UConn Mitre eCTF

Generated by Doxygen 1.8.8

Wed Mar 1 2017 20:08:49

Contents

1 Main Page

Code and Tools used for the 2017 MITRE eCTF Competition

ATMega1284_Boot

Contains MITRE's example bootloader code with minor edits to function in Atmel Studio. Will eventually contain full encrypted bootloader.

ATMega1284_Enc_noSCP

Performs encryption and decryption in CFB mode. Also computes hashes using CBC_MAC. Does not have any side channel resistance.

ATMega1284_Firm_Ex

Contains the simplest possible test code I could think of. It's just a 1Hz blink sketch. Used to test bootloader functionality.

ATMega1284_TRNG

Contains test code for a True Random Number Generator (TRNG) based on watchdog timer jitter. The code has been tested. The randomness has not.

ATMega1284_VarClk

Contains test code for a randomly varying clock for the ATMega. Switches back and forth between 1MHz and 8MHz successfully. Has functions for reading and writing from/to UART0 & UART1. Only the writing functions (and clock switching) have been tested. NOTE: UART will not work in anything but 8MHz mode. Prepare accordingly.

host_tools

Contains host tools being worked on by Luke, James, and Cameron.

Other

Blink_FW0.json

Example blink firmware for use by fw_update

2 README

3 Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

4 Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

5 File Index

5.1 File List

Here is a list of all files with brief descriptions:

6 Namespace Documentation

6.1 AES Namespace Reference

Functions

- def encryptFileAES
- def AESHash
- def decryptFileAES
- def generate128Key

6.1.1 Function Documentation

6.1.1.1 def AES.AESHash (*key, iv, input_file, output_file*)

6.1.1.2 def AES.decryptFileAES (*key, iv, input_file, output_file*)

```
Takes in a key, initialization vector, and a file location of the input, and location of the output
```

6.1.1.3 def AES.encryptFileAES (*key, iv, input_file, output_file*)

```
Takes in a key, initialization vector, and a file location of the input, and location of the output
```

6.1.1.4 def AES.generate128Key ( )

6.2 AESandRandNums Namespace Reference

Functions

- def encryptFileAES
- def AESHash
- def decryptFileAES
- def generate128Key

6.2.1 Function Documentation

6.2.1.1 def AESandRandNums.AESHash (*key, iv, input_file, output_file*)

6.2.1.2 def AESandRandNums.decryptFileAES (*key, iv, input_file, output_file*)

```
Takes in a key, initialization vector, and a file location of the input, and location of the output
```

6.2.1.3 def AESandRandNums.encryptFileAES (*key, iv, input_file, output_file*)

```
Takes in a key, initialization vector, and a file location of the input, and location of the output
```

6.2.1.4 def AESandRandNums.generate128Key ( )

6.3 bl_build Namespace Reference

Functions

- def grabKeys
- def CMACHash
- def bytesToHexList
- def bytesToCString
- def generate128Entropy
- def generate256Entropy
- def make_bootloader
- def copy_artifacts
- def write_fuse_file
- def generate_secrets
- def make_secrets_file
- def stripLine
- def addrOf
- def dataLen
- def genMem

Variables

- tuple FILE_DIR = os.path.abspath(os.path.dirname(__file__))
- tuple secrets = grabKeys()
- tuple bootFlash = genMem("flash.hex")
- tuple flashHash = CMACHash(secrets["H_KEY"],bootFlash)

6.3.1 Function Documentation

6.3.1.1 def bl_build.addrOf (*intHexLine*)

```
Extract the address bytes from an intel hex line.
```

6.3.1.2 def bl_build.bytesToCString (*data*)

6.3.1.3 def bl_build.bytesToHexList (*data*)

6.3.1.4 def bl_build.CMACHash (*key, inBytes*)

6.3.1.5 def bl_build.copy_artifacts ( )

```
Copy bootloader build artifacts into the host tools directory.
```

6.3.1.6 def bl_build.dataLen (*intHexLine*)

```
Extract the length of the true data section to be safe.
```

6.3.1.7 def bl_build.generate128Entropy ( )

6.3.1.8 def bl_build.generate256Entropy ( )

6.3.1.9 def bl_build.generate_secrets ( )

```
Generate the keys and Password.
```

6.3.1.10 def bl_build.genMem (*intHex*)

6.3.1.11 def bl_build.grabKeys ( )

6.3.1.12 def bl_build.make_bootloader (*password* = None)

```
Build the bootloader from source.
Return:
    True if successful, False otherwise.
```

6.3.1.13 def bl_build.make_secrets_file (*secrets*)

```
Construct secret_build_output.txt with all of the necessary keys and
passwords.
```

6.3.1.14 def bl_build.stripLine (*intelLine*)

```
Pulls the data out of a line of Intel hex and packs it as bytes.
```

6.3.1.15 def bl_build.write_fuse_file (*fuse_name, fuse_value*)

6.3.2 Variable Documentation

6.3.2.1 tuple bootFlash = genMem("flash.hex")

6.3.2.2 tuple FILE_DIR = os.path.abspath(os.path.dirname(__file__))

6.3.2.3 tuple flashHash = CMACHash(secrets["H_KEY"],bootFlash)

6.3.2.4 tuple secrets = grabKeys()

6.4 bl_configure Namespace Reference

Functions

- def generate_secret_file
- def configure_bootloader
- def grabKeys

Variables

- tuple FILE_PATH = os.path.abspath(__file__)
- tuple parser = argparse.ArgumentParser(description='Bootloader Config Tool')
- required = True)
- tuple args = parser.parse_args()
- tuple serial_port = serial.Serial(args.port)

6.4.1 Function Documentation

6.4.1.1 def bl_configure.configure_bootloader (*serial_port*)

```
Configure bootloader using serial connection.
```

6.4.1.2 def bl_configure.generate_secret_file ( )

```
Compile all secrets from build and configuration and store to secret file.
```

6.4.1.3 def bl_configure.grabKeys ( )

6.4.2 Variable Documentation

6.4.2.1 tuple args = parser.parse_args()

6.4.2.2 tuple FILE_PATH = os.path.abspath(__file__)

6.4.2.3 tuple parser = argparse.ArgumentParser(description='Bootloader Config Tool')

6.4.2.4 required = True)

6.4.2.5 tuple serial_port = serial.Serial(args.port)

6.5 bytesManipulators Namespace Reference

Functions

- def writeBytesToFile
- def appendBytesToFile
- def bytesToCString
- def bytesToHexList

6.5.1 Function Documentation

6.5.1.1 def bytesManipulators.appendBytesToFile (*data, targetName*)

6.5.1.2 def bytesManipulators.bytesToCString (*data*)

6.5.1.3 def bytesManipulators.bytesToHexList (*data*)

6.5.1.4 def bytesManipulators.writeBytesToFile (*data, targetName*)

6.6 fw_protect Namespace Reference

Functions

- def grabKeys
- def stripLine
- def CMACHash
- def encryptCBC
- def encryptAES

Variables

- tuple parser = argparse.ArgumentParser(description='Firmware Update Tool')
- string help = "Path to the firmware image to protect."
- required = True)
- tuple args = parser.parse_args()

- tuple keyMap = grabKeys()
- list finalList = []
- list firmwareSections = []
- tuple version = struct.pack(">h",int(args.version))
- tuple tempMSG = (args.message)
- string finalBytes = b''
- int padSize = 125
- tuple CBCHash = CMACHash(keyMap["H_KEY"],finalBytes)

6.6.1 Function Documentation

6.6.1.1 def fw_protect.CMACHash (*key, inBytes*)

6.6.1.2 def fw_protect.encryptAES (*key, iv, inBytes*)

```
Takes in a key, initialization vector, and a file location of the     input, and location of the output
```

6.6.1.3 def fw_protect.encryptCBC (*key, iv, inBytes, outfile*)

```
Takes in a key, initialization vector, and a file location of the     input, and location of the output
```

6.6.1.4 def fw_protect.grabKeys ( )

```
Parses secret_build_output.txt to read in all secret names and values.
These are stored in a dictionary.
```

6.6.1.5 def fw_protect.stripLine (*intelLine*)

6.6.2 Variable Documentation

6.6.2.1 tuple args = parser.parse_args()

6.6.2.2 tuple CBCHash = CMACHash(keyMap["H_KEY"],finalBytes)

6.6.2.3 tuple finalBytes = b''

6.6.2.4 list finalList = []

6.6.2.5 list firmwareSections = []

6.6.2.6 string help = "Path to the firmware image to protect."

6.6.2.7 tuple keyMap = grabKeys()

6.6.2.8 int padSize = 125

6.6.2.9 tuple parser = argparse.ArgumentParser(description='Firmware Update Tool')

6.6.2.10 required = True)

6.6.2.11 tuple tempMSG = (args.message)

6.6.2.12 tuple version = struct.pack(">h",int(args.version))

6.7 fw_update Namespace Reference

Variables

- string RESP_OK = b'\x06'
- tuple parser = argparse.ArgumentParser(description='Firmware Update Tool')
- required = True)
- string action = 'store_true'
- tuple args = parser.parse_args()
- tuple ser = serial.Serial(args.port, baudrate=115200, timeout=2)
- tuple chunk = firmware.read(256)
- int i = 0
- tuple resp = ser.read()

6.7.1 Variable Documentation

6.7.1.1 string action = 'store_true'

6.7.1.2 tuple args = parser.parse_args()

6.7.1.3 tuple chunk = firmware.read(256)

6.7.1.4 int i = 0

6.7.1.5 tuple parser = argparse.ArgumentParser(description='Firmware Update Tool')

6.7.1.6 required = True)

6.7.1.7 tuple resp = ser.read()

6.7.1.8 string RESP_OK = b'\x06'

6.7.1.9 tuple ser = serial.Serial(args.port, baudrate=115200, timeout=2)

6.8 readback Namespace Reference

Functions

- def CMACHash
- def encryptCBC
- def encryptAES
- def decryptAES
- def readSecrets
- def construct_request

Variables

- tuple secrets = readSecrets()
- tuple parser = argparse.ArgumentParser(description='Memory Readback Tool')
- required = True)
- tuple args = parser.parse_args()
- tuple request = construct_request(int(args.address), int(args.num_bytes))

- list SECRET_KEY = secrets['RB_KEY']
- list IV = secrets['RB_IV']
- list HASH_KEY = secrets['RBH_KEY']
- tuple request_hash = CMACHash(HASH_KEY, request)
- tuple ser = serial.Serial(args.port, baudrate=115200)
- tuple data = ser.read(int(args.num_bytes))
- list printable = ["{:02x}".format(ord(x)) for x in data]

6.8.1 Function Documentation

6.8.1.1 def readback.CMACHash (*key, inBytes*)

6.8.1.2 def readback.construct_request (*start_addr, num_bytes*)

```
Construct a request frame to send the the AVR.
The frame consists of a 24 byte password followed by the start address (4
bytes) and then the number of bytes to read (4 bytes).
```

6.8.1.3 def readback.decryptAES (*key, iv, inBytes*)

```
Takes in a key, initialization vector, and a file location of the     input, and location of the output
```

6.8.1.4 def readback.encryptAES (*key, iv, inBytes*)

```
Takes in a key, initialization vector, and a file location of the     input, and location of the output
```

6.8.1.5 def readback.encryptCBC (*key, iv, inBytes, outfile*)

```
Takes in a key, initialization vector, and a file location of the     input, and location of the output
```

6.8.1.6 def readback.readSecrets ( )

6.8.2 Variable Documentation

6.8.2.1 tuple args = parser.parse_args()

6.8.2.2 tuple data = ser.read(int(args.num_bytes))

6.8.2.3 list HASH_KEY = secrets['RBH_KEY']

6.8.2.4 list IV = secrets['RB_IV']

6.8.2.5 tuple parser = argparse.ArgumentParser(description='Memory Readback Tool')

6.8.2.6 list printable = ["{:02x}".format(ord(x)) for x in data]

6.8.2.7 tuple request = construct_request(int(args.address), int(args.num_bytes))

6.8.2.8 tuple request_hash = CMACHash(HASH_KEY, request)

6.8.2.9 required = True)

6.8.2.10 list SECRET_KEY = secrets['RB_KEY']

6.8.2.11 tuple secrets = readSecrets()

6.8.2.12 tuple ser = serial.Serial(args.port, baudrate=115200)

6.9 test_grab Namespace Reference

Functions

- def grabKeys

6.9.1 Function Documentation

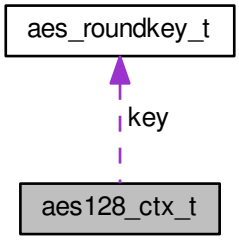
6.9.1.1 def test_grab.grabKeys ( )

7 Data Structure Documentation

7.1 aes128_ctx_t Struct Reference

`#include <aes_types.h>`

Collaboration diagram for aes128_ctx_t:



Data Fields

- aes_roundkey_t key [10+1]

7.1.1 Field Documentation

7.1.1.1 aes_roundkey_t key[10+1]

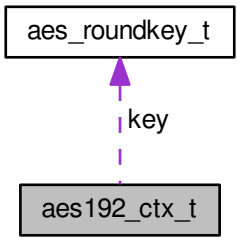
The documentation for this struct was generated from the following file:

- ATMega1284P_Boot/ATMega1284_Boot/AES_lib/aes_types.h

7.2 aes192_ctx_t Struct Reference

`#include <aes_types.h>`

Collaboration diagram for aes192_ctx_t:



Data Fields

- aes_roundkey_t key [12+1]

7.2.1 Field Documentation

7.2.1.1 aes_roundkey_t key[12+1]

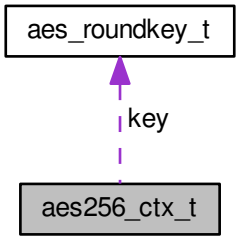
The documentation for this struct was generated from the following file:

- ATMega1284P_Boot/ATMega1284_Boot/AES_lib/aes_types.h

7.3 aes256_ctx_t Struct Reference

`#include <aes_types.h>`

Collaboration diagram for aes256_ctx_t:



Data Fields

- aes_roundkey_t key [14+1]

7.3.1 Field Documentation

7.3.1.1 aes_roundkey_t key[14+1]

The documentation for this struct was generated from the following file:

- ATMega1284P_Boot/ATMega1284_Boot/AES_lib/aes_types.h

7.4 aes_cipher_state_t Struct Reference

```
#include <aes_types.h>
```

Data Fields

- uint8_t s [16]

7.4.1 Field Documentation

7.4.1.1 uint8_t s[16]

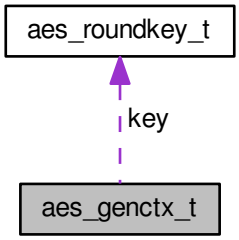
The documentation for this struct was generated from the following file:

- ATMega1284P_Boot/ATMega1284_Boot/AES_lib/aes_types.h

7.5 aes_genctx_t Struct Reference

```
#include <aes_types.h>
```

Collaboration diagram for aes_genctx_t:



Data Fields

- aes_roundkey_t key [1]

7.5.1 Field Documentation

7.5.1.1 aes_roundkey_t key[1]

The documentation for this struct was generated from the following file:

- ATMega1284P_Boot/ATMega1284_Boot/AES_lib/aes_types.h

7.6 aes_roundkey_t Struct Reference

`#include <aes_types.h>`

Data Fields

- uint8_t ks [16]

7.6.1 Field Documentation

7.6.1.1 uint8_t ks[16]

The documentation for this struct was generated from the following file:

- ATMega1284P_Boot/ATMega1284_Boot/AES_lib/aes_types.h

7.7 keysize_desc_arg_range_t Struct Reference

`#include <keysize_descriptor.h>`

Data Fields

- uint16_t min
- uint16_t max
- uint16_t distance
- uint16_t offset

7.7.1 Field Documentation

7.7.1.1 uint16_t distance

7.7.1.2 uint16_t max

7.7.1.3 uint16_t min

7.7.1.4 uint16_t offset

The documentation for this struct was generated from the following file:

- ATMega1284P_Boot/ATMega1284_Boot/AES_lib/keysize_descriptor.h

7.8 keysize_desc_list_t Struct Reference

```
#include <keysize_descriptor.h>
```

Data Fields

- uint8_t n_items
- uint16_t items []

7.8.1 Field Documentation

7.8.1.1 uint16_t items[]

7.8.1.2 uint8_t n_items

The documentation for this struct was generated from the following file:

- ATMega1284P_Boot/ATMega1284_Boot/AES_lib/keysize_descriptor.h

7.9 keysize_desc_range_t Struct Reference

```
#include <keysize_descriptor.h>
```

Data Fields

- uint16_t min
- uint16_t max

7.9.1 Field Documentation

7.9.1.1 uint16_t max

7.9.1.2 uint16_t min

The documentation for this struct was generated from the following file:

- ATMega1284P_Boot/ATMega1284_Boot/AES_lib/keysize_descriptor.h

8 File Documentation

8.1 AESandRandNums.py File Reference

Namespaces

- AESandRandNums

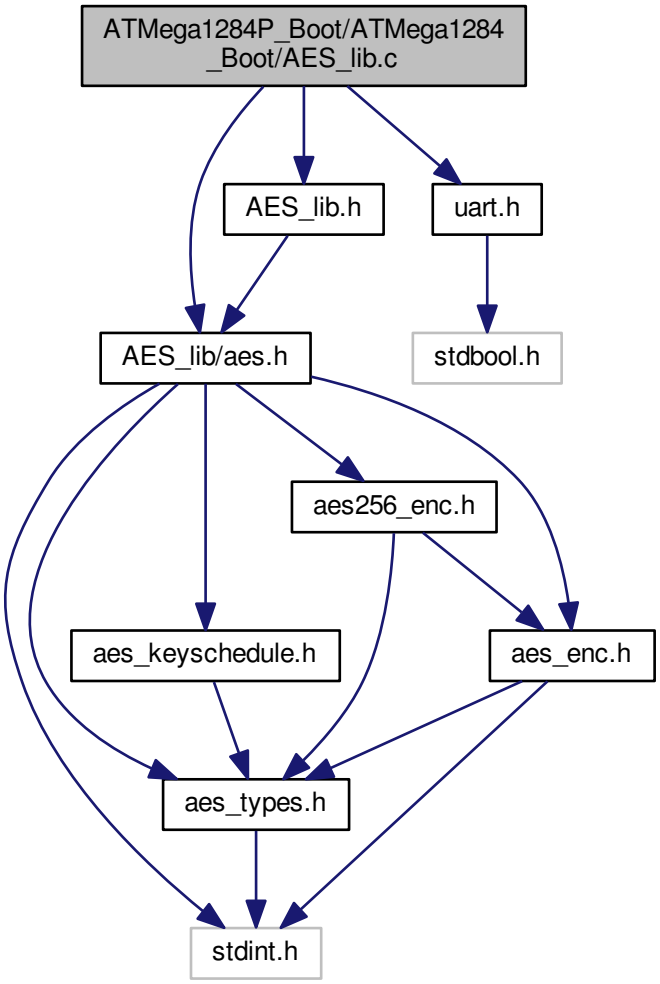
Functions

- def encryptFileAES
- def AESHash
- def decryptFileAES
- def generate128Key

8.2 ATMega1284P_Boot/ATMega1284_Boot/AES_lib.c File Reference

```
#include "AES_lib.h"
#include "AES_lib/aes.h"
#include "uart.h"
```

Include dependency graph for AES_lib.c:



Functions

- void strtEncCFB (uint8_t *key, uint8_t *firstBlockPlaintext, uint8_t *IV, aes256_ctx_t *ctx, uint8_t *firstBlock↩ Ciphertext)

  Begins encryption using AES-256 in CFB Mode on a single block.
- void contEncCFB (aes256_ctx_t *ctx, uint8_t *nextBlockPlaintext, uint8_t *prevBlockCiphertext, uint8_t *next↩ BlockCiphertext)

  Continues encryption using AES-256 in CFB Mode on a single block.
- void encCFB (uint8_t *key, uint8_t *data, uint8_t *IV, uint16_t size)

  Encrypts data in-place using AES-256 in CFB Mode.
- void strtDecCFB (uint8_t *key, uint8_t *firstBlockCiphertext, uint8_t *IV, aes256_ctx_t *ctx, uint8_t *firstBlock↩ Plaintext)

Begins decryption using AES-256 in CFB Mode on a single block.

- void contDecCFB (aes256_ctx_t *ctx, uint8_t *nextBlockCiphertext, uint8_t *prevBlockCiphertext, uint8_t *next↩ BlockPlaintext)

  Continues decryption using AES-256 in CFB Mode on a single block.

- void decCFB (uint8_t *key, uint8_t *data, uint8_t *IV, uint16_t size)

  Decrypts data in-place using AES-256 in CFB Mode.

8.2.1 Detailed Description

Author

Patrick Dunham

Date

2017-02-26 GPLv3 or later

8.2.2 Function Documentation

8.2.2.1 void contDecCFB (aes256_ctx_t * *ctx*, uint8_t * *nextBlockCiphertext*, uint8_t * *prevBlockCiphertext*, uint8_t * *nextBlockPlaintext*)

Continues decryption using AES-256 in CFB Mode on a single block.

This method uses a standard AES-256 encryption kernel to decrypt data in CFB Mode. This essentially turns the block cipher into a self-correcting stream cipher. Unlike other block cipher modes, the decryption method can also use the AES-256 encryption kernel, saving code space. The data array byte count MUST be divisible by 16. This should have been accounted for in the encryption function.

Parameters

ctx	Pointer to AES-256 Keyschedule.
nextBlock↩ Ciphertext	Pointer to data holding ciphertext block to be decrypted.
prevBlock↩ Ciphertext	Pointer to data holding previous block of ciphertext
nextBlock↩ Plaintext	Pointer to memory that will hold next block of plaintext

8.2.2.2 void contEncCFB (aes256_ctx_t * *ctx*, uint8_t * *nextBlockPlaintext*, uint8_t * *prevBlockCiphertext*, uint8_t * *nextBlockCiphertext*)

Continues encryption using AES-256 in CFB Mode on a single block.

This method uses a standard AES-256 encryption kernel to encrypt data in CFB Mode. This essentially turns the block cipher into a self-correcting stream cipher. In addition, the corresponding decryption method can also use the AES-256 encryption kernel, saving code space. The data array byte count MUST be divisible by 16. This can be achieved through padding.

Parameters

ctx	Pointer to AES-256 Keyschedule.
nextBlock↩ Plaintext	Pointer to data holding plaintext block to be encrypted.
prevBlock↩ Ciphertext	Pointer to data holding previous block of ciphertext
nextBlock↩ Ciphertext	Pointer to memory that will hold next block of ciphertext

8.2.2.3 void decCFB (uint8_t * *key,* uint8_t * *data,* uint8_t * *IV,* uint16_t *size*)

Decrypts data in-place using AES-256 in CFB Mode.

This method uses a standard AES-256 encryption kernel to decrypt data in CFB Mode. This essentially turns the block cipher into a self-correcting stream cipher. Unlike other block cipher modes, the decryption method can also use the AES-256 encryption kernel, saving code space. The code will be decrypted block by block and stored in the same memory space used for the ciphertext. This results in drastic memory savings. The data array byte count MUST be divisible by 16. This should have been accounted for in the encryption function.

Parameters

key	Pointer to 32-byte array containing the AES-256 key.
data	Pointer to data array. Begins as ciphertext, ends as plaintext.
IV	Pointer to a 16-byte random Initialization Vector
size	Size in bytes of data array. Must be divisible by 16.

8.2.2.4 void encCFB (uint8_t * *key,* uint8_t * *data,* uint8_t * *IV,* uint16_t *size*)

Encrypts data in-place using AES-256 in CFB Mode.

This method uses a standard AES-256 encryption kernel to encrypt data in CFB Mode. This essentially turns the block cipher into a self-correcting stream cipher. In addition, the corresponding decryption method can also use the AES-256 encryption kernel, saving code space. The code will be encrypted block by block and stored in the same memory space used for the plaintext. This results in drastic memory savings. The data array byte count MUST be divisible by 16. This can be achieved through padding.

Parameters

key	Pointer to 32-byte array containing the AES-256 key.
data	Pointer to data array. Begins as plaintext, ends as ciphertext.
IV	Pointer to a 16-byte random Initialization Vector
size	Size in bytes of data array. Must be divisible by 16.

8.2.2.5 void strtDecCFB (uint8_t * *key,* uint8_t * *firstBlockCiphertext,* uint8_t * *IV,* **aes256_ctx_t** * *ctx,* uint8_t * *firstBlockPlaintext*)

Begins decryption using AES-256 in CFB Mode on a single block.

This method uses a standard AES-256 encryption kernel to decrypt data in CFB Mode. This essentially turns the block cipher into a self-correcting stream cipher. Unlike other block cipher modes, the decryption method can also use the AES-256 encryption kernel, saving code space. The data array byte count MUST be divisible by 16. This should have been accounted for in the encryption function.

Parameters

key	Pointer to 32-byte AES-256 key.
firstBlock↩ Ciphertext	Pointer to data holding ciphertext block to be decrypted.
IV	Pointer to 16-byte Initialization Vector
ctx	Pointer to AES-256 Keyschedule.
firstBlockPlaintext	Pointer to memory that will hold block of plaintext

8.2.2.6 void strtEncCFB (uint8_t * *key*, uint8_t * *firstBlockPlaintext*, uint8_t * *IV*, aes256_ctx_t * *ctx*, uint8_t * *firstBlockCiphertext*)

Begins encryption using AES-256 in CFB Mode on a single block.

This method uses a standard AES-256 encryption kernel to encrypt data in CFB Mode. This essentially turns the block cipher into a self-correcting stream cipher. In addition, the corresponding decryption method can also use the AES-256 encryption kernel, saving code space. The data array byte count MUST be divisible by 16. This can be achieved through padding.

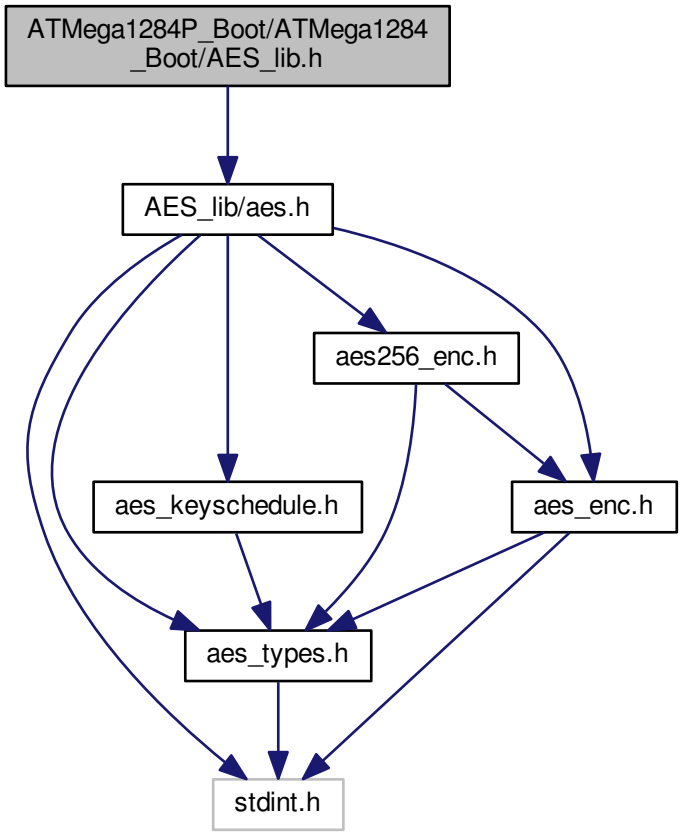
Parameters

key	Pointer to 32-byte AES-256 key.
firstBlockPlaintext	Pointer to data holding plaintext block to be encrypted.
IV	Pointer to 16-byte Initialization Vector
ctx	Pointer to AES-256 Keyschedule.
firstBlock↩ Ciphertext	Pointer to memory that will hold block of ciphertext

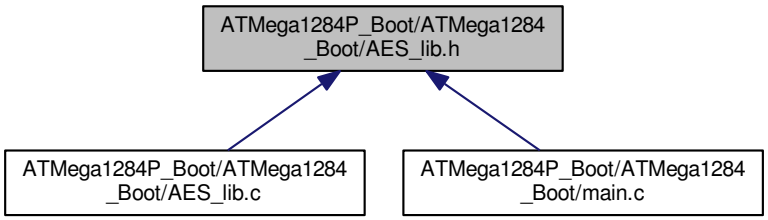
8.3 ATMega1284P_Boot/ATMega1284_Boot/AES_lib.h File Reference

```
#include "AES_lib/aes.h"
```

Include dependency graph for AES_lib.h:



This graph shows which files directly or indirectly include this file:



Functions

- void encCFB (uint8_t *key, uint8_t *data, uint8_t *IV, uint16_t size)

  Encrypts data in-place using AES-256 in CFB Mode.
- void strtEncCFB (uint8_t *key, uint8_t *firstBlockPlaintext, uint8_t *IV, aes256_ctx_t *ctx, uint8_t *firstBlock↩ Ciphertext)

  Begins encryption using AES-256 in CFB Mode on a single block.
- void contEncCFB (aes256_ctx_t *ctx, uint8_t *nextBlockPlaintext, uint8_t *prevBlockCiphertext, uint8_t *next↩ BlockCiphertext)

  Continues encryption using AES-256 in CFB Mode on a single block.
- void decCFB (uint8_t *key, uint8_t *data, uint8_t *IV, uint16_t size)

  Decrypts data in-place using AES-256 in CFB Mode.
- void strtDecCFB (uint8_t *key, uint8_t *firstBlockCiphertext, uint8_t *IV, aes256_ctx_t *ctx, uint8_t *firstBlock↩ Plaintext)

  Begins decryption using AES-256 in CFB Mode on a single block.
- void contDecCFB (aes256_ctx_t *ctx, uint8_t *nextBlockCiphertext, uint8_t *prevBlockCiphertext, uint8_t *next↩ BlockPlaintext)

  Continues decryption using AES-256 in CFB Mode on a single block.
- void hashCBC (uint8_t *key, uint8_t *data, uint8_t *hash, uint16_t size)

8.3.1 Detailed Description

Author

Patrick Dunham

Date

2017-02-26 GPLv3 or later

8.3.2 Function Documentation

8.3.2.1 void contDecCFB (aes256_ctx_t * *ctx*, uint8_t * *nextBlockCiphertext*, uint8_t * *prevBlockCiphertext*, uint8_t * *nextBlockPlaintext*)

Continues decryption using AES-256 in CFB Mode on a single block.

This method uses a standard AES-256 encryption kernel to decrypt data in CFB Mode. This essentially turns the block cipher into a self-correcting stream cipher. Unlike other block cipher modes, the decryption method can also use the AES-256 encryption kernel, saving code space. The data array byte count MUST be divisible by 16. This should have been accounted for in the encryption function.

Parameters

ctx	Pointer to AES-256 Keyschedule.
nextBlock↩ Ciphertext	Pointer to data holding ciphertext block to be decrypted.

prevBlock↩ Ciphertext	Pointer to data holding previous block of ciphertext
nextBlock↩ Plaintext	Pointer to memory that will hold next block of plaintext

8.3.2.2 void contEncCFB (aes256_ctx_t * *ctx*, uint8_t * *nextBlockPlaintext*, uint8_t * *prevBlockCiphertext*, uint8_t * *nextBlockCiphertext*)

Continues encryption using AES-256 in CFB Mode on a single block.

This method uses a standard AES-256 encryption kernel to encrypt data in CFB Mode. This essentially turns the block cipher into a self-correcting stream cipher. In addition, the corresponding decryption method can also use the AES-256 encryption kernel, saving code space. The data array byte count MUST be divisible by 16. This can be achieved through padding.

Parameters

ctx	Pointer to AES-256 Keyschedule.
nextBlock↩ Plaintext	Pointer to data holding plaintext block to be encrypted.
prevBlock↩ Ciphertext	Pointer to data holding previous block of ciphertext
nextBlock↩ Ciphertext	Pointer to memory that will hold next block of ciphertext

8.3.2.3 void decCFB (uint8_t * *key*, uint8_t * *data*, uint8_t * *IV*, uint16_t *size*)

Decrypts data in-place using AES-256 in CFB Mode.

This method uses a standard AES-256 encryption kernel to decrypt data in CFB Mode. This essentially turns the block cipher into a self-correcting stream cipher. Unlike other block cipher modes, the decryption method can also use the AES-256 encryption kernel, saving code space. The code will be decrypted block by block and stored in the same memory space used for the ciphertext. This results in drastic memory savings. The data array byte count MUST be divisible by 16. This should have been accounted for in the encryption function.

Parameters

key	Pointer to 32-byte array containing the AES-256 key.
data	Pointer to data array. Begins as ciphertext, ends as plaintext.
IV	Pointer to a 16-byte random Initialization Vector
size	Size in bytes of data array. Must be divisible by 16.

8.3.2.4 void encCFB (uint8_t * *key*, uint8_t * *data*, uint8_t * *IV*, uint16_t *size*)

Encrypts data in-place using AES-256 in CFB Mode.

This method uses a standard AES-256 encryption kernel to encrypt data in CFB Mode. This essentially turns the block cipher into a self-correcting stream cipher. In addition, the corresponding decryption method can also use the AES-256 encryption kernel, saving code space. The code will be encrypted block by block and stored in the same memory space used for the plaintext. This results in drastic memory savings. The data array byte count MUST be divisible by 16. This can be achieved through padding.

Parameters

key	Pointer to 32-byte array containing the AES-256 key.
data	Pointer to data array. Begins as plaintext, ends as ciphertext.
IV	Pointer to a 16-byte random Initialization Vector
size	Size in bytes of data array. Must be divisible by 16.

8.3.2.5 void hashCBC (uint8_t * *key*, uint8_t * *data*, uint8_t * *hash*, uint16_t *size*)

8.3.2.6 void strtDecCFB (uint8_t * *key*, uint8_t * *firstBlockCiphertext*, uint8_t * *IV*, aes256_ctx_t * *ctx*, uint8_t * *firstBlockPlaintext*)

Begins decryption using AES-256 in CFB Mode on a single block.

This method uses a standard AES-256 encryption kernel to decrypt data in CFB Mode. This essentially turns the block cipher into a self-correcting stream cipher. Unlike other block cipher modes, the decryption method can also use the AES-256 encryption kernel, saving code space. The data array byte count MUST be divisible by 16. This should have been accounted for in the encryption function.

Parameters

key	Pointer to 32-byte AES-256 key.
firstBlock↩ Ciphertext	Pointer to data holding ciphertext block to be decrypted.
IV	Pointer to 16-byte Initialization Vector
ctx	Pointer to AES-256 Keyschedule.
firstBlockPlaintext	Pointer to memory that will hold block of plaintext

8.3.2.7 void strtEncCFB (uint8_t * *key*, uint8_t * *firstBlockPlaintext*, uint8_t * *IV*, aes256_ctx_t * *ctx*, uint8_t * *firstBlockCiphertext*)

Begins encryption using AES-256 in CFB Mode on a single block.

This method uses a standard AES-256 encryption kernel to encrypt data in CFB Mode. This essentially turns the block cipher into a self-correcting stream cipher. In addition, the corresponding decryption method can also use the AES-256 encryption kernel, saving code space. The data array byte count MUST be divisible by 16. This can be achieved through padding.

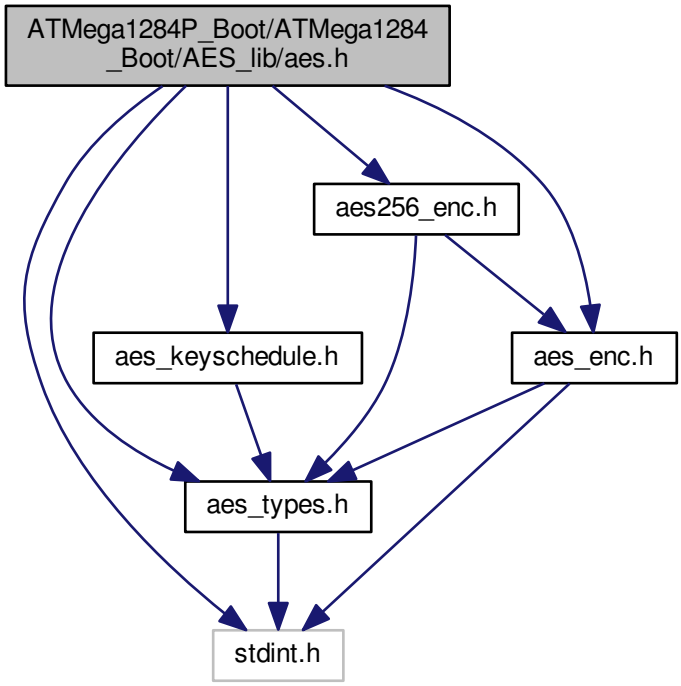
Parameters

key	Pointer to 32-byte AES-256 key.
firstBlockPlaintext	Pointer to data holding plaintext block to be encrypted.
IV	Pointer to 16-byte Initialization Vector
ctx	Pointer to AES-256 Keyschedule.
firstBlock↩ Ciphertext	Pointer to memory that will hold block of ciphertext

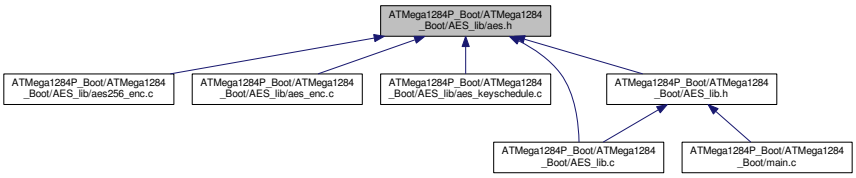
8.4 ATMega1284P_Boot/ATMega1284_Boot/AES_lib/aes.h File Reference

```
#include <stdint.h>
#include "aes_types.h"
#include "aes256_enc.h"
#include "aes_enc.h"
#include "aes_keyschedule.h"
```

Include dependency graph for aes.h:



This graph shows which files directly or indirectly include this file:



8.4.1 Detailed Description

bg@nerilex.org

Author

Daniel Otte

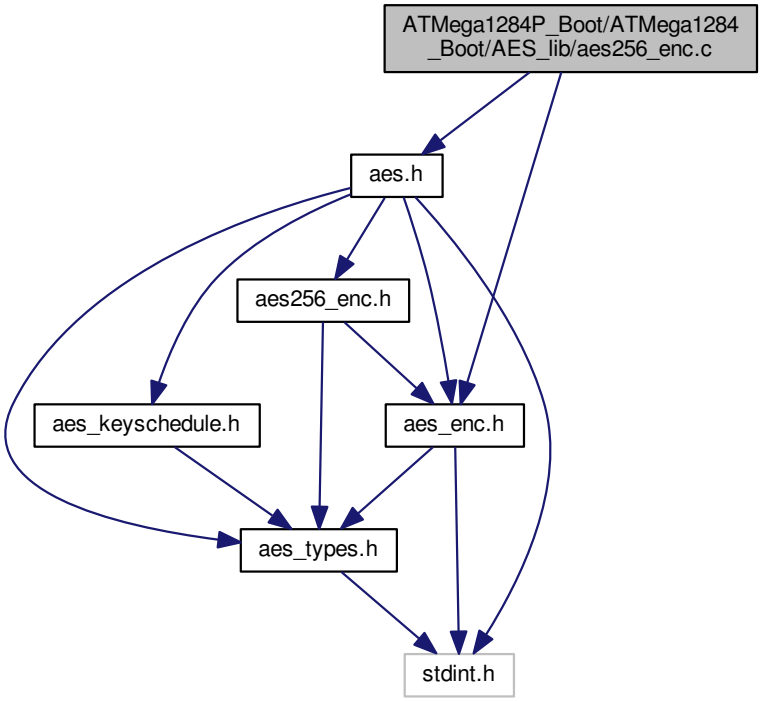
Date

2008-12-30 GPLv3 or later

8.5 ATMega1284P_Boot/ATMega1284_Boot/AES_lib/aes256_enc.c File Reference

```
#include "aes.h"
#include "aes_enc.h"
```

Include dependency graph for aes256_enc.c:



Functions

- void aes256_enc (void *buffer, aes256_ctx_t *ctx)

  encrypt with 256 bit key.

8.5.1 Detailed Description

bg@nerilex.org

Author

Daniel Otte

Date

2008-12-31 GPLv3 or later

8.5.2 Function Documentation

8.5.2.1 void aes256_enc (void * *buffer*, aes256_ctx_t * *ctx*)

encrypt with 256 bit key.

This function encrypts one block with the AES algorithm under control of a keyschedule produced from a 256 bit key.

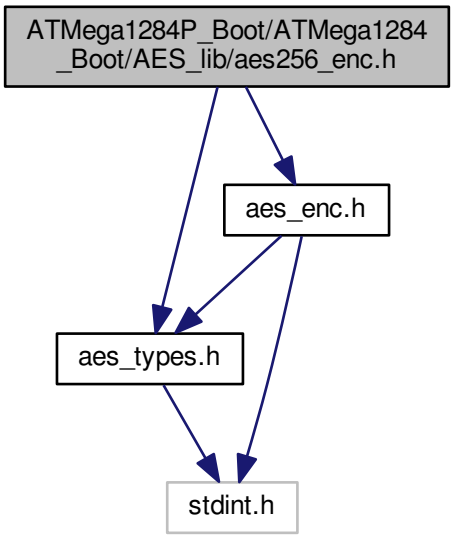
Parameters

buffer	pointer to the block to encrypt
ctx	pointer to the key schedule

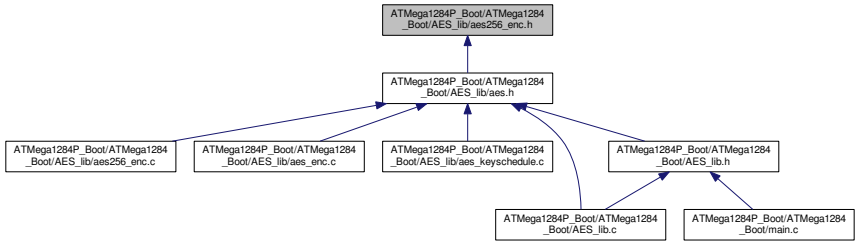
8.6 ATMega1284P_Boot/ATMega1284_Boot/AES_lib/aes256_enc.h File Reference

```
#include "aes_types.h"
#include "aes_enc.h"
```

Include dependency graph for aes256_enc.h:



This graph shows which files directly or indirectly include this file:



Functions

- void aes256_enc (void *buffer, aes256_ctx_t *ctx)
  encrypt with 256 bit key.

8.6.1 Detailed Description

bg@nerilex.org

Author

Daniel Otte

Date

2008-12-31 GPLv3 or later

8.6.2 Function Documentation

8.6.2.1 void aes256_enc (void * *buffer*, aes256_ctx_t * *ctx*)

encrypt with 256 bit key.

This function encrypts one block with the AES algorithm under control of a keyschedule produced from a 256 bit key.

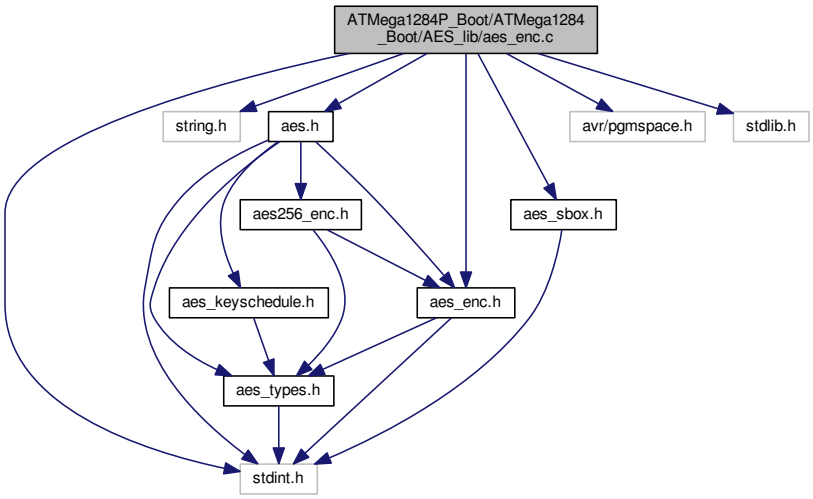
Parameters

buffer	pointer to the block to encrypt
ctx	pointer to the key schedule

8.7 ATMega1284P_Boot/ATMega1284_Boot/AES_lib/aes_enc.c File Reference

```
#include <stdint.h>
#include <string.h>
#include "aes.h"
#include "aes_sbox.h"
#include "aes_enc.h"
#include <avr/pgmspace.h>
#include <stdlib.h>
```

Include dependency graph for aes_enc.c:



Macros

- #define NUM_DUMMY_OP 5

Functions

- void aes_shiftcol (void *data, uint8_t shift)
- uint8_t xtime (uint8_t x)
- void aes_encrypt_core (aes_cipher_state_t *state, const aes_genctx_t *ks, uint8_t rounds)

8.7.1 Detailed Description

bg@nerilex.org

Author

Daniel Otte

Date

2008-12-30 GPLv3 or later

8.7.2 Macro Definition Documentation

8.7.2.1 #define NUM_DUMMY_OP 5

8.7.3 Function Documentation

8.7.3.1 void aes_encrypt_core (aes_cipher_state_t * *state*, const aes_genctx_t * *ks*, uint8_t *rounds*)

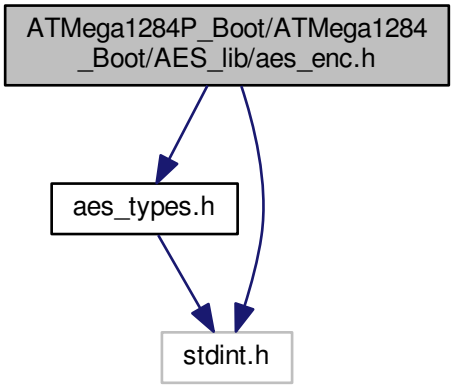
8.7.3.2 void aes_shiftcol (void * *data*, uint8_t *shift*)

8.7.3.3 uint8_t xtime (uint8_t *x*)

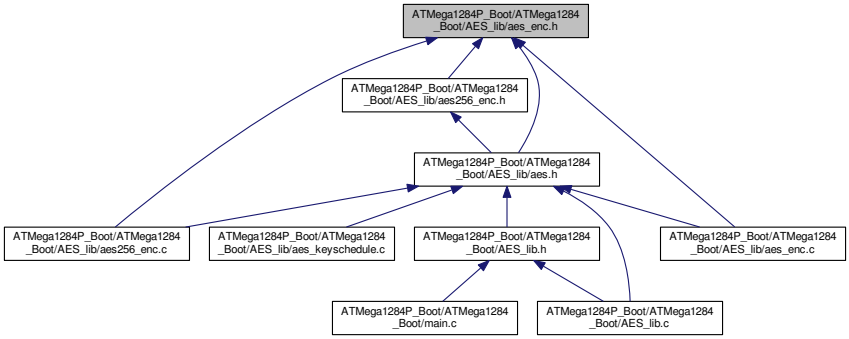
8.8 ATMega1284P_Boot/ATMega1284_Boot/AES_lib/aes_enc.h File Reference

```
#include "aes_types.h"
#include <stdint.h>
```

Include dependency graph for aes_enc.h:



This graph shows which files directly or indirectly include this file:



Functions

- void aes_encrypt_core (aes_cipher_state_t *state, const aes_genctx_t *ks, uint8_t rounds)

8.8.1 Detailed Description

bg@nerilex.org

Author

Daniel Otte

Date

2008-12-30 GPLv3 or later

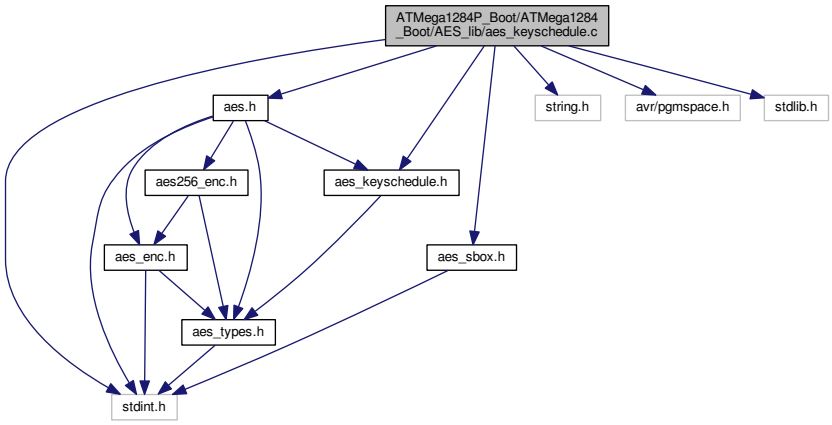
8.8.2 Function Documentation

8.8.2.1 void aes_encrypt_core (aes_cipher_state_t * *state*, const aes_genctx_t * *ks*, uint8_t *rounds*)

8.9 ATMega1284P_Boot/ATMega1284_Boot/AES_lib/aes_keyschedule.c File Reference

```
#include <stdint.h>
#include "aes.h"
#include "aes_keyschedule.h"
#include "aes_sbox.h"
#include <string.h>
#include <avr/pgmspace.h>
#include <stdlib.h>
```

Include dependency graph for aes_keyschedule.c:



Macros

- #define NUM_DUMMY_OP 5

Functions

- void aes_init (const void *key, uint16_t keysize_b, aes_genctx_t *ctx)

initialize the keyschedule

- void aes256_init (const void *key, aes256_ctx_t *ctx)

  initialize the keyschedule for 256 bit key

Variables

- const uint8_t rc_tab[] PROGMEM

8.9.1 Detailed Description

bg@nerilex.org

Author

Daniel Otte

Date

2008-12-30 GPLv3 or later

8.9.2 Macro Definition Documentation

8.9.2.1 #define NUM_DUMMY_OP 5

8.9.3 Function Documentation

8.9.3.1 void aes256_init (const void * *key*, aes256_ctx_t * *ctx*)

initialize the keyschedule for 256 bit key

This function computes the keyschedule from a given 256 bit key and stores it in the context variable

Parameters

key	pointer to the key material
ctx	pointer to the context where the keyschedule should be stored

8.9.3.2 void aes_init (const void * *key*, uint16_t *keysize_b*, aes_genctx_t * *ctx*)

initialize the keyschedule

This function computes the keyschedule from a given key with a given length and stores it in the context variable

Parameters

key	pointer to the key material
keysize_b	length of the key in bits (valid are 128, 192 and 256)
ctx	pointer to the context where the keyschedule should be stored

8.9.4 Variable Documentation

8.9.4.1 const uint8_t rc_tab [] PROGMEM

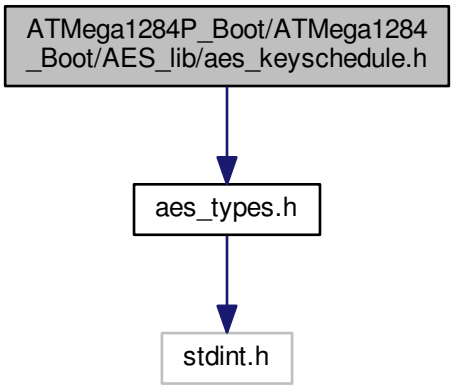
Initial value:

```
= { 0x01, 0x02, 0x04, 0x08,
        0x10, 0x20, 0x40, 0x80,
        0x1b, 0x36 }
```

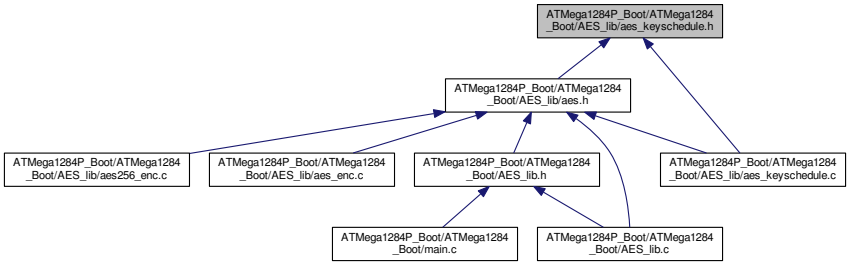
8.10 ATMega1284P_Boot/ATMega1284_Boot/AES_lib/aes_keyschedule.h File Reference

```
#include "aes_types.h"
```

Include dependency graph for aes_keyschedule.h:



This graph shows which files directly or indirectly include this file:



Functions

- void aes_init (const void *key, uint16_t keysize_b, aes_genctx_t *ctx)

  initialize the keyschedule
- void aes128_init (const void *key, aes128_ctx_t *ctx)

  initialize the keyschedule for 128 bit key
- void aes192_init (const void *key, aes192_ctx_t *ctx)

  initialize the keyschedule for 192 bit key

- void aes256_init (const void *key, aes256_ctx_t *ctx)
  initialize the keyschedule for 256 bit key

8.10.1 Detailed Description

bg@nerilex.org

Author

Daniel Otte

Date

2008-12-30 GPLv3 or later

8.10.2 Function Documentation

8.10.2.1 void aes128_init (const void * *key*, aes128_ctx_t * *ctx*)

initialize the keyschedule for 128 bit key

This function computes the keyschedule from a given 128 bit key and stores it in the context variable

Parameters

key	pointer to the key material
ctx	pointer to the context where the keyschedule should be stored

8.10.2.2 void aes192_init (const void * *key*, aes192_ctx_t * *ctx*)

initialize the keyschedule for 192 bit key

This function computes the keyschedule from a given 192 bit key and stores it in the context variable

Parameters

key	pointer to the key material
ctx	pointer to the context where the keyschedule should be stored

8.10.2.3 void aes256_init (const void * *key*, aes256_ctx_t * *ctx*)

initialize the keyschedule for 256 bit key

This function computes the keyschedule from a given 256 bit key and stores it in the context variable

Parameters

key	pointer to the key material
ctx	pointer to the context where the keyschedule should be stored

8.10.2.4 void aes_init (const void * *key*, uint16_t *keysize_b*, aes_genctx_t * *ctx*)

initialize the keyschedule

This function computes the keyschedule from a given key with a given length and stores it in the context variable

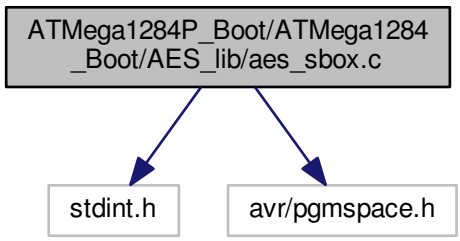
Parameters

key	pointer to the key material
keysize_b	length of the key in bits (valid are 128, 192 and 256)
ctx	pointer to the context where the keyschedule should be stored

8.11 ATMega1284P_Boot/ATMega1284_Boot/AES_lib/aes_sbox.c File Reference

```
#include <stdint.h>
#include <avr/pgmspace.h>
```

Include dependency graph for aes_sbox.c:



Variables

- const uint8_t aes_sbox[256] PROGMEM

8.11.1 Variable Documentation

8.11.1.1 const uint8_t aes_sbox [256] PROGMEM

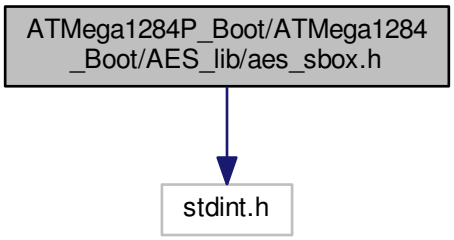
Initial value:

```
= {
 0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
 0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
 0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
 0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
 0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
 0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
 0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
 0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
 0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
 0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
 0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
 0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
 0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
 0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
 0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
 0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16
}
```

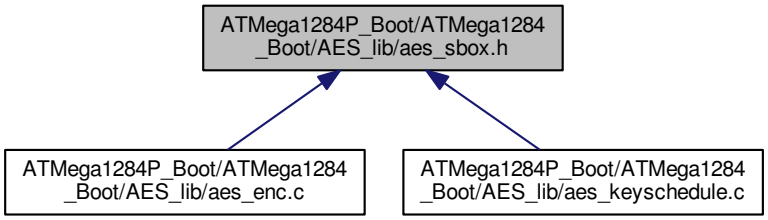
8.12 ATMega1284P_Boot/ATMega1284_Boot/AES_lib/aes_sbox.h File Reference

```
#include <stdint.h>
```

Include dependency graph for aes_sbox.h:



This graph shows which files directly or indirectly include this file:



Variables

- uint8_t aes_sbox [ ]

8.12.1 Detailed Description

bg@nerilex.org

Author

Daniel Otte

Date

2008-12-30 GPLv3 or later

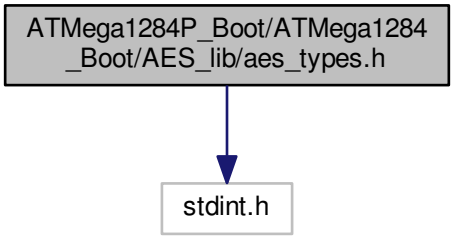
8.12.2 Variable Documentation

8.12.2.1 uint8_t aes_sbox[ ]

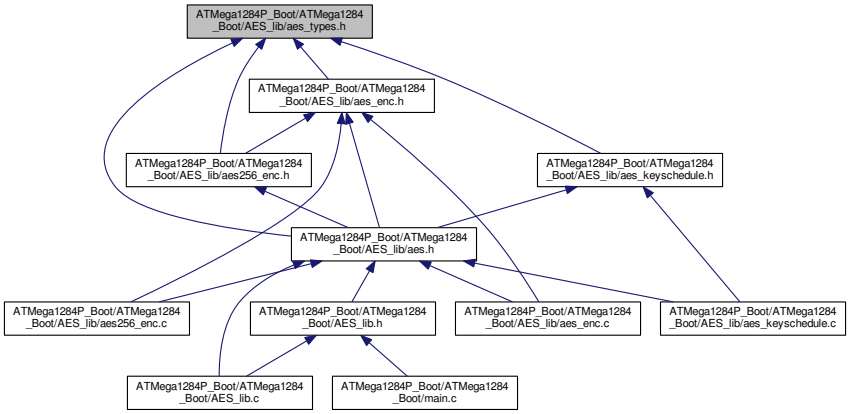
8.13 ATMega1284P_Boot/ATMega1284_Boot/AES_lib/aes_types.h File Reference

```
#include <stdint.h>
```

Include dependency graph for aes_types.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct aes_roundkey_t
- struct aes128_ctx_t
- struct aes192_ctx_t
- struct aes256_ctx_t
- struct aes_genctx_t
- struct aes_cipher_state_t

8.13.1 Detailed Description

bg@nerilex.org

Author

Daniel Otte

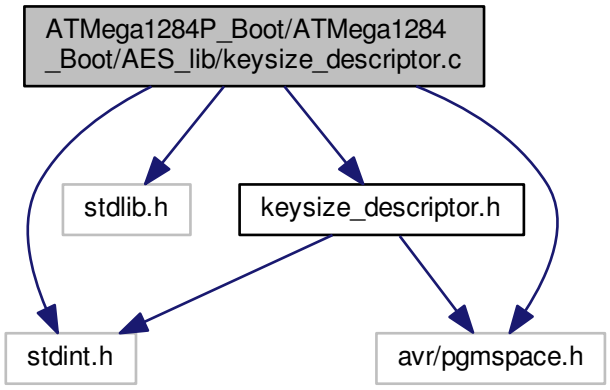
Date

2008-12-30 GPLv3 or later

8.14 ATMega1284P_Boot/ATMega1284_Boot/AES_lib/keysize_descriptor.c File Reference

```
#include <stdint.h>
#include <stdlib.h>
#include <avr/pgmspace.h>
#include "keysize_descriptor.h"
```

Include dependency graph for keysize_descriptor.c:



Functions

- uint8_t is_valid_keysize_P (PGM_VOID_P ks_desc, uint16_t keysize)
- uint16_t get_keysize (PGM_VOID_P ks_desc)
- uint16_t get_keysizes (PGM_VOID_P ks_desc, uint16_t **list)

8.14.1 Detailed Description

Author

Daniel Otte daniel.otte@rub.de

Date

2009-01-07 GPLv3 or later

8.14.2 Function Documentation

8.14.2.1 uint16_t get_keysize (PGM_VOID_P *ks_desc*)

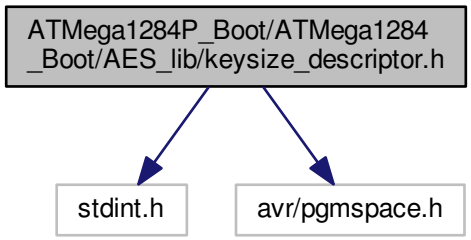
8.14.2.2 uint16_t get_keysizes (PGM_VOID_P *ks_desc,* uint16_t ** *list*)

8.14.2.3 uint8_t is_valid_keysize_P (PGM_VOID_P *ks_desc,* uint16_t *keysize*)

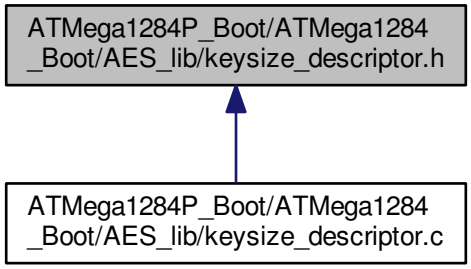
8.15 ATMega1284P_Boot/ATMega1284_Boot/AES_lib/keysize_descriptor.h File Reference

```
#include <stdint.h>
#include <avr/pgmspace.h>
```

Include dependency graph for keysize_descriptor.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct keysize_desc_list_t
- struct keysize_desc_range_t
- struct keysize_desc_arg_range_t

Macros

- #define KS_TYPE_TERMINATOR 0x00
- #define KS_TYPE_LIST 0x01
- #define KS_TYPE_RANGE 0x02
- #define KS_TYPE_ARG_RANGE 0x03
- #define KS_INT(a) ((a)&0xFF), ((a)>>8)

Functions

- uint8_t is_valid_keysize_P (PGM_VOID_P ks_desc, uint16_t keysize)
- uint16_t get_keysize (PGM_VOID_P ks_desc)
- uint16_t get_keysizes (PGM_VOID_P ks_desc, uint16_t **list)

8.15.1 Detailed Description

Author

Daniel Otte daniel.otte@rub.de

Date

2009-01-07 GPLv3 or later

8.15.2 Macro Definition Documentation

8.15.2.1 #define KS_INT(*a*) ((a)&0xFF), ((a)>>8)

8.15.2.2 #define KS_TYPE_ARG_RANGE 0x03

8.15.2.3 #define KS_TYPE_LIST 0x01

8.15.2.4 #define KS_TYPE_RANGE 0x02

8.15.2.5 #define KS_TYPE_TERMINATOR 0x00

8.15.3 Function Documentation

8.15.3.1 uint16_t get_keysize (PGM_VOID_P *ks_desc*)

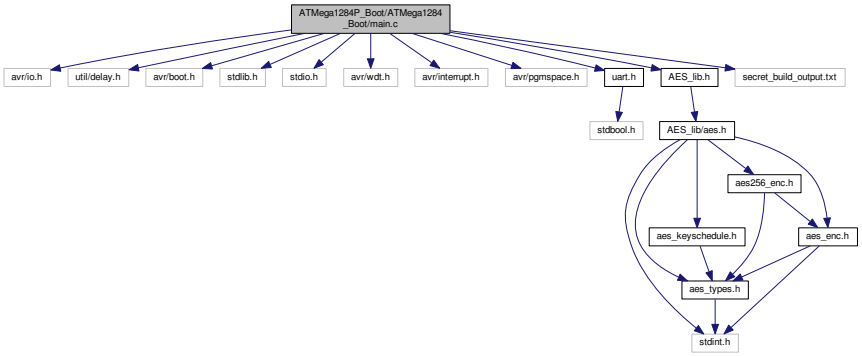
8.15.3.2 uint16_t get_keysizes (PGM_VOID_P *ks_desc,* uint16_t ** *list*)

8.15.3.3 uint8_t is_valid_keysize_P (PGM_VOID_P *ks_desc,* uint16_t *keysize*)

8.16 ATMega1284P_Boot/ATMega1284_Boot/main.c File Reference

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/boot.h>
#include <stdlib.h>
#include <stdio.h>
#include <avr/wdt.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include "uart.h"
#include "AES_lib.h"
#include "secret_build_output.txt"
```

Include dependency graph for main.c:



Macros

- #define F_CPU 7300000UL
- #define BAUD 115200UL
- #define LED PINB0
- #define UPDATE_PIN PINB2
- #define READBACK_PIN PINB3
- #define CONFIGURE_PIN PINB4
- #define RAND_CLOCK_SWITCH 10
- #define OK ((unsigned char)0x00)
- #define ERROR ((unsigned char)0x01)
- #define ACK ((unsigned char)0x06)
- #define NACK ((unsigned char)0x15)
- #define BLOCK_SIZE 16UL
- #define KEY_SIZE 32UL
- #define READBACK_PASSWORD_SIZE 24UL
- #define BOOTLDR_SIZE 8192UL
- #define EEPROM_SIZE 4096UL
- #define MAX_PAGE_NUMBER 126UL
- #define READBACK_REQUEST_SIZE 48UL
- #define APPLICATION_SECTION 0UL ∗ MAX_PAGE_NUMBER ∗ SPM_PAGESIZE
- #define MESSAGE_SECTION 1UL ∗ (MAX_PAGE_NUMBER - 6) ∗ SPM_PAGESIZE

- #define ENCRYPTED_SECTION 1UL ∗ MAX_PAGE_NUMBER ∗ SPM_PAGESIZE
- #define DECRYPTED_SECTION 2UL ∗ MAX_PAGE_NUMBER ∗ SPM_PAGESIZE
- #define BOOTLDR_SECTION 480UL ∗ SPM_PAGESIZE
- #define FIRM_HASH_SECTION 479UL ∗ SPM_PAGESIZE
- #define BOOT_HASH_SECTION 511UL ∗ SPM_PAGESIZE
- #define EPRM_HASH_SECTION

Functions

- void initTimer0 (void)

  Initialize Timer0 for clock switching.
- void enableClockSwitching (void)

  Starts Timer0, enabling Clock Switching.
- void disableClockSwitching (void)

  Stops Timer0, disabling Clock Switching.
- void calcHash (uint8_t ∗key, uint16_t startPage, uint16_t endPage, uint8_t ∗hash, uint8_t EEFlag)

  Calculates a hash of a memory section.
- void program_flash (uint32_t page_address, unsigned char ∗data)

  Programs a page of ATMega1284P flash memory.
- void load_firmware (void)

  Loads a new firmware image and release message.
- void boot_firmware (void)

  Ensures the firmware is loaded correctly and boots it up.
- void readback (void)

  Interfaces with host readback tool.
- void configure (void)

  Interfaces with host configure tool.
- int main (void)
- ISR (TIMER0_COMPA_vect)

Variables

- uint16_t fw_version EEMEM = 1
- uint8_t fastClock = 0
- unsigned char CONFIG_ERROR_FLAG = OK
- uint8_t hashKey [KEY_SIZE] = H_KEY
- uint8_t readbackHashKey [KEY_SIZE] = RBH_KEY
- uint8_t firmwareKey [KEY_SIZE] = FW_KEY
- uint8_t readbackKey [KEY_SIZE] = RB_KEY
- uint8_t firmwareIV [BLOCK_SIZE] = FW_IV
- uint8_t readbackIV [BLOCK_SIZE] = RB_IV
- uint8_t readbackPassword [READBACK_PASSWORD_SIZE] = RB_PW

8.16.1 Macro Definition Documentation

8.16.1.1 #define ACK ((unsigned char)0x06)

8.16.1.2 #define APPLICATION_SECTION 0UL * MAX_PAGE_NUMBER * SPM_PAGESIZE

8.16.1.3 #define BAUD 115200UL

8.16.1.4 #define BLOCK_SIZE 16UL

8.16.1.5 #define BOOT_HASH_SECTION 511UL * SPM_PAGESIZE

8.16.1.6 #define BOOTLDR_SECTION 480UL * SPM_PAGESIZE

8.16.1.7 #define BOOTLDR_SIZE 8192UL

8.16.1.8 #define CONFIGURE_PIN PINB4

8.16.1.9 #define DECRYPTED_SECTION 2UL * MAX_PAGE_NUMBER * SPM_PAGESIZE

8.16.1.10 #define EEPROM_SIZE 4096UL

8.16.1.11 #define ENCRYPTED_SECTION 1UL * MAX_PAGE_NUMBER * SPM_PAGESIZE

8.16.1.12 #define EPRM_HASH_SECTION

8.16.1.13 #define ERROR ((unsigned char)0x01)

8.16.1.14 #define F_CPU 7300000UL

bootloader.c

If Port B Pin 2 (PB2 on the ProtoStack board) is pulled to ground the bootloader will wait for data to appear on UART1 (which will be interpreted as an updated firmware package).

If the PB2 pin is NOT pulled to ground, but Port B Pin 3 (PB3 on the ProtoStack board) is pulled to ground, then the bootloader will enter flash memory readback mode.

If the PB3 pin is NOT pulled to ground, but Port B Pin 4 (PB4 on the ProtoStack board) is pulled to ground, then the bootloader will enter bootloader configure mode.

If NEITHER of these pins are pulled to ground, then the bootloader will execute the application from flash.

The load_firmware function details the structure of the firmware upload messages.

The readback function details the structure of the readback request messages.

See program_flash() for information on the process of programming the flash memory. Note that if no data is received after 2 seconds, the bootloader will time out and reset.

8.16.1.15 #define FIRM_HASH_SECTION 479UL * SPM_PAGESIZE

8.16.1.16 #define KEY_SIZE 32UL

8.16.1.17 #define LED PINB0

8.16.1.18 #define MAX_PAGE_NUMBER 126UL

8.16.1.19 #define MESSAGE_SECTION 1UL * (MAX_PAGE_NUMBER - 6) * SPM_PAGESIZE

8.16.1.20 #define NACK ((unsigned char)0x15)

8.16.1.21 #define OK ((unsigned char)0x00)

8.16.1.22 #define RAND_CLOCK_SWITCH 10

8.16.1.23 #define READBACK_PASSWORD_SIZE 24UL

8.16.1.24 #define READBACK_PIN PINB3

8.16.1.25 #define READBACK_REQUEST_SIZE 48UL

8.16.1.26 #define UPDATE_PIN PINB2

8.16.2 Function Documentation

8.16.2.1 void boot_firmware (void)

Ensures the firmware is loaded correctly and boots it up.

This function calculates the hash of the firmware section and compares it to the one currently stored.

HASH CORRECT - The function prints a release message if available, and boots. The Watchdog Timer is disabled before boot.

HASH INCORRECT - The function indicates firmware error and resets.

8.16.2.2 void calcHash (uint8_t * *key,* uint16_t *startPage,* uint16_t *endPage,* uint8_t * *hash,* uint8_t *EEFlag*)

Calculates a hash of a memory section.

This function calculates the CBC-MAC hash of a section in flash or in EEPROM. The section is indexed by pages, where 1 page = 256 bytes. Both EEPROM and FLASH are indexed through the same method. The final byte is used as a flag to indicate which region of memory is being hashed.

```
0 - Hash Flash  Memory
1 - Hash EEPROM Memory
```

The hash array fed to this function MUST be filled with zeros initially, or the hashing will not work correctly. The startPage parameter refers to the first page to be hashed. The endPage parameter does not refer to the last page to be hashed, but to the first page to NOT hash.

Parameters

key	Pointer to 32-byte array containing the AES-256 key.
startPage	Starting 256-byte page of memory to hash (this page WILL be hashed)
endPage	Ending 256-byte page of memory to hash (this page will NOT be hashed)
hash	Pointer to a 16-byte hash array. Must be initialized to all zeros.
EEFlag	Flag to indicate memory type. 0 - Flash; 1 - EEPROM

8.16.2.3 void configure (void)

Interfaces with host configure tool.

The procedure followed is outlined below.

1 - The routine waits to receive an ACK from the configure tool. 2 - The bootloader now calculates the hash of the bootloader in memory. 3 - The hash is now sent to the tools over UART1. IF CORRECT - The bootloader proceeds to check the EEPROM installation IF INCORRECT - The bootloader sets a flag and terminates 3 - The routine waits for an ACK from the host tools. 4 - The hash of the EEPROM is now calculated. 5 - That hash is sent over UART1 to the host

tools. IF CORRECT - The bootloader terminates. IF INCORRECT - A flag is set and the bootloader terminates.

8.16.2.4 void disableClockSwitching (void)

Stops Timer0, disabling Clock Switching.

8.16.2.5 void enableClockSwitching (void)

Starts Timer0, enabling Clock Switching.

8.16.2.6 void initTimer0 (void)

Initialize Timer0 for clock switching.

8.16.2.7 ISR (TIMER0_COMPA_vect)

8.16.2.8 void load_firmware (void)

Loads a new firmware image and release message.

This function securely loads a new firmware image onto flash. Firmware is encrypted using AES-256 in CFB Mode, and sent one page at a time. It is in the following format:

------—32000 Bytes------— -—256 Bytes-—- [Encrypted Firmware Update] [Message MAC Page]

The Encrypted Firmware Update section is broken into the following pages:

–256 Bytes— –30720 Bytes— –1024 Bytes— [Version Page] [Firmware Pages] [Message Pages]

The version page is constructed in the following format:

-—-2 Bytes-—- —254 Bytes-—- [Version Number] [Random Padding]

Each Firmware Page consists of 256 bytes of raw flash. These are obtained via the PC stripping the .hex file generated from the firmware Makefile. Address offsets are added, and blank Flash in the Application Section will be written to 0xFF.

The Message Page is 1024 bytes input by the user at FW_PROTECT time. The string is null terminated, and empty bytes in the Message Section will be written to 0xFF

The Message MAC page is constructed in the following format:

–16 Bytes— —240 Bytes-—- [Message MAC] [Random Padding]

The procedure followed is outlined below.

1 - The encrypted firmware image is loaded into the ENCRYPTED_SECTION of flash. 2 - The CBC-MAC of the encrypted firmware image is computed, and compared to the CBC-MAC sent. IF CORRECT - The bootloader proceeds with the firmware upload. IF INCORRECT - The bootloader erases ENCRYPTED_SECTION and terminates. 3 - The firmware image is decrypted and stored in the DECRYPTED_SECTION of flash. 4 - The version number is checked versus the current version, and updated in EEPROM IF CORRECT - The bootloader proceeds with the firmware upload. IF INCORRECT - The bootloader erases ENCRYPTED_SECTION and DECRYPTED_SECTION and terminate. 5 - The release message is written to the MESSAGE_SECTION 6 - The firmware is written to the APPLICATION_SECTION 7 - The bootloader erases ENCRYPTED_SECTION and DECRYPTED_SECTION and terminates

8.16.2.9 int main (void)

8.16.2.10 void program_flash (uint32_t *page_address,* unsigned char * *data*)

Programs a page of ATMega1284P flash memory.

To program flash, you need to access and program it in pages On the atmega1284p, each page is 128 words, or 256

bytes

Programing involves four things,

1. Erasing the page
2. Filling a page buffer
3. Writing a page
4. When you are done programming all of your pages, enable the flash

You must fill the buffer one word at a time

Parameters

page_address	Starting address of page to be programmed
data	256-byte array of data to be programmed

8.16.2.11 void readback (void)

Interfaces with host readback tool.

This function allows the factory to read back sections of Application Flash. The tool DOES NOT read from EEPROM or Bootloader Flash. The 48-byte readback request is sent in the following format:

-—32 Bytes—— ——_16 Bytes—— [Encrypted Request] [Readback Request MAC]

The Encrypted Request encrypted using AES-256 in CFB mode, with a specific Readback Key and Readback Initialization Vector. It is in the following format:

—24 Bytes—— -—4 Bytes—— —4 Bytes— [Readback Password] [Start Address] [End Address]

Although the addresses are byte-addressable, this function will dump flash pages (each of which is 256 bytes). The function will begin returning the page containing the starting address, and finish returning the page returning the ending address. Each page is encrypted using the same key and IV and sent back to the PC.

The procedure followed is outlined below.

1 - The encrypted readback request is received 2 - The CBC-MAC of the encrypted readback request is calculated, and compared to the CBC-MAC sent. IF CORRECT - The bootloader proceeds with the Readback Request IF INCORRECT - The bootloader terminates 3 - The readback request is decrypted using the Readback Key and IV 4 - The readback password is checked versus the password stored in the bootloader IF CORRECT - The bootloader proceeds with the readback request IF INCORRECT - The bootloader terminates 5 - The bootloader reads in the start address and end address and converts them to start page and end page. The end page is truncated to the page before the BOOTL↩ OADER_SECTION. 6 - The bootloader begins reading the flash data a page at a time. Each page is encrypted using AES-256 in CFB mode using the Readback Key and IV before being sent to PC.

8.16.3 Variable Documentation

8.16.3.1 unsigned char CONFIG_ERROR_FLAG = OK

8.16.3.2 uint16_t fw_version EEMEM = 1

8.16.3.3 uint8_t fastClock = 0

8.16.3.4 uint8_t firmwareIV[BLOCK_SIZE] = FW_IV

8.16.3.5 uint8_t firmwareKey[KEY_SIZE] = FW_KEY

8.16.3.6 uint8_t hashKey[KEY_SIZE] = H_KEY

8.16.3.7 uint8_t readbackHashKey[KEY_SIZE] = RBH_KEY

8.16.3.8 uint8_t readbackIV[BLOCK_SIZE] = RB_IV

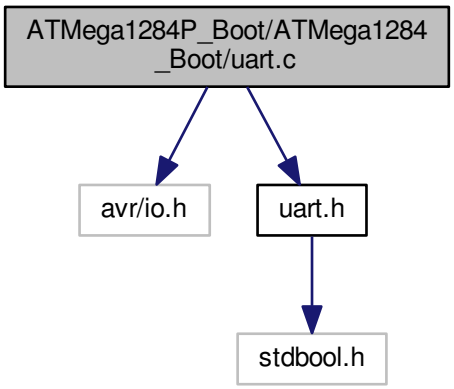
8.16.3.9 uint8_t readbackKey[KEY_SIZE] = RB_KEY

8.16.3.10 uint8_t readbackPassword[READBACK_PASSWORD_SIZE] = RB_PW

8.17 ATMega1284P_Boot/ATMega1284_Boot/uart.c File Reference

```
#include <avr/io.h>
#include "uart.h"
```

Include dependency graph for uart.c:



Macros

- #define F_CPU 7300000UL
- #define BAUD 115200UL

Functions

- void UART1_init (void)
- void UART1_putchar (unsigned char data)
- bool UART1_data_available (void)
- unsigned char UART1_getchar (void)
- void UART1_flush (void)
- void UART1_putstring (char ∗str)
- void UART0_init (void)
- void UART0_putchar (unsigned char data)
- bool UART0_data_available (void)

- unsigned char UART0_getchar (void)
- void UART0_flush (void)
- void UART0_putstring (char ∗str)

8.17.1 Macro Definition Documentation

8.17.1.1 #define BAUD 115200UL

8.17.1.2 #define F_CPU 7300000UL

8.17.2 Function Documentation

8.17.2.1 bool UART0_data_available (void)

8.17.2.2 void UART0_flush (void)

8.17.2.3 unsigned char UART0_getchar (void)

8.17.2.4 void UART0_init (void)

8.17.2.5 void UART0_putchar (unsigned char *data*)

8.17.2.6 void UART0_putstring (char ∗ *str*)

8.17.2.7 bool UART1_data_available (void)

8.17.2.8 void UART1_flush (void)

8.17.2.9 unsigned char UART1_getchar (void)

8.17.2.10 void UART1_init (void)

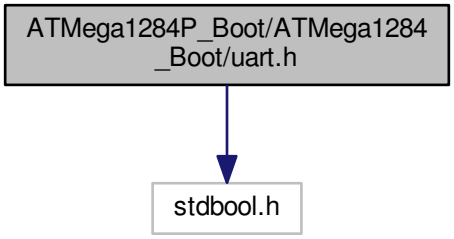
8.17.2.11 void UART1_putchar (unsigned char *data*)

8.17.2.12 void UART1_putstring (char ∗ *str*)

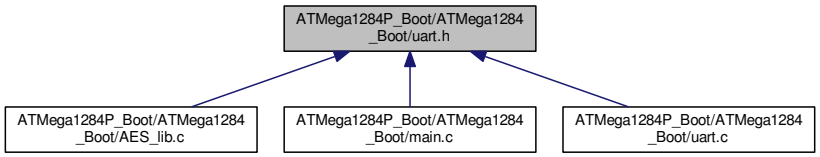
8.18 ATMega1284P_Boot/ATMega1284_Boot/uart.h File Reference

```
#include <stdbool.h>
```

Include dependency graph for uart.h:



This graph shows which files directly or indirectly include this file:



Functions

- void UART1_init (void)
- void UART1_putchar (unsigned char data)
- bool UART1_data_available (void)
- unsigned char UART1_getchar (void)
- void UART1_flush (void)
- void UART1_putstring (char ∗str)
- void UART0_init (void)
- void UART0_putchar (unsigned char data)
- bool UART0_data_available (void)
- unsigned char UART0_getchar (void)
- void UART0_flush (void)
- void UART0_putstring (char ∗str)

8.18.1 Function Documentation

8.18.1.1 bool UART0_data_available (void)

8.18.1.2 void UART0_flush (void)

8.18.1.3 unsigned char UART0_getchar (void)

8.18.1.4 void UART0_init (void)

8.18.1.5 void UART0_putchar (unsigned char *data*)

8.18.1.6 void UART0_putstring (char * *str*)

8.18.1.7 bool UART1_data_available (void)

8.18.1.8 void UART1_flush (void)

8.18.1.9 unsigned char UART1_getchar (void)

8.18.1.10 void UART1_init (void)

8.18.1.11 void UART1_putchar (unsigned char *data*)

8.18.1.12 void UART1_putstring (char * *str*)

8.19 host_tools/AES.py File Reference

Namespaces

- AES

Functions

- def encryptFileAES
- def AESHash
- def decryptFileAES
- def generate128Key

8.20 host_tools/bl_build.py File Reference

Namespaces

- bl_build

Functions

- def grabKeys
- def CMACHash
- def bytesToHexList
- def bytesToCString
- def generate128Entropy
- def generate256Entropy
- def make_bootloader
- def copy_artifacts
- def write_fuse_file
- def generate_secrets
- def make_secrets_file

- def stripLine
- def addrOf
- def dataLen
- def genMem

Variables

- tuple FILE_DIR = os.path.abspath(os.path.dirname(__file__))
- tuple secrets = grabKeys()
- tuple bootFlash = genMem("flash.hex")
- tuple flashHash = CMACHash(secrets["H_KEY"],bootFlash)

8.21 host_tools/bl_configure.py File Reference

Namespaces

- bl_configure

Functions

- def generate_secret_file
- def configure_bootloader
- def grabKeys

Variables

- tuple FILE_PATH = os.path.abspath(__file__)
- tuple parser = argparse.ArgumentParser(description='Bootloader Config Tool')
- required = True)
- tuple args = parser.parse_args()
- tuple serial_port = serial.Serial(args.port)

8.22 host_tools/bytesManipulators.py File Reference

Namespaces

- bytesManipulators

Functions

- def writeBytesToFile
- def appendBytesToFile
- def bytesToCString
- def bytesToHexList

8.23 host_tools/fw_protect.py File Reference

Namespaces

- fw_protect

Functions

- def grabKeys
- def stripLine
- def CMACHash
- def encryptCBC
- def encryptAES

Variables

- tuple parser = argparse.ArgumentParser(description='Firmware Update Tool')
- string help = "Path to the firmware image to protect."
- required = True)
- tuple args = parser.parse_args()
- tuple keyMap = grabKeys()
- list finalList = [ ]
- list firmwareSections = [ ]
- tuple version = struct.pack(">h",int(args.version))
- tuple tempMSG = (args.message)
- string finalBytes = b''
- int padSize = 125
- tuple CBCHash = CMACHash(keyMap["H_KEY"],finalBytes)

8.24 host_tools/fw_update.py File Reference

Namespaces

- fw_update

Variables

- string RESP_OK = b'\x06'
- tuple parser = argparse.ArgumentParser(description='Firmware Update Tool')
- required = True)
- string action = 'store_true'
- tuple args = parser.parse_args()
- tuple ser = serial.Serial(args.port, baudrate=115200, timeout=2)
- tuple chunk = firmware.read(256)
- int i = 0
- tuple resp = ser.read()

8.25 host_tools/readback.py File Reference

Namespaces

- readback

Functions

- def CMACHash
- def encryptCBC
- def encryptAES
- def decryptAES
- def readSecrets
- def construct_request

Variables

- tuple secrets = readSecrets()
- tuple parser = argparse.ArgumentParser(description='Memory Readback Tool')
- required = True)
- tuple args = parser.parse_args()
- tuple request = construct_request(int(args.address), int(args.num_bytes))
- list SECRET_KEY = secrets['RB_KEY']
- list IV = secrets['RB_IV']
- list HASH_KEY = secrets['RBH_KEY']
- tuple request_hash = CMACHash(HASH_KEY, request)
- tuple ser = serial.Serial(args.port, baudrate=115200)
- tuple data = ser.read(int(args.num_bytes))
- list printable = ["{:02x}".format(ord(x)) for x in data]

8.26 host_tools/test_grab.py File Reference

Namespaces

- test_grab

Functions

- def grabKeys

8.27 README.md File Reference

Index