Category:Security

Contents

Stable: 17.06.2020 - 15:27 / Revision: 16.01.2020 - 08:05

Category:Security

Stable: 17.06.2020 - 10:27 | Revision: 16.01.2020 - 08:08

A quality version of this page, approved on *17 June 2020*, was based off this revision.

This category groups together all articles and subcategories related to the Linux® **security** software frameworks such as: hardware random, TSS, and so on.

Subcategories

This category has only the following subcategory.

- Crypto (4 P)

Pages in category "Security"

The following 3 pages are in this category, out of 3 total.

- Hardware random overview
- How to control a RNG in userspace
- RNG device tree configuration

Stable: 17.02.2021 - 19:55 / Revision: 17.02.2021 - 19:55

A quality version of this page, approved on *17 February 2021*, was based off this revision.

This article gives information about the Linux® hardware random framework.

Contents

1 Article Purpose

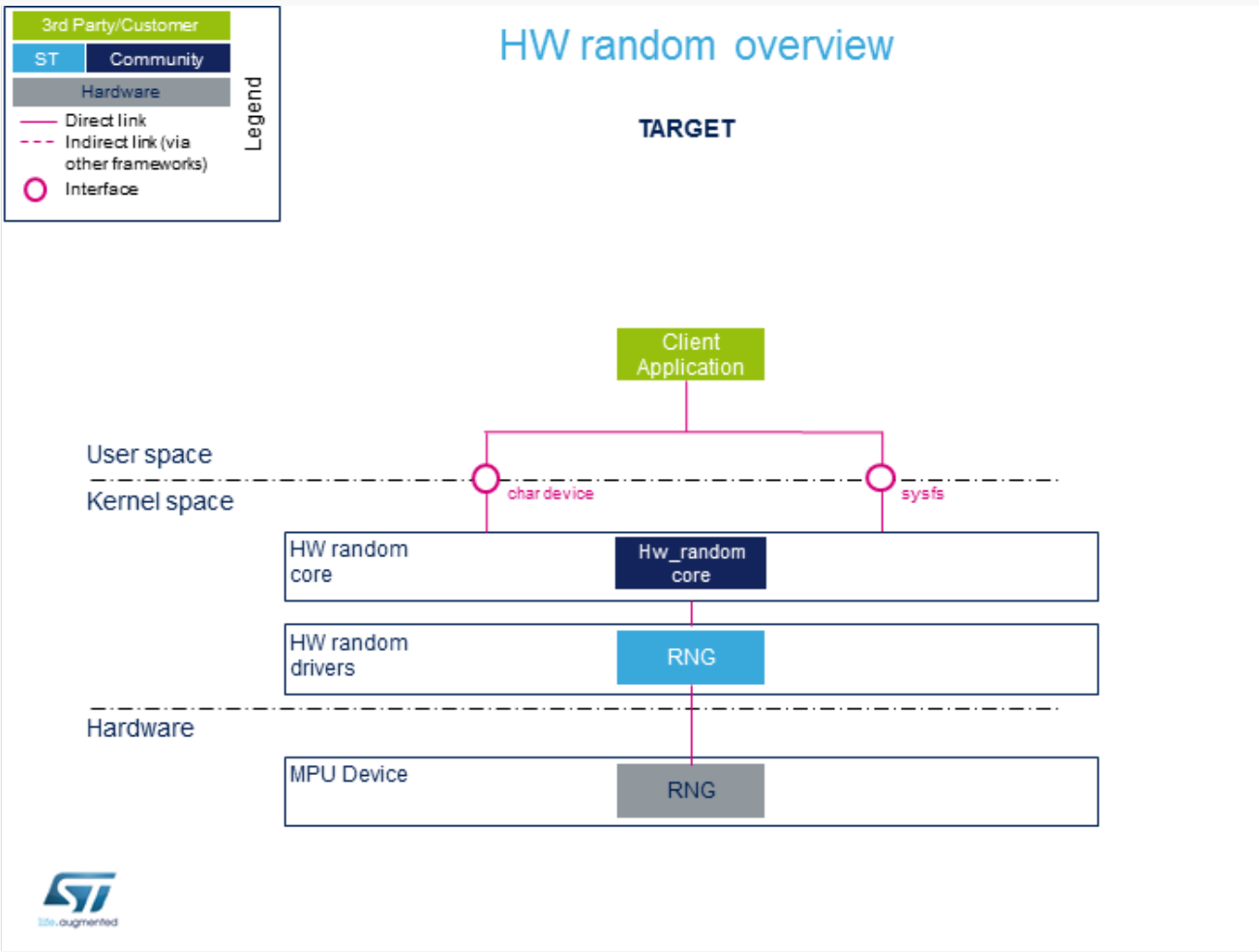
This article gives information about the hardware random (HWRNG) framework.

2 Framework purpose

The Hardware random framework is integrated in the kernel. It provides access to RNG peripherals and focuses on supporting the hardware number generator.

3 System overview

The HW random framework allows retrieving random numbers in userland.



Hw Random

3.1 Component description

- **HW random core** (Kernel space)

Generic interface in kernel space. This layer is in charge of creating the character device (char device) and sysfs to access hw_random.

- **RNG** (Kernel space)

Hardware random Linux® drivers handling the HW blocks.

- **RNG** (Hardware)

HW blocks handling the RNG peripheral.

3.2 API description

The Hardware random framework uses char device API[1] ioctl operations. For additional information, refer to:

- sysfs interface.
- Kernel Documentation directory[2]

4 Configuration

4.1 Kernel configuration

The Hardware random support is activated by default in ST deliveries. No specific configuration is required apart from enabling or disabling peripheral support using Linux® Menuconfig tool. Refer to Menuconfig or how to configure kernel and select:

```
[*] Device Drivers --->
  [*] Character devices --->
    [*] Hardware Random Number Generator Core support --->
      [*] STMicroelectronics STM32 random number generator
```

4.2 Device tree configuration

DT configuration can be done thanks to the STM32CubeMX.

A detailed device tree configuration is described in RNG device tree configuration.

5 How to use the framework

The framework provides external interfaces from userland : How to control RNG.

5.1 How to use from char device

The community tool for using Hardware random framework is rng_tools[3] which provides a complete set of utilities related to random number generators:

- **rngd**: runs a background daemon that opens /dev/hwrng file (default) to connect and retrieve random numbers.
- **rngtest**: runs different tests that check the entropy and verify the compliance regarding FIPS 140-2 standard.

5.2 How to use from sysfs

Available devices compatible with Hardware framework can be listed using sysfs commands:

```
Board $> cat /sys/class/misc/hw_random/rng_available
stm32-rng
```

The selected device is shown here:

```
Board $> cat /sys/class/misc/hw_random/rng_current
stm32-rng
```

To select a different device:

```
Board $> echo "stm32-rng"> /sys/class/misc/hw_random/rng_current
```

6 How to trace and debug the framework

Light information on the framework can be accessed by using sysfs.

By default, the framework does not provide any specific debug output or dynamic debugging tool.

7 Source code location

Hardware random drivers and framework are available here[4].

8 To go further

Code examples are directly available from rng-tools[3] github.

9 References

- https://bootlin.com/doc/legacy/accessing-hardware/accessing-hardware.pdf
- Documentation/admin-guide/hw_random.rst
- 3.03.1 Rng_tools source code
- drivers/char/hw_random , Hw_random sources

Stable: 03.02.2020 - 08:42 / Revision: 03.02.2020 - 08:27

A quality version of this page, approved on *3 February 2020*, was based off this revision.

Contents

1 Purpose

Hardware random framework offers the interface to control RNG devices from userspace.
This article shows two ways to control a RNG in userspace:

- using /dev/random command to generate a random number
- using rng-tools to validate the RNG

2 RNG control through /dev/random

/dev/random is a special file that can be used to generate random numbers based on a pseudo-random generator. It uses noise collected from device drivers and hardware random sources to generate data. od (octal dump) command is used to extract the number of bytes and display the decimal number.
Ex: - Random number (0 - 255):

```
Board $> od -An -N1 -i /dev/random
        172
```

- Random number (0 - 65535):

```
Board $> od -An -N2 -i /dev/random
        20041
```

3 RNG control through rng-tools

rng_tools[1] is a set of tools related to random number generation.
rng-tools will connect to the hardware random number generator through /dev/hwrng. rngtest is a basic test that checks data using FIPS 140-2 tests[2] which is a security requirement test for cryptographic module compliance.

```
Board $> rngtest -c 100 </dev
/hwrng

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: bits received from input: 2000032
rngtest: FIPS 140-2 successes: 100
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=33.154; avg=33.656; max=34.217) Kibits/s
rngtest: FIPS tests speed: (min=21.193; avg=23.180; max=23.403) Mibits/s
rngtest: Program run time: 58114432 microseconds
```

It is normal for any random generator to fail in small number of tests, but failures must not exceed around 10.

4 References

- https://git.kernel.org/pub/scm/utils/kernel/rng-tools/rng-tools.git/
- https://en.wikipedia.org/wiki/FIPS_140-2

Stable: 13.05.2020 - 08:40 / Revision: 13.05.2020 - 08:39

A quality version of this page, approved on *13 May 2020*, was based off this revision.

Contents

1 Article purpose

This article explains how to configure the **RNG** internal peripheral when it is assigned to the Linux® OS. In that case, it is controlled by the Hardware random framework.

The configuration is performed using the device tree mechanism that provides a hardware description of the RNG peripheral, used by the STM32 RNG Linux driver.

If the peripheral is assigned to another execution context, refer to How to assign an internal peripheral to a runtime context article for guidelines on peripheral assignment and configuration.

2 DT bindings documentation

The *RNG* is represented by the *STM32 RNG device tree bindings*[1]

3 DT configuration

This hardware description is a combination of the **STM32 microprocessor** device tree files (*.dtsi* extension) and **board** device tree files (*.dts* extension). See the Device tree for an explanation of the device tree file split.

STM32CubeMX can be used to generate the board device tree. Refer to How to configure the DT using STM32CubeMX for more details.

3.1 DT configuration (STM32 level)

The RNG node is declared in stm32mp151.dtsi[2]. It describes the hardware register address, clock and reset.

```
 rng1: rng@54003000 {                                          Comments
         compatible = "st,stm32-rng";
         reg = <0x54003000 0x400>;                              --> Register location
and length
         clocks = <&scmi0_clk CK_SCMI0_RNG1>;
         resets = <&scmi0_reset RST_SCMI0_RNG1>;
         status = "disabled";
 };
```

Warning

This device tree part is related to STM32 microprocessors. It must be kept as is, without being modified by the end-user.

3.2 DT configuration (board level)

This part is used to enable the RNG used on a board which is done by setting the **status** property to **okay**.

A clock-error-detect property is available depending the clock choosen for entropy. It can be enabled to manage the clock detection.

3.3 DT configuration examples

```
&rng1 {
        status = "okay";
        clock-error-detect;
};
```

4 How to configure the DT using STM32CubeMX

The STM32CubeMX tool can be used to configure the STM32MPU device and get the corresponding platform configuration device tree files.
The STM32CubeMX may not support all the properties described in the above DT bindings documentation paragraph. If so, the tool inserts **user sections** in the generated device tree. These sections can then be edited to add some properties and they are preserved from one generation to another. Refer to STM32CubeMX user manual for further information.

5 References

Please refer to the following links for additional information:

- Device tree bindings
- STM32MP151 device tree