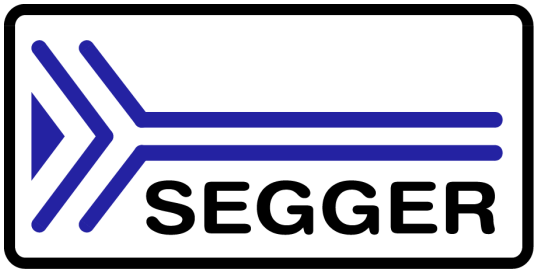
emLib

Library collection

User & Reference Guide

Document: UM12001
Software version: 1.0
Revision: 0
Date: July 17, 2013



A product of SEGGER Microcontroller GmbH & Co. KG

www.segger.com

Disclaimer

Specifications written in this document are believed to be accurate, but are not guaranteed to be entirely free of error. The information in this manual is subject to change for functional or performance improvements without notice. Please make sure your manual is the latest edition. While the information herein is assumed to be accurate, SEGGER Microcontroller GmbH & Co. KG (SEGGER) assumes no responsibility for any errors or omissions. SEGGER makes and you receive no warranties or conditions, express, implied, statutory or in any communication with you. SEGGER specifically disclaims any implied warranty of merchantability or fitness for a particular purpose.

Copyright notice

You may not extract portions of this manual or modify the PDF file in any way without the prior written permission of SEGGER. The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such a license.

© 2013 SEGGER Microcontroller GmbH & Co. KG, Hilden / Germany

Trademarks

Names mentioned in this manual may be trademarks of their respective companies.

Brand and product names are trademarks or registered trademarks of their respective holders.

Contact address

SEGGER Microcontroller GmbH & Co. KG

In den Weiden 11
D-40721 Hilden

Germany

Tel.+49 2103-2878-0
Fax.+49 2103-2878-28
E-mail: support@segger.com
Internet: http://www.segger.com

Manual versions

This manual describes the current software version. If any error occurs, inform us and we will try to assist you as soon as possible.
Contact us for further information on topics or routines not yet specified.

Print date: July 17, 2013

Software	Revision	Date	By	Description
1.00	0	130321	JL	Initial release

© 2013 SEGGER Microcontroller GmbH & Co. KG

 © 2013 SEGGER Microcontroller GmbH & Co. KG

About this document

Assumptions

This document assumes that you already have a solid knowledge of the following:

- The software tools used for building your application (assembler, linker, C compiler)
- The C programming language
- The target processor
- DOS command line

If you feel that your knowledge of C is not sufficient, we recommend The C Programming Language by Kernighan and Richie (ISBN 0-13-1103628), which describes the standard in C-programming and, in newer editions, also covers the ANSI C standard.

How to use this manual

This manual explains all the functions and macros that the product offers. It assumes you have a working knowledge of the C language. Knowledge of assembly programming is not required.

Typographic conventions for syntax

This manual uses the following typographic conventions:

Style	Used for
Body	Body text.
`Keyword`	Text that you enter at the command-prompt or that appears on the display (that is system functions, file- or pathnames).
`Parameter`	Parameters in API functions.
`Sample`	Sample code in program examples.
`Sample comment`	Comments in programm examples.
Reference	Reference to chapters, sections, tables and figures or other documents.
GUIElement	Buttons, dialog boxes, menu names, menu commands.
Emphasis	Very important sections.

Table 1.1: Typographic conventions

© 2013 SEGGER Microcontroller GmbH & Co. KG

SEGGER Microcontroller GmbH & Co. KG develops and distributes software development tools and ANSI C software components (middleware) for embedded systems in several industries such as telecom, medical technology, consumer electronics, automotive industry and industrial automation.

SEGGER's intention is to cut software development time for embedded applications by offering compact flexible and easy to use middleware, allowing developers to concentrate on their application.

Our most popular products are emWin, a universal graphic software package for embedded applications, and embOS, a small yet efficient real-time kernel. emWin, written entirely in ANSI C, can easily be used on any CPU and most any display. It is complemented by the available PC tools: Bitmap Converter, Font Converter, Simulator and Viewer. embOS supports most 8/16/32-bit CPUs. Its small memory footprint makes it suitable for single-chip applications.

Apart from its main focus on software tools, SEGGER develops and produces programming tools for flash micro controllers, as well as J-Link, a JTAG emulator to assist in development, debugging and production, which has rapidly become the industry standard for debug access to ARM cores.

Corporate Office:
http://www.segger.com

United States Office:
http://www.segger-us.com

EMBEDDED SOFTWARE (Middleware)

emWin
Graphics software and GUI
emWin is designed to provide an efficient, processor- and display controller-independent graphical user interface (GUI) for any application that operates with a graphical display.

embOS
Real Time Operating System
embOS is an RTOS designed to offer the benefits of a complete multitasking system for hard real time applications with minimal resources.

embOS/IP
TCP/IP stack
embOS/IP a high-performance TCP/IP stack that has been optimized for speed, versatility and a small memory footprint.

emFile
File system
emFile is an embedded file system with FAT12, FAT16 and FAT32 support. Various Device drivers, e.g. for NAND and NOR flashes, SD/MMC and CompactFlash cards, are available.

USB-Stack
USB device/host stack
A USB stack designed to work on any embedded system with a USB controller. Bulk communication and most standard device classes are supported.

SEGGER TOOLS

Flasher
Flash programmer
Flash Programming tool primarily for micro controllers.

J-Link
JTAG emulator for ARM cores
USB driven JTAG interface for ARM cores.

J-Trace
JTAG emulator with trace
USB driven JTAG interface for ARM cores with Trace memory. supporting the ARM ETM (Embedded Trace Macrocell).

J-Link / J-Trace Related Software
Add-on software to be used with SEGGER's industry standard JTAG emulator, this includes flash programming software and flash breakpoints.

© 2013 SEGGER Microcontroller GmbH & Co. KG

Table of Contents

1 Introduction to emLib 9
1.1 What is emLib 10
1.2 Features 10
1.3 Available modules 10
1.3.1 Cryptographic modules 10
2 AES 11
2.1 What is AES? 12
2.2 Using emLib AES 13
2.3 AES API functions 14
2.4 Example codes 26
2.5 Sample applications 30
2.5.1 AESCrypt 31
2.5.2 AESSpeedtest 32
2.5.3 AESValidate 33
2.6 Performance and memory footprint 34
2.6.1 Performance test 34
3 DES 35
3.1 What is DES? 36
3.2 Usind emLib DES 37
3.3 DES API functions 38
3.4 Example codes 45
3.5 Sample applications 47
3.5.1 DESSpeedtest 48
3.5.2 DESValidate 49
3.6 Performance and memory footprint 50
3.6.1 Performance test 50

 © 2013 SEGGER Microcontroller GmbH & Co. KG

 © 2013 SEGGER Microcontroller GmbH & Co. KG

Chapter 1

Introduction to emLib

This chapter provides an introduction to emLib. It explains the basic concept behind emLib and its modules.

 © 2013 SEGGER Microcontroller GmbH & Co. KG

1.1 What is emLib

emLib is a collection of software modules for different purposes. It currently includes AES and DES encryption. Modules for CRC, compression and aysmmetric encryption/ decryption are planned.
The software is designed for portability to any device. The modules can be used in PC applications, as well as on embedded target devices.
emLib is optimized for speed performance and a small memory footprint.
The sources are completely written in ANSI-C and MISRA-C 2004 compliant.
Validation code for the APIs using standard test patterns is included.

1.2 Features

emLib is written in ANSI-C and can be used on virtually any CPU.

Some features of emLib:

- Easy to integrate by using a simple API.
- Same modules and same API can be used in PC programs as well as on embedded targets.
- Sample applications for tests and validation of the modules included.

1.3 Available modules

1.3.1 Cryptographic modules

AES module

Implemention of the AES 128 bit and 256 bit algorithm including chained block processing for en-/decryption of more than 16 Byte of data.

DES module

Implementation of the DES (56 bit) algorithm, also including CBC for processing more than 8 Byte of data.
The DES functions can be called multiple times to achieve a higher security (TDES, tripel-DES).

 © 2013 SEGGER Microcontroller GmbH & Co. KG

Chapter 2

AES

The emLib AES module allows encryption and decryption of data using AES, the Advanced Encryption Standard as standardized by NIST in 2001. This chapter describes the AES API functions and shows their usage based on example codes.

 © 2013 SEGGER Microcontroller GmbH & Co. KG

2.1 What is AES?

The Advanced Encryption Standard, short AES, is a symmetric-key algorithm used for encryption an decryption of data. It was established by the U.S. National Institute of Standards and Technology (NIST) and is the standard for encrypting electronic data since 2001. AES supersedes the Data Encryption Standard (DES).

AES is a substitution-permutation network block cipher using a fixed block size of 128 bits and a key size of 128, 192 or 256 bits.
The data block is stored in a 4-row matrix with a cell size of 8 bits. Based on the key length, these blocks are transformed using parts of the key in a number of rounds. AES 128 uses 10 rounds, AES 256 14. Therefore encryption with AES 256 is ~40% slower than AES 128.
In each round a round key is derived from the original key. Afterwards each byte is non-linear substituted according to a lookup table, the rows of the data matrix are shifted cyclically and mixed.

emLib AES uses a key of 128 or 256 bits to encrypt a block of 16 bytes of data at a time. To optimize the performance of the algorithms the generation of the round keys can be done before the actual encryption or decryption and used more than one time. For the substitution and mixing steps, emLib can be built with pre-calculated lookup tables, to increase the speed performance. emLib can also be built without these tables, to save memory.

AES can also be used in cipher block chaining (CBC) mode to process a multiple of 16 Bytes.
In CBC mode every chunk of 16 Bytes is XOR linked with the result of the previous encryption (the cipher text), before being encrypted. To decrypt one block, all previous blocks have to be known.
For the encryption of the first block an initialization vector which will be linked with the block, can be used to make sure the first block cannot be brute-force decrypted by comparing it to common first data blocks.

 © 2013 SEGGER Microcontroller GmbH & Co. KG

2.2 Using emLib AES

The emLib AES module has a simple yet powerful API. It can be easily integrated into an existing application.
The code is completely written in ANSI-C and MISRA-C compliant.

All functionality can be verified with standard test patterns using the Validation API functions. The functions for generating the tables used for higher optimization levels are also included for full transparency.

The module can be built with configurable optimizations to fit any requirement of high speed or low memory usage.

To simply encrypt or decrypt data the application would only need to call one function.
If more than one block needs to be processed with the same key, a context containing the round keys calculated from the key can be prepared and directly used by the encryption and decryption functions. For more than one call of these functions this method results in a slightly higher processing speed.

The following section lists and describes the available API functions of the emLib AES module.

 © 2013 SEGGER Microcontroller GmbH & Co. KG

2.3 AES API functions

The table below lists the available API functions.

Function	Description
AES128_CBC_Decrypt()	Decrypts data with AES 128 Bit using CBC.
AES128_CBC_Encrypt()	Encrypts data with AES 128 Bit using CBC.
AES128_Decrypt()	Decrypts 16 Bytes with AES 128 Bit.
AES128_Encrypt()	Encrypts 16 Bytes with AES 128 Bit.
AES128_Prepare()	Prepares the context for de-/encryption.
AES256_CBC_Decrypt()	Decrypts data with AES 256 Bit using CBC.
AES256_CBC_Encrypt()	Encrypts data with AES 256 Bit using CBC.
AES256_Decrypt()	Decrypts 16 Bytes with AES 256 Bit.
AES256_Encrypt()	Encrypts 16 Bytes with AES 256 Bit.
AES256_Prepare()	Prepares the context for de-/encryption.
AES_Validate()	Test function for validation of AES.

Table 2.1: AES API function overview

 © 2013 SEGGER Microcontroller GmbH & Co. KG

2.3.1 AES128_Prepare()

Description

Prepares the context depending on the key used for AES 128bit de-/encryption.

Prototype

```
void AES128_Prepare (AES_CONTEXT * pContext,
                     const U8    * pKey);
```

Parameter	Description
pContext	Pointer to the context for de-/encryption.
pKey	Pointer to the buffer which holds the encryption key (128bit).

Table 2.2: AES128_Prepare() parameter list

Additional information

The key has to be 128 bit (16 Byte) long.

Example

See *AES 128bit en-/decryption of 16 Bytes* on page 26

 © 2013 SEGGER Microcontroller GmbH & Co. KG

2.3.2 AES128_Encrypt()

Description

Encrypts a block of 16 Bytes (128 bit) using a context prepared with a 128 bit key.

Prototype

```
void AES128_Encrypt (AES_CONTEXT * pContext,
                     U8 *          pDest,
                     const U8 *    pSrc);
```

Parameter	Description
pContext	Pointer to the previously prepared context.
pDest	Pointer to the buffer which will hold the encrypted data.
pSrc	Pointer to the buffer which holds the unencrypted data.

Table 2.3: AES128_Encrypt() parameter list

Additional information

The data which will be encrypted has to be 16 Bytes.

For more than 16 Bytes see *AES128_CBC_Encrypt()* on page 21.

Example

See *AES 128bit en-/decryption of 16 Bytes* on page 26

 © 2013 SEGGER Microcontroller GmbH & Co. KG

2.3.3 AES128_Decrypt()

Description

Decrypts a block of 16 Bytes (128 bit) using a context prepared with a 128 bit key.

Prototype

```
void AES128_Decrypt (AES_CONTEXT * pContext,
                     U8 *          pDest,
                     const U8 *    pSrc);
```

Parameter	Description
pContext	Pointer to the previously prepared context.
pDest	Pointer to the buffer which will hold the decrypted data.
pSrc	Pointer to the buffer which holds the encrypted data.

Table 2.4: AES128_Decrypt() parameter list

Additional information

The key has to be the same as the one used for encryption.

The data which will be decrypted has to be 16 Bytes.

For more than 16 Bytes see *AES128_CBC_Decrypt()* on page 22.

Example

See *AES 128bit en-/decryption of 16 Bytes* on page 26

 © 2013 SEGGER Microcontroller GmbH & Co. KG

2.3.4 AES256_Prepare()

Description

Prepares the context depending on the key used for AES 256bit de-/encryption.

Prototype

```
void AES256_Prepare (AES_CONTEXT * pContext, const U8 * pKey);
```

Parameter	Description
pContext	Pointer to the context for de-/encryption.
pKey	Pointer to the buffer which holds the encryption key (256bit).

Table 2.5: AES256_Prepare() parameter list

Additional information

The key has to be 256 bit (32 Byte) long.

Example

See *AES 256bit en-/decryption of 16 Bytes* on page 28

 © 2013 SEGGER Microcontroller GmbH & Co. KG

2.3.5 AES256_Encrypt()

Description

Encrypts a block of 16 Bytes (128 bit) using a context prepared with the 256 bit key.

Prototype

```
void AES128_Encrypt (AES_CONTEXT * pContext, U8 * pDest, const U8 * pSrc);
```

Parameter	Description
pContext	Pointer to the previously prepared context.
pDest	Pointer to the buffer which will hold the encrypted data.
pSrc	Pointer to the buffer which holds the unencrypted data.

Table 2.6: AES256_Encrypt() parameter list

Additional information

The data which will be encrypted has to be 16 Bytes.

For more than 16 Bytes see *AES256_CBC_Encrypt()* on page 23.

Example

See *AES 256bit en-/decryption of 16 Bytes* on page 28

 © 2013 SEGGER Microcontroller GmbH & Co. KG

2.3.6 AES256_Decrypt()

Description

Decrypts a block of 16 Bytes (128 bit) using a context prepared with the 256 bit key.

Prototype

```
void AES128_Decrypt (AES_CONTEXT * pContext, U8 * pDest, const U8 * pSrc);
```

Parameter	Description
pContext	Pointer to the previously prepared context.
pDest	Pointer to the buffer which will hold the decrypted data.
pSrc	Pointer to the buffer which holds the encrypted data.

Table 2.7: AES256_Decrypt() parameter list

Additional information

The key has to be the same as the one used for encryption.

The data which will be decrypted has to be 16 Bytes.

For more than 16 Bytes see *AES256_CBC_Decrypt()* on page 24.

Example

See *AES 256bit en-/decryption of 16 Bytes* on page 28

 © 2013 SEGGER Microcontroller GmbH & Co. KG

2.3.7 AES128_CBC_Encrypt()

Description

Encrypts data using cypher block chaining and a 128 bit key.

Prototype

```
void AES128_CBC_Encrypt (AES_CONTEXT * pContext, U8 * pDest, const U8 *
pSrc, int NumBytes, const U8 * pIV);
```

Parameter	Description
pContext	Pointer to the previously prepared context.
pDest	Pointer to the buffer which will hold the encrypted data.
pSrc	Pointer to the buffer which holds the data.
NumBytes	Number of Bytes which have to be encrypted
pIV	Pointer to the buffer which holds the initialization verctor.

Table 2.8: AES128_CBC_Encrypt() parameter list

Additional information

The length of the data has to be a multiple of 16 bytes.

If pIV is NULL the first block will not be linked.

pDest and pSrc may be the same, if the plain data is not needed after encryption.

Example

See *AES 128bit en-/decryption of 32 Bytes using CBC* on page 27

 © 2013 SEGGER Microcontroller GmbH & Co. KG

2.3.8 AES128_CBC_Decrypt()

Description

Decrypts data using cypher block chaining and a 128 bit key.

Prototype

```
void AES128_CBC_Decrypt (AES_CONTEXT * pContext, U8 * pDest, const U8 *
pSrc, int NumBytes, const U8 * pIV);
```

Parameter	Description
pContext	Pointer to the previously prepared context.
pDest	Pointer to the buffer which will hold the decrypted data.
pSrc	Pointer to the buffer which holds the encrypted data.
NumBytes	Number of Bytes which have to be decrypted
pIV	Pointer to the buffer which holds the initialization verctor.

Table 2.9: AES128_CBC_Decrypt() parameter list

Additional information

The key and the initialization vector have to be the same as used for encryption.

The length of the data has to be a multiple of 16 bytes.

If pIV is NULL the first block will not be linked.

pDest and pSrc must be different.

Example

See *AES 128bit en-/decryption of 32 Bytes using CBC* on page 27

 © 2013 SEGGER Microcontroller GmbH & Co. KG

2.3.9 AES256_CBC_Encrypt()

Description

Encrypts data using cypher block chaining and a 256 bit key.

Prototype

```
void AES128_CBC_Encrypt (AES_CONTEXT * pContext, U8 * pDest, const U8 *
pSrc, int NumBytes, const U8 * pIV);
```

Parameter	Description
pContext	Pointer to the previously prepared context.
pDest	Pointer to the buffer which will hold the encrypted data.
pSrc	Pointer to the buffer which holds the data.
NumBytes	Number of Bytes which have to be encrypted
pIV	Pointer to the buffer which holds the initialization vector.

Table 2.10: AES256_CBC_Encrypt() parameter list

Additional information

The length of the data has to be a multiple of 16 bytes.

If pIV is NULL the first block will not be linked.

pDest and pSrc may be the same, if the plain data is not needed after encryption.

Example

See *AES 256bit en-/decryption of 32 Bytes using CBC* on page 29

 © 2013 SEGGER Microcontroller GmbH & Co. KG

2.3.10 AES256_CBC_Decrypt()

Description

Decrypts data using cypher block chaining and a 256 bit key.

Prototype

```
void AES128_CBC_Decrypt (AES_CONTEXT * pContext, U8 * pDest, const U8 *
pSrc, int NumBytes, const U8 * pIV);
```

Parameter	Description
pContext	Pointer to the previously prepared context.
pDest	Pointer to the buffer which will hold the decrypted data.
pSrc	Pointer to the buffer which holds the encrypted data.
NumBytes	Number of Bytes which have to be decrypted
pIV	Pointer to the buffer which holds the initialization vector.

Table 2.11: AES256_CBC_Decrypt() parameter list

Additional information

The key and the initialization vector have to be the same as used for encryption.

The length of the data has to be a multiple of 16 bytes.

If pIV is NULL the first block will not be linked.

pDest and pSrc must be different.

Example

See *AES 256bit en-/decryption of 32 Bytes using CBC* on page 29

 © 2013 SEGGER Microcontroller GmbH & Co. KG

2.3.11 AES_Validate()

Description

This function can be used to test the AES implementation.

It en- and decrypts specified data and checks for valid output.

Prototype

```
int AES_Validate (void);
```

Return values

0: O.K. No error.

<0: Error. The implementation is not working correctly.

Additional information

The data for the validation is taken from RFC 3062 (http://www.rfc-editor.org/rfc/rfc3602.txt Chapter 4).

 © 2013 SEGGER Microcontroller GmbH & Co. KG

2.4 Example codes

2.4.1 AES 128bit en-/decryption of 16 Bytes

This sample shows how to encrypt and afterwards decrypt 16 bytes of data with AES and a 128 bit key.

```
#include "AES.h"

const U8 _aKey[16] =       {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
                            0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f};

const U8 _aPlaintext[16] = {0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
                            0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff};

int main() {
  U8 aEncrypted[16];
  U8 aDecrypted[16];
  AES_CONTEXT Context;
  //
  // Prepares the AES Context with _aKey
  //
  AES128_Prepare(&Context, &_aKey[0]);
  //
  // Encrypts the data from _aPlaintext and stores it in aEncrypted
  //
  AES128_Encrypt(&Context, &aEncrypted[0], &_aPlaintext[0]);
  //
  // Decrypts the data from aEncrypted and stores it in aDecrypted
  //
  AES128_Decrypt(&Context, &aDecrypted[0], &aEncrypted[0]);
  //
  // Check if aDecrypted is the same as _aPlaintext
  //
  if (memcmp(&aDecrypted[0], _aPlaintext, 16)) {
    return -1;
  }
  return 0;
}
```

 © 2013 SEGGER Microcontroller GmbH & Co. KG

2.4.2 AES 128bit en-/decryption of 32 Bytes using CBC

This sample shows how to encrypt and afterwards decrypt 32 bytes of data with AES and a 128 bit key using Cipher Block Chaining.

```
#include "AES.h"

const U8 _aKey[16] = {              0xc2, 0x86, 0x69, 0x6d, 0x88, 0x7c, 0x9a, 0xa0,
                                    0x61, 0x1b, 0xbb, 0x3e, 0x20, 0x25, 0xa4, 0x5a};

static const U8 _aIV[16] = {        0x56, 0x2e, 0x17, 0x99, 0x6d, 0x09, 0x3d, 0x28,
                                    0xdd, 0xb3, 0xba, 0x69, 0x5a, 0x2e, 0x6f, 0x58};

static const U8 _aPlaintext[32] = {
                                    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
                                    0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
                                    0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
                                    0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f};

static const U8 _aCiphertext[32] = {
                                    0xd2, 0x96, 0xcd, 0x94, 0xc2, 0xcc, 0xcf, 0x8a,
                                    0x3a, 0x86, 0x30, 0x28, 0xb5, 0xe1, 0xdc, 0x0a,
                                    0x75, 0x86, 0x60, 0x2d, 0x25, 0x3c, 0xff, 0xf9,
                                    0x1b, 0x82, 0x66, 0xbe, 0xa6, 0xd6, 0x1a, 0xb1};

int main() {
  U8          aEnc[32];
  U8          aPlain[32];
  AES_CONTEXT Context;
  //
  // Prepare the context with _aKey
  //
  AES128_Prepare(&Context, &_aKey[0]);
  //
  // Encrypt the data of _aPlaintext
  // and compare it with the desired result.
  //
  AES128_CBC_Encrypt(&Context, &aEnc[0], &_aPlaintext[0], 32, &_aIV[0]);
  if (memcmp(&aEnc[0], &_aCiphertext[0], 32)) {
    return -1;
  }
  //
  // Decrypt the data of aEnc
  // and compare it with the previously used _aPlaintext
  //
  AES128_CBC_Decrypt(&Context, &aPlain[0], &aEnc[0], 32, &_aIV[0]);
  if (memcmp(&aPlain[0], &_aPlaintext[0], 32)) {
    return -1;
  }
  return 0; // AES 128 CBC works fine.
}
```

 © 2013 SEGGER Microcontroller GmbH & Co. KG

2.4.3 AES 256bit en-/decryption of 16 Bytes

This sample shows how to encrypt and afterwards decrypt 16 bytes of data with AES and a 256 bit key.

```
#include "AES.h"

const U8 _aKey[32] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
                      0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
                      0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
                      0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f};

const U8 _aPlaintext[16] = {0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
                            0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff};

int main() {
  U8 aEncrypted[16];
  U8 aDecrypted[16];
  AES_CONTEXT Context;
  //
  // Prepares the AES Context with _aKey
  //
  AES256_Prepare(&Context, &_aKey[0]);
  //
  // Encrypts the data from _aPlaintext and stores it in aEncrypted
  //
  AES256_Encrypt(&Context, &aEncrypted[0], &_aPlaintext[0]);
  //
  // Decrypts the data from aEncrypted and stores it in aDecrypted
  //
  AES256_Decrypt(&Context, &aDecrypted[0], &aEncrypted[0]);
  //
  // Check if aDecrypted is the same as _aPlaintext
  //
  if (memcmp(&aDecrypted[0], _aPlaintext, 16)) {
    return -1;
  }
  return 0;
}
```

 © 2013 SEGGER Microcontroller GmbH & Co. KG

2.4.4 AES 256bit en-/decryption of 32 Bytes using CBC

This sample shows how to encrypt and afterwards decrypt 32 bytes of data with AES and a 256 bit key using Cipher Block Chaining.

```
#include "AES.h"

const U8 _aKey[32] = {              0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
                                    0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
                                    0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
                                    0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f};

static const U8 _aIV[16] = {        0x56, 0x2e, 0x17, 0x99, 0x6d, 0x09, 0x3d, 0x28,
                                    0xdd, 0xb3, 0xba, 0x69, 0x5a, 0x2e, 0x6f, 0x58};

static const U8 _aPlaintext[32] = {
                                    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
                                    0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
                                    0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
                                    0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f};

int main() {
  U8          aEnc[32];
  U8          aPlain[32];
  AES_CONTEXT Context;
  //
  // Prepare the context with _aKey
  //
  AES256_Prepare(&Context, &_aKey[0]);
  //
  // Encrypt the data of _aPlaintext
  //
  AES256_CBC_Encrypt(&Context, &aEnc[0], &_aPlaintext[0], 32, &_aIV[0]);
  //
  // Decrypt the data of aEnc
  // and compare it with the previously used _aPlaintext
  //
  AES256_CBC_Decrypt(&Context, &aPlain[0], &aEnc[0], 32, &_aIV[0]);
  if (memcmp(&aPlain[0], &_aPlaintext[0], 32)) {
    return -1;
  }
  return 0; // AES 256 CBC works fine.
}
```

 © 2013 SEGGER Microcontroller GmbH & Co. KG

2.5 Sample applications

emLib includes some sample applications to show the modules functionality and provide an easy to use starting point for your application.
The application's source code is included within the module.

The following applications are included in emLib AES:

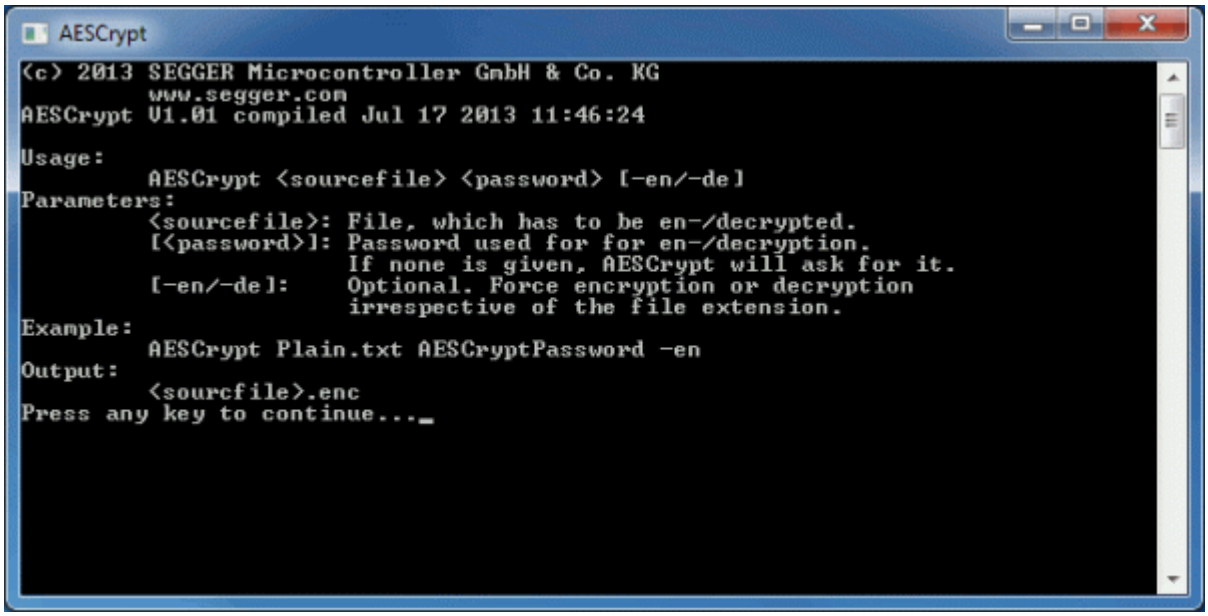
Application name	Target platform	Description
AESCrypt.exe	Windows	Commandline tool to en-/decrypt a file using AES 256.
AESSpeedtest.exe	Windows	Console application testing the speed of emLib AES.
AESValidate.exe	Windows	Console application validating emLib AES with standard test patterns.

Table 2.12: Sample Applications

 © 2013 SEGGER Microcontroller GmbH & Co. KG

2.5.1 AESCrypt

AESCrypt is a windows application, encrypting and decrypting a file with the given keyword. The tool can be used to easily keep files secured.



Usage

"AESCrypt" <sourcefile> [<password>] [<option>]

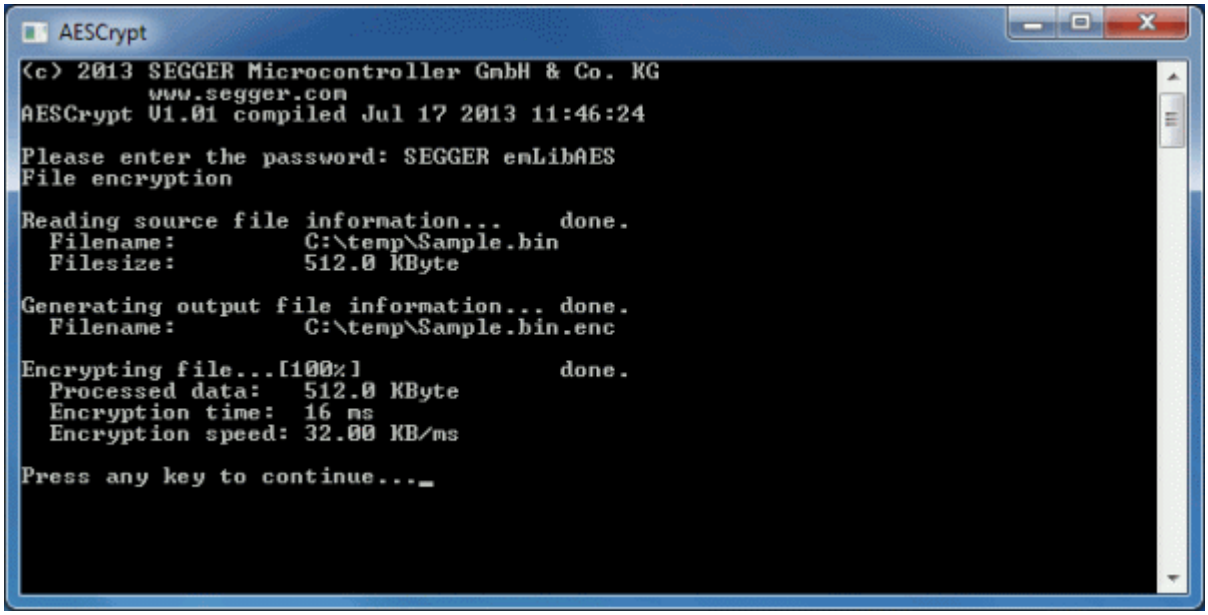
Parameter	Description
<sourcefile>	Path to the file, which has to be en-/decrypted.
<password>	Password used for en-/decryption.
<option>	(optional) "-en": Force encryption of the source file. "-de": Force decryption of the source file. If no option is given, operation depends on source file extension.

Table 2.13: AESCrypt parameter list

Additional information

The password can contain any character and does not have a fixed required length. The output file after encryption will have the extension ".enc".

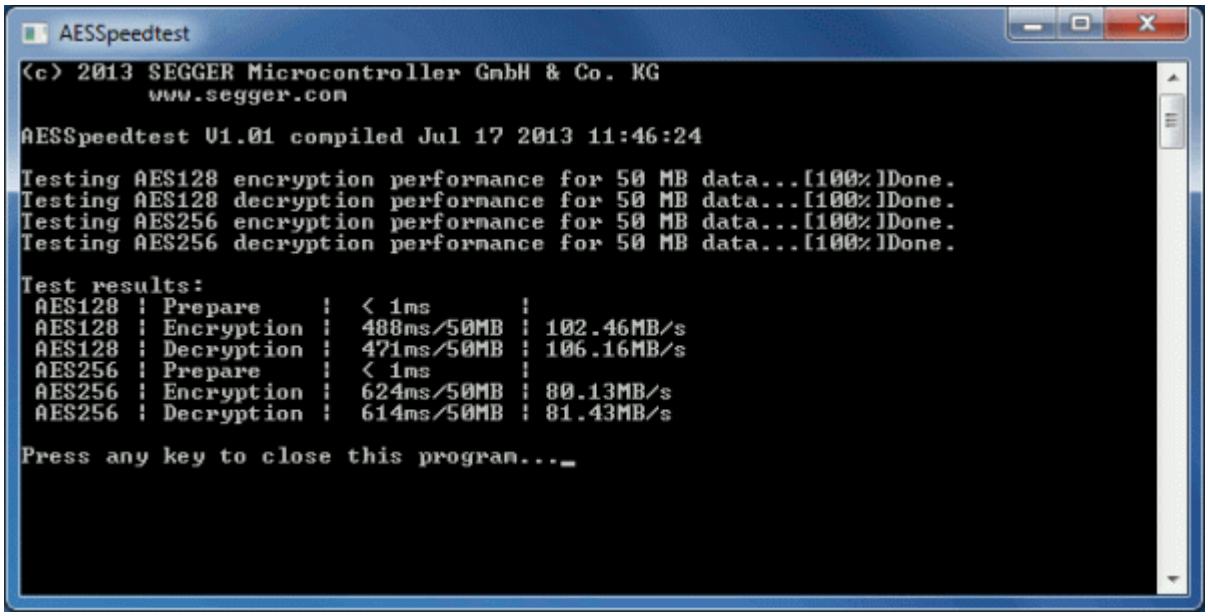
If present, the original file will be renamed to <Filename>.orig, when decrypting a file with the same name.



 © 2013 SEGGER Microcontroller GmbH & Co. KG

2.5.2 AESSpeedtest

AESSpeedtest is a windows application, testing the performance of the emLib AES algorithms.



 © 2013 SEGGER Microcontroller GmbH & Co. KG

2.5.3 AESValidate

AESValidate is a Windows application used to test and validate the implementation of the AES algorithms.
The application uses the Validation API and compares the results of encryption and decryption with the expected results.

AESValidate will show an error message, if a validation test fails.

 © 2013 SEGGER Microcontroller GmbH & Co. KG

2.6 Performance and memory footprint

emLib AES aims for portability and is designed to fit speed and size requirements for different targets.
It includes configurable defines to switch between speed and size optimizations.
The values can be changed in `AES_Config.h`.

#define	Values	Description
OPTIMIZE_MIX_SUBST	0 (No opt.) 1 (default)	Use a 32-bit table to perform "MixColumns" and "SubBytes" at the same time.
OPTIMIZE_MIX_COLUMNS	0 (default) 1 2 (highest)	Use tables for matrix multiplication.

Table 2.14: Optimization defines

2.6.1 Performance test

The following system has been used to measure the performance and memory footprint of the module with different optimization levels.

Detail	Description:
Target	STM32F417 running at 168 MHz, internal flash used
Tool chain	IAR EWARM V6.40E

Table 2.15: Performance test configuration

Results

The following table shows the en- and decryption speed of emLib AES128:

Compiler options	Module #defines	Speed	ROM usage
Optimize high for speed	OPTIMIZE_MIX_SUBST 1 OPTIMIZE_MIX_COLUMNS 2	~1.3 MByte/sec	~11.8 KBytes
Optimize high for size	OPTIMIZE_MIX_SUBST 0 OPTIMIZE_MIX_COLUMNS 0	~0.4 MByte/sec	~3.4 KBytes

The performance depends on the MCU speed and the flash memory speed. Results may vary if a different setup is used.

 © 2013 SEGGER Microcontroller GmbH & Co. KG

Chapter 3

DES

The emLib DES module allows encryption and dycryption of data using DES, the Data Encryption Standard as published in 1976. This chapter describes the DES API functions and shows their usage based on example codes.

In this chapter, you will find a description of the DES module API functions and samples for their implementation.

 © 2013 SEGGER Microcontroller GmbH & Co. KG

3.1 What is DES?

The Data Encryption Standard, short DES, is a symmetric-key algorithm for en- and decryption of data. It was developed in the 1970's and established as a standard for the United States by the National Bureau of Standards (NBS, now NIST). DES has been superseded by AES.

DES is a block cypher, taking a fixed-length block of data (64 bits). The key used for processing consists of 64 bits, where only 56 are actually used for transformations and 8 bits are used for parity checks.
DES performs an initial permitation of the data, 16 rounds of transformation, and a final permitation, the inverse of the initial permutation. In the transformations the data block is initially splitted in two 32 bit blocks where the first block is transformated with the round key using a Feistel cipher and XOR-linked with the second block. The first block and the resulting block are used for the next round.

emLib DES uses a key of 64 bits to encrypt a block of 68 bits of data at a time. To optimize the performance of the algorithms the generation of the round keys can be done before the actual encryption or decryption and used more than one time.

DES can also be used in cipher block chaining (CBC) mode to process more than 64 bits.
In CBC mode every chunk of 64 bits is XOR linked with the result of the previous encryption (the cipher text), before being encrypted. To decrypt one block, all previous blocks have to be known.
For the encryption of the first block an initialization vector which will be linked with the block, can be used to make sure the first block cannot be brute-force decrypted by comparing it to common first data blocks.

 © 2013 SEGGER Microcontroller GmbH & Co. KG

3.2 Usind emLib DES

The emLib DES module has a simple yet powerful API. It can be easily integrated into an existing application.
The code is completely written in ANSI-C and MISRA-C compliant.

All functionality can be verified with standard test patterns using the Validation API functions. The functions for generating the tables used for higher optimization levels are also included for full transparency.

To simply encrypt or decrypt data the application would only need to call one function. If more than one block needs to be processed with the same key, a context containing the round keys calculated from the key can be prepared and directly used by the encryption and decryption functions. For more than one call of these functions this method results in a slightly higher processing speed.

The following section lists and describes the available API functions of the emLib DES module.

 © 2013 SEGGER Microcontroller GmbH & Co. KG

3.3 DES API functions

The table below lists the available API functions.

Function	Description
DES_CBC_Encrypt()	Encrypts data with DES using CBC.
DES_CBC_Decrypt()	Decrypts data with DES using CBC.
DES_Decrypt()	Decrypts 8 Bytes with DES.
DES_Encrypt()	Encrypts 8 Bytes with DES.
DES_Prepare()	Prepares the context for de-/encryption.
DES_Validate()	Test function for validation of DES.

Table 3.1: DES API function overview

 © 2013 SEGGER Microcontroller GmbH & Co. KG

3.3.1 DES_Prepare()

Description

Prepares the context depending on the 64bit key used for DES de-/encryption.

Prototype

```
void DES_Prepare(DES_CONTEXT * pContext, const U8 * pKey);
```

Parameter	Description
pContext	Pointer to the context for de-/encryption.
pKey	Pointer to the buffer which holds the encryption key (64bit).

Table 3.2: DES_Prepare() parameter list

Additional information

The key has 1 parity bit per byte, so the effective key length is 56bit.

A pointer to a 64bit key has to be provided to the function.

Example

See *DES en-/decryption of 8 Bytes* on page 45

 © 2013 SEGGER Microcontroller GmbH & Co. KG

3.3.2 DES_Encrypt()

Description

Encrypts a block of 8 Bytes (64 bit) using a context prepared with the 64 bit key.

Prototype

```
void DES_Encrypt (DES_CONTEXT * pContext, U8 * pDest, const U8 * pSrc);
```

Parameter	Description
pContext	Pointer to the prepared context for DES encryption.
pDest	Pointer to the buffer for the encrypted data
pSrc	Pointer to the plain text data buffer which has to be encrypted.

Table 3.3: DES_Encrypt() parameter list

Additional information

The data has to be 64bit.

For more than 64 bit see *DES_CBC_Encrypt()* on page 42.

Example

See *DES en-/decryption of 8 Bytes* on page 45

 © 2013 SEGGER Microcontroller GmbH & Co. KG

3.3.3 DES_Decrypt()

Description

Decrypts a block of 8 Bytes (64 bit) using a context prepared with the 64 bit key.

Prototype

```
void DES_Decrypt (DES_CONTEXT * pContext, U8 * pDest, const U8 * pSrc);
```

Parameter	Description
pContext	Pointer to the prepared DES conctext.
pDest	Pointer to the buffer for the decrypted data.
pSrc	Pointer to the buffer with the encrypted data.

Table 3.4: DES_Decrypt() parameter list

Additional information

The key has to be the same as the one used for encryption.

The data which will be decrypted has to be 64bit long.

For more than 64bit see *DES_CBC_Decrypt()* on page 43.

Example

See *DES en-/decryption of 8 Bytes* on page 45

 © 2013 SEGGER Microcontroller GmbH & Co. KG

3.3.4 DES_CBC_Encrypt()

Description

Encrypts a block of data using DES with cypher blcok chaining.

Prototype

```
void DES_CBC_Encrypt (DES_CONTEXT * pContext, U8 * pDest, const U8 * pSrc,
int NumBytes, const U8 * pIV);
```

Parameter	Description
pContext	Pointer to the DES context.
pDest	Pointer to the data buffer for the encrypted data.
pSrc	Pointer to the plain data buffer.
NumBytes	Number of Bytes, which has to be encrypted.
pIV	[optional] Initialization vector for the first block of data.

Table 3.5: DES_CBC_Encrypt() parameter list

Additional information

The data has to be a multiple of 8 Byte.

To prepare the context use DES_Prepare().

If pIV is NULL, an initialization vector of 0 is used.

Example

See *DES en-/decryption of 16 Bytes using CBC* on page 46

 © 2013 SEGGER Microcontroller GmbH & Co. KG

3.3.5 DES_CBC_Decrypt()

Description

Decrypts a data block using DES with cypher block chaining.

Prototype

```
void DES_CBC_Decrypt (DES_CONTEXT * pContext, U8 * pDest, const U8 * pSrc,
int NumBytes, const U8 * pIV);
```

Parameter	Description
pContext	Pointer to the DES context.
pDest	Pointer to the buffer for the decrypted data.
pSrc	Pointer to the buffer with encrypted data.
NumBytes	Number of Bytes which has to be decrypted.
pIV	[optional] Initialization vector used for the first data block.

Table 3.6: DES_CBC_Decrypt() parameter list

Additional information

The context has to be generated with the same key as for encryption. The initialization vector has to be the same as for encryption.

If pIV is NULL an initialization vector of 0 is used.

The data has to be a multiple of 8 Bytes.

Example

See *DES en-/decryption of 16 Bytes using CBC* on page 46

 © 2013 SEGGER Microcontroller GmbH & Co. KG

3.3.6 DES_Validate()

Description

This function is used to test the DES implementation.

It uses defined plain data and a defined key for encryption and checks if the encryption result is correct. The initialization vector is 0.

Prototype

```
int  DES_Validate       (void);
```

Return values

0: O.K. No error.

-1: Error. Encryption failed. The implementation is not working correctly.

-2: Error. Decryption failed. The implementation is not working correctly.

Additional information

Validation set from NIST special publication 800-17.

 © 2013 SEGGER Microcontroller GmbH & Co. KG

3.4 Example codes

3.4.1 DES en-/decryption of 8 Bytes

```
#include <DES.h>

int main(void) {
  DES_CONTEXT Context;
  const U8  aKey[8]       = {0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF};
  const U8  aPlain[8]     = {0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xE7};
        U8  aRefPlain[8];
        U8  aCipher[8];
        int r;

  //
  // Prepare the DES Context with aKey
  //
  DES_PrepareKey(&Context, &aKey[0]);
  //
  // Encrypt the data of aPlain
  //
  DES_Encrypt(&Context, &aCipher[0], &aPlain[0]);
  //
  // Decrypt the data of aCipher
  //
  DES_Decrypt(&Context, &aRefPlain[0], &aCipher[0]);
  r = memcmp(&aPlain[0], &aRefPlain[0], sizeof(aRefPlain));
  if (r != 0) {
    return -2;
  }
  return r; // DES works fine.
}
```

 © 2013 SEGGER Microcontroller GmbH & Co. KG

3.4.2 DES en-/decryption of 16 Bytes using CBC

```
#include <DES.h>

int main(void) {
  DES_CONTEXT Context;
  const U8  aKey[8]        = {0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF};
  const U8  aPlain[16]     = {0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xE7,
                              0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xE7};
        U8  aRefPlain[16];
        U8  aCipher[16];
        int r;

  //
  // Prepare the DES Context with aKey
  //
  DES_PrepareKey(&Context, &aKey[0]);
  //
  // Encrypt the data of aPlain
  //
  DES_CBC_Encrypt(&Context, &aCipher[0], &aPlain[0], sizeof(aPlain), NULL);
  //
  // Decrypt the data of aCipher
  //
  DES_CBC_Decrypt(&Context, &aRefPlain[0], &aCipher[0], sizeof(aCipher), NULL);
  r = memcmp(&aPlain[0], &aRefPlain[0], sizeof(aRefPlain));
  if (r != 0) {
    return -2;
  }
  return r; // DES works fine.
}
```

 © 2013 SEGGER Microcontroller GmbH & Co. KG

3.5 Sample applications

emLib includes some sample applications to show the modules functionality and provide an easy to use starting point for your application.
The application’s source code is included within the module.

The following applications are included in emLib DES:

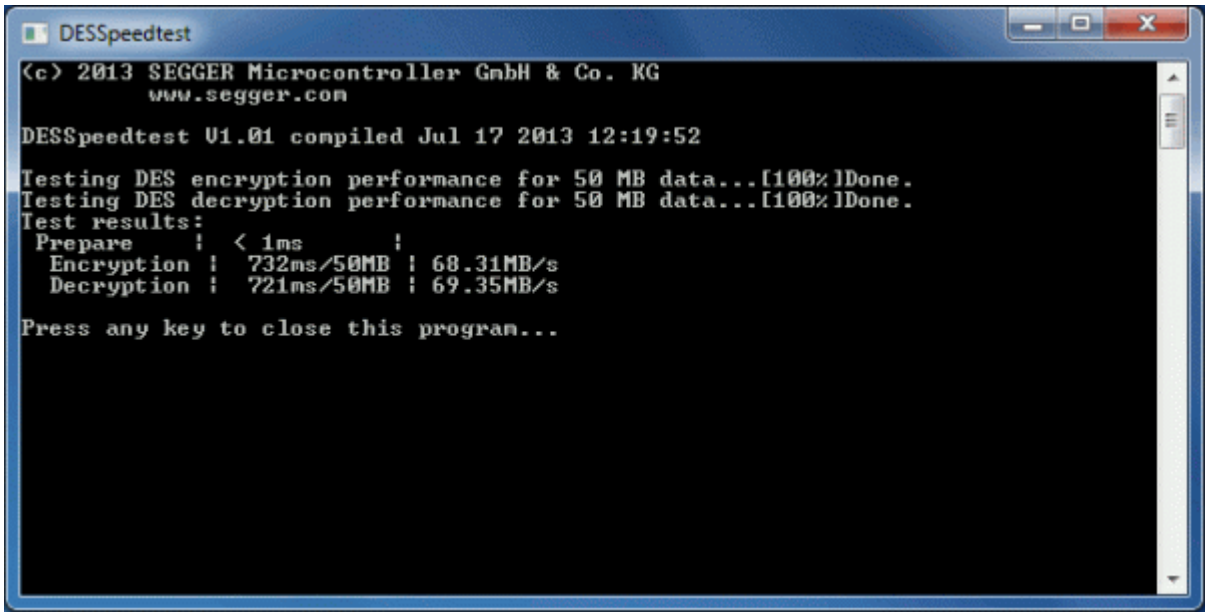
Application name	Target platform	Description
DESSpeedtest.exe	Windows	Console application testing the speed of emLib DES.
DESValidate.exe	Windows	Console application validating emLib DES with standard test patterns.

Table 3.7: Sample Applications

 © 2013 SEGGER Microcontroller GmbH & Co. KG

3.5.1 DESSpeedtest

DESSpeedtest is a windows application, testing the performance of the emLib DES algorithms.



 © 2013 SEGGER Microcontroller GmbH & Co. KG

3.5.2 DESValidate

DESValidate is a Windows application used to test and validate the implementation of the DES algorithms.
The application uses the Validation API and compares the results of encryption and decryption with the expected results.

DESValidate will show an error message, if a validation test fails.

 © 2013 SEGGER Microcontroller GmbH & Co. KG

3.6 Performance and memory footprint

emLib DES aims for portability and is designed to fit speed and size requirements for different targets.

3.6.1 Performance test

The following system has been used to measure the performance and memory footprint of the module with different optimization levels.

Detail	Description:
Target	STM32F417 running at 168 MHz, internal flash used
Tool chain	IAR EWARM V6.40E

Table 3.8: Performance test configuration

Results

The following table shows the en- and decryption speed of emLib DES:

Compiler options	Speed	ROM usage
Optimize high for speed	~0.8 MByte/sec	~3.2 KBytes
Optimize high for size	~0.6 MByte/sec	~3.0 KBytes

The performance depends on the MCU speed and the flash memory speed. Results may vary if a different setup is used.

 © 2013 SEGGER Microcontroller GmbH & Co. KG

Index

A
AES 12

C
CBC 36
Ciphertext 12

D
DES 36

F
Feistel cipher 36

I
Initialization Vector 12

S
Syntax, conventions used 5

 © 2013 SEGGER Microcontroller GmbH & Co. KG

 © 2013 SEGGER Microcontroller GmbH & Co. KG

 © 2013 SEGGER Microcontroller GmbH & Co. KG