EFM32

... the world's most energy friendly microcontrollers

Fortuna Cryptographically Secure PRNG

AN0806 - Application Note

Introduction

This application note describes how to get started running the Fortuna cryptographically secure pseudo random number generator (PRNG) on the EFM32 family of microcontrollers from Silicon Labs.

This application note includes:

- This PDF document
- Source files (zip)
  - Example C-code
  - Multiple IDE projects

1 Fortuna Overview

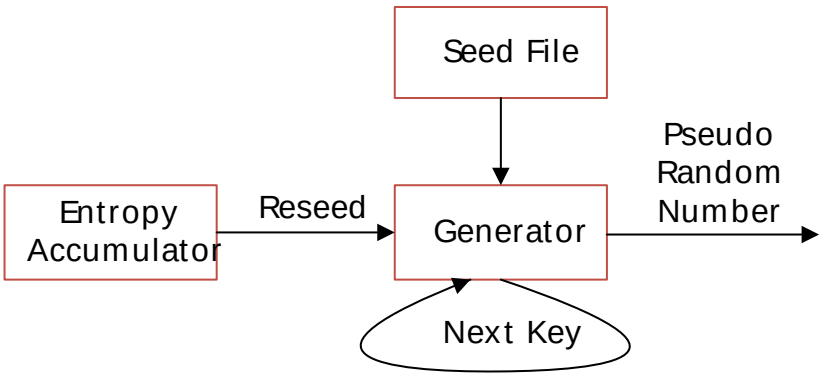
Fortuna is a cryptographically secure PRNG by Bruce Schneier and Niels Ferguson [1]. The design leaves some choices open to implementers, and the specific implementation included in this application note is based on "Implementation of Fortuna on EFM32 Microcontroller and Differential Power Analysis" by Abreham Emishaw Delelegn [2].

Being cryptographically secure means that Fortuna can be used in many cryptographic applications including digital signatures, authentication protocols, and data encryption. Its robust design against cryptographic threats and the capability of the algorithm to reuse existing modules makes it useful for embedded applications.

Fortuna is composed of the following main components:

- The generator itself, which once seeded will produce an indefinite quantity of pseudo random data.
- The entropy accumulator, which collects genuinely random data from various sources and uses it to reseed the generator when enough new randomness has arrived.
- The seed file, which stores enough state to enable the computer to start generating random numbers as soon as it has booted.

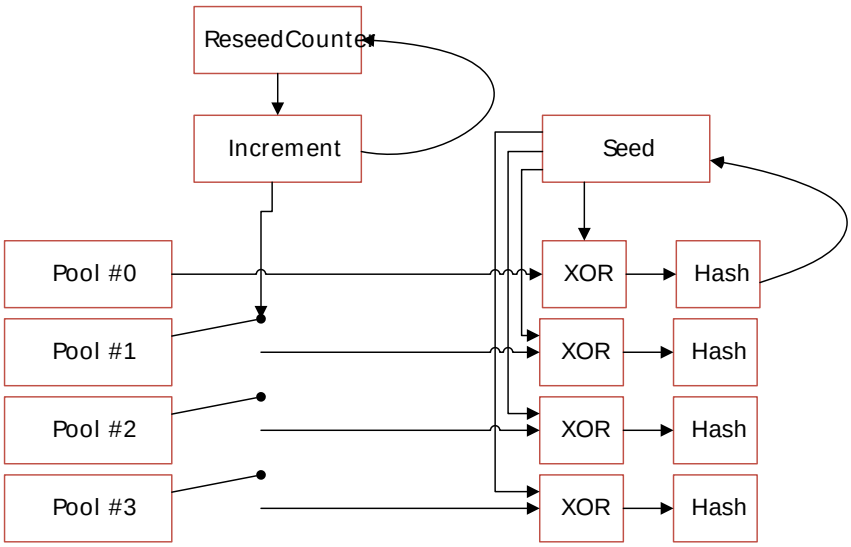
Figure 1.1. Fortuna Block Diagram



1.1 Why Fortuna?

Fortuna is an improvement of *Yarrow* which used entropy estimators and various heuristic rules in order to protect the internal states of the PRNG against attackers. The problem with the Yarrow approach is to know how much entropy to collect before using it to reseed the PRNG.

Fortuna solves this problem by accumulating entropy randomly from various entropy sources and distributes the entropy evenly to **32 entropy pools** in a round-robin fashion. The reseed algorithm uses only a selection of the pools when reseeding the PRNG. The pool selection depends on an internal 32-bit **reseed counter** which is incremented for each reseed operation. If 2^i (where i is the pool number) is a divisor of the reseed counter (without remainder), then the pool is selected in the new seed. Thus pool #0 is used for every reseed, pool #1 is used every second reseed, ..., and pool #31 is used every 2^31 reseed which is usually years in time given a typical reseed period. This provides for forward secrecy since randomly generated entropy is distributed evenly over long bit extractions. An attacker may know about some entropy sources, but as long as he does not control every entropy source there will always be a pool that defeats him. Fortuna is designed for long lived systems with periodic reseeding. Please refer to [1] and [2] for more details.



1.2 Fortuna on EFM32

This application note was developed and tested on the following 3 EFM32 kits, but the source code should compile and run on any EFM32 that incorporates an AES peripheral.

- EFM32TG_STK3300
- EFM32GG_STK3700
- EFM32GG_DK3750

The source code is based on the implementation of Abreham Emishaw Delelegn in [2], and has been reworked in order to simplify the user interface including the API and the configuration interface. Entropy is generated by sampling the internal temperature sensor of the EFM32 with the ADC. [2] concludes that the ADC samples has entropy of just above one bit per sample, indicating that at least the least significant bit is truly random. However the example implementation of the entropy source in this application note feeds the whole 32-bit ADC sample to the Fortuna PRNG. This should not diminish the quality of the output of the Fortuna PRNG because the sample is mixed and hashed with previous entropy stored internally in the Fortuna PRNG. However the user is encouraged to implement additional entropy sources from random input events or similar.

The Fortuna PRNG for EFM32 has been tested and verified with the following programs:

- **'ent'**

  A Pseudorandom Number Sequence Test Program by John Walker, January 28th, 2008,

  http://www.fourmilab.ch/random/

- **'sts-2.1'**

  NIST Statistical Test Suite,

  http://csrc.nist.gov/groups/ST/toolkit/rng/documentation_software.html

2 User Interfaces

This chapter gives a short description of the EFM32 Fortuna user interfaces, including the API and the configuration user interafce.

2.1 API

This section gives a short description of the EFM32 Fortuna API. Please refer to the function headers in the file fortuna.c for a more detailed description of each API function. The API is declared in the file fortuna.h, and consists of the functions listed below. All functions return an integer in order to report the status of the function. The possible status codes are defined in fortuna.h.

- **int FORTUNA_Init (void)**

  Reads the seed file and initializes the internal resources.

- **int FORTUNA_AddRandomEvent (int sourceId, uint32_t data)**

  Add entropy to the entropy pools. This function is called randomly by the included entropy sources when there is random data available. The user is responsible for making sure the entropy sources generate entropy and call this function in order to feed the PRNG with fresh entropy while Fortuna is running.

- **int FORTUNA_RandomDataGet (void* data, int numBytes)**

  Returns a number of pseudo random data bytes in the data buffer.

- **int FORTUNA_Rand (uint32_t* pseudoRandomNumber)**

  Returns a 32-bit pseduo random number.

- **int FORTUNA_Shutdown (void)**

  Clear and release all state variables and resources used by Fortuna.

2.2 Configuration User Interface

This section gives a short description of the EFM32 Fortuna configuration interface. The EFM32 Fortuna configuration may be customized by the user in order to satisfy specific application requiremets. The configuration interface is defined in the file fortuna.h, and consists of the following definitions:

- **NUM_POOLS**

  Number of pools used for entropy acquisition. The default is 32 pools. The maximum is 32 pools.

- **MAX_ENTROPY_SOURCES**

  Maximum number of entropy sources allowed to feed Fortuna with entropy via the FORTUNA_AddRandomEvent function. The default is one source which is the ADC controller sampling the internal temperature sensor. The user is encouraged to implement additional entropy sources.

- **RESEED_PERIOD_TICKS**

  Minimum time between reseed events in ticks. The default is 100 ticks which corresponds to 100 milliseconds, because the tick counter is incremented for each system tick in the SysTickHandler implemented in fortuna_main.c.

- **RESEED_LIMIT**

  Maximum number of pseudo random data bytes between reseed events. The default is 1Mbytes. This parameter controls the maximum amount of data can be generated for each seed. Fortuna should reseed with fresh entropy in order to stay cryptographically secure.

- **SEED_FILE_UPDATE_PERIOD_TICKS**

  Minimum time between seed file updates in ticks. The default is 10000 ticks which by default corresponds to 10 seconds, because the tick counter is incremented for each system tick in the SysTickHandler implemented in fortuna_main.c.

  NOTE: The seed file is written to the Flash memory device which supports only a limited number of erase and write cycles. Therefore the user should consider to reduce the update period in order to extend the life time of the flash memory device.

- **SEED_FILE_FLASH_ADDRESS**

  Base address of the seed file in flash memory. The default is the last page in flash memory. The user should make sure the selected page is not used for other purposes in the application. In case of conflicts and the application requires a significant amount of flash data storage, it is recommended to consider to use a wear leveling file system, e.g. as described in the application note AN0019 EEPROM Emulation.

- **FORTUNA_MAX_DATA_SIZE**

  Maximum number of random data bytes that can be requested when calling the FORTUNA_RandomDataGet function. The FORTUNA_RandomDataGet function reseeds for each call. Therefore this parameter controls the maximum amount of data can be requested for each seed, i.e. fresh entropy in the PRNG. The default is 1MB.

3 Functional Operation

This chapter quickly describes how the user should operate Fortuna on the EFM32. The following description refers to the source code and especially the file called fortuna_main.c.

3.1 Initialization

First of all the EFM32 MCU and required peripherals should be initialized. In the demo program included in this application note we enable the Serial Wire Output (SWO) interface (part of the serial wire debug interface) in order to direct stdout to the SWO interface. On EFM32GG_DK3750 we redirect the stdout to the serial port (for details see #ifdef EFM32GG_DK3750 statements in fortuna_main.c).

Fortuna will need to keep track of time, and in the EFM32 Fortuna we provide a function called SysTickGet which is implemented in fortuna_main.c. Fortuna calls this function which returns a tick counter which is incremented every millisecond (by default) in SysTick_Handler. The tick rate is setup in EFM32_Init.

The default operation of Fortuna on EFM32 is to use the ADC as entropy source. Everything that is required for the ADC to generate samples with entropy is setup in the function AdcRandomInitialize in fortuna_adc.c.

In order to start using Fortuna on the EFM32 the FORTUNA_Init function must be called. This will initialize the internal resources and initialize the seed from the seed file if present. If the seed file is not present, Fortuna needs to be initially seeded from the entropy pools. The entropy pools will be empty when FORTUNA_Init returns, so the user is responsible for setting up and starting the entropy sources which should feed Fortuna with real random data by calling the FORTUNA_AddRandomEvent function. In the demo program, the function AddInitialEntropy is called to fill the entropy pools initially.

3.2 Adding Entropy

In order for Fortuna to run properly the entropy pools must be updated with fresh entropy at an appropriate rate. EFM32 Fortuna incorporates 32 entropy pools of 32 bytes each, or rather 8 32-bit elements. The function FORTUNA_AddRandomEvent receives 32 bits of fresh entropy for each call and hashes the 32 bits of fresh entropy into one of the 8 32-bit elements of the next pool in a round robin fashion. This way, in order for all pools to be updated inside each reseed period (100ms by default) the FORTUNA_AddRandomEvent must be called 32 times for each reseed period. Pool#0 is always used to generate a new seed. Whether the other pools are included in the reseed operation depends on the reseed counter. Therefore, it is recommended to call FORTUNA_AddRandomEvent at least 32 times in every reseed period in order to make sure pool#0 is updated. The reseed period is defined in fortuna.h, and is 100ms by default which is considered quite short and safe. The user can consider to change the reseed period depending on the frequency of fresh entropy and the security requirements of the application. In general the reseed period should be kept short in order to reduce the amount of pseudo random data generated per seed which can be traced by attackers. On the other hand, if the addition of fresh entropy is infrequent resulting in that there is no fresh entropy in the new seed, the value of reseeding is decreased. The user can reduce the number of entropy pools in order to reduce the entropy hunger of Fortuna and call FORTUNA_AddRandomEvent less frequently. The user is also encouraged to implement additional entropy sources in order to make it more difficult for attackers to get information about the entropy being provided to Fortuna.

3.3 Requesting Pseudo Random Data

The user can call either FORTUNA_RandomDataGet or FORTUNA_Rand in order to request pseudo random data from Fortuna on the EFM32. FORTUNA_RandomDataGet generates a buffer of pseudo random data of the size requested by the user. FORTUNA_Rand just calls FORTUNA_RandomDataGet with the request size of 32 bits and returns the 32 bits of pseudo random data to the user.

3.4 Shutdown

When the application stops using Fortuna, the user should call FORTUNA_Shutdown in order to update the seed file the finally and clear all internal state data structures.

4 Getting Started

This chapter describes how to compile and run the Fortuna source code accompanying this application note.

4.1 Quick Start

Connect your favorite EFM32 kit to your host development computer. In order for this quick start routine to work the kit must be one of EFM32TG_STK3300, EFM32GG_STK3700 or EFM32GG_DK3750. Additional kits should easily be supported but may need some extra setup in order to work.

From Simplicity Studio the user can select the Fortuna application note, then click on the "Source" button which will automatically open the Fortuna source code in the preconfigured IDE. From within the IDE the user can easily build and run the Fortuna demo program. For instance, In IAR, click the green arrow button labeled "Download and Debug" which will compile, download and break at the start of the main program ready to debug. The user can now run or step through the program line by line. Please refer to the documentation of the IDE for details on how to operate the IDE.

4.2 Source Code Overview

The EFM32 Fortuna source code is distributed in 3 folders named **inc**, **src** and **src**. The inc folder includes the h-files and especially fortuna.h which defines the API of the Fortuna PRNG on EFM32. The **src** folder includes the source code of the Fortuna PRNG and consists of the following C files:

- **fortuna_prng.c**

  The core module of Fortuna PRNG that handles reseeding and pseudo random number generation with the AES block cipher function.

- **fortuna_entropy_accumulator.c**

  The entropy accumulator implements the acquisition of true random data from entropy sources into internal entropy pools that are used to reseed the Fortuna PRNG.

- **fortuna_seed_manager.c**

  The seed manager handles how the seed file is stored and updated in flash memory.

- **fortuna.c**

  This file implements the API functions of the Fortuna PRNG on EFM2.

The **demo** folder includes a demo program which implements an entropy source using the ADC on the EFM32, and consists of the following C files:

- **fortuna_main.c**

  This file includes an example program that demonstrates how to setup and use the Fortuna PRNG on the EFM32. First, the demo program initializes the required system resources, the ADC entropy source and the Fortuna PRNG by calling the FORTUNA_Init function. Then some initial entropy is generated and fed to the FORTUNA_AddRandomEvent function in order to arm the PRNG to generate pseudo random data. Then FORTUNA_RandomDataGet is called to retrieve pseudo random data into a user buffer which is printed to the stdout with printf. Then FORTUNA_Rand is called to demonstrate generation of a 32-bit pseudo random word. And finally FORTUNA_Shutdown is called in order to shutdown the PRNG in a secure fashion.

NOTE: On the EFM32GG_DK3750, the demo uses the serial port to output the example pseudo random data generated. The call to RETARGET_SerialInit redirects standard I/O (stdin/stdout) to the serial port (UART1 on the EFM32GG_DK3750). The default baud rate is 115200bps which is relatively high and should therefore run with the HFXO as core clock source, instead of the default HFRCO which is inaccurate and may cause data errors, especially loss of data.

- **fortuna_adc.c**

  This file implements an entropy source based on sampling the internal temperature sensor of the EFM with the ADC. This is the only entropy source included by default in the current version of the Fortuna PRNG for EFM32. Users are encouraged to implement additional entropy sources.

5 References

- [1] N. Ferguson and B. Schneier, Practical Cryptography, Wiley Publishing Inc., 2003.
- [2] Abreham Emishaw Delelegn, Implementation of Fortuna on EFM32 Microcontroller and Differential Power Analysis, Master Thesis 2012, University of Southampton Faculty of Physical and Applied Sciences School of Electronics and Computer Science

6 Revision History

6.1 Revision 1.02

2013-12-19

Changed application note numbering to 4 digits.

6.2 Revision 1.01

2013-11-25

Fixed typos.

6.3 Revision 1.00

2013-11-21

Initial revision.

A Disclaimer and Trademarks

A.1 Disclaimer

Silicon Laboratories intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Laboratories products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Laboratories reserves the right to make changes without further notice and limitation to product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Silicon Laboratories shall have no liability for the consequences of use of the information supplied herein. This document does not imply or express copyright licenses granted hereunder to design or fabricate any integrated circuits. The products must not be used within any Life Support System without the specific written consent of Silicon Laboratories. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Laboratories products are generally not intended for military applications. Silicon Laboratories products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons.

A.2 Trademark Information

Silicon Laboratories Inc., Silicon Laboratories, Silicon Labs, SiLabs and the Silicon Labs logo, CMEMS®, EFM, EFM32, EFR, Energy Micro, Energy Micro logo and combinations thereof, "the world's most energy friendly microcontrollers", Ember®, EZLink®, EZMac®, EZRadio®, EZRadioPRO®, DSPLL®, ISOmodem®, Precision32®, ProSLIC®, SiPHY®, USBXpress® and others are trademarks or registered trademarks of Silicon Laboratories Inc. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. All other products or brand names mentioned herein are trademarks of their respective holders.

B Contact Information

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701

Please visit the Silicon Labs Technical Support web page:
http://www.silabs.com/support/pages/contacttechnicalsupport.aspx
and register to submit a technical support request.

Table of Contents

List of Figures

silabs.com


ZERO
ARM Cortex-M0+
TINY
ARM Cortex-M3
GECKO
ARM Cortex-M3
LEOPARD
ARM Cortex-M3
GIANT
ARM Cortex-M3
WONDER
ARM Cortex-M4