (or later after some data has been extracted from it), the yellow LED on the controller is lite. When the pool is completely filled the green LED on the controller lights and the yellow LED turns off. The red LED on the controller is used to indicate an error condition.

Once the pool is filled, whenever a new value is read from the ADC, the following steps are taken:

1. The entire pool is hashed using SHA-256 and the resulting hash value is appended to the pool, causing the oldest 32 bytes to be discarded.
2. The new value from the ADC is appended to the pool, causing the oldest byte to be discarded.

Thus 33 bytes are discarded from the pool for each new byte added. However, those bytes participate in the hash that gets appended (and that hash is further hashed again, etc.).

A second task on the controller interacts with the USB interface, receiving commands from the host computer, interpreting those commands, and acting on them. This “interface task” prepares random data from the entropy pool in a separate buffer area with a size equal to the amount of random data requested (which might be smaller or larger than the size of the active pool). The task takes data from the pool, oldest data first, thereby reducing the active pool size, waiting for more data to appear in the pool if necessary to completely fill the result buffer. Note that when the pool is less than completely filled, the hashing process described above is temporarily suspended until the pool is filled.

3.1.2 Command Language

This subsection describes the command language used between the host computer and the hardware controller. These commands and their corresponding responses flow over a serial connection and, for convenience, are entirely ASCII text. The generator hardware never sends an unsolicited response. Furthermore, every command sent by the PC elicits a response of some kind (even if only a success code). This behavior facilities testing the generator hardware using a standard terminal emulation program.

Every command and every response is terminated by a CR/LF pair. Lines will never exceed 80 characters in length. If a command line is longer than 80 characters, the extra characters are ignored. Unless otherwise mentioned below, commands and responses are only one line. Unless otherwise mentioned below, responses are returned in a ”timely” manner. Commands are case insensitive.

```
Command: CHECK
Purpose:
  Verify proper operation of the generator hardware

Possile responses:
   "!OK"
   "!OK: STATISTICS UNKNOWN"
   "!HARDWARE FAILURE: [message]"
   "!BAD STATISTICS"
   "!SOFTWARE FAILURE: [message]"
   "!UNKNOWN ERROR."
```

Note that support for hardware evaluation of statistics is not required. The generator hardware should probably provide a way to tell the calling software if it does statistical analysis or not. How that might be done is currently unspecified.

```
Command: GET IMMEDIATE n
Purpose:
  Obtain n bits of random data "immediately" (without blocking)

Possible responses:
  A string of '0' and '1' characters of size n bits
  One of the same responses to the CHECK command
  "!INSUFFICIENT DATA: n"
```

Errors are indicated as for the CHECK command. The !INSUFFICIENT DATA response is sent if there is not enough entropy to satisfy the request immediately. The value of n returned by !INSUFFICIENT DATA is the number of bits currently in the buffer. A GET IMMEDIATE with that value of n or less will definitely succeed. Use GET IMMEDIATE 0 to elicit an insufficient data response regardless of the amount of data available, and to thus learn how much data is available.

If ¡n¿ is greater than 80, the data will be delivered in multiple lines. The precise number of lines is unspecified, but no line will be blank. The value of n in the GET IMMEDIATE command must be greater than or equal to zero.

```
Command: GET n
Purpose:
  Obtain n bits of random data, possibly blocking

Possible responses:
  A string of '0' and '1' characters of size n bits
  One of the same responses to the CHECK command
```

The returned data might be possibly split over several lines as described above, until a total of n bits are returned. If all the bits are not yet available, the generator hardware will wait until they are, either returning the bits as they are produced (meaning in spurts) or in a single block once they are all produced. The generator hardware will accept no other commands while it is generating data. However, the unit may output an error response (as defined for the CHECK command) on a line by itself at any time during the random output.

3.1.3 Driver/Library Architecture

There are two parts to the software that runs on the computer host. The first part is a kernel level driver that presents a device file such that when the device file is read, random binary data is returned following the usual semantics for reading devices. The driver should also provide a "raw" mode that allows applications to write commands directly in the command language described above and read the corresponding responses. This allows applications to be built that interact directly with the controller hardware for special purposes.

The second software component is a library that uses the device in its raw mode but presents a simple API to the application for reading random binary data. That that respect the library provides the same interface as the driver, except in application space. However, this allows the library to also do significant statistical analysis on the data stream before returning it to the application. That analysis can be used to monitor the statistical properties of the data and alert the application if any biases are found. The goal of the library is thus to keep the statistical analysis out of kernel space.

3.2 Development Environment Architecture

Since the development is done using SPARK/Ada, AdaCore's GPS is the primary development environment. That environment includes the ability to program and debug code on the STM32F4DISCOVERY board. See Section 5 for more information about the test plan for the system.

For convenience, the development platform is the same as the initial target platform: Ubuntu Linux 20.04. Driver development takes place inside a Virtual machine to avoid the possibility of destablizing the development platform while debugging the driver.

4 Design

In this section we describe the design of Random Thoughts, including various design trade-offs made. This information is primarily of interest to developers looking to enhance Random Thoughts. Developers interested in only using Random Thoughts do not need to read this section unless they are interested in the rationale behind some of the design decisions made.

5 Test Plan

In this section we describe the test plan for Random Thoughts.

6 Manual

This section contains the user's manual for Random Thoughts. The intended audience of this chapter is both application developers and application users.

Bibliography