

A man with a beard and a small robot on his head is looking at a tablet. The background is a blurred office setting with windows and plants.

arm

Securing IoT Applications with mbed TLS

Hannes Tschofenig

Part#3:
Random Number Generators (RNGs)

July 2018
Munich

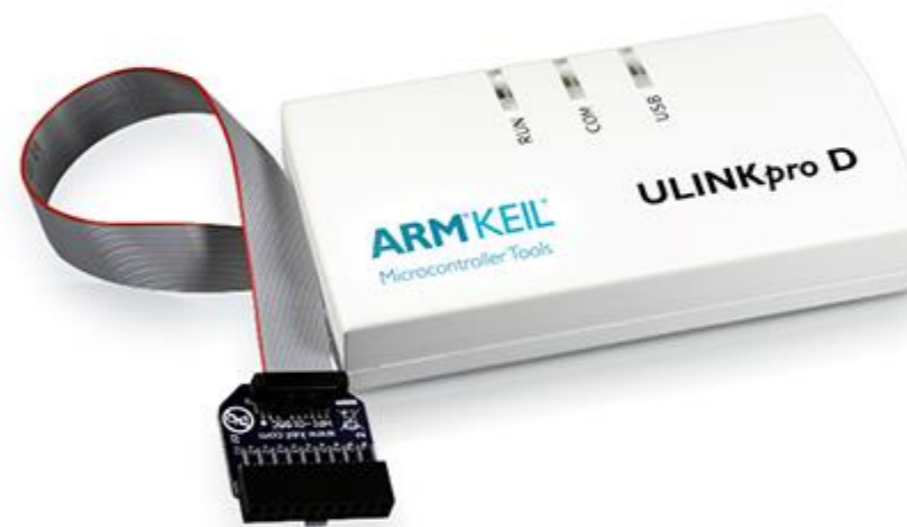
Why should we talk about RNGs?

- The importance of random numbers for security protocols was already discussed in the [first webinar](#).
- As a reminder, watch Paul Bakker talk about “[Entropy Requirements in IoT](#)” on the [Arm mbed Youtube channel](#).
- Example code so far did not use an RNG because of the following “development only” settings in the mbedTLS_config.h file:
 - `#define MBEDTLS_NO_DEFAULT_ENTROPY_SOURCES`
 - `#define MBEDTLS_TEST_NULL_ENTROPY`
- A compile-time warning is the result:
`THIS BUILD IS *NOT* SUITABLE FOR PRODUCTION USE`
- We will enhance our code to make use of a hardware-based random number generator.

Our Hardware

Keil Evaluation Board

- We are still using the [Keil MCBSTM32F400 Evaluation Board](#), which uses the [STM32F407IG](#) MCU.
- This MCU uses an Arm Cortex M4 processor and offers a hardware-based random number generator.



RNG Availability?

- ST alone offers 410 MCUs with RNG support. This is only ST and other vendors have similar offers.

The screenshot shows the ST MCU Finder interface. On the left, the 'Peripheral Choice' menu has 'TRNG' selected and highlighted with a red box. A red arrow points from the 'TRNG' selection to a red circle around the text 'MCUs List: 410 items' in the main results area. The main area displays a table of MCUs with columns for Part No, Reference, Marketing Status, Unit Price for 10KU (US\$), Board, Package, Flash, RAM, IO, and Freq. A message 'Datasheet document is not available' is visible above the table.

Part No	Reference	Marketing Status	Unit Price for 10KU (US\$)	Board	Package	Flash	RAM	IO	Freq.
STM32F215RE	STM32F215RETx	Active	4.721		LQFP64 (10x10mm)	512 kBytes	128 kBytes	51	120 MHz
STM32F215RG	STM32F215RGTx	Active	5.646		LQFP64 (10x10mm)	1024 kBytes	128 kBytes	51	120 MHz
STM32F215VE	STM32F215VETx	Active	5.091		LQFP100 (14x14mm)	512 kBytes	128 kBytes	82	120 MHz
STM32F215VG	STM32F215VGTx	Active	6.017		LQFP100 (14x14mm)	1024 kBytes	128 kBytes	82	120 MHz
STM32F215ZE	STM32F215ZETx	Active	5.808		LQFP144 (20x20mm)	512 kBytes	128 kBytes	114	120 MHz
STM32F215ZG	STM32F215ZGTx	Active	6.734		LQFP144 (20x20mm)	1024 kBytes	128 kBytes	114	120 MHz
STM32F217IE	STM32F217IEHx	Active	6.41		UFBGA176 (10x10mm)	512 kBytes	128 kBytes	140	120 MHz
STM32F217IETx		Active	6.41		LQFP176 (24x24mm)	512 kBytes	128 kBytes	140	120 MHz
STM32F217IG	STM32F217IGHx	Active	7.336	STM3221G-EVAL	UFBGA176 (10x10mm)	1024 kBytes	128 kBytes	140	120 MHz
STM32F217IGTx		Active	7.336	STM3221G-EVAL	LQFP176 (24x24mm)	1024 kBytes	128 kBytes	140	120 MHz
STM32F217VE	STM32F217VETx	Active	5.461		LQFP100 (14x14mm)	512 kBytes	128 kBytes	82	120 MHz
STM32F217VG	STM32F217VGTx	Active	6.387		LQFP100 (14x14mm)	1024 kBytes	128 kBytes	82	120 MHz
STM32F217ZE	STM32F217ZETx	Active	6.179		LQFP144 (20x20mm)	512 kBytes	128 kBytes	114	120 MHz
STM32F217ZG	STM32F217ZGTx	Active	7.104		LQFP144 (20x20mm)	1024 kBytes	128 kBytes	114	120 MHz
STM32F405OE	STM32F405OEYx	Active	4.371		WLCSP90	512 kBytes	192 kBytes	72	168 MHz
STM32F405OG	STM32F405OGYx	Active	5.297		WLCSP90	1024 kBytes	192 kBytes	72	168 MHz
STM32F405RG	STM32F405RGTx	Active	5.829		LQFP64 (10x10mm)	1024 kBytes	192 kBytes	51	168 MHz
STM32F405VG	STM32F405VGTx	Active	6.2		LQFP100 (14x14mm)	1024 kBytes	192 kBytes	82	168 MHz
STM32F405ZG	STM32F405ZGTx	Active	6.662		LQFP144 (20x20mm)	1024 kBytes	192 kBytes	114	168 MHz
STM32F407IE	STM32F407IEHx	Active	6.338		UFBGA176 (10x10mm)	512 kBytes	192 kBytes	140	168 MHz
STM32F407IETx		Active	6.338		LQFP176 (24x24mm)	512 kBytes	192 kBytes	140	168 MHz
STM32F407IG	STM32F407IGHx	Active	7.264	STM3240G-EVAL	UFBGA176 (10x10mm)	1024 kBytes	192 kBytes	140	168 MHz
STM32F407IGTx		Active	7.264	STM3240G-EVAL	LQFP176 (24x24mm)	1024 kBytes	192 kBytes	140	168 MHz
STM32F407VE	STM32F407VETx	Active	5.644		LQFP100 (14x14mm)	512 kBytes	192 kBytes	82	168 MHz
STM32F407VG	STM32F407VGTx	Active	6.57	STM32F4DISCOVERY	LQFP100 (14x14mm)	1024 kBytes	192 kBytes	82	168 MHz
STM32F407ZE	STM32F407ZETx	Active	6.107		LQFP144 (20x20mm)	512 kBytes	192 kBytes	114	168 MHz
STM32F407ZG	STM32F407ZGTx	Active	7.033		LQFP144 (20x20mm)	1024 kBytes	192 kBytes	114	168 MHz
STM32F413CG	STM32F413CGUx	Active	4.796		UQFPN48 (7x7mm)	1024 kBytes	320 kBytes	36	100 MHz

Accessing the RNG

STM32F407IG RNG

- The [STM32F407IG product webpage](#) contains pointers to reference manuals, including [RM0090](#).
- Section 24 describes the RNG functionality. In a nutshell:
 - The RNG delivers a 32-bit random number with every access.
 - The peripheral is attached to the AHB2 bus and is memory mapped.
 - The RNG can be operated in two modes; in an Interrupt-driven mode and in a polling mode.
 - There are three registers dedicated to the use with the RNG, namely (1) a control register (CR), a status register (SR), and a data register (DR).

RNG Register Map

Offset	Register name reset value	Register size																															
		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	RNG_CR 0x00000000	Reserved																										IE	RNGEN	Reserved			
0x04	RNG_SR 0x00000000	Reserved																										SEIS	CEIS	Reserved	SECS	CECS	DRDY
0x08	RNG_DR 0x00000000	RNDATA[31:0]																															

Steps for accessing the RNG

Enable RNG peripheral via RNGEN bit in the RNG_CR register

Note: The first random number generated after setting the RNGEN bit should not be used, according to FIPS (Federal Information Processing Standard Publication) PUB 140-2.

RNG working

Check for errors and random number availability

Perform three checks via the RNG_SR register:

1. Is there a seed error? (SEIS bit)
2. Is there a clock error? (CEIS bit)
3. Is a random number ready? (DRDY bit)

No error and data is ready

Note: Each generated random number has to be compared with the previously generated number. The test fails if any two compared numbers are equal.

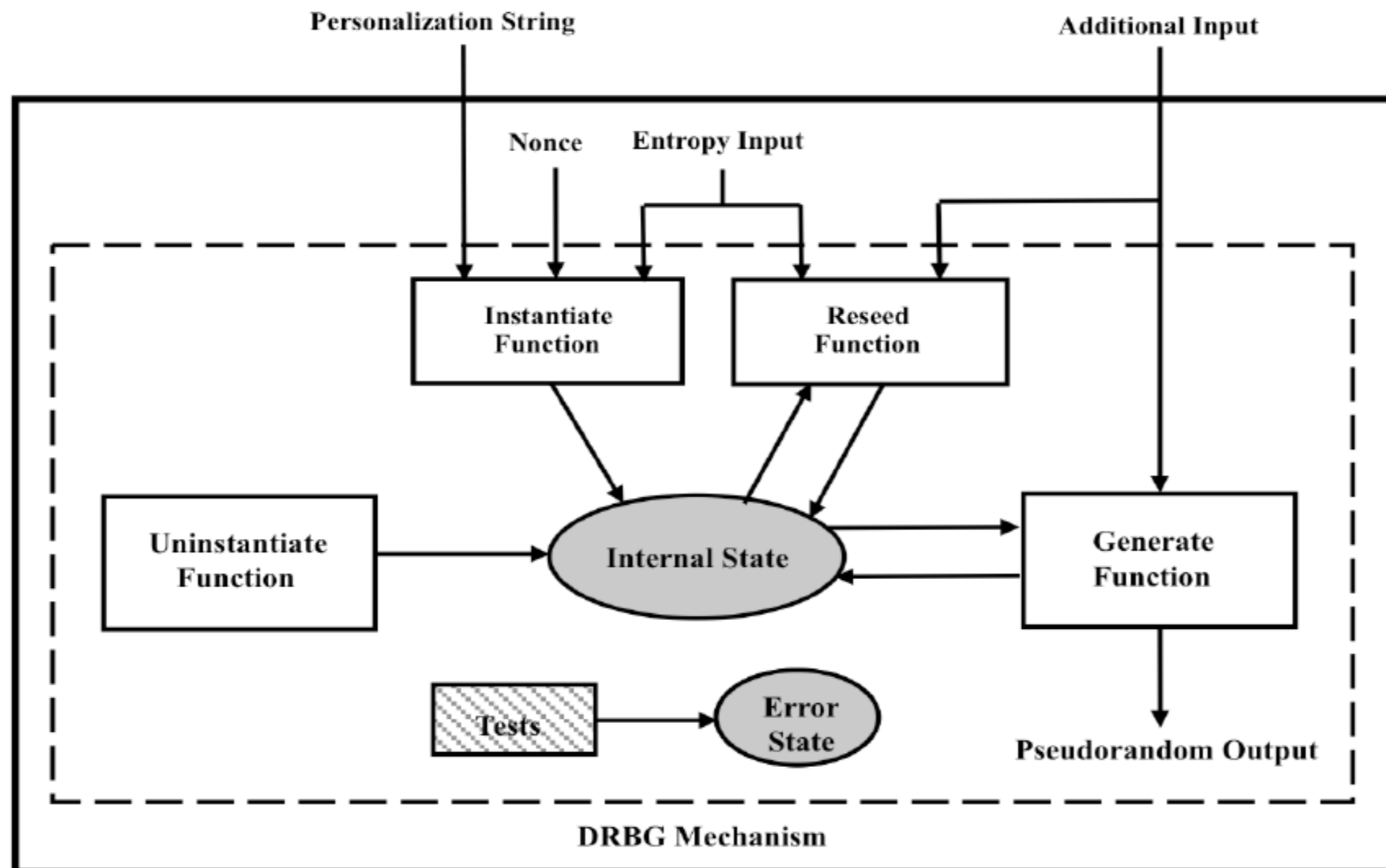
Read the random number

Hands-On: Testing the RNG Hardware

NIST SP 800-90A rev 1

Deterministic random bit generator (DRBG)

- [NIST SP 800-90A rev1](#) specifies the generation of random bits using deterministic methods based on either hash functions or block cipher algorithms.



Mbed TLS Entropy & DRBG API

Entropy accumulator implementation

```
// Entropy context structure  
mbedtls_entropy_context entropy;
```

```
// Entropy context initialization  
mbedtls_entropy_init(...)
```

```
// Adds an entropy source to poll  
mbedtls_entropy_add_source(...)
```

Add our
hardware-based
RNG

```
// Free entropy context  
mbedtls_entropy_free(...)
```

DRBG Mechanism Based on Block Ciphers

```
// CTR_DRBG context structure  
mbedtls_ctr_drbg_context ctr_drbg;
```

```
// Initializes the CTR_DRBG context  
mbedtls_ctr_drbg_init(...)
```

```
// Seed the CTR_DRBG  
mbedtls_ctr_drbg_seed(...)
```

Link with
entropy context

```
// Turns prediction resistance on or off  
mbedtls_ctr_drbg_set_prediction_resistance(...)
```

```
// Clears CTR_DRBG context  
mbedtls_ctr_drbg_free(...)
```

```
// Configure Mbed TLS with RNG callback  
mbedtls_ssl_conf_rng(...)
```

Link with
CTR_DRBG context

Hands-On: Integrating the RNG functionality

Summary

- Random numbers are important for security.
- Pick the appropriate hardware for your task. For our security application we need a hardware-based random number generator.
- Use your favorite MCU and follow the steps to integrate your RNG into mbed TLS.