

Hackable ESP device

Generated by Doxygen 1.9.3

1 Hackable ESP8266 device	1
1.1 Getting Started	1
1.1.1 Prerequisites	1
1.1.2 Dependencies	1
1.1.3 Installing	1
1.1.3.1 Option 1: Arduino IDE	2
1.1.3.2 Option 2: Visual Studio Code + Platformio	2
1.1.4 Manual Platformio Prep	3
1.1.5 Running	3
1.1.5.1 Wifi Manager First Boot	3
1.1.6 Customization of Hackable ESP (Contains spoilers) (Look in raw version of readme.md)	3
1.2 Hardware	3
1.3 Questions or Feedback?	3
1.4 Authors	4
2 Class Index	5
2.1 Class List	5
3 File Index	7
3.1 File List	7
4 Class Documentation	9
4.1 AES Class Reference	9
4.1.1 Detailed Description	9
4.1.2 Constructor & Destructor Documentation	10
4.1.2.1 AES()	10
4.1.3 Member Function Documentation	10
4.1.3.1 calc_size_n_pad()	10
4.1.3.2 cbc_decrypt() [1/2]	10
4.1.3.3 cbc_decrypt() [2/2]	11
4.1.3.4 cbc_encrypt() [1/2]	11
4.1.3.5 cbc_encrypt() [2/2]	11
4.1.3.6 CheckPad()	12
4.1.3.7 clean()	12
4.1.3.8 copy_n_bytes()	13
4.1.3.9 decrypt()	13
4.1.3.10 do_aes_decrypt() [1/2]	14
4.1.3.11 do_aes_decrypt() [2/2]	14
4.1.3.12 do_aes_encrypt() [1/2]	15
4.1.3.13 do_aes_encrypt() [2/2]	15
4.1.3.14 encrypt()	16
4.1.3.15 get_IV()	16
4.1.3.16 get_pad()	17

4.1.3.17	get_size()	17
4.1.3.18	iv_inc()	17
4.1.3.19	padPlaintext()	17
4.1.3.20	printArray() [1/2]	18
4.1.3.21	printArray() [2/2]	18
4.1.3.22	set_IV()	18
4.1.3.23	set_key()	20
4.1.3.24	set_size()	20
4.2	BufferOverflow Class Reference	21
4.2.1	Detailed Description	21
4.2.2	Constructor & Destructor Documentation	21
4.2.2.1	BufferOverflow()	21
4.2.3	Member Function Documentation	21
4.2.3.1	Is()	21
4.2.3.2	objectDump()	22
4.2.3.3	runCProgram()	22
4.2.3.4	vi()	22
4.3	CbcEncryptor Class Reference	22
4.3.1	Detailed Description	23
4.3.2	Constructor & Destructor Documentation	23
4.3.2.1	CbcEncryptor()	23
4.3.3	Member Function Documentation	23
4.3.3.1	decryptFile()	23
4.3.3.2	decryptLine()	24
4.3.3.3	encryptFile()	24
4.3.3.4	encryptLine()	24
4.3.3.5	setKey()	25
4.4	SerialCommandExecuter Class Reference	25
4.4.1	Detailed Description	26
4.4.2	Constructor & Destructor Documentation	26
4.4.2.1	SerialCommandExecuter()	26
4.4.3	Member Function Documentation	26
4.4.3.1	executeCommand()	26
4.4.3.2	setUsers()	26
4.5	UserHandler Class Reference	27
4.5.1	Detailed Description	27
4.5.2	Constructor & Destructor Documentation	27
4.5.2.1	UserHandler()	27
4.5.3	Member Function Documentation	27
4.5.3.1	checkPermission()	28
4.5.3.2	getNumberOfUsers()	28
4.5.3.3	getUsers()	28

4.5.3.4 updateUsers()	28
5 File Documentation	29
5.1 hackableEspDevice/AES.cpp File Reference	29
5.1.1 Macro Definition Documentation	30
5.1.1.1 d2	30
5.1.1.2 DPOLY	30
5.1.1.3 f2	30
5.1.1.4 WPOLY	30
5.2 AES.cpp	31
5.3 hackableEspDevice/AES.h File Reference	38
5.4 AES.h	38
5.5 hackableEspDevice/AES_config.h File Reference	40
5.5.1 Macro Definition Documentation	41
5.5.1.1 FAILURE	41
5.5.1.2 KEY_SCHEDULE_BYTES	41
5.5.1.3 N_BLOCK	42
5.5.1.4 N_COL	42
5.5.1.5 N_MAX_ROUNDS	42
5.5.1.6 N_ROW	42
5.5.1.7 SUCCESS	42
5.6 AES_config.h	43
5.7 hackableEspDevice/BufferOverflow.cpp File Reference	44
5.8 BufferOverflow.cpp	44
5.9 hackableEspDevice/BufferOverflow.h File Reference	47
5.9.1 Macro Definition Documentation	48
5.9.1.1 ADDRESS_LENGTH	49
5.9.1.2 MAX_NUM_CHARS	49
5.9.1.3 OVERFLOW_BEGIN	49
5.9.1.4 OVERFLOW_LENGTH	49
5.9.1.5 RETURN_ADDRESS	49
5.10 BufferOverflow.h	50
5.11 hackableEspDevice/CbcEncryptor.cpp File Reference	51
5.12 CbcEncryptor.cpp	51
5.13 hackableEspDevice/CbcEncryptor.h File Reference	54
5.14 CbcEncryptor.h	54
5.15 hackableEspDevice/Debugger.cpp File Reference	55
5.15.1 Function Documentation	56
5.15.1.1 debug()	56
5.15.1.2 debugIn()	56
5.15.1.3 getDebugEnabled()	56
5.15.1.4 setDebugEnabled()	57

5.16 Debugger.cpp	57
5.17 hackableEspDevice/Debugger.h File Reference	58
5.17.1 Macro Definition Documentation	59
5.17.1.1 ENABLE_DEBUG_FLAG_ADDRESS	59
5.17.2 Function Documentation	59
5.17.2.1 debug()	59
5.17.2.2 debugln()	59
5.17.2.3 getDebugEnabled()	60
5.17.2.4 setDebugEnabled()	60
5.18 Debugger.h	60
5.19 hackableEspDevice/hackableEspDevice.ino File Reference	61
5.19.1 Macro Definition Documentation	62
5.19.1.1 MIN_BRIGHTNESS	62
5.19.1.2 OFF	62
5.19.1.3 ON	63
5.19.2 Function Documentation	63
5.19.2.1 getContentType()	63
5.19.2.2 handleFileDownload()	63
5.19.2.3 handleFileRequest()	63
5.19.2.4 handleFileUpload()	64
5.19.2.5 initializeHostname()	64
5.19.2.6 initializeServer()	64
5.19.2.7 loop()	64
5.19.2.8 sendToFrontend()	65
5.19.2.9 server()	65
5.19.2.10 setup()	65
5.19.2.11 setupWifi()	65
5.19.3 Variable Documentation	65
5.19.3.1 brightness	65
5.19.3.2 cliExecuter	66
5.19.3.3 fsUploadFile	66
5.19.3.4 ledState	66
5.19.3.5 server	66
5.20 hackableEspDevice.ino	66
5.21 hackableEspDevice/HostnameWrite.cpp File Reference	71
5.21.1 Function Documentation	72
5.21.1.1 checkEepromCommit()	72
5.21.1.2 getHostname()	72
5.21.1.3 setEepromToNull()	72
5.21.1.4 writeHostname()	73
5.22 HostnameWrite.cpp	73
5.23 hackableEspDevice/HostnameWrite.h File Reference	74

5.23.1 Function Documentation	75
5.23.1.1 checkEepromCommit()	75
5.23.1.2 getHostname()	75
5.23.1.3 setEepromToNull()	76
5.23.1.4 writeHostname()	76
5.24 HostnameWrite.h	76
5.25 hackableEspDevice/SerialCommandExecuter.cpp File Reference	77
5.26 SerialCommandExecuter.cpp	77
5.27 hackableEspDevice/SerialCommandExecuter.h File Reference	82
5.27.1 Macro Definition Documentation	84
5.27.1.1 ARG_DEBUG_OFF	84
5.27.1.2 ARG_DEBUG_ON	84
5.27.1.3 ARG_GPG_DECRYPT	85
5.27.1.4 ARG_GPG_ENCRYPT	85
5.27.1.5 ARG_HELP_LONG	85
5.27.1.6 ARG_HELP_SHORT	85
5.27.1.7 ARG_HOSTNAME_DEFAULT	85
5.27.1.8 ARG_HOSTNAME_SET	85
5.27.1.9 ARG_LS_FILE_1_1	86
5.27.1.10 ARG_LS_FILE_1_2	86
5.27.1.11 ARG_LS_FILE_2_1	86
5.27.1.12 ARG_LS_FILE_2_2	86
5.27.1.13 COMMAND_DEBUG	86
5.27.1.14 COMMAND_GPG	86
5.27.1.15 COMMAND_HELP	87
5.27.1.16 COMMAND_HOSTNAME	87
5.27.1.17 COMMAND_KEYS	87
5.27.1.18 COMMAND_LS	87
5.27.1.19 COMMAND_OBJDUMP	87
5.27.1.20 COMMAND_RESTART	87
5.27.1.21 COMMAND_RUN	88
5.27.1.22 COMMAND_SU	88
5.27.1.23 COMMAND_USERS	88
5.27.1.24 COMMAND_VI	88
5.27.1.25 COMMAND_WHOAMI	88
5.27.1.26 ERROR_CMD_NOT_FOUND	88
5.27.1.27 ERROR_NO_FILE	89
5.27.1.28 ERROR_NO_FILE_DIR	89
5.27.1.29 ERROR_NO_PERMISSION	89
5.27.1.30 ERROR_NO_VALID_KEY	89
5.27.1.31 ERROR_PERM_DENIED	89
5.27.1.32 ERROR_TOO_FEW_ARGS	89

5.27.1.33 ERROR_TOO_MANY_ARGS	90
5.27.1.34 ERROR_WRONG_ARGS	90
5.27.1.35 ERROR_WRONG_PWD	90
5.27.1.36 MAX_NUMBER_PARAMS	90
5.27.1.37 MESS_SUPER_USER	90
5.28 SerialCommandExecuter.h	91
5.29 hackableEspDevice/StartupText.h File Reference	92
5.29.1 Macro Definition Documentation	92
5.29.1.1 LENGTH	93
5.29.2 Function Documentation	93
5.29.2.1 printStartupText()	93
5.29.2.2 printStringInBytes()	93
5.30 StartupText.h	94
5.31 hackableEspDevice/userHandler.cpp File Reference	98
5.32 userHandler.cpp	98
5.33 hackableEspDevice/userHandler.h File Reference	100
5.34 userHandler.h	101
5.35 README.md File Reference	101
6 Example Documentation	103
6.1 aes.ino	103
6.2 aes.cpp	103
6.3 test_vectors.ino	103
6.4 test_vectors.cpp	103
Index	105

Chapter 1

Hackable ESP8266 device

Firmware for ESP8266 based device (D1 Mini board) with designed vulnerabilities to practice ethical hacking. The software is tested on the following boards:

- [D1 Mini](#)

1.1 Getting Started

These instructions will get you a copy of the project up and running on your D1 Mini (or other ESP8266 based boards) for development or hacking purposes.

1.1.1 Prerequisites

The software is written, compiled and uploaded using the [Arduino IDE](#). Platform.io and Visual Studio Code can be used as well. Use the script to convert the project to a Platform.io.

1.1.2 Dependencies

- ESP for Arduino IDE
- ESP Async WebServer V1.2.3
- Wifimanager V0.16.0

1.1.3 Installing

General Install

1. Install the [driver](#) for the esp8266.
2. Clone the repository.

There are multiple ways to upload the program files to the board. The two ways listed here are using Arduino IDE and Platformio on Visual Studio Code.

1.1.3.1 **Option 1: Arduino IDE**

1. Install the `Arduino IDE`
2. `Add the esp8266 libraries to Arduino IDE.`
3. Follow `this` tutorial about the SPIFFS.
4. Navigate to the `hackableEspDevice` folder.
5. Open `hackableEspDevice.ino`.
6. Upload the files in the `data` folder (see the tutorial).
7. Upload the program to the device.
8. Connect to the `Configure Smartlight Wifi AP` to configure the wifi.

ESP8266 Sketch Data Upload

- (a) The Arduino IDE won't have the option 'ESP8266 Sketch Data Upload'.
- (b) You can download it from this `link`.
- (c) The file should be unpacked at `<home_dir>/Arduino-<version>/tools/ESP8266↵FS/tools/`.
 - If the directory `tools` does not exist you should create it. You have to create a new file named "tools" if it doesn't exist already inside of `Arduino file`.

1.1.3.2 **Option 2: Visual Studio Code + Platformio**

1. Install the `Platformio` plugin.
2. Prepare files for platformio.
 - Run the `toPlatformio.ps1` script and select the copy or symbolic option.
 - Symbolic changes the original ideal for editing the files.
 - Copy simply copies the files to a new location for platformio files.
 - Run the `toPlatformio.ps1` script and select fix.
 - Or prepare the files manually see manual prep platformio.
3. Open visual studio code in the `HackableEspDevicePlatformio` directory.
4. In visual studio code open the project in the platformio addon. (`Platformio > Projects > open HackableEspDevicePlatformio`).
5. Upload the program (`project tasks > General> Upload`).
6. Upload the filesystem Image (`Project tasks > Platform > Upload filesystem Image`).
7. Done. The device should now be ready for use.

1.1.4 Manual Platformio Prep

1. Create the correct hierarchy.

```
|HackableEspDevicePlatformio\  
|--- platformio.ini  
|--- src\  
|--- src\main.cpp\  
|--- data\  
|
```

1. The src dir needs to contain all the files from the hackableEspDevice directory except the data directory.
2. Rename the `hackableEspDevice.ino` to `main.cpp`.
3. In `main.cpp` add a reference to all functions in `main` e.g.

```
void setup();  
void setup();  
void initializeHostname();  
void connectWifi();  
void initializeServer();  
void loop();  
String getContentType(String filename);  
void handleFileRequest(String path, uint8_t permissionLevel);  
void handleFileUpload();  
void handleFileDownload();
```
4. Move the `platformio.ini` file from the root dir to the `hackableEspDevicePlatformio` dir.
5. Copy all files from `hackableEspDevice\data` to `hackableEspDevicePlatformio\data`.

1.1.5 Running

1.1.5.1 Wifi Manager First Boot

1. Start up the device.
2. Connect to the `Configure Smartlight Wifi` via a mobile device.
3. Go to the IP address listed in the serial monitor. Most of the time this is `http://192.168.4.1`.
4. Follow the steps on the website to configure a wifi connection.
5. The device should now restart, connect to the selected wifi network and be ready for use.

1.1.6 Customization of Hackable ESP (Contains spoilers) (Look in raw version of readme.md)

1.2 Hardware

- 1x D1 Mini Board
- 1x USB to USB-mini cable
- 1x ESP8266 casing

1.3 Questions or Feedback?

There is technical documentation available if you want to contribute to this project. There is a user manual as well, contact us for information. You can open an issue if you have questions or feedback for this repository.

1.4 Authors

- **Luke de Munk** - *Head author* - [LinkedIn](#)
- **Thijs Takken** - *Head author* - [LinkedIn](#)
- **Christina Kostine** - *Head author* - [LinkedIn](#)
- **Twenne Elffers** - *Head author* - [LinkedIn](#)

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

AES	9
BufferOverflow	21
CbcEncryptor	22
SerialCommandExecuter	25
UserHandler	27

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

hackableEspDevice/ AES.cpp	29
hackableEspDevice/ AES.h	38
hackableEspDevice/ AES_config.h	40
hackableEspDevice/ BufferOverflow.cpp	44
hackableEspDevice/ BufferOverflow.h	47
hackableEspDevice/ CbcEncryptor.cpp	51
hackableEspDevice/ CbcEncryptor.h	54
hackableEspDevice/ Debugger.cpp	55
hackableEspDevice/ Debugger.h	58
hackableEspDevice/ hackableEspDevice.ino	61
hackableEspDevice/ HostnameWrite.cpp	71
hackableEspDevice/ HostnameWrite.h	74
hackableEspDevice/ SerialCommandExecuter.cpp	77
hackableEspDevice/ SerialCommandExecuter.h	82
hackableEspDevice/ StartupText.h	92
hackableEspDevice/ userHandler.cpp	98
hackableEspDevice/ userHandler.h	100

Chapter 4

Class Documentation

4.1 AES Class Reference

```
#include <AES.h>
```

Public Member Functions

- [AES](#) ()
AES constructor.
- byte [set_key](#) (byte key[], int keylen)
- void [clean](#) ()
- void [copy_n_bytes](#) (byte *AEST, byte *src, byte n)
- byte [encrypt](#) (byte plain[N_BLOCK], byte cipher[N_BLOCK])
- byte [cbc_encrypt](#) (byte *plain, byte *cipher, int n_block, byte iv[N_BLOCK])
- byte [cbc_encrypt](#) (byte *plain, byte *cipher, int n_block)
- byte [decrypt](#) (byte cipher[N_BLOCK], byte plain[N_BLOCK])
- byte [cbc_decrypt](#) (byte *cipher, byte *plain, int n_block, byte iv[N_BLOCK])
- byte [cbc_decrypt](#) (byte *cipher, byte *plain, int n_block)
- void [set_IV](#) (unsigned long long int IVCI)
- void [iv_inc](#) ()
- int [get_size](#) ()
- void [set_size](#) (int sizel)
- int [get_pad](#) ()
- void [get_IV](#) (byte *out)
- void [calc_size_n_pad](#) (int p_size)
- void [padPlaintext](#) (void *in, byte *out)
- bool [CheckPad](#) (byte *in, int size)
- void [printArray](#) (byte output[], bool p_pad=true)
- void [printArray](#) (byte output[], int sizel)
- void [do_aes_encrypt](#) (byte *plain, int size_p, byte *cipher, byte *key, int bits, byte iv[N_BLOCK])
- void [do_aes_encrypt](#) (byte *plain, int size_p, byte *cipher, byte *key, int bits)
- void [do_aes_decrypt](#) (byte *cipher, int size_c, byte *plain, byte *key, int bits, byte iv[N_BLOCK])
- void [do_aes_decrypt](#) (byte *cipher, int size_c, byte *plain, byte *key, int bits)

4.1.1 Detailed Description

Definition at line 39 of file [AES.h](#).

4.1.2 Constructor & Destructor Documentation

4.1.2.1 AES()

```
AES::AES ( )
```

[AES](#) constructor.

This function initialized an instance of [AES](#).

Definition at line 231 of file [AES.cpp](#).

4.1.3 Member Function Documentation

4.1.3.1 calc_size_n_pad()

```
void AES::calc_size_n_pad (
    int p_size )
```

Calculates the size of the plaintext and the padding.

Calculates the size of the plaintext with the padding and the size of the padding needed. Moreover it stores them in their class variables.

Parameters

<i>p_size</i>	the size of the byte array ex sizeof(plaintext)
---------------	---

Definition at line 492 of file [AES.cpp](#).

4.1.3.2 cbc_decrypt() [1/2]

```
byte AES::cbc_decrypt (
    byte * cipher,
    byte * plain,
    int n_block )
```

CBC decrypt a number of blocks (input and return an IV)

Parameters

<i>*cipher</i>	Pointer, points to the ciphertext that will be created.
<i>*plain</i>	Pointer, points to the plaintext.
<i>n_block</i>	integer, indicated the number of blocks to be ciphered. @Return 0 if SUCCESS or -1 if FAILURE

Definition at line 432 of file [AES.cpp](#).

4.1.3.3 cbc_decrypt() [2/2]

```
byte AES::cbc_decrypt (
    byte * cipher,
    byte * plain,
    int n_block,
    byte iv[N_BLOCK] )
```

CBC decrypt a number of blocks (input and return an IV)

Parameters

<i>*cipher</i>	Pointer, points to the ciphertext that will be created.
<i>*plain</i>	Pointer, points to the plaintext.
<i>n_block</i>	integer, indicated the number of blocks to be ciphered.
<i>iv[N_BLOCK]</i>	byte Array that holds the IV (initialization vector). @Return 0 if SUCCESS or -1 if FAILURE

Definition at line 414 of file [AES.cpp](#).

4.1.3.4 cbc_encrypt() [1/2]

```
byte AES::cbc_encrypt (
    byte * plain,
    byte * cipher,
    int n_block )
```

CBC encrypt a number of blocks (input and return an IV).

Parameters

<i>*plain</i>	Pointer, points to the plaintext.
<i>*cipher</i>	Pointer, points to the ciphertext that will be created.
<i>n_block</i>	integer, indicated the number of blocks to be ciphered. @Return 0 if SUCCESS or -1 if FAILURE

Definition at line 375 of file [AES.cpp](#).

4.1.3.5 cbc_encrypt() [2/2]

```
byte AES::cbc_encrypt (
    byte * plain,
```

```

byte * cipher,
int n_block,
byte iv[N_BLOCK] )

```

CBC encrypt a number of blocks (input and return an IV).

Parameters

<i>*plain</i>	Pointer, points to the plaintext.
<i>*cipher</i>	Pointer, points to the ciphertext that will be created.
<i>n_block</i>	integer, indicated the number of blocks to be ciphered.
<i>iv[N_BLOCK]</i>	byte Array that holds the IV (initialization vector). @Return 0 if SUCCESS or -1 if FAILURE

Definition at line 359 of file [AES.cpp](#).

4.1.3.6 CheckPad()

```

bool AES::CheckPad (
    byte * in,
    int size )

```

Check the if the padding is correct.

This functions checks the padding of the plaintext.

Parameters

<i>in</i>	the string of the plaintext in a byte array
<i>size</i>	the size of the string

Returns

true if correct / false if not

Definition at line 509 of file [AES.cpp](#).

4.1.3.7 clean()

```

void AES::clean ( )

```

clean up subkeys after use.

Definition at line 311 of file [AES.cpp](#).

4.1.3.8 copy_n_bytes()

```
void AES::copy_n_bytes (
    byte * AEst,
    byte * src,
    byte n )
```

copying and xoring utilities.

Parameters

<i>*AEst</i>	byte pointer of the AEstination array.
<i>*src</i>	byte pointer of the source array.
<i>n</i>	byte, indicating the sizeof the bytes to be copied.

Note

this is an alternative for memcpy(void *s1,const void *s2, site_t n), i have not updated the function in the implementation yet, but it is considered a future plan.

Definition at line 320 of file [AES.cpp](#).

4.1.3.9 decrypt()

```
byte AES::decrypt (
    byte cipher[N_BLOCK],
    byte plain[N_BLOCK] )
```

Decrypt a single block of 16 bytes

Parameters

<i>cipher[N_BLOCK]</i>	Array of the ciphertext.
<i>plain[N_BLOCK]</i>	Array of the plaintext.

Note

The N_BLOCK is defined in [AES_config.h](#) as,

```
#define N_ROW      4
#define N_COL      4
#define N_BLOCK    (N_ROW * N_COL)
```

Changed to that will change the Block_size. @Return 0 if SUCCESS or -1 if FAILURE

Definition at line 391 of file [AES.cpp](#).

4.1.3.10 do_aes_decrypt() [1/2]

```
void AES::do_aes_decrypt (
    byte * cipher,
    int size_c,
    byte * plain,
    byte * key,
    int bits )
```

User friendly implementation of AES-CBC decryption.

Parameters

<i>*cipher</i>	pointer to the ciphertext
<i>size_c</i>	size of the ciphertext
<i>*plain</i>	pointer to the plaintext
<i>*key</i>	pointer to the key that will be used.
<i>bits</i>	bits of the encryption/decryption

Note

The key will be stored in class variable.

Definition at line 585 of file [AES.cpp](#).

4.1.3.11 do_aes_decrypt() [2/2]

```
void AES::do_aes_decrypt (
    byte * cipher,
    int size_c,
    byte * plain,
    byte * key,
    int bits,
    byte ivl[N_BLOCK] )
```

User friendly implementation of AES-CBC decryption.

Parameters

<i>*cipher</i>	pointer to the ciphertext
<i>size_c</i>	size of the ciphertext
<i>*plain</i>	pointer to the plaintext
<i>*key</i>	pointer to the key that will be used.
<i>bits</i>	bits of the encryption/decryption
<i>ivl[N_BLOCK]</i>	the initialization vector IV that will be used for decryption.

Note

The key will be stored in class variable.

Definition at line 576 of file [AES.cpp](#).

4.1.3.12 do_aes_encrypt() [1/2]

```
void AES::do_aes_encrypt (
    byte * plain,
    int size_p,
    byte * cipher,
    byte * key,
    int bits )
```

User friendly implementation of AES-CBC encryption.

Parameters

<i>*plain</i>	pointer to the plaintext
<i>size_p</i>	size of the plaintext
<i>*cipher</i>	pointer to the ciphertext
<i>*key</i>	pointer to the key that will be used.
<i>bits</i>	bits of the encryption/decryption

Note

The key will be stored in class variable.

Definition at line 565 of file [AES.cpp](#).

4.1.3.13 do_aes_encrypt() [2/2]

```
void AES::do_aes_encrypt (
    byte * plain,
    int size_p,
    byte * cipher,
    byte * key,
    int bits,
    byte ivl[N_BLOCK] )
```

User friendly implementation of AES-CBC encryption.

Parameters

<i>*plain</i>	pointer to the plaintext
<i>size_p</i>	size of the plaintext
<i>*cipher</i>	pointer to the ciphertext
<i>*key</i>	pointer to the key that will be used.
<i>bits</i>	bits of the encryption/decryption
<i>ivl[N_BLOCK]</i>	the initialization vector IV that will be used for encryption.

Note

The key will be stored in class variable.

Definition at line 554 of file [AES.cpp](#).

4.1.3.14 encrypt()

```
byte AES::encrypt (
    byte plain[N_BLOCK],
    byte cipher[N_BLOCK] )
```

Encrypt a single block of 16 bytes .

Parameters

<i>plain[N_BLOCK]</i>	Array of the plaintext.
<i>cipher[N_BLOCK]</i>	Array of the ciphertext.

Note

The N_BLOCK is defined in [AES_config.h](#) as,

```
#define N_ROW      4
#define N_COL      4
#define N_BLOCK    (N_ROW * N_COL)
```

Changed to that will change the Block_size. @Return 0 if SUCCESS or -1 if FAILURE

Definition at line 336 of file [AES.cpp](#).

4.1.3.15 get_IV()

```
void AES::get_IV (
    byte * out )
```

Getter method for IV

This function return the IV

Parameters

<i>out</i>	byte pointer that gets the IV.
------------	--------------------------------

Returns

none, the IV is writed to the out pointer.

Definition at line 485 of file [AES.cpp](#).

4.1.3.16 get_pad()

```
int AES::get_pad ( )
```

Definition at line 472 of file [AES.cpp](#).

4.1.3.17 get_size()

```
int AES::get_size ( )
```

Getter method for size

This function return the size

Returns

an integer, that is the size of the of the padded plaintext, thus, the size of the ciphertext.

Definition at line 466 of file [AES.cpp](#).

4.1.3.18 iv_inc()

```
void AES::iv_inc ( )
```

increase the iv (initialization vector) and IVC (IV counter) by 1

This function increased the VI by one step in order to have a different IV each time

Definition at line 458 of file [AES.cpp](#).

4.1.3.19 padPlaintext()

```
void AES::padPlaintext (
    void * in,
    byte * out )
```

Pads the plaintext

This function pads the plaintext and returns an char array with the plaintext and the padding in order for the plaintext to be compatible with 16bit size blocks required by [AES](#)

Parameters

<i>in</i>	the string of the plaintext in a byte array
<i>out</i>	The string of the out array.

Returns

no return, The padded plaintext is stored in the out pointer.

Definition at line [499](#) of file [AES.cpp](#).

4.1.3.20 printArray() [1/2]

```
void AES::printArray (
    byte output[],
    bool p_pad = true )
```

Prints the array given.

This function prints the given array and pad, It is mainly used for debugging purposes or to output the string.

Parameters

<i>output[]</i>	the string of the text in a byte array
<i>p_pad</i>	optional, used to print with out the padding characters

Definition at line [525](#) of file [AES.cpp](#).

4.1.3.21 printArray() [2/2]

```
void AES::printArray (
    byte output[],
    int size )
```

Prints the array given.

This function prints the given array in Hexadecimal.

Parameters

<i>output[]</i>	the string of the text in a byte array
<i>size</i>	the size of the array.

Definition at line [542](#) of file [AES.cpp](#).

4.1.3.22 set_IV()

```
void AES::set_IV (
    unsigned long long int IVCl )
```

Sets IV (initialization vector) and IVC (IV counter). This function changes the ivc and iv variables needed for [AES](#).

Parameters

<i>IVC/</i>	int or hex value of iv , ex. 0x0000000000000001
-------------	---

Note

example:

```
unsigned long long int my_iv = 01234567;
```

Definition at line 450 of file [AES.cpp](#).

4.1.3.23 set_key()

```
byte AES::set_key (
    byte key[],
    int keylen )
```

Set the cipher key for the pre-keyed version.

Parameters

<i>key[]</i>	pointer to the key string.
<i>keylen</i>	Integer that indicates the length of the key.

Note

NOTE: If the length_type used for the key length is an unsigned 8-bit character, a key length of 256 bits must be entered as a length in bytes (valid inputs are hence 128, 192, 16, 24 and 32).

Definition at line 255 of file [AES.cpp](#).

4.1.3.24 set_size()

```
void AES::set_size (
    int size1 )
```

Setter method for size

This function sets the size of the plaintext+pad

Definition at line 478 of file [AES.cpp](#).

The documentation for this class was generated from the following files:

- hackableEspDevice/[AES.h](#)
- hackableEspDevice/[AES.cpp](#)

4.2 BufferOverflow Class Reference

```
#include <BufferOverflow.h>
```

Public Member Functions

- [BufferOverflow](#) ()
Constructor.
- void [ls](#) ()
Prints the fake list of files.
- void [vi](#) ()
Prints the vulnerable testprogram.
- void [objectDump](#) ()
Prints the disassembled code of the vulnerable testprogram.
- bool [runCProgram](#) (String arg)
Simulates the vulnerable testprogram.

4.2.1 Detailed Description

Definition at line 21 of file [BufferOverflow.h](#).

4.2.2 Constructor & Destructor Documentation

4.2.2.1 BufferOverflow()

```
BufferOverflow::BufferOverflow ( )
```

Constructor.

Definition at line 17 of file [BufferOverflow.cpp](#).

4.2.3 Member Function Documentation

4.2.3.1 ls()

```
void BufferOverflow::ls ( )
```

Prints the fake list of files.

Definition at line 26 of file [BufferOverflow.cpp](#).

4.2.3.2 objectDump()

```
void BufferOverflow::objectDump ( )
```

Prints the disassembled code of the vulnerable testprogram.

Definition at line 72 of file [BufferOverflow.cpp](#).

4.2.3.3 runCProgram()

```
bool BufferOverflow::runCProgram (
    String arg )
```

Simulates the vulnerable testprogram.

Parameters

<i>arg</i>	Given argument
------------	----------------

Returns

bool True if the buffer overflow attack is done correctly

Definition at line 129 of file [BufferOverflow.cpp](#).

4.2.3.4 vi()

```
void BufferOverflow::vi ( )
```

Prints the vulnerable testprogram.

Definition at line 36 of file [BufferOverflow.cpp](#).

The documentation for this class was generated from the following files:

- [hackableEspDevice/BufferOverflow.h](#)
- [hackableEspDevice/BufferOverflow.cpp](#)

4.3 CbcEncryptor Class Reference

```
#include <CbcEncryptor.h>
```

Public Member Functions

- [CbcEncryptor](#) ()
Constructor.
- bool [encryptFile](#) (String filename)
Encrypts the SPIFFS file if it exists.
- bool [decryptFile](#) (String filename)
Decrypts the SPIFFS file if it exists.
- String [encryptLine](#) (String line)
Encrypts a line of text.
- String [decryptLine](#) (String line)
Decrypts a line of text.
- bool [setKey](#) (String key)
Sets the encryption key.

4.3.1 Detailed Description

Definition at line 15 of file [CbcEncryptor.h](#).

4.3.2 Constructor & Destructor Documentation

4.3.2.1 CbcEncryptor()

```
CbcEncryptor::CbcEncryptor ( )
```

Constructor.

Definition at line 16 of file [CbcEncryptor.cpp](#).

4.3.3 Member Function Documentation

4.3.3.1 decryptFile()

```
bool CbcEncryptor::decryptFile (  
    String filename )
```

Decrypts the SPIFFS file if it exists.

Parameters

<i>filename</i>	Name of the file to decrypt
-----------------	-----------------------------

Returns

bool True if decryption is successfull

Definition at line 75 of file [CbcEncryptor.cpp](#).

4.3.3.2 decryptLine()

```
String CbcEncryptor::decryptLine (  
    String line )
```

Decrypts a line of text.

Parameters

<i>line</i>	Line to decrypt
-------------	-----------------

Returns

String Decrypted string

Definition at line 168 of file [CbcEncryptor.cpp](#).

4.3.3.3 encryptFile()

```
bool CbcEncryptor::encryptFile (  
    String filename )
```

Encrypts the SPIFFS file if it exists.

Parameters

<i>filename</i>	Name of the file to encrypt
-----------------	-----------------------------

Returns

bool True if encryption is successfull

Definition at line 33 of file [CbcEncryptor.cpp](#).

4.3.3.4 encryptLine()

```
String CbcEncryptor::encryptLine (  
    String line )
```

Encrypts a line of text.

Parameters

<i>line</i>	Line to encrypt
-------------	-----------------

Returns

String Encrypted string

Definition at line 137 of file [CbcEncryptor.cpp](#).

4.3.3.5 setKey()

```
bool CbcEncryptor::setKey (
    String key )
```

Sets the encryption key.

Parameters

<i>key</i>	Encryption key
------------	----------------

Returns

bool True if is successfull

Definition at line 115 of file [CbcEncryptor.cpp](#).

The documentation for this class was generated from the following files:

- [hackableEspDevice/CbcEncryptor.h](#)
- [hackableEspDevice/CbcEncryptor.cpp](#)

4.4 SerialCommandExecuter Class Reference

```
#include <SerialCommandExecuter.h>
```

Public Member Functions

- [SerialCommandExecuter](#) ()
Constructor.
- void [executeCommand](#) ()
Reads the commands and sends them to the parser.
- void [setUsers](#) (String *users, uint8_t numUsers)
Sets the users for user list.

4.4.1 Detailed Description

Definition at line 66 of file [SerialCommandExecuter.h](#).

4.4.2 Constructor & Destructor Documentation

4.4.2.1 SerialCommandExecuter()

```
SerialCommandExecuter::SerialCommandExecuter ( )
```

Constructor.

Definition at line 16 of file [SerialCommandExecuter.cpp](#).

4.4.3 Member Function Documentation

4.4.3.1 executeCommand()

```
void SerialCommandExecuter::executeCommand ( )
```

Reads the commands and sends them to the parser.

Definition at line 40 of file [SerialCommandExecuter.cpp](#).

4.4.3.2 setUsers()

```
void SerialCommandExecuter::setUsers (
    String * users,
    uint8_t numUsers )
```

Sets the users for user list.

Parameters

<i>users</i>	Array of the users
<i>numUsers</i>	Number of users

Definition at line 27 of file [SerialCommandExecuter.cpp](#).

The documentation for this class was generated from the following files:

- hackableEspDevice/[SerialCommandExecuter.h](#)
- hackableEspDevice/[SerialCommandExecuter.cpp](#)

4.5 UserHandler Class Reference

```
#include <userHandler.h>
```

Public Member Functions

- [UserHandler](#) (ESP8266WebServer *[server](#))
Constructor.
- void [updateUsers](#) ()
Updates the users from the config file in RAM.
- String * [getUsers](#) ()
Gets users.
- uint8_t [getNumberOfUsers](#) ()
Gets number of users.
- bool [checkPermission](#) (uint8_t permissionLevel, ESP8266WebServer *[server](#))
Checks if user has permission.

4.5.1 Detailed Description

Definition at line 17 of file [userHandler.h](#).

4.5.2 Constructor & Destructor Documentation

4.5.2.1 UserHandler()

```
UserHandler::UserHandler (
    ESP8266WebServer * server )
```

Constructor.

Parameters

<i>server</i>	Webserver object
---------------	------------------

Definition at line 17 of file [userHandler.cpp](#).

4.5.3 Member Function Documentation

4.5.3.1 checkPermission()

```
bool UserHandler::checkPermission (
    uint8_t permissionLevel,
    ESP8266WebServer * server )
```

Checks if user has permission.

Parameters

<i>permissionLevel</i>	0 = not logged in, 1 = user, 2 = admin
<i>server</i>	Webserver object

Returns

bool True if user has permission

Definition at line 94 of file [userHandler.cpp](#).

4.5.3.2 getNumberOfUsers()

```
uint8_t UserHandler::getNumberOfUsers ( )
```

Gets number of users.

Returns

Uint8_t Number of users

Definition at line 82 of file [userHandler.cpp](#).

4.5.3.3 getUsers()

```
String * UserHandler::getUsers ( )
```

Gets users.

Returns

String* String array of users

Definition at line 72 of file [userHandler.cpp](#).

4.5.3.4 updateUser()

```
void UserHandler::updateUsers ( )
```

Updates the users from the config file in RAM.

Definition at line 26 of file [userHandler.cpp](#).

The documentation for this class was generated from the following files:

- [hackableEspDevice/userHandler.h](#)
- [hackableEspDevice/userHandler.cpp](#)

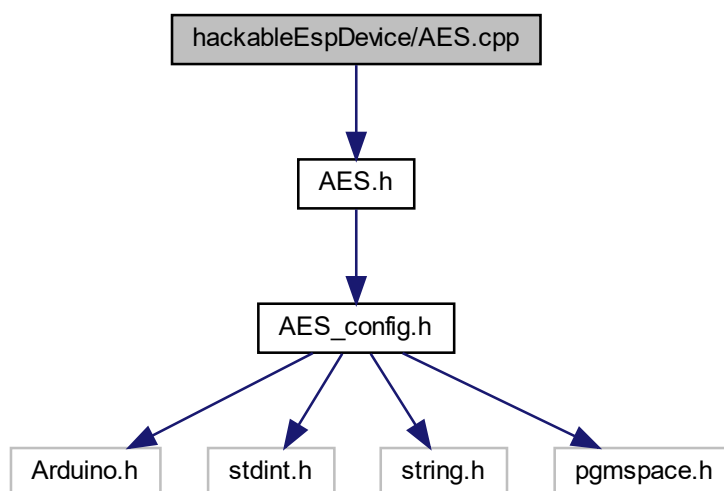
Chapter 5

File Documentation

5.1 hackableEspDevice/AES.cpp File Reference

```
#include "AES.h"
```

Include dependency graph for AES.cpp:



Macros

- `#define WPOLY 0x011B`
- `#define DPOLY 0x008D`
- `#define f2(x) ((x) & 0x80 ? (x << 1) ^ WPOLY : x << 1)`
- `#define d2(x) (((x) >> 1) ^ ((x) & 1 ? DPOLY : 0))`

5.1.1 Macro Definition Documentation

5.1.1.1 d2

```
#define d2(  
    x ) ((x) >> 1) ^ ((x) & 1 ? DPOLY : 0)
```

Definition at line 110 of file [AES.cpp](#).

5.1.1.2 DPOLY

```
#define DPOLY 0x008D
```

Definition at line 66 of file [AES.cpp](#).

5.1.1.3 f2

```
#define f2(  
    x ) ((x) & 0x80 ? (x << 1) ^ WPOLY : x << 1)
```

Definition at line 109 of file [AES.cpp](#).

5.1.1.4 WPOLY

```
#define WPOLY 0x011B
```

Definition at line 65 of file [AES.cpp](#).

5.2 AES.cpp

[Go to the documentation of this file.](#)

```

00001 #include "AES.h"
00002
00003 /*
00004 -----
00005 Copyright (c) 1998-2008, Brian Gladman, Worcester, UK. All rights reserved.
00006
00007 LICENSE TERMS
00008
00009 The redistribution and use of this software (with or without changes)
00010 is allowed without the payment of fees or royalties provided that:
00011
00012 1. source code distributions include the above copyright notice, this
00013    list of conditions and the following disclaimer;
00014
00015 2. binary distributions include the above copyright notice, this list
00016    of conditions and the following disclaimer in their documentation;
00017
00018 3. the name of the copyright holder is not used to endorse products
00019    built using this software without specific written permission.
00020
00021 DISCLAIMER
00022
00023 This software is provided 'as is' with no explicit or implied warranties
00024 in respect of its properties, including, but not limited to, correctness
00025 and/or fitness for purpose.
00026 -----
00027 Issue 09/09/2006
00028
00029 This is an AES implementation that uses only 8-bit byte operations on the
00030 cipher state (there are options to use 32-bit types if available).
00031
00032 The combination of mix columns and byte substitution used here is based on
00033 that developed by Karl Malbrain. His contribution is acknowledged.
00034 */
00035
00036 /* This version derived by Mark Tillotson 2012-01-23, tidied up, slimmed down
00037    and tailored to 8-bit microcontroller abilities and Arduino datatypes.
00038
00039    The s-box and inverse s-box were retained as tables (0.5kB PROGMEM) but all
00040    the other transformations are coded to save table space. Many efficiency
00041    improvements to the routines mix_sub_columns() and inv_mix_sub_columns()
00042    (mainly common sub-expression elimination).
00043
00044    Only the routines with precalculated subkey schedule are retained (together
00045    with set_key() - this does however mean each AES object takes 240 bytes of
00046    RAM, alas)
00047
00048    The CBC routines side-effect the iv argument (so that successive calls work
00049    together correctly).
00050
00051    All the encryption and decryption routines work with plain == cipher for
00052    in-place encryption, note.
00053
00054 */
00055
00056 /* functions for finite field multiplication in the AES Galois field */
00057
00058 /* code was modified by george spanos <spaniakos@gmail.com>
00059    * 16/12/14
00060    */
00061
00062 // GF(2^8) stuff
00063
00064 #define WPOLY 0x011B
00065 #define DPOLY 0x008D
00066
00067 static const byte s_fwd [0x100] PROGMEM =
00068 {
00069     0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
00070     0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
00071     0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
00072     0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
00073     0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
00074     0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
00075     0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
00076     0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
00077     0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
00078     0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
00079     0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
00080     0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
00081     0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0xa8,

```

```

00083 0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
00084 0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
00085 0x8c, 0x0c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16,
00086 } ;
00087
00088 static const byte s_inv [0x100] PROGMEM =
00089 {
00090 0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb,
00091 0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb,
00092 0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e,
00093 0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25,
00094 0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92,
00095 0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84,
00096 0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06,
00097 0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b,
00098 0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73,
00099 0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e,
00100 0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b,
00101 0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4,
00102 0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f,
00103 0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef,
00104 0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61,
00105 0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d,
00106 } ;
00107
00108 // times 2 in the GF(2^8)
00109 #define f2(x) ((x) & 0x80 ? (x << 1) ^ WPOLY : x << 1)
00110 #define d2(x) (((x) >> 1) ^ ((x) & 1 ? DPOLY : 0))
00111
00112 static byte s_box (byte x)
00113 {
00114 // return fwd_affine (pgm_read_byte (&inv [x])) ;
00115 return pgm_read_byte (& s_fwd [x]) ;
00116 }
00117
00118 // Inverse Sbox
00119 static byte is_box (byte x)
00120 {
00121 // return pgm_read_byte (&inv [inv_affine (x)]) ;
00122 return pgm_read_byte (& s_inv [x]) ;
00123 }
00124
00125
00126 static void xor_block (byte * d, byte * s)
00127 {
00128 for (byte i = 0 ; i < N_BLOCK ; i += 4)
00129 {
00130 *d++ ^= *s++ ; // some unrolling
00131 *d++ ^= *s++ ;
00132 *d++ ^= *s++ ;
00133 *d++ ^= *s++ ;
00134 }
00135 }
00136
00137 static void copy_and_key (byte * d, byte * s, byte * k)
00138 {
00139 for (byte i = 0 ; i < N_BLOCK ; i += 4)
00140 {
00141 *d++ = *s++ ^ *k++ ; // some unrolling
00142 *d++ = *s++ ^ *k++ ;
00143 *d++ = *s++ ^ *k++ ;
00144 *d++ = *s++ ^ *k++ ;
00145 }
00146 }
00147
00148 // #define add_round_key(d, k) xor_block (d, k)
00149
00150 /* SUB ROW PHASE */
00151
00152 static void shift_sub_rows (byte st [N_BLOCK])
00153 {
00154 st [0] = s_box (st [0]) ; st [4] = s_box (st [4]) ;
00155 st [8] = s_box (st [8]) ; st [12] = s_box (st [12]) ;
00156
00157 byte tt = st [1] ;
00158 st [1] = s_box (st [5]) ; st [5] = s_box (st [9]) ;
00159 st [9] = s_box (st [13]) ; st [13] = s_box (tt) ;
00160
00161 tt = st [2] ; st [2] = s_box (st [10]) ; st [10] = s_box (tt) ;
00162 tt = st [6] ; st [6] = s_box (st [14]) ; st [14] = s_box (tt) ;
00163
00164 tt = st [15] ;
00165 st [15] = s_box (st [11]) ; st [11] = s_box (st [7]) ;
00166 st [7] = s_box (st [3]) ; st [3] = s_box (tt) ;
00167 }
00168
00169 static void inv_shift_sub_rows (byte st[N_BLOCK])

```



```

00170 {
00171     st [0] = is_box (st[0]) ; st [4] = is_box (st [4]);
00172     st [8] = is_box (st[8]) ; st [12] = is_box (st [12]);
00173
00174     byte tt = st[13] ;
00175     st [13] = is_box (st [9]) ; st [9] = is_box (st [5]) ;
00176     st [5] = is_box (st [1]) ; st [1] = is_box (tt) ;
00177
00178     tt = st [2] ; st [2] = is_box (st [10]) ; st [10] = is_box (tt) ;
00179     tt = st [6] ; st [6] = is_box (st [14]) ; st [14] = is_box (tt) ;
00180
00181     tt = st [3] ;
00182     st [3] = is_box (st [7]) ; st [7] = is_box (st [11]) ;
00183     st [11] = is_box (st [15]) ; st [15] = is_box (tt) ;
00184 }
00185
00186 /* SUB COLUMNS PHASE */
00187
00188 static void mix_sub_columns (byte dt[N_BLOCK], byte st[N_BLOCK])
00189 {
00190     byte j = 5 ;
00191     byte k = 10 ;
00192     byte l = 15 ;
00193     for (byte i = 0 ; i < N_BLOCK ; i += N_COL)
00194     {
00195         byte a = st [i] ;
00196         byte b = st [j] ; j = (j+N_COL) & 15 ;
00197         byte c = st [k] ; k = (k+N_COL) & 15 ;
00198         byte d = st [l] ; l = (l+N_COL) & 15 ;
00199         byte a1 = s_box (a), b1 = s_box (b), c1 = s_box (c), d1 = s_box (d) ;
00200         byte a2 = f2(a1), b2 = f2(b1), c2 = f2(c1), d2 = f2(d1) ;
00201         dt[i] = a2 ^ b2^b1 ^ c1 ^ d1 ;
00202         dt[i+1] = a1 ^ b2 ^ c2^c1 ^ d1 ;
00203         dt[i+2] = a1 ^ b1 ^ c2 ^ d2^d1 ;
00204         dt[i+3] = a2^a1 ^ b1 ^ c1 ^ d2 ;
00205     }
00206 }
00207
00208 static void inv_mix_sub_columns (byte dt[N_BLOCK], byte st[N_BLOCK])
00209 {
00210     for (byte i = 0 ; i < N_BLOCK ; i += N_COL)
00211     {
00212         byte a1 = st [i] ;
00213         byte b1 = st [i+1] ;
00214         byte c1 = st [i+2] ;
00215         byte d1 = st [i+3] ;
00216         byte a2 = f2(a1), b2 = f2(b1), c2 = f2(c1), d2 = f2(d1) ;
00217         byte a4 = f2(a2), b4 = f2(b2), c4 = f2(c2), d4 = f2(d2) ;
00218         byte a8 = f2(a4), b8 = f2(b4), c8 = f2(c4), d8 = f2(d4) ;
00219         byte a9 = a8 ^ a1, b9 = b8 ^ b1, c9 = c8 ^ c1, d9 = d8 ^ d1 ;
00220         byte ac = a8 ^ a4, bc = b8 ^ b4, cc = c8 ^ c4, dc = d8 ^ d4 ;
00221
00222         dt[i] = is_box (ac^a2 ^ b9^b2 ^ cc^c1 ^ d9) ;
00223         dt[(i+5)&15] = is_box (a9 ^ bc^b2 ^ c9^c2 ^ dc^d1) ;
00224         dt[(i+10)&15] = is_box (ac^a1 ^ b9 ^ cc^c2 ^ d9^d2) ;
00225         dt[(i+15)&15] = is_box (a9^a2 ^ bc^b1 ^ c9 ^ dc^d2) ;
00226     }
00227 }
00228
00229 /*****
00230
00231 AES::AES() {
00232     byte ar_iv[8] = { 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x01 } ;
00233     memcpy(iv,ar_iv,8);
00234     memcpy(iv+8,ar_iv,8);
00235     arr_pad[0] = 0x01;
00236     arr_pad[1] = 0x02;
00237     arr_pad[2] = 0x03;
00238     arr_pad[3] = 0x04;
00239     arr_pad[4] = 0x05;
00240     arr_pad[5] = 0x06;
00241     arr_pad[6] = 0x07;
00242     arr_pad[7] = 0x08;
00243     arr_pad[8] = 0x09;
00244     arr_pad[9] = 0x0a;
00245     arr_pad[10] = 0x0b;
00246     arr_pad[11] = 0x0c;
00247     arr_pad[12] = 0x0d;
00248     arr_pad[13] = 0x0e;
00249     arr_pad[14] = 0x0f;
00250     arr_pad[15] = 0x10;
00251 }
00252
00253 /*****
00254
00255 byte AES::set_key (byte key [], int keylen)
00256 {

```

```

00257     byte hi ;
00258     switch (keylen)
00259     {
00260         case 16:
00261         case 128:
00262             keylen = 16; // 10 rounds
00263             round = 10 ;
00264             break;
00265         case 24:
00266         case 192:
00267             keylen = 24; // 12 rounds
00268             round = 12 ;
00269             break;
00270         case 32:
00271         case 256:
00272             keylen = 32; // 14 rounds
00273             round = 14 ;
00274             break;
00275         default:
00276             round = 0;
00277             return FAILURE;
00278     }
00279     hi = (round + 1) << 4 ;
00280     copy_n_bytes (key_sched, key, keylen) ;
00281     byte t[4] ;
00282     byte next = keylen ;
00283     for (byte cc = keylen, rc = 1 ; cc < hi ; cc += N_COL)
00284     {
00285         for (byte i = 0 ; i < N_COL ; i++)
00286             t[i] = key_sched [cc-4+i] ;
00287         if (cc == next)
00288         {
00289             next += keylen ;
00290             byte ttt = t[0] ;
00291             t[0] = s_box (t[1]) ^ rc ;
00292             t[1] = s_box (t[2]) ;
00293             t[2] = s_box (t[3]) ;
00294             t[3] = s_box (ttt) ;
00295             rc = f2 (rc) ;
00296         }
00297         else if (keylen == 32 && (cc & 31) == 16)
00298         {
00299             for (byte i = 0 ; i < 4 ; i++)
00300                 t[i] = s_box (t[i]) ;
00301         }
00302         byte tt = cc - keylen ;
00303         for (byte i = 0 ; i < N_COL ; i++)
00304             key_sched [cc + i] = key_sched [tt + i] ^ t[i] ;
00305     }
00306     return SUCCESS ;
00307 }
00308
00309 /*****
00310
00311 void AES::clean ()
00312 {
00313     for (byte i = 0 ; i < KEY_SCHEDULE_BYTES ; i++)
00314         key_sched [i] = 0 ;
00315     round = 0 ;
00316 }
00317
00318 /*****
00319
00320 void AES::copy_n_bytes (byte * d, byte * s, byte nn)
00321 {
00322     while (nn >= 4)
00323     {
00324         *d++ = *s++ ; // some unrolling
00325         *d++ = *s++ ;
00326         *d++ = *s++ ;
00327         *d++ = *s++ ;
00328         nn -= 4 ;
00329     }
00330     while (nn--)
00331         *d++ = *s++ ;
00332 }
00333
00334 /*****
00335
00336 byte AES::encrypt (byte plain [N_BLOCK], byte cipher [N_BLOCK])
00337 {
00338     if (round)
00339     {
00340         byte s1 [N_BLOCK], r ;
00341         copy_and_key (s1, plain, (byte*) (key_sched)) ;
00342
00343         for (r = 1 ; r < round ; r++)

```

```

00344     {
00345         byte s2 [N_BLOCK] ;
00346         mix_sub_columns (s2, s1) ;
00347         copy_and_key (s1, s2, (byte*) (key_sched + r * N_BLOCK)) ;
00348     }
00349     shift_sub_rows (s1) ;
00350     copy_and_key (cipher, s1, (byte*) (key_sched + r * N_BLOCK)) ;
00351 }
00352 else
00353     return FAILURE ;
00354 return SUCCESS ;
00355 }
00356
00357 /*****
00358
00359 byte AES::cbc_encrypt (byte * plain, byte * cipher, int n_block, byte iv [N_BLOCK])
00360 {
00361     while (n_block--)
00362     {
00363         xor_block (iv, plain) ;
00364         if (encrypt (iv, iv) != SUCCESS)
00365             return FAILURE ;
00366         copy_n_bytes (cipher, iv, N_BLOCK) ;
00367         plain += N_BLOCK ;
00368         cipher += N_BLOCK ;
00369     }
00370     return SUCCESS ;
00371 }
00372
00373 /*****
00374
00375 byte AES::cbc_encrypt (byte * plain, byte * cipher, int n_block)
00376 {
00377     while (n_block--)
00378     {
00379         xor_block (iv, plain) ;
00380         if (encrypt (iv, iv) != SUCCESS)
00381             return FAILURE ;
00382         copy_n_bytes (cipher, iv, N_BLOCK) ;
00383         plain += N_BLOCK ;
00384         cipher += N_BLOCK ;
00385     }
00386     return SUCCESS ;
00387 }
00388
00389 /*****
00390
00391 byte AES::decrypt (byte plain [N_BLOCK], byte cipher [N_BLOCK])
00392 {
00393     if (round)
00394     {
00395         byte s1 [N_BLOCK] ;
00396         copy_and_key (s1, plain, (byte*) (key_sched + round * N_BLOCK)) ;
00397         inv_shift_sub_rows (s1) ;
00398
00399         for (byte r = round ; --r ; )
00400         {
00401             byte s2 [N_BLOCK] ;
00402             copy_and_key (s2, s1, (byte*) (key_sched + r * N_BLOCK)) ;
00403             inv_mix_sub_columns (s1, s2) ;
00404         }
00405         copy_and_key (cipher, s1, (byte*) (key_sched)) ;
00406     }
00407     else
00408         return FAILURE ;
00409     return SUCCESS ;
00410 }
00411
00412 /*****
00413
00414 byte AES::cbc_decrypt (byte * cipher, byte * plain, int n_block, byte iv [N_BLOCK])
00415 {
00416     while (n_block--)
00417     {
00418         byte tmp [N_BLOCK] ;
00419         copy_n_bytes (tmp, cipher, N_BLOCK) ;
00420         if (decrypt (cipher, plain) != SUCCESS)
00421             return FAILURE ;
00422         xor_block (plain, iv) ;
00423         copy_n_bytes (iv, tmp, N_BLOCK) ;
00424         plain += N_BLOCK ;
00425         cipher += N_BLOCK ;
00426     }
00427     return SUCCESS ;
00428 }
00429
00430 /*****

```

```

00431
00432 byte AES::cbc_decrypt (byte * cipher, byte * plain, int n_block)
00433 {
00434     while (n_block--)
00435     {
00436         byte tmp [N_BLOCK] ;
00437         copy_n_bytes (tmp, cipher, N_BLOCK) ;
00438         if (decrypt (cipher, plain) != SUCCESS)
00439             return FAILURE ;
00440         xor_block (plain, iv) ;
00441         copy_n_bytes (iv, tmp, N_BLOCK) ;
00442         plain += N_BLOCK ;
00443         cipher += N_BLOCK;
00444     }
00445     return SUCCESS ;
00446 }
00447
00448 /*****
00449
00450 void AES::set_IV(unsigned long long int IVc1){
00451     memcpy(iv,&IVc1,8);
00452     memcpy(iv+8,&IVc1,8);
00453     IVC = IVc1;
00454 }
00455
00456 /*****
00457
00458 void AES::iv_inc(){
00459     IVC += 1;
00460     memcpy(iv,&IVC,8);
00461     memcpy(iv+8,&IVC,8);
00462 }
00463
00464 /*****
00465
00466 int AES::get_size(){
00467     return size;
00468 }
00469
00470 /*****
00471
00472 int AES::get_pad(){
00473     return pad;
00474 }
00475
00476 /*****
00477
00478 void AES::set_size(int sizel){
00479     size = sizel;
00480 }
00481
00482
00483 /*****
00484
00485 void AES::get_IV(byte *out){
00486     memcpy(out,&IVC,8);
00487     memcpy(out+8,&IVC,8);
00488 }
00489
00490 /*****
00491
00492 void AES::calc_size_n_pad(int p_size){
00493     pad = N_BLOCK - p_size % N_BLOCK;
00494     size = p_size + pad;
00495 }
00496
00497 /*****
00498
00499 void AES::padPlaintext(void* in,byte* out)
00500 {
00501     memcpy(out,in,size);
00502     for (int i = size-pad; i < size; i++){
00503         out[i] = arr_pad[pad - 1];
00504     }
00505 }
00506
00507 /*****
00508
00509 bool AES::CheckPad(byte* in,int lsize){
00510     if (in[lsize-1] <= 0x10){
00511         int lpad = (int)in[lsize-1];
00512         for (int i = lsize - 1; i >= lsize-lpad; i--){
00513             if (arr_pad[lpad - 1] != in[i]){
00514                 return false;
00515             }
00516         }
00517     }else{

```

```

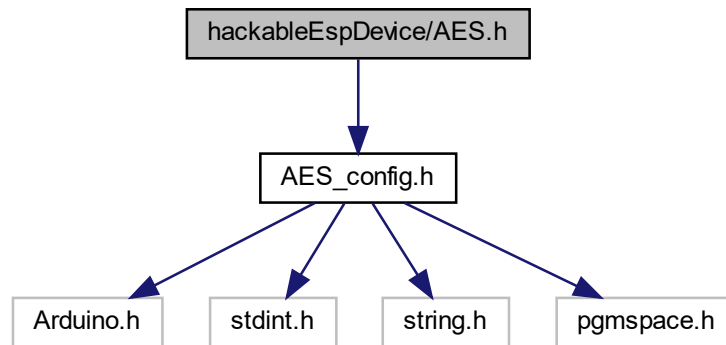
00518         return true;
00519     }
00520     return true;
00521 }
00522
00523 /*****
00524
00525 void AES::printArray(byte output[],bool p_pad)
00526 {
00527     uint8_t i,j;
00528     uint8_t loops = size/N_BLOCK;
00529     uint8_t outp = N_BLOCK;
00530     for (j = 0; j < loops; j += 1){
00531         if (p_pad && (j == (loops - 1)) ) { outp = N_BLOCK - pad; }
00532         for (i = 0; i < outp; i++)
00533         {
00534             printf_P(PSTR("%c"),output[j*N_BLOCK + i]);
00535         }
00536     }
00537     printf_P(PSTR("\n"));
00538 }
00539
00540 /*****
00541
00542 void AES::printArray(byte output[],int sizel)
00543 {
00544     for (int i = 0; i < sizel; i++)
00545     {
00546         printf_P(PSTR("%02x"),output[i]); // print hex in fixed 2-cgar format
00547     }
00548     printf_P(PSTR("\n"));
00549 }
00550
00551
00552 /*****
00553
00554 void AES::do_aes_encrypt(byte *plain,int size_p,byte *cipher,byte *key, int bits, byte ivl [N_BLOCK]){
00555     calc_size_n_pad(size_p);
00556     byte plain_p[get_size()];
00557     padPlaintext(plain,plain_p);
00558     int blocks = get_size() / N_BLOCK;
00559     set_key (key, bits) ;
00560     cbc_encrypt (plain_p, cipher, blocks, ivl);
00561 }
00562
00563 /*****
00564
00565 void AES::do_aes_encrypt(byte *plain,int size_p,byte *cipher,byte *key, int bits){
00566     calc_size_n_pad(size_p);
00567     byte plain_p[get_size()];
00568     padPlaintext(plain,plain_p);
00569     int blocks = get_size() / N_BLOCK;
00570     set_key (key, bits) ;
00571     cbc_encrypt (plain_p, cipher, blocks);
00572 }
00573
00574 /*****
00575
00576 void AES::do_aes_decrypt(byte *cipher,int size_c,byte *plain,byte *key, int bits, byte ivl [N_BLOCK]){
00577     set_size(size_c);
00578     int blocks = size_c / N_BLOCK;
00579     set_key (key, bits);
00580     cbc_decrypt (cipher,plain, blocks, ivl);
00581 }
00582
00583 /*****
00584
00585 void AES::do_aes_decrypt(byte *cipher,int size_c,byte *plain,byte *key, int bits){
00586     set_size(size_c);
00587     int blocks = size_c / N_BLOCK;
00588     set_key (key, bits);
00589     cbc_decrypt (cipher,plain, blocks);
00590 }
00591
00592
00593 /*****
00594
00595 #if defined(AES_LINUX)
00596 double AES::millis(){
00597     gettimeofday(&tv, NULL);
00598     return (tv.tv_sec + 0.000001 * tv.tv_usec);
00599 }
00600 #endif

```

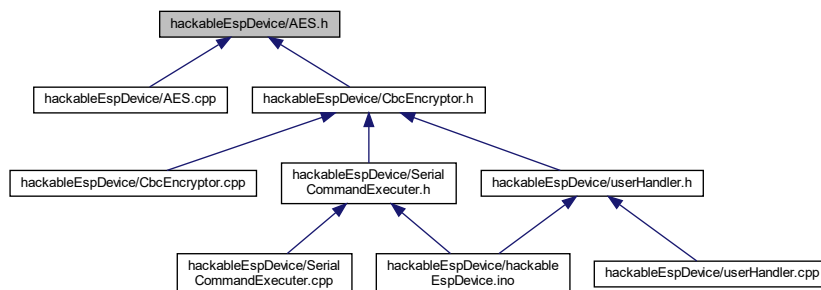
5.3 hackableEspDevice/AES.h File Reference

```
#include "AES_config.h"
```

Include dependency graph for AES.h:



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



Classes

- class [AES](#)

5.4 AES.h

[Go to the documentation of this file.](#)

```

00001 #ifndef __AES_H__
00002 #define __AES_H__
00003
00004 #include "AES_config.h"
00005 /*
00006 -----
00007 Copyright (c) 1998-2008, Brian Gladman, Worcester, UK. All rights reserved.
00008

```

```

00009  LICENSE TERMS
00010
00011  The redistribution and use of this software (with or without changes)
00012  is allowed without the payment of fees or royalties provided that:
00013
00014      1. source code distributions include the above copyright notice, this
00015         list of conditions and the following disclaimer;
00016
00017      2. binary distributions include the above copyright notice, this list
00018         of conditions and the following disclaimer in their documentation;
00019
00020      3. the name of the copyright holder is not used to endorse products
00021         built using this software without specific written permission.
00022
00023  DISCLAIMER
00024
00025  This software is provided 'as is' with no explicit or implied warranties
00026  in respect of its properties, including, but not limited to, correctness
00027  and/or fitness for purpose.
00028  -----
00029  Issue 09/09/2006
00030
00031  This is an AES implementation that uses only 8-bit byte operations on the
00032  cipher state.
00033  */
00034
00035  /* code was modified by george spanos <spaniakos@gmail.com>
00036  * 16/12/14
00037  */
00038
00039  class AES
00040  {
00041  public:
00042
00043      /* The following calls are for a precomputed key schedule
00044
00045         NOTE: If the length_type used for the key length is an
00046         unsigned 8-bit character, a key length of 256 bits must
00047         be entered as a length in bytes (valid inputs are hence
00048         128, 192, 16, 24 and 32).
00049      */
00050      AES();
00051
00052      byte set_key (byte key[], int keylen) ;
00053
00054      void clean () ; // delete key schedule after use
00055
00056      void copy_n_bytes (byte * AEst, byte * src, byte n) ;
00057
00058      byte encrypt (byte plain [N_BLOCK], byte cipher [N_BLOCK]) ;
00059
00060      byte cbc_encrypt (byte * plain, byte * cipher, int n_block, byte iv [N_BLOCK]) ;
00061
00062      byte cbc_encrypt (byte * plain, byte * cipher, int n_block) ;
00063
00064      byte decrypt (byte cipher [N_BLOCK], byte plain [N_BLOCK]) ;
00065
00066      byte cbc_decrypt (byte * cipher, byte * plain, int n_block, byte iv [N_BLOCK]) ;
00067
00068      byte cbc_decrypt (byte * cipher, byte * plain, int n_block) ;
00069
00070      void set_IV(unsigned long long int IVCl);
00071
00072      void iv_inc();
00073
00074      int get_size();
00075
00076      void set_size(int sizel);
00077
00078      int get_pad();
00079
00080      void get_IV(byte *out);
00081
00082      void calc_size_n_pad(int p_size);
00083
00084      void padPlaintext(void* in,byte* out);
00085
00086      bool CheckPad(byte* in,int size);
00087
00088      void printArray(byte output[],bool p_pad = true);
00089
00090      void printArray(byte output[],int sizel);
00091
00092      void do_aes_encrypt(byte *plain,int size_p,byte *cipher,byte *key, int bits, byte ivl [N_BLOCK]);
00093
00094      void do_aes_encrypt (byte *plain,int size_p,byte *cipher,byte *key, int bits);

```

```

00266
00277 void do_aes_decrypt(byte *cipher,int size_c,byte *plain,byte *key, int bits, byte ivl [N_BLOCK]);
00278
00288 void do_aes_decrypt(byte *cipher,int size_c,byte *plain,byte *key, int bits);
00289
00290 #if defined(AES_LINUX)
00296     double millis();
00297 #endif
00298 private:
00299     int round ;
00300     byte key_sched [KEY_SCHEDULE_BYTES] ;
00301     unsigned long long int IVC;
00302     byte iv[16];
00303     int pad;
00304     int size;
00305     #if defined(AES_LINUX)
00306         timeval tv;
00307         byte arr_pad[16];
00308     #else
00309         byte arr_pad[16] = {
00310             0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0a,0x0b,0x0c,0x0d,0x0e,0x0f,0x10 };
00311     #endif
00312 } ;
00313
00314 #endif
00315

```

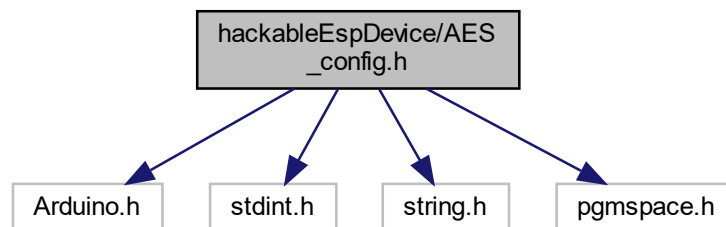
5.5 hackableEspDevice/AES_config.h File Reference

```

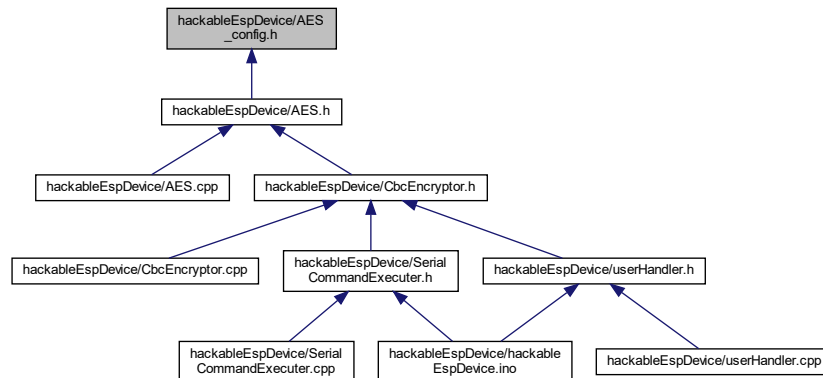
#include <Arduino.h>
#include <stdint.h>
#include <string.h>
#include <pgmspace.h>

```

Include dependency graph for AES_config.h:



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



Macros

- `#define N_ROW 4`
- `#define N_COL 4`
- `#define N_BLOCK (N_ROW * N_COL)`
- `#define N_MAX_ROUNDS 14`
- `#define KEY_SCHEDULE_BYTES ((N_MAX_ROUNDS + 1) * N_BLOCK)`
- `#define SUCCESS (0)`
- `#define FAILURE (-1)`

5.5.1 Macro Definition Documentation

5.5.1.1 FAILURE

```
#define FAILURE (-1)
```

Definition at line 53 of file [AES_config.h](#).

5.5.1.2 KEY_SCHEDULE_BYTES

```
#define KEY_SCHEDULE_BYTES ((N_MAX_ROUNDS + 1) * N_BLOCK)
```

Definition at line 51 of file [AES_config.h](#).

5.5.1.3 N_BLOCK

```
#define N_BLOCK (N_ROW * N_COL)
```

Definition at line 49 of file [AES_config.h](#).

5.5.1.4 N_COL

```
#define N_COL 4
```

Definition at line 48 of file [AES_config.h](#).

5.5.1.5 N_MAX_ROUNDS

```
#define N_MAX_ROUNDS 14
```

Definition at line 50 of file [AES_config.h](#).

5.5.1.6 N_ROW

```
#define N_ROW 4
```

Definition at line 47 of file [AES_config.h](#).

5.5.1.7 SUCCESS

```
#define SUCCESS (0)
```

Definition at line 52 of file [AES_config.h](#).

5.6 AES_config.h

[Go to the documentation of this file.](#)

```

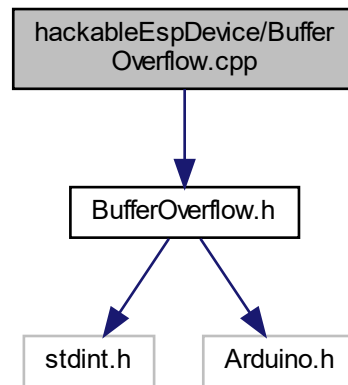
00001 /* code was modified by george spanos <spaniakos@gmail.com>
00002  * 16/12/14
00003  */
00004
00005 #ifndef __AES_CONFIG_H__
00006 #define __AES_CONFIG_H__
00007
00008 #if (defined(__linux) || defined(linux)) && !(defined(__ARDUINO_X86__) || defined(__arm__))
00009
00010     #define AES_LINUX
00011
00012     #include <stdint.h>
00013     #include <stdio.h>
00014     #include <stdlib.h>
00015     #include <string.h>
00016     #include <sys/time.h>
00017     #include <unistd.h>
00018 #else
00019     #include <Arduino.h>
00020 #endif
00021
00022 #include <stdint.h>
00023 #include <string.h>
00024
00025 #if defined(__ARDUINO_X86__) || defined(__arm__) || (defined (__linux) || defined (linux))
00026     #undef PROGMEM
00027     #define PROGMEM __attribute__(( section(".progmem.data") ))
00028     #define pgm_read_byte(p) (*(p))
00029     typedef unsigned char byte;
00030     #define printf_P printf
00031     #ifndef PSTR
00032         #define PSTR(x) (x)
00033     #endif
00034 #elif defined ( ESP8266 )
00035     #include <pgmspace.h>
00036     #ifndef PSTR
00037         #define PSTR(x) (x)
00038     #endif
00039 #else
00040     #if (defined(__AVR__))
00041         #include <avr/pgmspace.h>
00042     #else
00043         #include <pgmspace.h>
00044     #endif
00045 #endif
00046
00047 #define N_ROW 4
00048 #define N_COL 4
00049 #define N_BLOCK (N_ROW * N_COL)
00050 #define N_MAX_ROUNDS 14
00051 #define KEY_SCHEDULE_BYTES ((N_MAX_ROUNDS + 1) * N_BLOCK)
00052 #define SUCCESS (0)
00053 #define FAILURE (-1)
00054
00055 #endif

```

5.7 hackableEspDevice/BufferOverflow.cpp File Reference

```
#include "BufferOverflow.h"
```

Include dependency graph for BufferOverflow.cpp:



5.8 BufferOverflow.cpp

[Go to the documentation of this file.](#)

```

00001 /*
00002  * File:      BufferOverflow.h
00003  * Author:    Luke de Munk
00004  * Class:     BufferOverflow
00005  * Version:   1.0
00006  *
00007  * Buffer overflow simulator based on a Linux cli.
00008  * All elements of the bufferflow are in this class.
00009  */
00010 #include "BufferOverflow.h"
00011
00012 /*****
00016 /*****
00017 BufferOverflow::BufferOverflow() {
00018     _clearInput();                                     //First time call is the
00019     declaration of the array.
00020 }
00021 /*****
00025 /*****
00026 void BufferOverflow::ls() {
00027     Serial.println(F("testprogram.c"));
00028     Serial.println(F("testprogram"));
00029 }
00030
00031 /*****
00035 /*****
00036 void BufferOverflow::vi() {
00037
00038     Serial.println(F("|-----FILENAME-----|-----TYPE-----|-----AUTHOR-----|"));
00039     Serial.println(F("|-----testprogram.c-----|-----READONLY-----|-----admin-----|"));
00040     Serial.println(F("|-----|-----|-----|"));
00041     Serial.println(F("1      /*
00042     |"));
00043     Serial.println(F("2      * File:  testprogram.c
00044     |"));
00045     Serial.println(F("3      * Author: admin
00046     |"));
  
```

```

00043     Serial.println(F("4      *
|"));
00044     Serial.println(F("5      * To test superuser login. DELETE WHEN FINISHING DEVELOPMENT!!!
|"));
00045     Serial.println(F("6      */
|"));
00046     Serial.println(F("7      #include <stdio.h>
|"));
00047     Serial.println(F("8      #include <string.h>
|"));
00048     Serial.println(F("9
|"));
00049     Serial.println(F("10
|"));
00050     Serial.println(F("11
|"));
00051     Serial.println(F("12      /*!
|"));
00052     Serial.println(F("13      @brief      Logs given user name in as superuser and logs out again.
|"));
00053     Serial.println(F("14      */
|"));
00054     Serial.println(F("15
|"));
00055     Serial.println(F("16      int main(int argc, char** argv) {
|"));
00056     Serial.println(F("17      char username[10];
|"));
00057     Serial.println(F("18      strcpy(username, argv[1]);
|"));
00058     Serial.println(F("19      login(*username);
|"));
00059     Serial.println(F("20      logout();
|"));
00060     Serial.println(F("21
|"));
00061     Serial.println(F("22      return 0;
|"));
00062     Serial.println(F("23      }
|"));
00063     Serial.println(F("-----|"));
00064     Serial.println(F(""));
00065 }
00066
00067 /*****
00071 /*****
00072 void BufferOverflow::objectDump() {
00073     Serial.println(F("testprogram:      file format elf32-littlearm"));
00074     Serial.println(F(""));
00075     Serial.println(F("Disassembly of section .init:"));
00076     Serial.println(F(""));
00077     Serial.println(F("00010438 <main>:"));
00078     Serial.println(F("      10438: e92d4800  push {fp, lr}));
00079     Serial.println(F("      1043c: e28db004  add fp, sp, #4));
00080     Serial.println(F("      10440: e24dd018  sub sp, sp, #24));
00081     Serial.println(F("      10444: e50b0018  str r0, [fp, #-24] ; 0xffffffffe8));
00082     Serial.println(F("      10448: e50b101c  str r1, [fp, #-28] ; 0xffffffffe4));
00083     Serial.println(F("      1044c: e51b301c  ldr r3, [fp, #-28] ; 0xffffffffe4));
00084     Serial.println(F("      10450: e2833004  add r3, r3, #4));
00085     Serial.println(F("      10454: e5932000  ldr r2, [r3]));
00086     Serial.println(F("      10458: e24b3010  sub r3, fp, #16));
00087     Serial.println(F("      1045c: e1a01002  mov r1, r2));
00088     Serial.println(F("      10460: e1a00003  mov r0, r3));
00089     Serial.println(F("      10464: ebffffab  bl 10318 <strcpy@plt>));
00090     Serial.println(F("      10468: e55b3010  ldrb r3, [fp, #-16]));
00091     Serial.println(F("      1046c: e1a00003  mov r0, r3));
00092     Serial.println(F("      10470: eb000004  bl 10488 <login>));
00093     Serial.println(F("      10474: eb00000d  bl 104b0 <logout>));
00094     Serial.println(F("      10478: e3a03000  mov r3, #0));
00095     Serial.println(F("      1047c: e1a00003  mov r0, r3));
00096     Serial.println(F("      10480: e24bd004  sub sp, fp, #4));
00097     Serial.println(F("      10484: e8bd8800  pop {fp, pc}));
00098     Serial.println(F(""));
00099     Serial.println(F("00010488 <login>:"));
00100     Serial.println(F("      10488: e92d4800  push {fp, lr}));
00101     Serial.println(F("      1048c: e28db004  add fp, sp, #4));
00102     Serial.println(F("      10490: e24dd008  sub sp, sp, #8));
00103     Serial.println(F("      10494: e50b0008  str r0, [fp, #-8]));
00104     Serial.println(F("      10498: e59f000c  ldr r0, [pc, #12] ; 104ac <login+0x24>));
00105     Serial.println(F("      1049c: ebffff9a  bl 1030c <printf@plt>));
00106     Serial.println(F("      104a0: e1a00000  nop      ; (mov r0, r0));
00107     Serial.println(F("      104a4: e24bd004  sub sp, fp, #4));
00108     Serial.println(F("      104a8: e8bd8800  pop {fp, pc}));
00109     Serial.println(F("      104ac: 0001053c  .word 0x0001053c));
00110     Serial.println(F(""));
00111     Serial.println(F("000104b0 <logout>:"));

```

```

00112     Serial.println(F("    104b0: e92d4800 push {fp, lr}"));
00113     Serial.println(F("    104b4: e28db004 add fp, sp, #4"));
00114     Serial.println(F("    104b8: e59f0008 ldr r0, [pc, #8] ; 104c8 <logout+0x18>"));
00115     Serial.println(F("    104bc: ebffff92 bl 1030c <printf@plt>"));
00116     Serial.println(F("    104c0: e1a00000 nop ; (mov r0, r0)"));
00117     Serial.println(F("    104c4: e8bd8800 pop {fp, pc}"));
00118     Serial.println(F("    104c8: 00010548 .word 0x00010548"));
00119     Serial.println(F(""));
00120 }
00121
00122 /*****
00128 /*****
00129 bool BufferOverflow::runCProgram(String arg) {
00130     _formatInput(arg);
00131
00132     if (_numChars < OVERFLOW_BEGIN) {
00133         Serial.println("You are now super user.");
00134         Serial.print("Hello ");
00135         Serial.println(arg);
00136         Serial.println("You are not longer super user.");
00137     } else {
00138         if (_checkBufferOverflow()) {
00139             return true;
00140         }
00141     }
00142     return false;
00143 }
00144
00145 /*****
00150 /*****
00151 bool BufferOverflow::_checkBufferOverflow() {
00152     if(_getOverflowPortion() == RETURN_ADDRESS) {
00153         return true;
00154     }
00155
00156     _printOverflowError(); //If the overflow is not
    correctly, print value of the return address pointer
00157     return false;
00158 }
00159
00160 /*****
00164 /*****
00165 void BufferOverflow::_printOverflowError() {
00166     Serial.println("Program received signal SIGSEGV, Segmentation fault.");
00167     Serial.print("0x");
00168     _getOverflowPortion(true);
00169     Serial.println(" in ?? ()");
00170 }
00171
00172 /*****
00177 /*****
00178 void BufferOverflow::_formatInput(String input) {
00179     String tmp = "";
00180
00181     _clearInput();
00182
00183     /* Set every character in an element */
00184     for (uint16_t i = 0; i < input.length(); i++) {
00185         if (input[i] == '\\') {
00186             _formattedInput[_numChars] = "\\x"; //Move all hex chars in
    one element (for ex.: '\x90')
00187             _formattedInput[_numChars] += input[i+2];
00188             _formattedInput[_numChars] += input[i+3];
00189             i += 3; //Increase with 3, because
    the number of chars taken for a hex is 4 ('\x90')
00190         } else {
00191             _formattedInput[_numChars] = input[i];
00192         }
00193         _numChars++;
00194     }
00195
00196     /* Turn the whole array, to simulate little endian systems */
00197     for (uint8_t i = 0; i < _numChars/2-1; i++) {
00198         tmp = _formattedInput[i];
00199         _formattedInput[i] = _formattedInput[_numChars-i-1];
00200         _formattedInput[_numChars-i-1] = tmp;
00201         tmp = "";
00202     }
00203 }
00204
00205 /*****
00211 /*****
00212 String BufferOverflow::_getOverflowPortion(bool print) {
00213     String overflowPortion = "";
00214
00215     if (_numChars < OVERFLOW_LENGTH) {
00216         uint8_t numMissingBytes = OVERFLOW_LENGTH - _numChars;

```

```

00217         overflowPortion += _generateRandomBytes(numMissingBytes);
00218
00219         if (print) {
00220             Serial.print(overflowPortion);
00221         }
00222
00223         /* To determine and print the overflow portion */
00224         for (uint8_t i = 0; i < ADDRESS_LENGTH - numMissingBytes; i++) {
00225             /* Check if is hex number, else print as hex */
00226             if (_formattedInput[i][0] == '\\') {
00227                 overflowPortion += _formattedInput[i][2];
00228                 overflowPortion += _formattedInput[i][3];
00229                 if (print) {
00230                     Serial.print(_formattedInput[i][2]);
00231                     Serial.print(_formattedInput[i][3]);
00232                 }
00233             } else {
00234                 overflowPortion += _formattedInput[i];
00235                 if (print) {
00236                     Serial.print(char(_formattedInput[i][0]), HEX);
00237                 }
00238             }
00239         }
00240     } else {
00241         /* To print the overflow portion */
00242         uint8_t delta = abs(_numChars - OVERFLOW_LENGTH);
00243         for (uint8_t i = delta; i < delta + ADDRESS_LENGTH; i++) {
00244             /* Check if is hex number, else print as hex */
00245             if (_formattedInput[i][0] == '\\') {
00246                 overflowPortion += _formattedInput[i][2];
00247                 overflowPortion += _formattedInput[i][3];
00248                 if (print) {
00249                     Serial.print(_formattedInput[i][2]);
00250                     Serial.print(_formattedInput[i][3]);
00251                 }
00252             } else {
00253                 if (print) {
00254                     Serial.print(char(_formattedInput[i][0]), HEX);
00255                 }
00256                 overflowPortion += _formattedInput[i];
00257             }
00258         }
00259     }
00260     return overflowPortion;
00261 }
00262
00263 /*****
00264 /*****
00265 void BufferOverflow::_clearInput() {
00266     for (uint16_t i = 0; i < MAX_NUM_CHARS; i++) {
00267         _formattedInput[i] = "";
00268     }
00269     _numChars = 0;
00270 }
00271
00272 /*****
00273 /*****
00274 String BufferOverflow::_generateRandomBytes(uint8_t numBytes) {
00275     String bytes = "";
00276     randomSeed(numBytes);
00277
00278     for (uint8_t i = 0; i < numBytes; i++) {
00279         bytes += String(random(127), HEX);
00280     }
00281     return bytes;
00282 }
00283

```

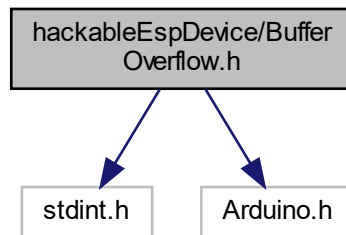
5.9 hackableEspDevice/BufferOverflow.h File Reference

```

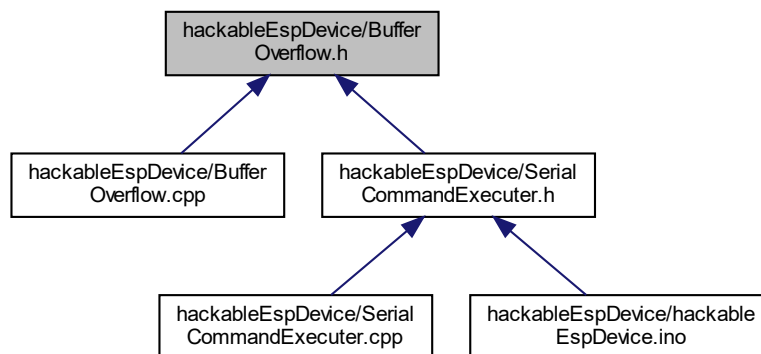
#include <stdint.h>
#include "Arduino.h"

```

Include dependency graph for BufferOverflow.h:



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



Classes

- class [BufferOverflow](#)

Macros

- `#define OVERFLOW_BEGIN 16`
- `#define ADDRESS_LENGTH 4`
- `#define OVERFLOW_LENGTH 20`
- `#define RETURN_ADDRESS "00010488"`
- `#define MAX_NUM_CHARS 256`

5.9.1 Macro Definition Documentation

5.9.1.1 ADDRESS_LENGTH

```
#define ADDRESS_LENGTH 4
```

Definition at line 16 of file [BufferOverflow.h](#).

5.9.1.2 MAX_NUM_CHARS

```
#define MAX_NUM_CHARS 256
```

Definition at line 19 of file [BufferOverflow.h](#).

5.9.1.3 OVERFLOW_BEGIN

```
#define OVERFLOW_BEGIN 16
```

Definition at line 15 of file [BufferOverflow.h](#).

5.9.1.4 OVERFLOW_LENGTH

```
#define OVERFLOW_LENGTH 20
```

Definition at line 17 of file [BufferOverflow.h](#).

5.9.1.5 RETURN_ADDRESS

```
#define RETURN_ADDRESS "00010488"
```

Definition at line 18 of file [BufferOverflow.h](#).

5.10 BufferOverflow.h

[Go to the documentation of this file.](#)

```

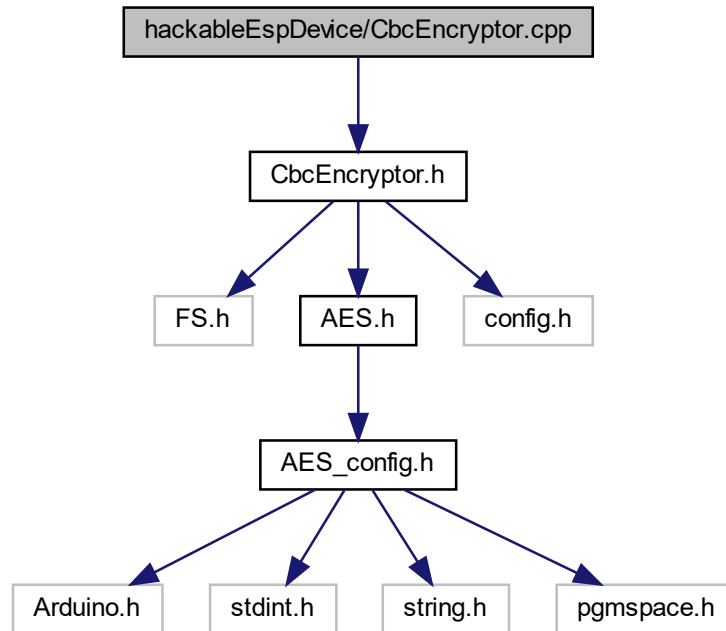
00001  /*
00002   * File:      BufferOverflow.h
00003   * Author:    Luke de Munk
00004   * Class:     BufferOverflow
00005   * Version:   1.0
00006   *
00007   * Buffer overflow simulator based on a Linux cli.
00008   * All elements of the bufferflow are in this class.
00009   */
00010 #ifndef BUFFER_OVERFLOW_H
00011 #define BUFFER_OVERFLOW_H
00012 #include <stdint.h>                                //For defining bits per
                                                    integer
00013 #include "Arduino.h"
00014
00015 #define OVERFLOW_BEGIN      16                    //Because array is in byte
                                                    resolution, 10 becomes 16. Then the return address pointer starts
00016 #define ADDRESS_LENGTH      4                    //Address is 32 bits long,
                                                    so 4 bytes
00017 #define OVERFLOW_LENGTH     20                    //OVERFLOW_BEGIN +
                                                    ADDRESS_LENGTH
00018 #define RETURN_ADDRESS      "00010488"           //0x00010488 == address of
                                                    login function.
00019 #define MAX_NUM_CHARS       256
00020
00021 class BufferOverflow
00022 {
00023     public:
00024         BufferOverflow();
00025         void ls();
00026         void vi();
00027         void objectDump();
00028         bool runCProgram(String arg);
00029
00030     private:
00031         bool _checkBufferOverflow();
00032         void _printOverflowError();
00033         void _formatInput(String input);
00034         String _getOverflowPortion(bool print = false);
00035         void _clearInput();
00036         String _generateRandomBytes(uint8_t numberOfBytes);
00037
00038         String _formattedInput[256];
00039         uint8_t _numChars;
00040 };
00041 #endif

```

5.11 hackableEspDevice/CbcEncryptor.cpp File Reference

```
#include "CbcEncryptor.h"
```

Include dependency graph for CbcEncryptor.cpp:



5.12 CbcEncryptor.cpp

[Go to the documentation of this file.](#)

```

00001 /*
00002  * File:      CbcEncryptor.cpp
00003  * Author:    Luke de Munk
00004  * Class:     CbcEncryptor
00005  * Version:   1.0
00006  *
00007  * Handles encryption and decryption of files. Uses AES encryption.
00008  */
00009 #include "CbcEncryptor.h"
00010
00011 /*****
00015 *****/
00016 CbcEncryptor::CbcEncryptor() {
00017     _aesInitVecInt = CBC_INIT_VECTOR;
00018     _aesKeyString = AES_KEY;
00019
00020     /* Convert key string to bytes */
00021     for (uint8_t i = 0; i < AES_KEY_SIZE; i++) {
00022         _aesKey[i] = (byte) _aesKeyString[i];
00023     }
00024 }
00025
00026 /*****
00032 *****/
00033 bool CbcEncryptor::encryptFile(String filename) {
00034     if (!SPIFFS.exists(filename)) {
00035         return false;
00036     }

```

```

00037
00038     setKey(AES_KEY); //Reset key
00039     _aesKeyString = AES_KEY;
00040
00041     File inputFile = SPIFFS.open(filename, "r"); //Plaintext file
00042     File tmpFile = SPIFFS.open("tmp_" + filename, "w"); //Temporary file to store
encrypted part
00043
00044     /* If already is plaintext, don't do anything */
00045     if (inputFile.readStringUntil('\n') == "format: encrypted") {
00046         return false;
00047     }
00048
00049     tmpFile.println("format: encrypted"); //Save format
00050     String line = inputFile.readStringUntil('\n');
00051
00052     _aes.iv_inc(); //Unique initialization
vector every encryption
00053
00054     /* Encrypt the file line by line */
00055     while (line != "") {
00056         tmpFile.println(encryptLine(line));
00057         line = inputFile.readStringUntil('\n'); //Read a line from the
file
00058     }
00059
00060     inputFile.close();
00061     tmpFile.close();
00062
00063     SPIFFS.remove(filename); //Remove plain text file
00064     SPIFFS.rename("tmp_" + filename, filename); //Rename tmp file
00065     return true;
00066 }
00067
00068 /*****/
00074 /*****/
00075 bool CbcEncryptor::decryptFile(String filename) {
00076     if (!SPIFFS.exists(filename)) {
00077         return false;
00078     }
00079
00080     setKey(AES_KEY); //Reset key
00081     _aesKeyString = AES_KEY;
00082
00083     File inputFile = SPIFFS.open(filename, "r");
00084     File tmpFile = SPIFFS.open("tmp_" + filename, "w");
00085
00086     /* If already is plaintext, don't do anything */
00087     if (inputFile.readStringUntil('\n') == "format: plaintext") {
00088         return false;
00089     }
00090
00091     tmpFile.println("format: plaintext"); //Save format
00092     String line = inputFile.readStringUntil('\n');
00093
00094     /* Encrypt the file line by line */
00095     while (line != "") {
00096         tmpFile.println(decryptLine(line));
00097         line = inputFile.readStringUntil('\n'); //Read a line from the
file
00098     }
00099
00100     inputFile.close();
00101     tmpFile.close();
00102
00103     SPIFFS.remove(filename); //Remove plain text file
00104     SPIFFS.rename("tmp_" + filename, filename); //Rename tmp file
00105     return true;
00106 }
00107
00108 /*****/
00114 /*****/
00115 bool CbcEncryptor::setKey(String key) {
00116     _aesKeyString = key;
00117
00118     /* Check if key is the right length */
00119     if (key.length() != AES_KEY_SIZE) {
00120         return false;
00121     }
00122
00123     /* Convert key string to bytes */
00124     for (uint8_t i = 0; i < AES_KEY_SIZE; i++) {
00125         _aesKey[i] = (byte) key[i];
00126     }
00127     return true;
00128 }
00129

```

```

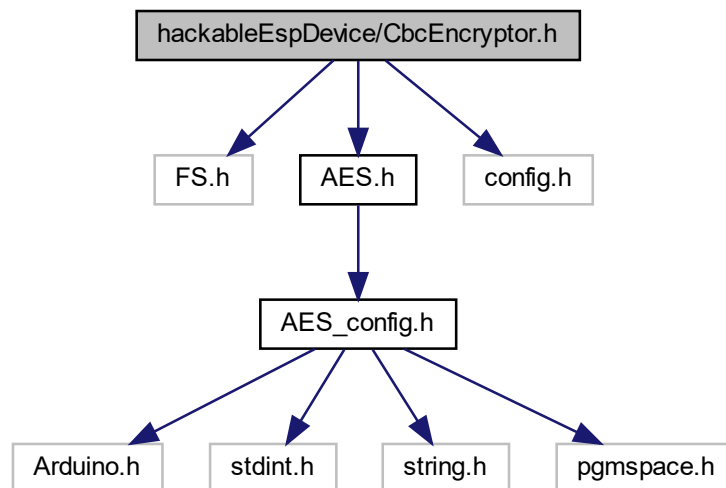
00130 /*****
00136 /*****
00137 String CbcEncryptor::encryptLine(String line) {
00138     uint16_t len = line.length();
00139     byte plain[len]; //To store plain bytes
00140     uint8_t paddedLength = len + N_BLOCK - len % N_BLOCK; //Calculate total length
00141     when padded
00142     byte encrypted[paddedLength]; //To store encrypted bytes
00143     char encryptedString[paddedLength*2+1]; //To store return value
00144     /* Convert string to bytes */
00145     for (uint8_t i = 0; i < len; i++) {
00146         plain[i] = (byte) line[i];
00147     }
00148     _aes.set_IV(_aesInitVecInt);
00149     _aes.get_IV(_aesInitVecInt);
00150     _aes.do_aes_encrypt(plain, len, encrypted, _aesKey, 128, _aesInitVecInt);
00151     /* Convert bytes to string */
00152     for (uint8_t i = 0; i < paddedLength; i++) {
00153         sprintf(encryptedString+2*i, "%02x", *(encrypted+i));
00154     }
00155     return encryptedString;
00156 }
00157
00161 /*****
00167 /*****
00168 String CbcEncryptor::decryptLine(String line) {
00169     uint16_t len = line.length()/2;
00170     byte encrypted[len]; //To store encrypted bytes
00171     byte decrypted[len]; //To store decrypted bytes
00172     uint8_t outputLen = 0; //To store plaintext
00173     length
00174     char decryptedString[len]; //To store return value
00175     /* Convert hex string to bytes */
00176     for (uint8_t i = 0; i < len; i++) {
00177         encrypted[i] = _hexCharToByte(line[i*2])<<4 | _hexCharToByte(line[i*2+1]);
00178     }
00179     _aes.set_IV(_aesInitVecInt);
00180     _aes.get_IV(_aesInitVecInt);
00181     _aes.do_aes_decrypt(encrypted, len, decrypted, _aesKey, 128, _aesInitVecInt);
00182     /* Convert bytes to string */
00183     for (uint8_t i = 0; i < len; i++) {
00184         /* If decrypted is printable character, save */
00185         if (decrypted[i] < 127 && decrypted[i] > 32) {
00186             decryptedString[i] = (char) decrypted[i];
00187         } else {
00188             outputLen = i;
00189             break;
00190         }
00191     }
00192     if (_aesKeyString != AES_KEY) {
00193         return String(decryptedString); //If another key is used,
00194         just throw the whole string back
00195     }
00196     return String(decryptedString).substring(0, outputLen); //Convert to string to
00197     trim the string with substring()
00198 }
00199
00201 /*****
00207 /*****
00208 byte CbcEncryptor::_hexCharToByte(char hexChar) {
00209     uint8_t c = hexChar;
00210     if (c <= '9' && c >= '0') {
00211         c -= '0';
00212     } else if (c <= 'f' && c >= 'a') {
00213         c -= ('a' - 0x0a);
00214     } else if (c <= 'F' && c >= 'A') {
00215         c -= ('A' - 0x0a);
00216     } else {
00217         return(1);
00218     }
00219     return (c);
00220 }
00221
00222 }

```

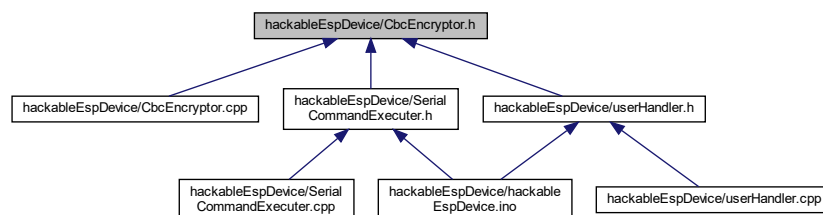
5.13 hackableEspDevice/CbcEncryptor.h File Reference

```
#include <FS.h>
#include "AES.h"
#include "config.h"
```

Include dependency graph for CbcEncryptor.h:



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



Classes

- class [CbcEncryptor](#)

5.14 CbcEncryptor.h

[Go to the documentation of this file.](#)

00001 /*

```

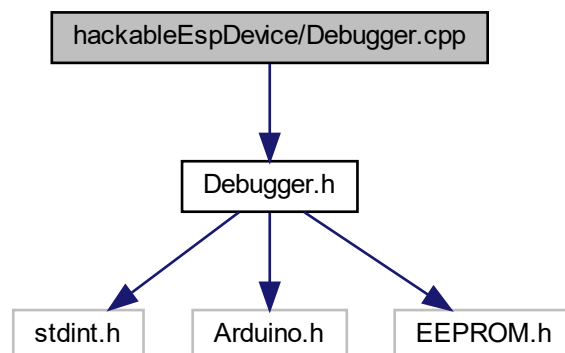
00002  * File:      CbcEncryptor.h
00003  * Author:    Luke de Munk
00004  * Class:     CbcEncryptor
00005  * Version:   1.0
00006  *
00007  * Handles encryption and decryption of files. Uses AES encryption.
00008  */
00009 #ifndef ENCRYPTOR_H
00010 #define ENCRYPTOR_H
00011 #include <FS.h>
00012 #include "AES.h"
00013 #include "config.h"
00014
00015 class CbcEncryptor
00016 {
00017     public:
00018         CbcEncryptor();
00019         bool encryptFile(String filename);
00020         bool decryptFile(String filename);
00021         String encryptLine(String line);
00022         String decryptLine(String line);
00023         bool setKey(String key);
00024
00025     private:
00026         byte _hexCharToByte(char hexChar);
00027
00028         String _aesKeyString;
00029         byte _aesKey[AES_KEY_SIZE];
00030         byte _aesInitVector[N_BLOCK];
00031         unsigned long long int _aesInitVecInt;
00032         AES _aes;
00033 };
00034 #endif

```

5.15 hackableEspDevice/Debugger.cpp File Reference

```
#include "Debugger.h"
```

Include dependency graph for Debugger.cpp:



Functions

- void `debug` (String text)
Prints text if debug is enabled.
- void `debugln` (String text)

- Prints text (+
) if debug is enabled.*
- bool `getDebugEnabled` ()
Gets if debug is enabled.
 - void `setDebugEnabled` (bool isEnabled)
Sets if debug is enabled.

5.15.1 Function Documentation

5.15.1.1 `debug()`

```
void debug (
    String text )
```

Prints text if debug is enabled.

Parameters

<i>text</i>	String of text that needs to be printed
-------------	---

Definition at line 16 of file [Debugger.cpp](#).

5.15.1.2 `debugln()`

```
void debugln (
    String text )
```

Prints text (+
) if debug is enabled.

Parameters

<i>text</i>	String of text that needs to be printed
-------------	---

Definition at line 32 of file [Debugger.cpp](#).

5.15.1.3 `getDebugEnabled()`

```
bool getDebugEnabled ( )
```

Gets if debug is enabled.

Returns

bool True if debug is enabled

Definition at line 47 of file [Debugger.cpp](#).

5.15.1.4 setDebugEnabled()

```
void setDebugEnabled (
    bool isEnabled )
```

Sets if debug is enabled.

Parameters

<i>isEnabled</i>	If debug is enabled (true == enabled)
------------------	---------------------------------------

Definition at line 60 of file [Debugger.cpp](#).

5.16 Debugger.cpp

[Go to the documentation of this file.](#)

```
00001 /*
00002  * File:      Debugger.h
00003  * Author:    Luke de Munk
00004  * Version:   1.0
00005  *
00006  * Class for handling the debug prints.
00007  */
00008 #include "Debugger.h"
00009
00010 /*****
00015 *****/
00016 void debug(String text) {
00017     EEPROM.begin(1);
00018     bool isEnabled = EEPROM.read(ENABLE_DEBUG_FLAG_ADDRESS);
00019
00020     if(isEnabled) {
00021         Serial.print(text);
00022     }
00023     EEPROM.end();
00024 }
00025
00026 /*****
00031 *****/
00032 void debugln(String text) {
00033     EEPROM.begin(1);
00034     bool isEnabled = EEPROM.read(ENABLE_DEBUG_FLAG_ADDRESS);
00035     if(isEnabled) {
00036         Serial.println(text);
00037     }
00038     EEPROM.end();
00039 }
00040
00041 /*****
00046 *****/
00047 bool getDebugEnabled() {
00048     EEPROM.begin(1);
00049     bool isEnabled = EEPROM.read(ENABLE_DEBUG_FLAG_ADDRESS);
00050     EEPROM.end();
00051     return isEnabled;
00052 }
00053
00054 /*****
```

```

00059 /*****
00060 void setDebugEnabled(bool isEnabled) {
00061     EEPROM.begin(1);
00062     EEPROM.write(ENABLE_DEBUG_FLAG_ADDRESS, (uint8_t) isEnabled);    //Set the debug flag
00063     EEPROM.commit();                                                  //Write to EEPROM
00064     EEPROM.end();
00065 }

```

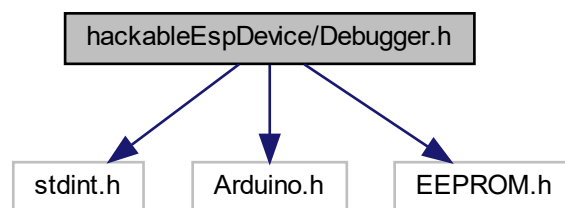
5.17 hackableEspDevice/Debugger.h File Reference

```

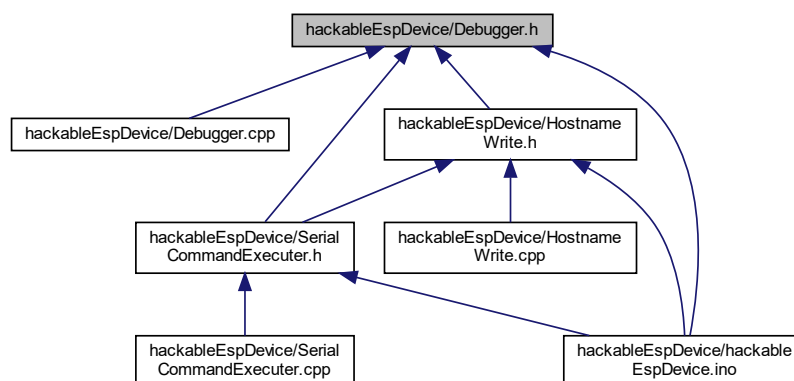
#include <stdint.h>
#include "Arduino.h"
#include <EEPROM.h>

```

Include dependency graph for Debugger.h:



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



Macros

- #define `ENABLE_DEBUG_FLAG_ADDRESS` 0

Functions

- void [debug](#) (String text)
Prints text if debug is enabled.
- void [debugln](#) (String text)
*Prints text (+'
) if debug is enabled.*
- bool [getDebugEnabled](#) ()
Gets if debug is enabled.
- void [setDebugEnabled](#) (bool isEnabled)
Sets if debug is enabled.

5.17.1 Macro Definition Documentation

5.17.1.1 ENABLE_DEBUG_FLAG_ADDRESS

```
#define ENABLE_DEBUG_FLAG_ADDRESS 0
```

Definition at line 21 of file [Debugger.h](#).

5.17.2 Function Documentation

5.17.2.1 debug()

```
void debug (  
    String text )
```

Prints text if debug is enabled.

Parameters

<i>text</i>	String of text that needs to be printed
-------------	---

Definition at line 16 of file [Debugger.cpp](#).

5.17.2.2 debugln()

```
void debugln (  
    String text )
```

Prints text (+'
) if debug is enabled.

Parameters

<i>text</i>	String of text that needs to be printed
-------------	---

Definition at line 32 of file [Debugger.cpp](#).

5.17.2.3 getDebugEnabled()

```
bool getDebugEnabled ( )
```

Gets if debug is enabled.

Returns

bool True if debug is enabled

Definition at line 47 of file [Debugger.cpp](#).

5.17.2.4 setDebugEnabled()

```
void setDebugEnabled (
    bool isEnabled )
```

Sets if debug is enabled.

Parameters

<i>isEnabled</i>	If debug is enabled (true == enabled)
------------------	---------------------------------------

Definition at line 60 of file [Debugger.cpp](#).

5.18 Debugger.h

[Go to the documentation of this file.](#)

```
00001 /*
00002  * File:      Debugger.h
00003  * Author:    Luke de Munk
00004  * Version:   1.0
00005  *
00006  * Class for handling the debug prints.
00007  */
00008 #ifndef DEBUGGER_H
00009 #define DEBUGGER_H
00010 #include <stdint.h>                                //For defining bits per
    integer
00011 #include "Arduino.h"
00012 #include <EEPROM.h>                                //For reading from and
    writing to flash memory, used for resetting wifi
```

```

00013
00014 /*
00015  * 1 byte to store the enable debug flag.
00016  * Is done in EEPROM, because the
00017  * flag is then non-volatile and can
00018  * be used by multiple classes. Also
00019  * is saved during restart.
00020  */
00021 #define ENABLE_DEBUG_FLAG_ADDRESS 0
00022
00023 void debug(String text);
00024 void debugln(String text);
00025 bool getDebugEnabled();
00026 void setDebugEnabled(bool isEnabled);
00027
00028 #endif

```

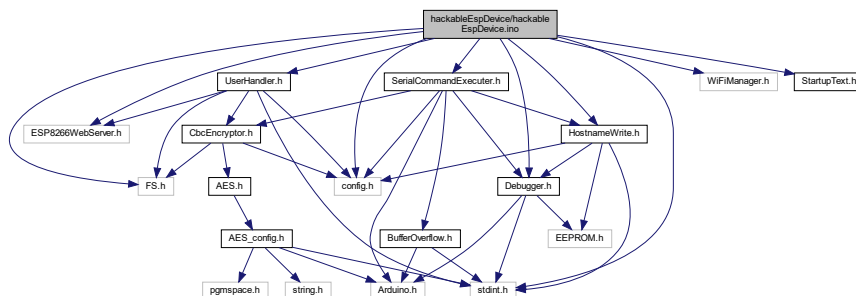
5.19 hackableEspDevice/hackableEspDevice.ino File Reference

```

#include <ESP8266WebServer.h>
#include <FS.h>
#include <stdint.h>
#include <WiFiManager.h>
#include "config.h"
#include "UserHandler.h"
#include "SerialCommandExecuter.h"
#include "Debugger.h"
#include "HostnameWrite.h"
#include "StartupText.h"

```

Include dependency graph for hackableEspDevice.ino:



Macros

- #define **ON** HIGH
- #define **OFF** LOW
- #define **MIN_BRIGHTNESS** 1022

Functions

- ESP8266WebServer **server** (80)
- void **setup** ()
Setup microchip.
- void **initializeHostname** ()
Initializes hostname.

- void [setupWifi](#) ()
Connects to WiFi if it can, otherwise starts as AP to configure WiFi.
- void [initializeServer](#) ()
Takes care of the webservices like pageloading.
- void [sendToFrontend](#) (String var)
Replaces placeholders with actual data in HTML page.
- void [loop](#) ()
Mainloop.
- String [getContentType](#) (String filename)
Converts the file extension to the MIME type.
- void [handleRequest](#) (String path, uint8_t permissionLevel)
Sends the requested file if the user has permission.
- void [handleFileUpload](#) ()
Handles the file upload to the SPIFFS.
- void [handleFileDownload](#) ()
Handles the file download to the SPIFFS.

Variables

- uint8_t [ledState](#) = [OFF](#)
- uint16_t [brightness](#) = 1023
- [UserHandler](#) userHandler & [server](#)
- [SerialCommandExecuter](#) cliExecuter
- File [fsUploadFile](#)

5.19.1 Macro Definition Documentation

5.19.1.1 MIN_BRIGHTNESS

```
#define MIN_BRIGHTNESS 1022
```

Definition at line 24 of file [hackableEspDevice.ino](#).

5.19.1.2 OFF

```
#define OFF LOW
```

Definition at line 22 of file [hackableEspDevice.ino](#).

5.19.1.3 ON

```
#define ON HIGH
```

Definition at line 21 of file [hackableEspDevice.ino](#).

5.19.2 Function Documentation

5.19.2.1 getContentType()

```
String getContentType (
    String filename )
```

Converts the file extension to the MIME type.

Parameters

<i>filename</i>	Name of the file
-----------------	------------------

Returns

String MIME type of the file

Definition at line 304 of file [hackableEspDevice.ino](#).

5.19.2.2 handleFileDownload()

```
void handleFileDownload ( )
```

Handles the file download to the SPIFFS.

Definition at line 388 of file [hackableEspDevice.ino](#).

5.19.2.3 handleFileRequest()

```
void handleFileRequest (
    String path,
    uint8_t permissionLevel )
```

Sends the requested file if the user has permission.

Parameters

<i>path</i>	Path to the file
<i>permissionLevel</i>	0 = not logged in, 1 = user, 2 = admin

Definition at line 321 of file [hackableEspDevice.ino](#).

5.19.2.4 handleFileUpload()

```
void handleFileUpload ( )
```

Handles the file upload to the SPIFFS.

Definition at line 353 of file [hackableEspDevice.ino](#).

5.19.2.5 initializeHostname()

```
void initializeHostname ( )
```

Initializes hostname.

Definition at line 74 of file [hackableEspDevice.ino](#).

5.19.2.6 initializeServer()

```
void initializeServer ( )
```

Takes care of the webservices like pageloading.

Definition at line 132 of file [hackableEspDevice.ino](#).

5.19.2.7 loop()

```
void loop ( )
```

Mainloop.

Definition at line 289 of file [hackableEspDevice.ino](#).

5.19.2.8 sendToFrontend()

```
void sendToFrontend (
    String var )
```

Replaces placeholders with actual data in HTML page.

Definition at line 276 of file [hackableEspDevice.ino](#).

5.19.2.9 server()

```
ESP8266WebServer server (
    80 )
```

5.19.2.10 setup()

```
void setup ( )
```

Setup microchip.

Definition at line 40 of file [hackableEspDevice.ino](#).

5.19.2.11 setupWifi()

```
void setupWifi ( )
```

Connects to WiFi if it can, otherwise starts as AP to configure WiFi.

Definition at line 105 of file [hackableEspDevice.ino](#).

5.19.3 Variable Documentation

5.19.3.1 brightness

```
uint16_t brightness = 1023
```

Definition at line 28 of file [hackableEspDevice.ino](#).

5.19.3.2 cliExecuter

```
SerialCommandExecuter cliExecuter
```

Definition at line 31 of file [hackableEspDevice.ino](#).

5.19.3.3 fsUploadFile

File fsUploadFile

Definition at line 33 of file [hackableEspDevice.ino](#).

5.19.3.4 ledState

```
uint8_t ledState = OFF
```

Definition at line 27 of file [hackableEspDevice.ino](#).

5.19.3.5 server

```
UserHandler userHandler& server
```

Definition at line 30 of file [hackableEspDevice.ino](#).

5.20 hackableEspDevice.ino

[Go to the documentation of this file.](#)

```
00001 /*
00002  * File:      hackableEspDevice.ino
00003  * Authors:   ESPinoza (Team 1)
00004  * Version:   1.0
00005  *
00006  * The main file of the firmware of a vunerable-by-design ESP8266 controller.
00007  * For more information, go to: https://gitlab.fdmci.hva.nl/munk1/hackable_esp_device
00008  *
00009  */
00010 #include <ESP8266WebServer.h>                                //For running the
00011     webserver
00012 #include <FS.h>                                              //For SPIFFS
00013 #include <stdint.h>                                          //For defining bits per
00014     integer
00015 #include <WiFiManager.h>                                     //For web-based wifi
00016     configuration
00017 #include "config.h"                                          //For the configuration
00018 #include "UserHandler.h"                                     //For handling the users
00019     from the config.conf
00020 #include "SerialCommandExecuter.h"                          //For handling serial
00021     commands
00022 #include "Debugger.h"                                        //For handling debug
00023     messages
00024 #include "HostnameWrite.h"                                   //For handling the
00025     hostname changes
```

```

00019 #include "StartupText.h" //For printing startup log
      files
00020
00021 #define ON HIGH
00022 #define OFF LOW
00023
00024 #define MIN_BRIGHTNESS 1022 //analogWrite() on ESP8266
      DI Mini board is inverted
00025
00026 ESP8266WebServer server(80); //Object that listens for
      HTTP requests on port 80
00027 uint8_t ledState = OFF; //For led state variable
00028 uint16_t brightness = 1023; //For led brightness
00029
00030 UserHandler userHandler(&server); //For handling the
      authentication
00031 SerialCommandExecuter cliExecuter; //For handling serial
      commands
00032
00033 File fsUploadFile; //A File object to
      temporarily store the received file
00034
00035 /*****
00039 /*****
00040 void setup() {
00041     Serial.begin(115200); //Serial port for
      debugging purposes
00042
00043     /* Initialize SPIFFS */
00044     if (!SPIFFS.begin()) {
00045         Serial.println("An Error has occurred while mounting SPIFFS");
00046         return;
00047     }
00048
00049     debugln("Debug is enabled");
00050
00051     /* If debug is enabled, the root password is printed in a big string of text */
00052     if (getDebugEnabled()) {
00053         String mess = "ROOT: " + String(ROOT_PASSWORD);
00054         printStartupText(mess);
00055     }
00056
00057     pinMode(LED_BUILTIN, OUTPUT);
00058     analogWrite(LED_BUILTIN, 1023);
00059
00060     initializeHostname();
00061     setupWifi();
00062     initializeServer();
00063     userHandler.updateUsers();
00064     cliExecuter.setUsers(userHandler.getUsers(), userHandler.getNumberOfUsers()); //Send users to the
      command executer for the 'users' command
00065
00066     Serial.println("Serial commands available. Typ 'help' for help.");
00067 }
00068
00069 /*****
00073 /*****
00074 void initializeHostname() {
00075     String customHostname = getHostname();
00076     /* Check if custom hostname is set, otherwise use default */
00077     if (customHostname != "") {
00078         /* Check if hostname can be set */
00079         if (WiFi.hostname(customHostname)) {
00080             debug(customHostname);
00081             debugln(" is the hostname.");
00082         } else {
00083             debug("Could not set '");
00084             debug(customHostname);
00085             debugln("' as hostname.");
00086         }
00087     } else {
00088         if (WiFi.hostname(DEFAULT_HOSTNAME)) {
00089             debug(DEFAULT_HOSTNAME);
00090             debugln(" is the hostname.");
00091         } else {
00092             debug("Could not set '");
00093             debug(DEFAULT_HOSTNAME);
00094             debugln("' as hostname.");
00095         }
00096     }
00097 }
00098
00099 /*****
00104 /*****
00105 void setupWifi() {
00106     WiFiManager wifiManager;
00107

```

```

00108     if (wifiManager.autoConnect(WIFI_CONF_AP_NAME)) {
00109         Serial.print("Connected to: ");
00110         Serial.println(WiFi.SSID());
00111         Serial.print("IP: ");
00112         Serial.println(WiFi.localIP());
00113     } else {
00114         Serial.println("Failed to connect, connect with AP");
00115         ESP.restart();
00116     }
00117
00118     debug("Copy and paste the following URL: http://");
00119
00120     if (WiFi.hostname(DEFAULT_HOSTNAME)) {
00121         debugln(DEFAULT_HOSTNAME);
00122     } else {
00123         debugln(WiFi.hostname().c_str());
00124     }
00125 }
00126
00127 /*****
00131 /*****
00132 void initializeServer() {
00133     /*
00134     * Routes for loading all the necessary files
00135     */
00136     /* Route for home page */
00137     server.on("/", HTTP_GET, []() {
00138         handleFileRequest("/index.html", PERMISSION_LVL_ALL);
00139     });
00140
00141     /* Route for sending c++ variables */
00142     server.on("/state", HTTP_GET, []() {
00143         sendToFrontend("ledState");
00144     });
00145
00146     /* Route for sending c++ variables */
00147     server.on("/brightness", HTTP_GET, []() {
00148         sendToFrontend("brightness");
00149     });
00150
00151     /* Route for admin controls */
00152     server.on("/admin", HTTP_GET, []() {
00153         handleFileRequest("/admin.html", PERMISSION_LVL_ADMIN);
00154     });
00155
00156     /* Route for user controls */
00157     server.on("/user", HTTP_GET, []() {
00158         handleFileRequest("/user.html", PERMISSION_LVL_USER);
00159     });
00160
00161     /* Route for file upload page */
00162     server.on("/upload", HTTP_GET, []() {
00163         handleFileRequest("/upload.html", PERMISSION_LVL_ADMIN);
00164     });
00165
00166     /* Route for file download page */
00167     server.on("/download", HTTP_GET, []() {
00168         handleFileRequest("/download.html", PERMISSION_LVL_USER);
00169     });
00170
00171     /* Load styles.css file, styling for desktop version */
00172     server.on("/styles.css", HTTP_GET, []() {
00173         handleFileRequest("/styles.css", PERMISSION_LVL_ALL);
00174     });
00175
00176     /* Load styles_mobile.css file, styling for mobile version */
00177     server.on("/styles_mobile.css", HTTP_GET, []() {
00178         handleFileRequest("/styles_mobile.css", PERMISSION_LVL_ALL);
00179     });
00180
00181     /* Load style_switch.css file, styling for the on/off switch */
00182     server.on("/style_switch.css", HTTP_GET, []() {
00183         handleFileRequest("/style_switch.css", PERMISSION_LVL_ALL);
00184     });
00185
00186     /* Load favicon.ico file, site icon */
00187     server.on("/favicon.ico", HTTP_GET, []() {
00188         handleFileRequest("/favicon.ico", PERMISSION_LVL_ALL);
00189     });
00190
00191     /* Load jquery.min.js file, for ajax */
00192     server.on("/jquery.min.js", HTTP_GET, []() {
00193         handleFileRequest("/jquery.min.js", PERMISSION_LVL_ALL);
00194     });
00195
00196     /* Load base.js file, JavaScript for site */
00197     server.on("/base.js", HTTP_GET, []() {

```

```

00198     handleFileRequest("/base.js", PERMISSION_LVL_ALL);
00199 });
00200
00201 /* Load switch.js file, JavaScript for on/off switch */
00202 server.on("/switch.js", HTTP_GET, []() {
00203     handleFileRequest("/switch.js", PERMISSION_LVL_ALL);
00204 });
00205 /*
00206  * End of file loading
00207 */
00208
00209 /*
00210  * Routes for JavaScript data receiving
00211 */
00212 /* Route for setting power */
00213 server.on("/set_power", HTTP_GET, []() {
00214     if (server.arg("state")) {
00215         ledState = atoi(server.arg("state").c_str());
00216         if(ledState == ON) {
00217             analogWrite(LED_BUILTIN, MIN_BRIGHTNESS-brightness);
00218         } else {
00219             analogWrite(LED_BUILTIN, 1023);
00220         }
00221     }
00222     handleFileRequest("/index.html", PERMISSION_LVL_ALL);
00223 });
00224
00225 /* Route for setting brightness */
00226 server.on("/set_brightness", HTTP_GET, []() {
00227     if (server.arg("brightness")) {
00228         brightness = atoi(server.arg("brightness").c_str());
00229         if(ledState == ON) {
00230             analogWrite(LED_BUILTIN, MIN_BRIGHTNESS-brightness);
00231         }
00232     }
00233     handleFileRequest("/index.html", PERMISSION_LVL_ALL);
00234 });
00235
00236 /* Route for restarting the server */
00237 server.on("/restart", HTTP_GET, []() {
00238     handleFileRequest("/", PERMISSION_LVL_ALL);
00239     ESP.restart();
00240 });
00241 /*
00242  * End of JavaScript data receiving
00243 */
00244
00245 /*
00246  * Routes for file management
00247 */
00248 /* Route for file upload request */
00249 server.on("/upload", HTTP_POST, []() {
00250     server.send(200); //HTTP code 200 == OK
00251     debugln("Wait, something got uploaded"); //Receive and save the
00252 }, handleFileUpload
00253     file
00254 );
00255 /* Route for file upload request */
00256 server.on("/download", HTTP_POST, []() {
00257     debugln("File download request"); //Receive and save the
00258 }, handleFileDownload
00259     file
00260 );
00261 /*
00262  * End of routes for file management
00263 */
00264 /* Not found */
00265 server.onNotFound([]() { //If the client requests
00266     any URI             //Send it if it exists
00267     handleFileRequest(server.uri(), PERMISSION_LVL_ALL);
00268     debugln("Route not found");
00269 });
00270 server.begin(); //Start server
00271
00272 void sendToFrontend(String var){
00273     if (var == "ledState") {
00274         server.send(200, "text/plain", String (ledState));
00275     } else if (var == "brightness") {
00276         server.send(200, "text/plain", String (brightness));
00277     }
00278 }
00279
00280
00281
00282
00283
00284

```

```

00288 /*****
00289 void loop() {
00290     server.handleClient();
00291
00292     if (Serial.available()) {
00293         cliExecuter.executeCommand();
00294     }
00295 }
00296
00297 /*****
00303 /*****
00304 String getContentType(String filename) {
00305     if (filename.endsWith(".html")) return "text/html";
00306     else if (filename.endsWith(".css")) return "text/css";
00307     else if (filename.endsWith(".js")) return "application/javascript";
00308     else if (filename.endsWith(".ico")) return "image/x-icon";
00309     else if (filename.endsWith(".gz")) return "application/x-gzip";
00310     else if (filename.endsWith(".txt")) return "text/plain";
00311     return "text/plain";
00312 }
00313
00314 /*****
00320 /*****
00321 void handleFileRequest(String path, uint8_t permissionLevel) {
00322     if(!userHandler.checkPermission(permissionLevel, &server)) {
00323         server.requestAuthentication();
00324         return;
00325     }
00326
00327     debugln(String("Requested file: ") + path);
00328
00329     String contentType = getContentType(path);           //Get the MIME type
00330     String pathWithGz = path + ".gz";
00331
00332     if (SPIFFS.exists(pathWithGz)) {                    //If there's a compressed
00333         version available                                //Use the compressed
00334         path += ".gz";
00335     }
00336
00337     if (SPIFFS.exists(path)) {
00338         File file = SPIFFS.open(path, "r");              //Open the file
00339         size_t sent = server.streamFile(file, contentType); //Send it to the client
00340         file.close();                                     //Close the file again
00341         debugln(String("Sent file: ") + path);
00342         return;
00343     }
00344     debugln(String("File Not Found: ") + path);          //If the file doesn't
00345     exist, return false
00346     server.send(404, "text/plain", "404: Not Found");    //otherwise, respond with
00347     a 404 (Not Found) error
00348 }
00349
00350 /*****
00352 /*****
00353 void handleFileUpload() {
00354     HTTPUpload& upload = server.upload();
00355
00356     if (upload.status == UPLOAD_FILE_START) {
00357         String filename = upload.filename;
00358
00359         if (!filename.startsWith("/")) {
00360             filename = "/" + filename;
00361         }
00362
00363         debugln(String("Upload file named: ") + filename);
00364
00365         fsUploadFile = SPIFFS.open(filename, "w");        //Open the file for
00366         writing in SPIFFS (create if it doesn't exist)
00367     } else if (upload.status == UPLOAD_FILE_WRITE && fsUploadFile ) {
00368         fsUploadFile.write(upload.buf, upload.currentSize); //Write the received bytes
00369         to the file
00370     } else if (upload.status == UPLOAD_FILE_END) {
00371         if (fsUploadFile) {                                //If the file was
00372             successfully created
00373             fsUploadFile.close();                          //Close the file again
00374             debugln(String("handleFileUpload Size: ") + upload.totalSize);
00375             server.sendHeader("Location", "/success.html"); //Redirect the client to
00376             the success page
00377             server.send(303);
00378             userHandler.updateUsers();
00379             cliExecuter.setUsers(userHandler.getUsers(), userHandler.getNumberOfUsers()); //Update
00380             users for cli as well
00381         } else {
00382             server.send(500, "text/plain", "500: couldn't create file");

```

```

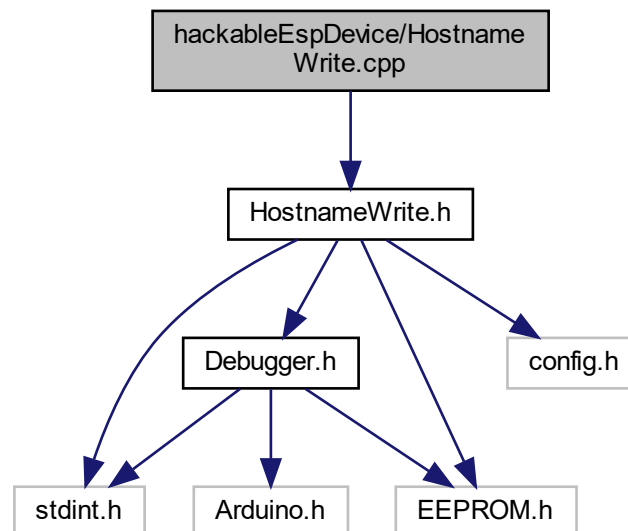
00379     }
00380 }
00381 }
00382
00383 /*****
00387 /*****
00388 void handleFileDownload() {
00389     String filename = server.arg("filekey");           //Get user input for
        filename
00390
00391     if (!filename.startsWith("/")) {
00392         filename = "/" + filename;
00393     }
00394
00395     if (!SPIFFS.exists(filename)) {
00396         server.send(404, "text/plain", "404: file not found!");
00397         return;
00398     }
00399
00400     File download = SPIFFS.open(filename, "r");
00401
00402     debugln("Start sending file");
00403
00404     server.sendHeader("Content-Type", "text/text");
00405     server.sendHeader("Content-Disposition", "attachment; filename="+filename);
00406     server.sendHeader("Connection", "close");
00407     server.streamFile(download, "application/octet-stream");
00408     download.close();
00409     server.send(200);           //HTTP code 200 == OK
00410 }

```

5.21 hackableEspDevice/HostnameWrite.cpp File Reference

```
#include "HostnameWrite.h"
```

Include dependency graph for HostnameWrite.cpp:



Functions

- String `getHostname()`

- Gets the hostname from the EEPROM.*
 - void [writeHostname](#) (char hostname[MAX_HOSTNAME_LENGTH])
 - Writes the new hostname to the EEPROM.*
 - void [setEepromToNull](#) (uint8_t writeLength, uint8_t startAddress)
 - Resets the given EEPROM addresses.*
 - void [checkEepromCommit](#) ()
 - Checks if the eeprom was actually committed.*

5.21.1 Function Documentation

5.21.1.1 [checkEepromCommit\(\)](#)

```
void checkEepromCommit ( )
```

Checks if the eeprom was actually committed.

Definition at line 79 of file [HostnameWrite.cpp](#).

5.21.1.2 [getHostname\(\)](#)

```
String getHostname ( )
```

Gets the hostname from the EEPROM.

Returns

String Current hostname

Definition at line 16 of file [HostnameWrite.cpp](#).

5.21.1.3 [setEepromToNull\(\)](#)

```
void setEepromToNull (
    uint8_t writeLength,
    uint8_t startAddress )
```

Resets the given EEPROM addresses.

Parameters

<i>writeLength</i>	Length to be erased
<i>startAddress</i>	Start address

Definition at line 56 of file [HostnameWrite.cpp](#).

5.21.1.4 writeHostname()

```
void writeHostname (
    char hostname[MAX_HOSTNAME_LENGTH] )
```

Writes the new hostname to the EEPROM.

Parameters

<i>hostname</i>	String that contains the hostname to be written
-----------------	---

Definition at line 37 of file [HostnameWrite.cpp](#).

5.22 HostnameWrite.cpp

[Go to the documentation of this file.](#)

```
00001 /*
00002  * File:      HostnameWrite.cpp
00003  * Author:    Twenne Elffers
00004  * Version:   1.0
00005  *
00006  * Handles hostname saving and reading to EEPROM.
00007  */
00008 #include "HostnameWrite.h"
00009
00010 /*****
00015 *****/
00016 String getHostname() {
00017     char hostname[MAX_HOSTNAME_LENGTH];
00018     EEPROM.begin(MAX_HOSTNAME_LENGTH);
00019
00020     for (uint8_t i = 0; i < MAX_HOSTNAME_LENGTH; i++) {
00021         EEPROM.get(HOSTNAME_ADRESS+i, hostname[i]);
00022         if (hostname[i] == 0xFF) {
00023             break; //Skips the unreadable
00024         }
00025     }
00026
00027     EEPROM.end();
00028     return String(hostname);
00029 }
00030
00031 /*****
00036 *****/
00037 void writeHostname(char hostname[MAX_HOSTNAME_LENGTH]) {
00038     EEPROM.begin(MAX_HOSTNAME_LENGTH);
00039
00040     for (uint8_t i = 0; i < MAX_HOSTNAME_LENGTH; i++){
00041         EEPROM.write(HOSTNAME_ADRESS+i, hostname[i]);
00042         yield();
00043     }
00044
00045     checkEepromCommit();
00046     EEPROM.end();
00047 }
00048
00049 /*****
00055 *****/
00056 void setEepromToNull(uint8_t writeLength, uint8_t startAddress) {
00057     EEPROM.begin(writeLength);
00058
00059     for (uint8_t i = 0; i < writeLength; i++){
00060         EEPROM.write(startAddress+i, 0);
00061         yield();
00062     }
00063 }
```

```

00062     }
00063
00064     checkEepromCommit();
00065
00066     debug("Reset Value at: ");
00067     debug(String(startAddress));
00068     debug(" till ");
00069     debugln(String(startAddress+writeLength));
00070
00071     EEPROM.end();
00072 }
00073
00074 /*****
00075 *****/
00076 /*****
00077 *****/
00079 void checkEepromCommit() {
00080     if (EEPROM.commit()) {
00081         Serial.println("Data written!");
00082     } else {
00083         Serial.println("ERROR! Data not written!");
00084     }
00085 }

```

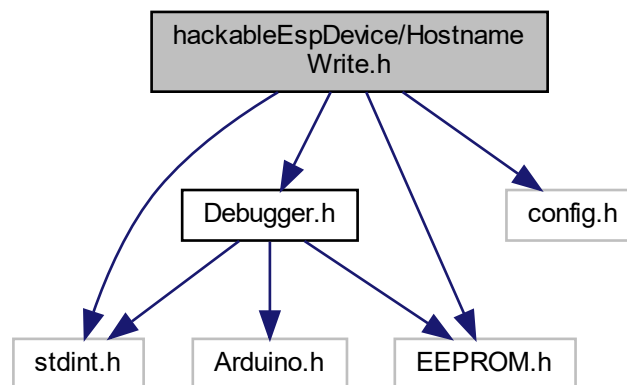
5.23 hackableEspDevice/HostnameWrite.h File Reference

```

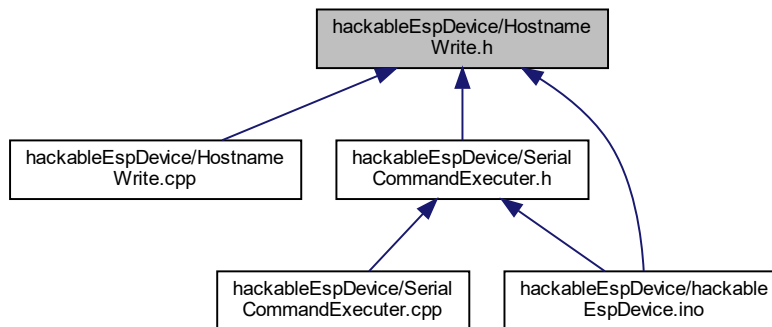
#include <stdint.h>
#include <EEPROM.h>
#include "Debugger.h"
#include "config.h"

```

Include dependency graph for HostnameWrite.h:



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



Functions

- String [getHostname](#) ()
Gets the hostname from the EEPROM.
- void [writeHostname](#) (char hostname[32])
- void [setEepromToNull](#) (uint8_t writeLength, uint8_t startAddress)
Resets the given EEPROM addresses.
- void [checkEepromCommit](#) ()
Checks if the eeprom was actually committed.

5.23.1 Function Documentation

5.23.1.1 checkEepromCommit()

```
void checkEepromCommit ( )
```

Checks if the eeprom was actually committed.

Definition at line 79 of file [HostnameWrite.cpp](#).

5.23.1.2 getHostname()

```
String getHostname ( )
```

Gets the hostname from the EEPROM.

Returns

String Current hostname

Definition at line 16 of file [HostnameWrite.cpp](#).

5.23.1.3 setEepromToNull()

```
void setEepromToNull (
    uint8_t writeLength,
    uint8_t startAddress )
```

Resets the given EEPROM addresses.

Parameters

<i>writeLength</i>	Length to be erased
<i>startAddress</i>	Start address

Definition at line 56 of file [HostnameWrite.cpp](#).

5.23.1.4 writeHostname()

```
void writeHostname (
    char hostname[32] )
```

5.24 HostnameWrite.h

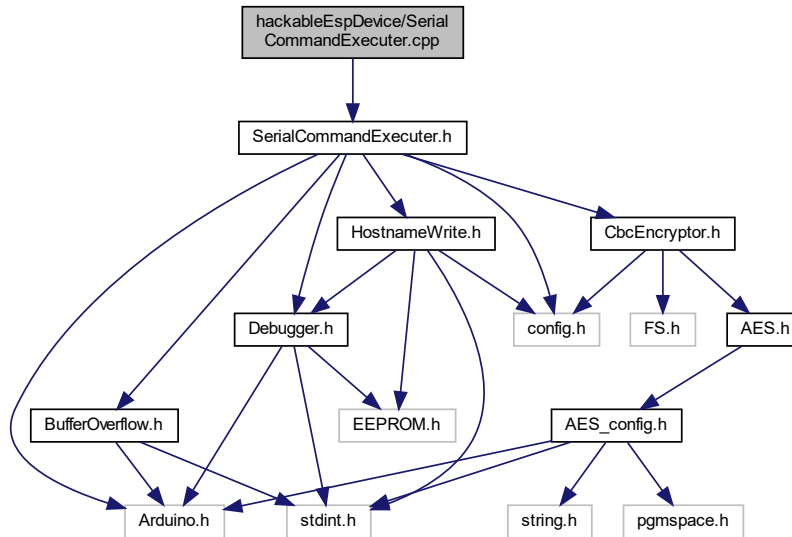
[Go to the documentation of this file.](#)

```
00001 /*
00002  * File:      HostnameWrite.h
00003  * Author:    Twenne Elffers
00004  * Version:   1.0
00005  *
00006  * Handles hostname saving and reading to EEPROM.
00007  */
00008 #ifndef HOSTNAME_WRITE_H
00009 #define HOSTNAME_WRITE_H
00010 #include <stdint.h>                                //For defining bits per
    integer
00011 #include <EEPROM.h>                                //For reading from and
    writing to EEPROM
00012 #include "Debugger.h"                              //For handling debug
    messages
00013 #include "config.h"                                //For the configuration
00014
00015 String getHostname();
00016 void writeHostname(char hostname[32]);
00017 void setEepromToNull(uint8_t writeLength, uint8_t startAddress);
00018 void checkEepromCommit();
00019 #endif
```

5.25 hackableEspDevice/SerialCommandExecuter.cpp File Reference

```
#include "SerialCommandExecuter.h"
```

Include dependency graph for SerialCommandExecuter.cpp:



5.26 SerialCommandExecuter.cpp

[Go to the documentation of this file.](#)

```

00001 /*
00002  * File:      SerialCommandExecuter.cpp
00003  * Author:    Luke de Munk & Twenne Elffers
00004  * Class:     SerialCommandExecuter
00005  * Version:   1.0
00006  *
00007  * Parses and executes serial terminal commands.
00008  */
00009 #include "SerialCommandExecuter.h"
00010
00011 /*****
00015 /*****
00016 SerialCommandExecuter::SerialCommandExecuter() {
00017     _isLoggedIn = false;
00018 }
00019
00020 /*****
00026 /*****
00027 void SerialCommandExecuter::setUsers(String* users, uint8_t numUsers) {
00028     /* Copy users */
00029     for (uint8_t i = 0; i < MAX_NUMBER_USERS*USER_INFO_LENGTH; i++) {
00030         _users[i] = users[i];
00031     }
00032     _numberUsers = numUsers;
00033 }
00034
00035 /*****
00039 /*****
00040 void SerialCommandExecuter::executeCommand() {
00041     String command = Serial.readString();
00042
00043     if (command != "") {
00044         if (_isLoggedIn) {
00045             Serial.print("~# ");
00046             superuser
00046         } else {
00046             //For the Linux feeling,
  
```

```

00047         Serial.print("~$ ");                                     //For the Linux feeling,
no superuser
00048     }
00049     Serial.print(command);                                       //Echo command (command
ends with \n)
00050
00051     _parseCommand(command);
00052 }
00053 }
00054
00055 /*****
00061 /*****
00062 bool SerialCommandExecutor::_parseCommand(String commandString) {
00063     String* trimmedCmdLine = _trimCommand(commandString);
00064     String command = trimmedCmdLine[0].c_str();
00065     String params[MAX_NUMBER_PARAMS] = {" "};
00066     uint8_t numParams = 0;
00067
00068     while (numParams < MAX_NUMBER_PARAMS) {
00069         if (trimmedCmdLine[numParams+1] == " ") {                //+1, because the command
is in the first cell
00070             break;
00071         }
00072         numParams++;
00073     }
00074
00075     for (uint8_t i = 1; i-1 < numParams; i++){                  //+1, because the command
is in the first cell
00076         params[i-1] = trimmedCmdLine[i].c_str();
00077     }
00078
00079     /* Check which command is given */
00080     if (command == COMMAND_HELP) {
00081         _printHelp(COMMAND_HELP);
00082         return true;
00083     } else {
00084         /* If help needs to be printed, print it and return */
00085         if (_checkHelp(params[0], command)) {
00086             return true;
00087         }
00088     }
00089
00090     if (command == COMMAND_DEBUG) {
00091         if (!_checkParams(numParams, 1, 1) || !_enableDebug(params[0])) {
00092             return false;
00093         }
00094     } else if (command == COMMAND_SU) {
00095         if (!_checkParams(numParams, 1, 1) || !_superUserLogin(params[0])) {
00096             return false;
00097         }
00098     } else if (command == COMMAND_KEYS) {
00099         if (!_viewKey()) {
00100             return false;
00101         }
00102     } else if ((command == COMMAND_RESTART)) {
00103         _restart();
00104     } else if (command == COMMAND_USERS) {
00105         if (!_viewUsers()) {
00106             return false;
00107         }
00108     } else if (command == COMMAND_HOSTNAME) {
00109         if (!_checkParams(numParams, 0, 2) || !_hostname(params)) {
00110             return false;
00111         }
00112     } else if (command == COMMAND_WHOAMI) {
00113         if (_isLoggedIn) {
00114             Serial.println("superuser");
00115         } else {
00116             Serial.println("user");
00117         }
00118         return true;
00119     } else if (command == COMMAND_LS) {
00120         buffOverflow.ls();
00121     } else if (command == COMMAND_VI) {
00122         if (_checkParams(numParams, 1, 1)) {
00123             if (params[0] == ARG_LS_FILE_1_1 || params[0] == ARG_LS_FILE_1_2) {
00124                 buffOverflow.vi();
00125             } else {
00126                 Serial.println(ERROR_NO_FILE);
00127                 return false;
00128             }
00129         }
00130     } else if (command.substring(0, 2) == COMMAND_RUN) {        //Substring == "./" the
rest is filename
00131         if (_checkParams(numParams, 0, 1)) {
00132             String filename = command.substring(2);              //The rest of the command
is filename

```

```

00133
00134         if (filename == ARG_LS_FILE_1_1) {
00135             Serial.println(ERROR_PERM_DENIED);
00136             return false;
00137         }
00138
00139         if (filename != ARG_LS_FILE_2_1) {
00140             Serial.println(ERROR_NO_FILE_DIR);
00141             return false;
00142         }
00143
00144         if (numParams == 1) {
00145             /* If buffer overflow is done correctly,
00146              * user is logged in.
00147              */
00148             if (buffOverflow.runCProgram(params[0])) {
00149                 _isLoggedIn = true;
00150                 Serial.println(MESS_SUPER_USER);
00151             }
00152         } else {
00153             buffOverflow.runCProgram("");
00154         }
00155     }
00156 } else if (command == COMMAND_OBJDUMP) {
00157     if (_checkParams(numParams, 2, 2)) {
00158         if (params[0] != "-d") {
00159             Serial.println(ERROR_WRONG_ARGS);
00160             return false;
00161         }
00162         if (params[1] == ARG_LS_FILE_2_1 || params[1] == ARG_LS_FILE_2_2) {
00163             buffOverflow.objectDump();
00164         } else {
00165             Serial.println(ERROR_NO_FILE);
00166             return false;
00167         }
00168     }
00169 } else if (command == COMMAND_GPG) {
00170     if (_checkParams(numParams, 3, 3)) {
00171         if (params[0] == ARG_GPG_ENCRYPT) {
00172             _encrypt(params);
00173         } else if (params[0] == ARG_GPG_DECRYPT) {
00174             _decrypt(params);
00175         } else {
00176             Serial.println(ERROR_WRONG_ARGS);
00177             return false;
00178         }
00179     }
00180 } else {
00181     Serial.println(ERROR_CMD_NOT_FOUND);
00182     return false;
00183 }
00184 return true;
00185 }
00186
00187 /*****
00193 *****/
00194 String* SerialCommandExecuter::_trimCommand(String commandString) {
00195     static String commandItems[1+MAX_NUMBER_PARAMS] = {" "}; //To save command and
00196     String item = " "; //Can be a command or
00197     parameter
00198     uint8_t paramCounter = 0;
00199
00200     /* Reset static array */
00201     for (uint16_t x = 0; x < 1+MAX_NUMBER_PARAMS; x++) {
00202         commandItems[x] = " ";
00203     }
00204
00205     /* Count number of parameters by adding to temp variable if not a whitespace or end of line*/
00206     for (uint16_t c = 0; c < commandString.length(); c++) {
00207         if (commandString[c] == ' ' || commandString[c] == '\n') {
00208             /* If item is not empty: add to item array */
00209             if (item != "") {
00210                 commandItems[paramCounter] = item;
00211                 item = " "; //Save param to items list
00212                 paramCounter++; //Reset item value
00213             }
00214         } else {
00215             item += commandString[c];
00216         }
00217     }
00218     return commandItems;
00219 }
00220
00221 /*****
00227 *****/
00228 bool SerialCommandExecuter::_checkParams(uint8_t numParams, uint8_t minNumberParams, uint8_t

```

```

        maxNumberParams) {
00229     if (numParams < minNumberParams) {
00230         Serial.println(ERROR_TOO_FEW_ARGS);
00231         return false;
00232     } else if (numParams > maxNumberParams) {
00233         Serial.println(ERROR_TOO_MANY_ARGS);
00234         return false;
00235     }
00236     return true;
00237 }
00238
00239 /*****
00240 /*****
00241 void SerialCommandExecuter::_printHelp(String command) {
00242     /* Print help lines according to command */
00243     if (command == "" || command == COMMAND_HELP) { //Default help
00244         Serial.println("|-----HELP-----|");
00245         Serial.println("This is a commandline interface that allows access to the smartlight config");
00246         _printCommands();
00247     } else if (command == COMMAND_DEBUG) {
00248         Serial.println("Usage: debug [--off]           Turns the debug off");
00249         Serial.println("          debug [--on]           Turns the debug on");
00250     } else if (command == COMMAND_SU) {
00251         Serial.println("Usage: su {passwd}           Login as superuser");
00252     } else if (command == COMMAND_KEYS) {
00253         Serial.println("Usage: privatekeys           Shows private encryption keys");
00254     } else if (command == COMMAND_RESTART) {
00255         Serial.println("Usage: reboot               Reboots the device");
00256     } else if (command == COMMAND_USERS) {
00257         Serial.println("Usage: users                 Shows usertable of website");
00258     } else if (command == COMMAND_HOSTNAME) {
00259         Serial.println("Usage: hostname             Gives the current hostname");
00260         Serial.println("          hostname [--set] {newhostname} Set new hostname. (needs reboot)");
00261         Serial.println("          hostname [--default]         Sets the hostname to the default
00262 hostname");
00263     } else if (command == COMMAND_LS) {
00264         Serial.println("Usage: ls                     Shows files in current folder");
00265     } else if (command == COMMAND_VI) {
00266         Serial.println("Usage: vi {filename}         Opens file in text editor");
00267     } else if (command == COMMAND_RUN) {
00268         Serial.println("Usage: ./ {filename}         Runs an executable file");
00269     } else if (command == COMMAND_OBJDUMP) {
00270         Serial.println("Usage: objdump -d {filename} Prints disassembled code of an
00271 executable file");
00272     } else if (command == COMMAND_GPG) {
00273         Serial.println("Usage: gpg --encrypt {key} {line} Prints disassembled code of an
00274 executable file");
00275         Serial.println("Usage: gpg --decrypt {key} {line} Prints disassembled code of an
00276 executable file");
00277     } else {
00278         Serial.println(ERROR_CMD_NOT_FOUND);
00279     }
00280 }
00281
00282 /*****
00283 /*****
00284 void SerialCommandExecuter::_printCommands() {
00285     Serial.println("Available commands:");
00286     Serial.println(COMMAND_HELP);
00287     Serial.println(COMMAND_DEBUG);
00288     Serial.println(COMMAND_SU);
00289     Serial.println(COMMAND_KEYS);
00290     Serial.println(COMMAND_RESTART);
00291     Serial.println(COMMAND_USERS);
00292     Serial.println(COMMAND_HOSTNAME);
00293     Serial.println(COMMAND_LS);
00294     Serial.println(COMMAND_VI);
00295     Serial.println(COMMAND_OBJDUMP);
00296     Serial.println(COMMAND_WHOAMI);
00297     Serial.println(COMMAND_GPG);
00298 }
00299
00300 /*****
00301 /*****
00302 bool SerialCommandExecuter::_enableDebug(String enable) {
00303     if (enable == ARG_DEBUG_ON) {
00304         setDebugEnabled(true);
00305         Serial.println("debug = true");
00306     } else if (enable == ARG_DEBUG_OFF) {
00307         setDebugEnabled(false);
00308         Serial.println("debug = false");
00309     } else {
00310         Serial.println(ERROR_WRONG_ARGS);
00311         return false;
00312     }
00313     return true;
00314 }
00315
00316 /*****
00317 /*****
00318
00319
00320
00321
00322
00323

```



```

00324
00325 /*****
00331 /*****
00332 bool SerialCommandExecuter::_superUserLogin(String password) {
00333     if (password == ROOT_PASSWORD) {
00334         _isLoggedIn = true;
00335         Serial.println(MESS_SUPER_USER);
00336     } else {
00337         Serial.println(ERROR_WRONG_PWD);
00338         return false;
00339     }
00340     return true;
00341 }
00342
00343 /*****
00348 /*****
00349 bool SerialCommandExecuter::_viewKey() {
00350     if (!_isLoggedIn) {
00351         Serial.println(ERROR_NO_PERMISSION);
00352         return false;
00353     }
00354     Serial.println("Private encryption keys (Don't share!!!):");
00355     Serial.println(AES_KEY);
00356     return true;
00357 }
00358
00359 /*****
00363 /*****
00364 void SerialCommandExecuter::_restart() {
00365     Serial.print("Restarting in ");
00366
00367     /* Wait 3 seconds */
00368     for (uint8_t s = 3; s > 0; s--) {
00369         Serial.print(s);
00370         Serial.print(" ");
00371         delay(1000);
00372     }
00373     ESP.restart();
00374 }
00375
00376 /*****
00381 /*****
00382 bool SerialCommandExecuter::_viewUsers() {
00383     String userPrints[USER_INFO_LENGTH] = {" "};
00384
00385     if (!_isLoggedIn) {
00386         Serial.println(ERROR_NO_PERMISSION);
00387         return false;
00388     }
00389
00390     Serial.println("|-USERNAME-----|-PASSWORD-----|-ROLE---|");
00391
00392     for (uint8_t i = 0; i < _numberUsers; i += 3) {
00393         userPrints[0] = _users[i].c_str(); //Username
00394         if (atoi(_users[i+2].c_str()) == PERMISSION_LVL_USER) {
00395             userPrints[1] = _users[i+1].c_str(); //Password
00396             userPrints[2] = "User"; //Permission level/role
00397         } else if (atoi(_users[i+2].c_str()) == PERMISSION_LVL_ADMIN) {
00398             userPrints[1] = "*****"; //Password, not printed
00399             userPrints[2] = "Admin"; //Permission level/role
00400         }
00401         Serial.printf("| %-15s| %-15s| %-7s|\n", userPrints[0].c_str(), userPrints[1].c_str(),
00402             userPrints[2].c_str());
00403         return true;
00404     }
00405
00406 /*****
00412 /*****
00413 bool SerialCommandExecuter::_hostname(String* params) {
00414     uint8_t numParams = params->length();
00415     if (numParams == 0) { //If empty: show hostname
00416         Serial.print("Hostname is: ");
00417         Serial.println(String(getHostname()));
00418         return true;
00419     }
00420
00421     if (params[0] == ARG_HOSTNAME_SET && params[1] != " ") { //If parameter 'set' check
00422         if next value is not empty
00423             char newHostname[MAX_HOSTNAME_LENGTH];
00424             params[1].toCharArray(newHostname, MAX_HOSTNAME_LENGTH);
00425             writeHostname(newHostname);
00426         } else if (params[0] == ARG_HOSTNAME_DEFAULT) {
00427             writeHostname(DEFAULT_HOSTNAME);
00428         } else {
00429             Serial.println(ERROR_WRONG_ARGS); //If it can't find
00430             suitable params: give error

```

```

00429         return false;
00430     }
00431     return true;
00432 }
00433
00434
00435 /*****
00441 /*****
00442 bool SerialCommandExecuter::_encrypt(String* params) {
00443     if (!cryptor.setKey(params[1])) {
00444         Serial.println(ERROR_NO_VALID_KEY);
00445         return false;
00446     }
00447
00448     Serial.print("Encrypted output: ");
00449     Serial.println(cryptor.encryptLine(params[2]));
00450     return true;
00451 }
00452
00453 /*****
00459 /*****
00460 bool SerialCommandExecuter::_decrypt(String* params) {
00461     if (!cryptor.setKey(params[1])) {
00462         Serial.println(ERROR_NO_VALID_KEY);
00463         return false;
00464     }
00465
00466     Serial.print("Decrypted output: ");
00467     Serial.println(cryptor.decryptLine(params[2]));
00468     return true;
00469 }
00470
00471 /*****
00479 /*****
00480 bool SerialCommandExecuter::_checkHelp(String param, String command) {
00481     if (param == ARG_HELP_SHORT || param == ARG_HELP_LONG) {
00482         _printHelp(command);
00483         return true;
00484     }
00485     return false;
00486 }

```

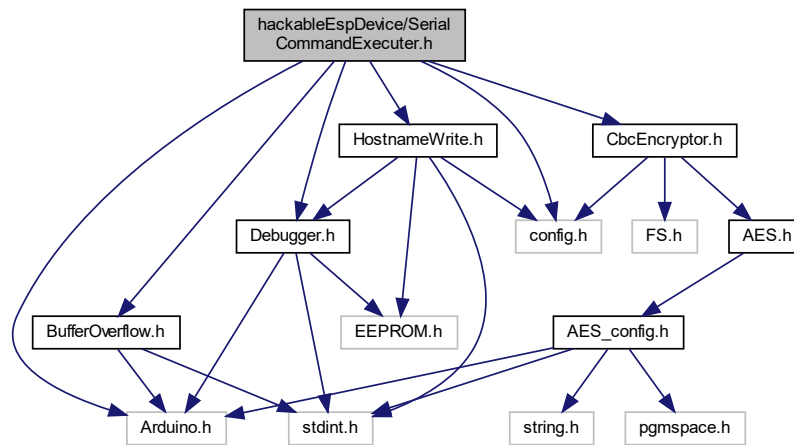
5.27 hackableEspDevice/SerialCommandExecuter.h File Reference

```

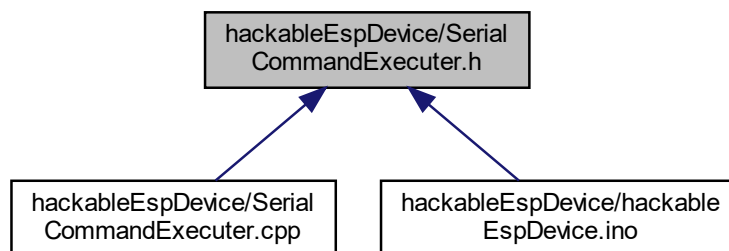
#include "Arduino.h"
#include "config.h"
#include "Debugger.h"
#include "HostnameWrite.h"
#include "BufferOverflow.h"
#include "CbcEncryptor.h"

```

Include dependency graph for SerialCommandExecuter.h:



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



Classes

- class [SerialCommandExecuter](#)

Macros

- `#define` [MAX_NUMBER_PARAMS](#) 3
- `#define` [COMMAND_HELP](#) "help"
- `#define` [COMMAND_DEBUG](#) "debug"
- `#define` [COMMAND_SU](#) "su"
- `#define` [COMMAND_KEYS](#) "privatekeys"
- `#define` [COMMAND_RESTART](#) "reboot"
- `#define` [COMMAND_USERS](#) "users"
- `#define` [COMMAND_HOSTNAME](#) "hostname"

- `#define COMMAND_WHOAMI "whoami"`
- `#define COMMAND_LS "ls"`
- `#define COMMAND_VI "vi"`
- `#define COMMAND_RUN "./"`
- `#define COMMAND_OBJDUMP "objdump"`
- `#define COMMAND_GPG "gpg"`
- `#define ARG_HELP_LONG "--help"`
- `#define ARG_HELP_SHORT "-h"`
- `#define ARG_DEBUG_ON "--on"`
- `#define ARG_DEBUG_OFF "--off"`
- `#define ARG_HOSTNAME_SET "--set"`
- `#define ARG_HOSTNAME_DEFAULT "--default"`
- `#define ARG_GPG_ENCRYPT "--encrypt"`
- `#define ARG_GPG_DECRYPT "--decrypt"`
- `#define ARG_LS_FILE_1_1 "testprogram.c"`
- `#define ARG_LS_FILE_1_2 "./testprogram.c"`
- `#define ARG_LS_FILE_2_1 "testprogram"`
- `#define ARG_LS_FILE_2_2 "./testprogram"`
- `#define MESS_SUPER_USER "You are now super user."`
- `#define ERROR_TOO_MANY_ARGS "Too many arguments. Add '-h' or '--help' to the command for help."`
- `#define ERROR_CMD_NOT_FOUND "Bash: command not found. Type 'help' for help."`
- `#define ERROR_PERM_DENIED "Bash: Permission denied"`
- `#define ERROR_WRONG_ARGS "Wrong argument(s). Add '-h' or '--help' to the command for help."`
- `#define ERROR_TOO_FEW_ARGS "Too few arguments. Add '-h' or '--help' to the command for help."`
- `#define ERROR_WRONG_PWD "Wrong password."`
- `#define ERROR_NO_PERMISSION "You are no super user. Use 'su {password}' to log in."`
- `#define ERROR_NO_FILE "No such file."`
- `#define ERROR_NO_FILE_DIR "No such file or directory."`
- `#define ERROR_NO_VALID_KEY "No valid key, needs to be 16 bytes long."`

5.27.1 Macro Definition Documentation

5.27.1.1 ARG_DEBUG_OFF

```
#define ARG_DEBUG_OFF "--off"
```

Definition at line 41 of file [SerialCommandExecuter.h](#).

5.27.1.2 ARG_DEBUG_ON

```
#define ARG_DEBUG_ON "--on"
```

Definition at line 40 of file [SerialCommandExecuter.h](#).

5.27.1.3 ARG_GPG_DECRYPT

```
#define ARG_GPG_DECRYPT "--decrypt"
```

Definition at line 45 of file [SerialCommandExecuter.h](#).

5.27.1.4 ARG_GPG_ENCRYPT

```
#define ARG_GPG_ENCRYPT "--encrypt"
```

Definition at line 44 of file [SerialCommandExecuter.h](#).

5.27.1.5 ARG_HELP_LONG

```
#define ARG_HELP_LONG "--help"
```

Definition at line 38 of file [SerialCommandExecuter.h](#).

5.27.1.6 ARG_HELP_SHORT

```
#define ARG_HELP_SHORT "-h"
```

Definition at line 39 of file [SerialCommandExecuter.h](#).

5.27.1.7 ARG_HOSTNAME_DEFAULT

```
#define ARG_HOSTNAME_DEFAULT "--default"
```

Definition at line 43 of file [SerialCommandExecuter.h](#).

5.27.1.8 ARG_HOSTNAME_SET

```
#define ARG_HOSTNAME_SET "--set"
```

Definition at line 42 of file [SerialCommandExecuter.h](#).

5.27.1.9 ARG_LS_FILE_1_1

```
#define ARG_LS_FILE_1_1 "testprogram.c"
```

Definition at line 46 of file [SerialCommandExecuter.h](#).

5.27.1.10 ARG_LS_FILE_1_2

```
#define ARG_LS_FILE_1_2 "./testprogram.c"
```

Definition at line 47 of file [SerialCommandExecuter.h](#).

5.27.1.11 ARG_LS_FILE_2_1

```
#define ARG_LS_FILE_2_1 "testprogram"
```

Definition at line 48 of file [SerialCommandExecuter.h](#).

5.27.1.12 ARG_LS_FILE_2_2

```
#define ARG_LS_FILE_2_2 "./testprogram"
```

Definition at line 49 of file [SerialCommandExecuter.h](#).

5.27.1.13 COMMAND_DEBUG

```
#define COMMAND_DEBUG "debug"
```

Definition at line 21 of file [SerialCommandExecuter.h](#).

5.27.1.14 COMMAND_GPG

```
#define COMMAND_GPG "gpg"
```

Definition at line 36 of file [SerialCommandExecuter.h](#).

5.27.1.15 COMMAND_HELP

```
#define COMMAND_HELP "help"
```

Definition at line 20 of file [SerialCommandExecuter.h](#).

5.27.1.16 COMMAND_HOSTNAME

```
#define COMMAND_HOSTNAME "hostname"
```

Definition at line 26 of file [SerialCommandExecuter.h](#).

5.27.1.17 COMMAND_KEYS

```
#define COMMAND_KEYS "privatekeys"
```

Definition at line 23 of file [SerialCommandExecuter.h](#).

5.27.1.18 COMMAND_LS

```
#define COMMAND_LS "ls"
```

Definition at line 30 of file [SerialCommandExecuter.h](#).

5.27.1.19 COMMAND_OBJDUMP

```
#define COMMAND_OBJDUMP "objdump"
```

Definition at line 33 of file [SerialCommandExecuter.h](#).

5.27.1.20 COMMAND_RESTART

```
#define COMMAND_RESTART "reboot"
```

Definition at line 24 of file [SerialCommandExecuter.h](#).

5.27.1.21 COMMAND_RUN

```
#define COMMAND_RUN "./"
```

Definition at line 32 of file [SerialCommandExecuter.h](#).

5.27.1.22 COMMAND_SU

```
#define COMMAND_SU "su"
```

Definition at line 22 of file [SerialCommandExecuter.h](#).

5.27.1.23 COMMAND_USERS

```
#define COMMAND_USERS "users"
```

Definition at line 25 of file [SerialCommandExecuter.h](#).

5.27.1.24 COMMAND_VI

```
#define COMMAND_VI "vi"
```

Definition at line 31 of file [SerialCommandExecuter.h](#).

5.27.1.25 COMMAND_WHOAMI

```
#define COMMAND_WHOAMI "whoami"
```

Definition at line 27 of file [SerialCommandExecuter.h](#).

5.27.1.26 ERROR_CMD_NOT_FOUND

```
#define ERROR_CMD_NOT_FOUND "Bash: command not found. Type 'help' for help."
```

Definition at line 56 of file [SerialCommandExecuter.h](#).

5.27.1.27 ERROR_NO_FILE

```
#define ERROR_NO_FILE "No such file."
```

Definition at line 62 of file [SerialCommandExecuter.h](#).

5.27.1.28 ERROR_NO_FILE_DIR

```
#define ERROR_NO_FILE_DIR "No such file or directory."
```

Definition at line 63 of file [SerialCommandExecuter.h](#).

5.27.1.29 ERROR_NO_PERMISSION

```
#define ERROR_NO_PERMISSION "You are no super user. Use 'su {password}' to log in."
```

Definition at line 61 of file [SerialCommandExecuter.h](#).

5.27.1.30 ERROR_NO_VALID_KEY

```
#define ERROR_NO_VALID_KEY "No valid key, needs to be 16 bytes long."
```

Definition at line 64 of file [SerialCommandExecuter.h](#).

5.27.1.31 ERROR_PERM_DENIED

```
#define ERROR_PERM_DENIED "Bash: Permission denied"
```

Definition at line 57 of file [SerialCommandExecuter.h](#).

5.27.1.32 ERROR_TOO_FEW_ARGS

```
#define ERROR_TOO_FEW_ARGS "Too few arguments. Add '-h' or '--help' to the command for help."
```

Definition at line 59 of file [SerialCommandExecuter.h](#).

5.27.1.33 ERROR_TOO_MANY_ARGS

```
#define ERROR_TOO_MANY_ARGS "Too many arguments. Add '-h' or '--help' to the command for help."
```

Definition at line 55 of file [SerialCommandExecuter.h](#).

5.27.1.34 ERROR_WRONG_ARGS

```
#define ERROR_WRONG_ARGS "Wrong argument(s). Add '-h' or '--help' to the command for help."
```

Definition at line 58 of file [SerialCommandExecuter.h](#).

5.27.1.35 ERROR_WRONG_PWD

```
#define ERROR_WRONG_PWD "Wrong password."
```

Definition at line 60 of file [SerialCommandExecuter.h](#).

5.27.1.36 MAX_NUMBER_PARAMS

```
#define MAX_NUMBER_PARAMS 3
```

Definition at line 18 of file [SerialCommandExecuter.h](#).

5.27.1.37 MESS_SUPER_USER

```
#define MESS_SUPER_USER "You are now super user."
```

Definition at line 52 of file [SerialCommandExecuter.h](#).

5.28 SerialCommandExecuter.h

[Go to the documentation of this file.](#)

```

00001 /*
00002  * File:      SerialCommandExecuter.h
00003  * Author:    Luke de Munk & Twenne Elffers
00004  * Class:     SerialCommandExecuter
00005  * Version:   1.0
00006  *
00007  * Parses and executes serial terminal commands.
00008  */
00009 #ifndef SERIAL_COMMAND_EXECUTER_H
00010 #define SERIAL_COMMAND_EXECUTER_H
00011 #include "Arduino.h"
00012 #include "config.h"
00013 #include "Debugger.h"
00014 #include "HostnameWrite.h"
00015 #include "BufferOverflow.h"
00016 #include "CbcEncryptor.h"
00017
00018 #define MAX_NUMBER_PARAMS 3
00019
00020 #define COMMAND_HELP "help"
00021 #define COMMAND_DEBUG "debug"
00022 #define COMMAND_SU "su"
00023 #define COMMAND_KEYS "privatekeys"
00024 #define COMMAND_RESTART "reboot"
00025 #define COMMAND_USERS "users"
00026 #define COMMAND_HOSTNAME "hostname"
00027 #define COMMAND_WHOAMI "whoami"
00028
00029 /* Used for buffer overflow */
00030 #define COMMAND_LS "ls"
00031 #define COMMAND_VI "vi"
00032 #define COMMAND_RUN "./"
00033 #define COMMAND_OBJDUMP "objdump"
00034
00035 /* Used for encryption */
00036 #define COMMAND_GPG "gpg"
00037
00038 #define ARG_HELP_LONG "--help"
00039 #define ARG_HELP_SHORT "-h"
00040 #define ARG_DEBUG_ON "--on"
00041 #define ARG_DEBUG_OFF "--off"
00042 #define ARG_HOSTNAME_SET "--set"
00043 #define ARG_HOSTNAME_DEFAULT "--default"
00044 #define ARG_GPG_ENCRYPT "--encrypt"
00045 #define ARG_GPG_DECRYPT "--decrypt"
00046 #define ARG_LS_FILE_1_1 "testprogram.c"
00047 #define ARG_LS_FILE_1_2 "./testprogram.c"
00048 #define ARG_LS_FILE_2_1 "testprogram"
00049 #define ARG_LS_FILE_2_2 "./testprogram"
00050
00051 #define MESS_SUPER_USER "You are now super user."
00052
00053 #define ERROR_TOO_MANY_ARGS "Too many arguments. Add '-h' or '--help' to the command for help."
00054 #define ERROR_CMD_NOT_FOUND "Bash: command not found. Type 'help' for help."
00055 #define ERROR_PERM_DENIED "Bash: Permission denied"
00056 #define ERROR_WRONG_ARGS "Wrong argument(s). Add '-h' or '--help' to the command for help."
00057 #define ERROR_TOO_FEW_ARGS "Too few arguments. Add '-h' or '--help' to the command for help."
00058 #define ERROR_WRONG_PWD "Wrong password."
00059 #define ERROR_NO_PERMISSION "You are no super user. Use 'su {password}' to log in."
00060 #define ERROR_NO_FILE "No such file."
00061 #define ERROR_NO_FILE_DIR "No such file or directory."
00062 #define ERROR_NO_VALID_KEY "No valid key, needs to be 16 bytes long."
00063
00064 class SerialCommandExecuter
00065 {
00066 public:
00067     SerialCommandExecuter();
00068     void executeCommand();
00069     void setUsers(String* users, uint8_t numUsers);
00070
00071 private:
00072     bool _parseCommand(String command);
00073     String* _trimCommand(String commandString);
00074     bool _checkParams(uint8_t numParams, uint8_t minNumberParams, uint8_t maxNumberParams);
00075
00076     void _printHelp(String command);
00077     void _printCommands();
00078     bool _enableDebug(String enable);
00079     bool _superUserLogin(String password);

```

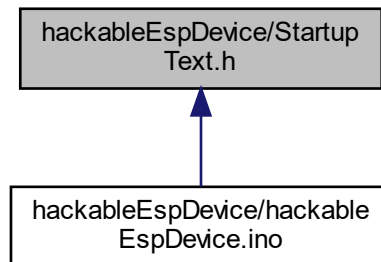
```

00082     bool _viewKey();
00083     void _restart();
00084     bool _viewUsers();
00085     bool _hostname(String* params);
00086     bool _encrypt(String* params);
00087     bool _decrypt(String* params);
00088     bool _checkHelp(String param, String command);
00089
00090     bool _isLoggedIn;
00091     String _users[MAX_NUMBER_USERS*USER_INFO_LENGTH];
00092     uint8_t _numberUsers;
00093     BufferOverflow buffOverflow;
00094     CbcEncryptor cryptor;
00095 };
00096 #endif

```

5.29 hackableEspDevice/StartupText.h File Reference

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



Macros

- `#define` [LENGTH](#) 20

Functions

- bool [printStartupText](#) (String hiddenMess)
Prints bytes of information with a message wrapped in it.
- bool [printStringInBytes](#) (String str)
Converts message in bytes and prints it.

5.29.1 Macro Definition Documentation

5.29.1.1 LENGTH

```
#define LENGTH 20
```

Definition at line 12 of file [StartupText.h](#).

5.29.2 Function Documentation

5.29.2.1 printStartupText()

```
bool printStartupText (  
    String hiddenMess )
```

Prints bytes of information with a message wrapped in it.

Parameters

<i>hiddenMess</i>	String of text that needs to be printed
-------------------	---

Returns

bool If conversion is successfull

Definition at line 25 of file [StartupText.h](#).

5.29.2.2 printStringInBytes()

```
bool printStringInBytes (  
    String str )
```

Converts message in bytes and prints it.

Parameters

<i>str</i>	String of text that needs to be printed
------------	---

Returns

bool If conversion is successfull

Definition at line 303 of file [StartupText.h](#).


```

00088     Serial.println(F("63 61 74 65 73 20 61 72 65 20 63 6F 6E 66 69 67 75 72 65"));
00089     Serial.println(F("64 20 6F 6E 20 49 64 50 2C 68 6F 77 65 76 65 72 20 63 61"));
00090     Serial.println(F("6E 20 6E 6F 74 20 64 65 74 65 72 6D 69 6E 65 20 63 65 72"));
00091     Serial.println(F("74 69 66 69 63 61 74 65 20 66 6F 72 20 73 69 67 6E 61 74"));
00092     Serial.println(F("75 72 65 0A 53 75 73 70 65 6E 64 20 6C 6F 67 20 63 6F 6C"));
00093     Serial.println(F("6C 65 63 74 6F 72 0A 46 61 69 6C 65 64 20 74 6F 20 6A 6F"));
00094     Serial.println(F("69 6E 20 74 6F 20 41 44 0A 49 67 6E 6F 72 65 20 4D 61 63"));
00095     Serial.println(F("68 69 6E 65 20 41 75 74 68 6F 72 69 7A 61 74 69 6F 6E 20"));
00096     Serial.println(F("50 41 43 20 72 65 71 75 65 73 74 20 62 65 63 61 75 73 65"));
00097     Serial.println(F("20 6F 66 20 63 75 72 72 65 6E 74 20 50 41 43 20 6F 66 20"));
00098     Serial.println(F("74 68 65 20 73 61 6D 65 20 74 79 70 65 20 77 61 73 20 75"));
00099     Serial.println(F("73 65 64 20 74 6F 20 73 6B 69 70 20 69 6E 6E 65 72 20 6D"));
00100     Serial.println(F("65 74 68 6F 64 0A 4E 54 50 20 53 65 72 76 65 72 20 73 65"));
00101     Serial.println(F("74 0A 43 68 69 70 20 69 73 20 45 53 50 38 32 36 36 45 58"));
00102     Serial.println(F("0A 46 65 61 74 75 72 65 73 3A 20 57 69 46 69 0A 43 72 79"));
00103     Serial.println(F("73 74 61 6C 20 69 73 20 32 36 4D 48 7A 0A 4D 41 43 3A 20"));
00104     Serial.println(F("38 63 3A 61 61 3A 62 35 3A 37 62 3A 65 30 3A 61 38 0A 43"));
00105     Serial.println(F("6F 6D 70 72 65 73 73 65 64 20 33 34 34 36 30 38 20 62 79"));
00106     Serial.println(F("74 65 73 20 74 6F 20 32 34 38 38 32 36 2E 2E 2E 0A 48 61"));
00107     Serial.println(F("73 68 20 6F 66 20 64 61 74 61 20 76 65 72 69 66 69 65 64"));
00108     Serial.println(F("2E 0A 43 6C 69 65 6E 74 20 63 65 72 74 69 66 69 63 61 74"));
00109     Serial.println(F("65 20 77 61 73 20 72 65 71 75 65 73 74 65 64 20 62 75 74"));
00110     Serial.println(F("20 6E 6F 74 20 72 65 63 65 69 76 65 64 20 69 6E 73 69 64"));
00111     Serial.println(F("65 20 74 68 65 20 74 75 6E 6E 65 6C 2E 20 57 69 6C 6C 20"));
00112     Serial.println(F("63 6F 6E 74 69 6E 75 65 20 77 69 74 68 20 69 6E 6E 65 72"));
00113     Serial.println(F("20 6D 65 74 68 6F 64 2E 0A 54 65 6C 65 6D 65 74 72 79 20"));
00114     Serial.println(F("6D 65 73 73 61 67 65 73 20 77 65 72 65 20 73 65 6E 74 20"));
00115     Serial.println(F("73 75 63 63 65 73 73 66 75 6C 6C 79 0A 44 65 6C 65 74 65"));
00116     Serial.println(F("20 6E 6F 64 65 20 66 61 69 6C 65 64 0A 50 72 6F 66 69 6C"));
00117     Serial.println(F("65 72 20 45 6E 64 50 6F 69 6E 74 20 63 6F 6C 6C 65 63 74"));
00118     Serial.println(F("69 6F 6E 20 65 76 65 6E 74 20 6F 63 63 75 72 72 65 64 0A"));
00119     Serial.println(F("52 41 44 49 55 53 20 44 54 4C 53 20 43 6F 41 20 68 61 6E"));
00120     Serial.println(F("64 73 68 61 6B 65 20 73 74 61 72 74 65 64 0A 52 75 6E"));
00121     Serial.println(F("69 6E 67 20 73 74 75 62 2E 2E 2E 0A 53 74 75 62 20 72 75"));
00122     Serial.println(F("6E 6E 69 6E 67 2E 2E 2E 0A 53 74 6F 70 70 65 64 20 54 41"));
00123     Serial.println(F("43 41 43 53 2B 20 6C 69 73 74 65 6E 65 72 0A 53 65 6C 65"));
00124     Serial.println(F("63 74 65 64 20 41 63 63 65 73 73 20 53 65 72 76 69 63 65"));
00125     Serial.println(F("20 74 79 70 65 20 69 73 20 6E 6F 74 20 44 65 76 69 63 65"));
00126     Serial.println(F("20 41 64 6D 69 6E 69 73 74 72 61 74 69 6F 6E 0A 4C 6F 63"));
00127     Serial.println(F("61 6C 20 6D 6F 64 65 0A 55 73 65 72 20 61 75 74 68 65 6E"));
00128     Serial.println(F("74 69 63 61 74 69 6F 6E 20 61 67 61 69 6E 73 74 20 41 63"));
00129     Serial.println(F("74 69 76 65 20 44 69 72 65 63 74 6F 72 79 20 66 61 69 6C"));
00130     Serial.println(F("65 64 20 73 69 6E 63 65 20 75 73 65 72 20 68 61 73 20 69"));
00131     Serial.println(F("6E 76 61 6C 69 64 20 63 72 65 64 65 6E 74 69 61 6C 73 0A"));
00132     Serial.println(F("43 41 20 73 65 72 76 69 63 65 20 64 69 73 61 62 6C 65 64"));
00133     Serial.println(F("0A 43 68 61 6E 67 69 6E 67 20 62 61 75 64 20 72 61 74 65"));
00134     Serial.println(F("20 74 6F 20 34 36 30 38 30 30 0A 43 6F 6E 66 69 67 75 72"));
00135     Serial.println(F("69 6E 67 20 66 6C 61 73 68 20 73 69 7A 65 2E 2E 2E 0A 41"));
00136     Serial.println(F("75 74 6F 2D 64 65 74 65 63 74 65 64 20 46 6C 61 73 68 20"));
00137     Serial.println(F("73 69 7A 65 3A 20 34 4D 42 0A 49 6E 76 61 6C 69 64 20 6E"));
00138     Serial.println(F("65 77 20 70 61 73 73 77 6F 72 64 2E 20 43 6F 6E 74 61 69"));
00139     Serial.println(F("6E 73 20 72 65 73 65 72 76 65 64 20 77 6F 72 64 0A 52 53"));
00140     Serial.println(F("41 20 61 67 65 6E 74 20 63 6F 6E 66 69 67 75 72 61 74 69"));
00141     Serial.println(F("20 75 70 64 61 74 65 64 2C 20 52 53 41 20 61 67 65"));
00142     Serial.println(F("6E 74 20 72 65 73 74 61 72 74 65 64 0A 4C 6F 6F 6B 75 70"));
00143     Serial.println(F("20 53 49 44 20 42 79 20 4E 61 6D 65 20 72 65 71 75 65 73"));
00144     Serial.println(F("74 20 66 61 69 6C 65 64 0A 53 74 61 72 74 20 6C 69 73 74"));
00145     Serial.println(F("65 6E 69 6E 67 20 74 6F 20 74 63 70 20 70 6F 72 74 0A 49"));
00146     Serial.println(F("67 6E 6F 72 65 20 4D 61 63 68 69 6E 65 20 41 75 74 68 6F"));
00147     Serial.println(F("72 69 7A 61 74 69 6F 6E 20 50 41 43 20 72 65 71 75 65 73"));
00148     Serial.println(F("74 20 62 65 63 61 75 73 65 20 6F 66 20 63 75 72 72 65 6E"));
00149     Serial.println(F("74 20 50 41 43 20 6F 66 20 74 68 65 20 73 61 6D 65 20 74"));
00150     Serial.println(F("79 70 65 20 77 61 73 20 75 73 65 64 20 74 6F 20 73 6B 69"));
00151     Serial.println(F("70 20 69 6E 6E 65 72 20 6D 65 74 68 6F 64 0A 49 53 45 20"));
00152     Serial.println(F("75 70 67 72 61 64 65 20 2D 20 4D 6E 54 0A 49 53 45 20 42"));
00153     Serial.println(F("61 63 6B 75 70 20 68 61 73 20 73 74 61 72 74 65 64 0A 54"));
00154     Serial.println(F("72 75 73 74 73 65 63 20 65 67 72 65 73 73 20 70 6F 6C 69 69"));
00155     Serial.println(F("63 79 20 77 61 73 20 73 75 63 63 65 73 73 66 75 6C 6C 79"));
00156     Serial.println(F("20 64 6F 77 6E 6C 6F 61 64 65 64 0A 52 41 44 49 55 53 20"));
00157     Serial.println(F("44 54 4C 53 3A 20 72 65 63 65 69 76 65 64 20 63 6C 69 65"));
00158     Serial.println(F("6E 74 20 68 65 6C 6C 6F 20 76 65 72 69 66 79 20 72 65 0A"));
00159     /* Print the message, return false if is not successfull */
00160     if (!printStringInBytes(hiddenMess)) {
00161         return false;
00162     }
00163     Serial.println(F("75 65 73 74 0A 47 75 65 73 74 20 73 65 73 73 69 6F 6E 20"));
00164     Serial.println(F("6C 69 6D 69 74 20 69 73 20 61 63 74 69 76 65 3B 20 72 65"));
00165     Serial.println(F("6D 6F 76 69 6E 67 20 6F 6C 64 65 72 20 67 75 65 73 74 20"));
00166     Serial.println(F("73 65 73 69 6F 6E 73 0A 49 67 6E 6F 72 65 20 4D 61 63"));
00167     Serial.println(F("68 69 6E 65 20 41 75 74 68 6F 72 69 7A 61 74 69 6F 6E 20"));
00168     Serial.println(F("50 41 43 20 72 65 71 75 65 73 74 20 62 65 63 61 75 73 65"));
00169     Serial.println(F("20 6F 66 20 63 75 72 72 65 6E 74 20 50 41 43 20 6F 66 20"));
00170     Serial.println(F("74 68 65 20 73 61 6D 65 20 74 79 70 65 20 77 61 73 20 75"));
00171     Serial.println(F("73 65 64 20 74 6F 20 73 6B 69 70 20 69 6E 6E 65 72 20 6D"));
00172     Serial.println(F("65 74 68 6F 64 0A 43 6C 69 65 6E 74 20 63 65 72 74 69 66"));
00173     Serial.println(F("69 63 61 74 65 20 77 61 73 20 72 65 71 75 65 73 74 65 64"));
00174     Serial.println(F("20 62 75 74 20 6E 6F 74 20 72 65 63 65 69 76 65 64 20 69"));

```

```
00175 Serial.println(F("6E 73 69 64 65 20 74 68 65 20 74 75 6E 6E 65 6C 2E 20 57")));
00176 Serial.println(F("69 6C 6C 20 63 6F 6E 74 69 6E 75 65 20 77 69 74 68 20 69")));
00177 Serial.println(F("6E 6E 65 72 20 6D 65 74 68 6F 64 2E 0A 54 65 6C 65 6D 65")));
00178 Serial.println(F("74 72 79 20 6D 65 73 73 61 67 65 73 20 77 65 72 65 20 73")));
00179 Serial.println(F("65 6E 74 20 73 75 63 63 65 73 73 66 75 6C 6C 79 0A 50 72")));
00180 Serial.println(F("6F 66 69 6C 65 72 20 45 6E 64 50 6F 69 6E 74 20 63 6F 6C")));
00181 Serial.println(F("6C 65 63 74 69 6F 6E 20 65 76 65 6E 74 20 6F 63 63 75 72")));
00182 Serial.println(F("72 65 64 0A 52 41 44 49 55 53 20 44 54 4C 53 20 43 6F 41")));
00183 Serial.println(F("20 68 61 6E 64 73 68 61 6B 65 20 73 74 61 72 74 65 64 0A")));
00184 Serial.println(F("53 74 6F 70 70 65 64 20 54 41 43 41 43 53 2B 20 6C 69 73")));
00185 Serial.println(F("74 65 6E 65 72 0A 53 65 6C 65 63 74 65 64 20 41 63 63 65")));
00186 Serial.println(F("69 73 20 53 65 72 76 69 63 65 20 74 79 70 65 20 69 73 20")));
00187 Serial.println(F("6E 6F 74 20 44 65 76 69 63 65 20 41 64 6D 69 6E 69 73 74")));
00188 Serial.println(F("72 61 74 69 6F 6E 0A 43 41 20 73 65 72 76 69 63 65 20 64")));
00189 Serial.println(F("69 73 61 62 6C 65 64 0A 52 53 41 20 61 67 65 6E 74 20 63")));
00190 Serial.println(F("6F 6E 66 69 67 75 72 61 74 69 6F 6E 20 75 70 64 61 74 65")));
00191 Serial.println(F("64 2C 20 52 53 41 20 61 67 65 6E 74 20 72 65 73 74 61 72")));
00192 Serial.println(F("74 65 64 0A 53 74 61 72 74 20 6C 69 73 74 65 6E 69 6E 67")));
00193 Serial.println(F("20 74 6F 20 74 63 70 20 70 6F 72 74 0A 49 67 6E 6F 72 65")));
00194 Serial.println(F("20 4D 61 63 68 69 6E 65 20 41 75 74 68 6F 72 69 7A 61 74")));
00195 Serial.println(F("69 6F 6E 20 50 41 43 20 72 65 71 75 65 73 74 20 62 65 63")));
00196 Serial.println(F("61 75 73 65 20 6F 66 20 63 75 72 72 65 6E 74 20 50 41 43")));
00197 Serial.println(F("20 6F 66 20 74 68 65 20 73 61 6D 65 20 74 79 70 65 20 77")));
00198 Serial.println(F("61 73 20 75 73 65 64 20 74 6F 20 73 6B 69 70 20 69 6E 6E")));
00199 Serial.println(F("65 72 20 6D 65 74 68 6F 64 0A 49 53 45 20 42 61 63 6B 75")));
00200 Serial.println(F("70 20 68 61 73 20 73 74 61 72 74 65 64 0A 53 6D 61 72 74")));
00201 Serial.println(F("20 4C 69 63 65 6E 73 69 6E 67 20 61 75 74 68 6F 72 69 7A")));
00202 Serial.println(F("61 74 69 6F 6E 20 72 65 6E 65 77 61 6C 20 73 75 63 63 65")));
00203 Serial.println(F("73 73 0A 52 65 6D 69 6E 64 65 72 3A 20 41 73 73 69 67 6E")));
00204 Serial.println(F("20 4E 41 44 20 50 72 6F 66 69 6C 65 73 2E 0A 52 41 44 49")));
00205 Serial.println(F("55 53 20 44 54 4C 53 3A 20 73 65 6E 74 20 66 69 6E 69 73")));
00206 Serial.println(F("68 65 64 20 6D 65 73 73 61 67 65 0A 50 72 65 70 61 72 65")));
00207 Serial.println(F("64 20 54 4C 53 20 53 65 72 76 65 72 4B 65 79 45 78 63 68")));
00208 Serial.println(F("61 6E 67 65 20 6D 65 73 73 61 67 65 0A 54 68 65 20 73 65")));
00209 Serial.println(F("63 75 72 69 64 20 66 69 6C 65 20 68 61 73 20 62 65 65 6E")));
00210 Serial.println(F("20 72 65 6D 6F 76 65 64 0A 55 70 64 61 74 65 64 20 45 41")));
00211 Serial.println(F("50 2D 54 4C 53 20 4D 61 73 74 65 72 20 4B 65 79 20 47 65")));
00212 Serial.println(F("6E 65 72 61 74 69 6F 6E 20 70 65 72 69 6F 64 0A 50 65 72")));
00213 Serial.println(F("66 6F 72 6D 65 64 20 66 61 6C 6C 62 61 63 6B 20 74 6F 20")));
00214 Serial.println(F("73 65 63 6F 6E 64 61 72 79 20 4F 43 53 50 20 73 65 72 76")));
00215 Serial.println(F("65 72 0A 49 53 45 20 68 61 73 20 72 65 66 72 65 73 68 65")));
00216 Serial.println(F("64 20 61 75 74 68 65 6E 74 69 63 61 74 69 6F 6E 20 61 67")));
00217 Serial.println(F("61 69 6E 73 74 20 41 50 49 43 20 73 75 63 63 65 73 73 66")));
00218 Serial.println(F("75 6C 6C 79 0A 52 41 44 49 55 53 20 44 54 4C 53 3A 20 53")));
00219 Serial.println(F("65 6E 74 20 61 6E 20 4F 43 53 50 20 72 65 71 75 65 73 74")));
00220 Serial.println(F("20 74 6F 20 74 68 65 20 70 72 69 6D 61 72 79 20 4F 43 53")));
00221 Serial.println(F("50 20 73 65 72 76 65 72 20 66 6F 72 20 74 68 65 20 43 41")));
00222 Serial.println(F("0A 55 73 65 72 20 6F 72 20 68 6F 73 74 20 64 69 73 61 62")));
00223 Serial.println(F("6C 65 64 20 69 6E 20 63 75 72 72 65 6E 74 20 49 44 53 74")));
00224 Serial.println(F("6F 72 65 20 69 6E 20 61 74 74 72 69 62 75 74 65 20 72 65")));
00225 Serial.println(F("74 72 69 65 76 61 6C 20 6D 6F 64 65 0A 53 6B 69 70 70 69")));
00226 Serial.println(F("6E 67 20 75 6E 75 73 61 62 6C 65 20 64 6F 6D 61 69 6E 0A")));
00227 Serial.println(F("50 72 65 70 61 72 65 64 20 45 41 50 2D 52 65 71 75 65 73")));
00228 Serial.println(F("74 20 77 69 74 68 20 61 6E 6F 74 68 65 72 20 45 41 50 2D")));
00229 Serial.println(F("4D 53 43 48 41 50 20 63 68 61 6C 6C 65 6E 67 65 0A 49 64")));
00230 Serial.println(F("65 6E 74 69 74 79 20 70 6F 6C 69 63 79 20 72 65 73 75 6C")));
00231 Serial.println(F("74 20 69 73 20 63 6F 6E 66 69 67 75 72 65 64 20 66 6F 72")));
00232 Serial.println(F("20 70 61 73 73 77 6F 72 64 20 62 61 73 65 64 20 61 75 74")));
00233 Serial.println(F("68 65 6E 74 69 63 61 74 69 6F 6E 20 6D 65 74 68 6F 64 73")));
00234 Serial.println(F("20 62 75 74 20 72 65 63 65 69 76 65 64 20 63 65 72 74 69")));
00235 Serial.println(F("66 69 63 61 74 65 20 62 61 73 65 64 20 61 75 74 68 65 6E")));
00236 Serial.println(F("74 69 63 61 74 69 6F 6E 20 72 65 71 75 65 73 74 0A 46 61")));
00237 Serial.println(F("69 6C 65 64 20 74 6F 20 66 6F 72 77 61 72 64 20 72 65 71")));
00238 Serial.println(F("75 65 73 74 20 74 6F 20 63 75 72 72 65 6E 74 20 72 65 6D")));
00239 Serial.println(F("6F 74 65 20 52 41 44 49 55 53 20 73 65 72 76 65 72 3B 20")));
00240 Serial.println(F("61 6E 20 69 6E 76 61 6C 69 64 20 72 65 73 70 6F 6E 73 65")));
00241 Serial.println(F("20 77 61 73 20 72 65 63 65 69 76 65 64 0A 55 73 65 72 20")));
00242 Serial.println(F("6C 6F 67 69 6E 20 74 6F 20 49 53 45 20 63 6F 6E 66 69 67")));
00243 Serial.println(F("75 72 61 74 69 6F 6E 20 6D 6F 64 65 20 66 61 69 6C 65 64")));
00244 Serial.println(F("0A 55 6E 61 62 6C 65 20 74 6F 20 66 69 6E 64 20 27 75 73")));
00245 Serial.println(F("65 72 6E 61 6D 65 27 20 61 74 74 72 69 62 75 74 65 20 61")));
00246 Serial.println(F("73 73 65 72 74 69 6F 6E 0A 56 61 6C 69 64 20 69 6E 63 6F")));
00247 Serial.println(F("6D 69 6E 67 20 61 63 63 6F 75 6E 74 69 6E 67 20 72 65 71")));
00248 Serial.println(F("75 65 73 74 0A 41 75 74 68 65 6E 74 69 63 61 74 69 6F 6E")));
00249 Serial.println(F("20 66 61 69 6C 65 64 20 62 65 63 61 75 73 65 20 4E 54 4C")));
00250 Serial.println(F("4D 20 77 61 73 20 62 6C 6F 63 6B 65 64 0A 53 6B 69 70 70")));
00251 Serial.println(F("69 6E 67 20 75 6E 6A 6F 69 6E 65 64 20 64 6F 6D 61 69 6E")));
00252 Serial.println(F("0A 54 68 65 20 75 73 65 72 20 69 73 20 6E 6F 74 20 66 6F")));
00253 Serial.println(F("75 6E 64 20 69 6E 20 74 68 65 20 69 6E 74 65 72 6E 61 6C")));
00254 Serial.println(F("20 67 75 65 73 74 73 20 69 64 65 6E 74 69 74 79 20 73 74")));
00255 Serial.println(F("6F 72 65 0A 43 68 61 6E 67 65 20 70 61 73 73 77 6F 72 64")));
00256 Serial.println(F("20 61 67 61 69 6E 73 74 20 41 63 74 69 76 65 20 44 69 72")));
00257 Serial.println(F("65 63 74 6F 72 79 20 66 61 69 6C 65 64 20 73 69 6E 63 65")));
00258 Serial.println(F("20 75 73 65 72 20 68 61 73 20 61 20 6E 6F 6E 2D 63 6F 6D")));
00259 Serial.println(F("70 6C 69 61 6E 74 20 70 61 73 73 77 6F 72 64 0A 41 70 70")));
00260 Serial.println(F("61 72 65 6E 74 20 6D 69 73 63 6F 6E 66 69 67 75 72 61 74")));
00261 Serial.println(F("69 6F 6E 20 6F 66 20 45 78 74 65 72 6E 61 6C 20 50 6F 6C")));
```



```

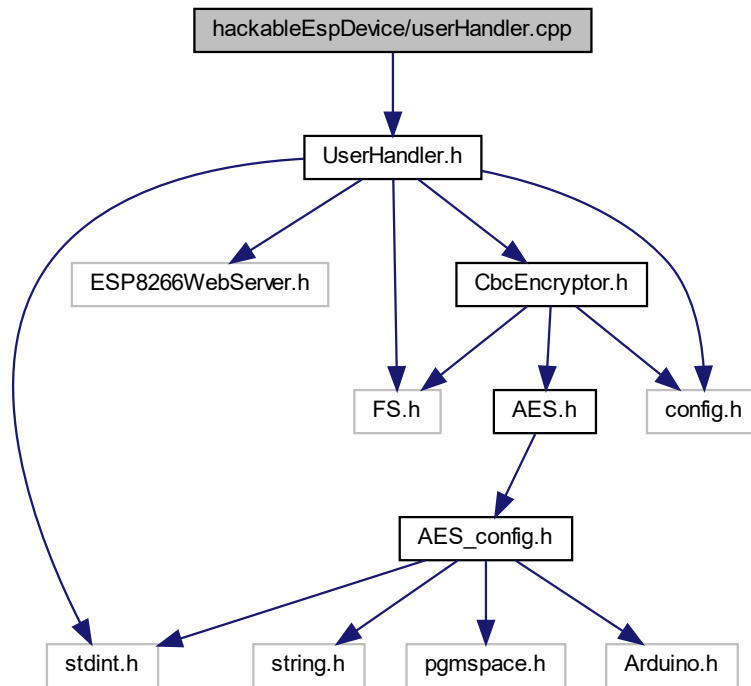
00262     Serial.println(F("69 63 79 20 53 65 72 76 65 72 0A 41 75 74 68 6F 72 69 7A"));
00263     Serial.println(F("61 74 69 6F 6E 20 70 72 6F 66 69 6C 65 2F 73 20 73 70 65"));
00264     Serial.println(F("63 69 66 69 65 64 20 61 72 65 20 6E 6F 74 20 73 75 69 74"));
00265     Serial.println(F("65 64 20 66 6F 72 20 74 68 69 73 20 4E 65 74 77 6F 72 6B"));
00266     Serial.println(F("20 41 63 63 65 73 73 20 44 65 76 69 63 65 0A 52 65 63 65"));
00267     Serial.println(F("69 76 65 64 20 61 20 72 65 61 75 74 68 65 6E 74 69 63 61"));
00268     Serial.println(F("74 65 20 72 65 73 70 6F 6E 73 65 0A 4C 6F 67 67 69 6E 67"));
00269     Serial.println(F("20 63 6F 6D 70 6F 6E 65 6E 74 20 6E 6F 77 20 72 65 61 64"));
00270     Serial.println(F("79 20 74 6F 20 72 65 63 65 69 76 65 20 63 6F 6E 66 69 67"));
00271     Serial.println(F("75 72 61 74 69 6F 6E 20 63 68 61 6E 67 65 73 0A 52 65 74"));
00272     Serial.println(F("75 72 6E 65 64 20 54 41 43 41 43 53 2B 20 41 75 74 68 65"));
00273     Serial.println(F("6E 74 69 63 61 74 69 6F 6E 20 52 65 70 6C 79 0A 45 76 61"));
00274     Serial.println(F("6C 75 61 74 69 6E 67 20 47 72 6F 75 70 20 4D 61 70 70 69"));
00275     Serial.println(F("6E 67 20 50 6F 6C 69 63 79 0A 4C 44 41 50 20 66 65 74 63"));
00276     Serial.println(F("68 20 66 6F 75 6E 64 20 6E 6F 20 6D 61 74 63 68 69 6E 67"));
00277     Serial.println(F("20 61 63 63 6F 75 6E 74 20 69 6E 20 64 6F 6D 61 69 6E 0A"));
00278     Serial.println(F("4D 61 63 68 69 6E 65 20 61 75 74 68 65 6E 74 69 63 61 74"));
00279     Serial.println(F("69 6F 6E 20 61 67 61 69 6E 73 74 20 41 63 74 69 76 65 20"));
00280     Serial.println(F("44 69 72 65 63 74 6F 72 79 20 66 61 69 6C 65 64 20 73 69"));
00281     Serial.println(F("6E 63 65 20 6D 61 63 68 69 6E 65 20 69 73 20 63 6F 6E 73"));
00282     Serial.println(F("69 64 65 72 65 64 20 74 6F 20 62 65 20 69 6E 20 72 65 73"));
00283     Serial.println(F("74 72 69 63 74 65 64 20 6C 6F 67 6F 6E 20 68 6F 75 72 73"));
00284     Serial.println(F("0A 41 73 73 65 72 74 69 6F 6E 20 64 6F 65 73 20 6E 6F 74"));
00285     Serial.println(F("20 63 6F 6E 74 61 69 6E 20 73 75 62 6A 65 63 74 20 63 6F"));
00286     Serial.println(F("6E 66 69 72 6D 61 74 69 6F 6E 0A 55 73 65 72 20 72 65 63"));
00287     Serial.println(F("6F 72 64 20 77 61 73 20 63 61 63 68 65 64 20 69 6E 20 50"));
00288     Serial.println(F("61 73 73 63 6F 64 65 20 63 61 63 68 65 0A 49 64 65 6E 74"));
00289     Serial.println(F("69 74 79 20 72 65 73 6F 6C 75 74 69 6F 6E 20 62 79 20 63"));
00290     Serial.println(F("65 72 74 69 66 69 63 61 74 65 20 66 6F 75 6E 64 20 61 6D"));
00291     Serial.println(F("62 69 67 75 6F 75 73 20 61 63 63 6F 75 6E 74 73 0A 53 74"));
00292     Serial.println(F("61 72 74 75 70 20 43 6F 6D 70 6C 65 74 65 21 2E 2E 2E 2E"));
00293     return true;
00294 }
00295
00296 /*****
00302 /*****
00303 bool printStringInBytes(String str) {
00304     uint8_t messLength = str.length() + 1;
00305     /* Check if string is not too long */
00306     if (messLength > LENGTH) {
00307         return false;
00308     }
00309
00310     unsigned char messBytes[messLength];
00311     str.getBytes(messBytes, messLength);
00312     uint8_t i;
00313
00314     for (i = 0; i < messLength; i++) {
00315         if (messBytes[i] != 0) {
00316             Serial.print(messBytes[i], HEX);
00317             Serial.print(" ");
00318         }
00319     }
00320     Serial.print("0A ");
00321     i++;
00322     /* Print . (2E) until end of line, to match random data */
00323     while (i < LENGTH-1) {
00324         Serial.print("2E ");
00325         i++;
00326     }
00327     Serial.println("2E");
00328     return true;
00329 }
00330 #endif

```

5.31 hackableEspDevice/userHandler.cpp File Reference

```
#include "UserHandler.h"
```

Include dependency graph for userHandler.cpp:



5.32 userHandler.cpp

[Go to the documentation of this file.](#)

```

00001 /*
00002  * File:      UserHandler.cpp
00003  * Author:    Luke de Munk & Twenne Elffers
00004  * Class:     UserHandler
00005  * Version:   1.0
00006  *
00007  * Class for the http authentication process.
00008  */
00009 #include "UserHandler.h"
00010
00011 /*****
00016 *****/
00017 UserHandler::UserHandler(ESP8266WebServer *server) {
00018     _numberUsers = 0;
00019 }
00020
00021 /*****
00025 *****/
00026 void UserHandler::updateUsers() {
00027     /* If there is no file, return 0 users */
00028     if (!SPIFFS.exists(HTTP_CONFIG_LOCATION)) {
00029         _numberUsers = 0;
00030         return;
00031     }
00032
00033     //Decrypt the file before reading
00034     if (!cryptor.decryptFile(HTTP_CONFIG_LOCATION)) {

```

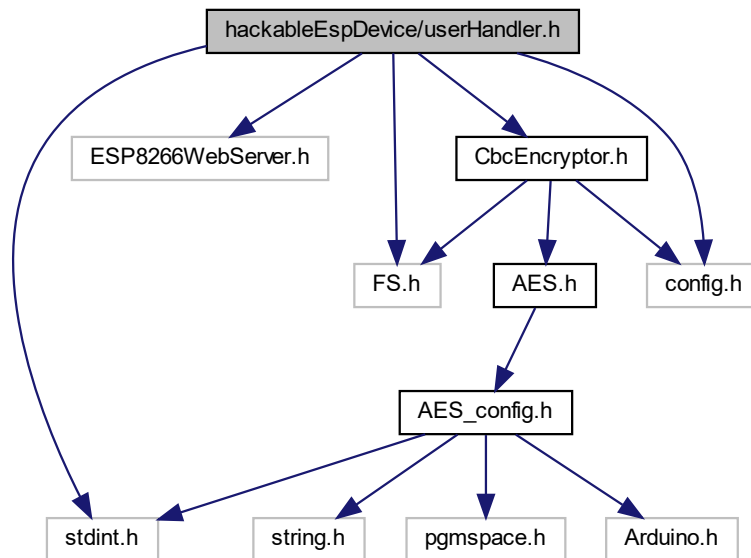
```

00035         return;
00036     }
00037
00038     File configFile = SPIFFS.open(HTTP_CONFIG_LOCATION, "r");
00039
00040     String line;
00041     String* user;
00042     configFile.readStringUntil('\n'); //Ignore first line
00043     (format)
00044     /* Extract user information line by line */
00045     for(uint8_t i = 0; i < MAX_NUMBER_USERS*USER_INFO_LENGTH; i+=USER_INFO_LENGTH) {
00046         line = configFile.readStringUntil('\n'); //Read a line from the
00047         file
00048         if (line != "" && line.indexOf(":") != -1) {
00049             user = _parseLine(line);
00050             _users[i] = user[0].c_str();
00051             _users[i+1] = user[1].c_str();
00052             _users[i+2] = user[2].c_str();
00053         } else {
00054             _numberUsers = i-1;
00055             break;
00056         }
00057         _numberUsers = i-1;
00058     }
00059     configFile.close();
00060
00061     /* Encrypt the file again */
00062     if (!cryptor.encryptFile(HTTP_CONFIG_LOCATION)) {
00063         return;
00064     }
00065 }
00066 /*****
00071 /*****
00072 String* UserHandler::getUsers() {
00073     return _users;
00074 }
00075
00076 /*****
00081 /*****
00082 uint8_t UserHandler::getNumberOfUsers() {
00083     return _numberUsers;
00084 }
00085
00086 /*****
00093 /*****
00094 bool UserHandler::checkPermission(uint8_t permissionLevel, ESP8266WebServer *server) {
00095     bool isLoggedIn = false;
00096     bool hasPermission = false;
00097     uint8_t userIndex = 0;
00098
00099     if (permissionLevel == PERMISSION_LVL_ALL) {
00100         return true;
00101     } else {
00102         for (uint8_t i = 0; i < _numberUsers; i += 3) {
00103             if (server->authenticate(_users[i].c_str(), _users[i+1].c_str())) {
00104                 userIndex = i;
00105                 isLoggedIn = true;
00106                 break;
00107             }
00108         }
00109
00110         if (isLoggedIn && atoi(_users[userIndex+2].c_str()) >= permissionLevel) {
00111             return true;
00112         }
00113     }
00114     return false;
00115 }
00116
00117 /*****
00123 /*****
00124 String* UserHandler::_parseLine(String line) {
00125     static String userInfo[3];
00126
00127     uint8_t indexForUsername = line.indexOf(":"); //gets loc of first ":"
00128     uint8_t indexForPassword = line.indexOf(":", indexForUsername+1); //gets loc of second ":"
00129
00130     userInfo[0] = line.substring(0, indexForUsername); //Selects xxxx from
00131     userInfo[1] = line.substring(indexForUsername+1, indexForPassword); //Selects yyyy from
00132     userInfo[2] = line.substring(indexForPassword+1); //Selects zzzz from
00133     userInfo[3] = line.substring(indexForPassword+1); //Selects zzzz from
00134     return userInfo;
00135 }

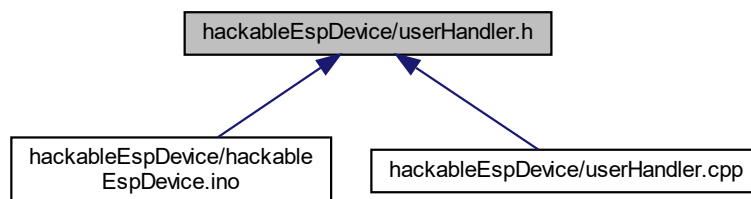
```

5.33 hackableEspDevice/userHandler.h File Reference

```
#include <stdint.h>
#include <ESP8266WebServer.h>
#include <FS.h>
#include "config.h"
#include "CbcEncryptor.h"
Include dependency graph for userHandler.h:
```



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



Classes

- class [UserHandler](#)

5.34 userHandler.h

[Go to the documentation of this file.](#)

```
00001 /*
00002  * File:      UserHandler.h
00003  * Author:    Luke de Munk
00004  * Class:     UserHandler
00005  * Version:   1.0
00006  *
00007  * Class for the http authentication process.
00008  */
00009 #ifndef USER_HANDLER_H
00010 #define USER_HANDLER_H
00011 #include <stdint.h>                                //For defining bits per
    integer
00012 #include <ESP8266WebServer.h>                       //For running the
    webserver
00013 #include <FS.h>                                     //For SPIFFS
00014 #include "config.h"                                //For the configuration
00015 #include "CbcEncryptor.h"                          //For decrypting file to
    obtain users
00016
00017 class UserHandler
00018 {
00019     public:
00020         UserHandler(ESP8266WebServer *server);
00021         void updateUsers();
00022         String* getUsers();
00023         uint8_t getNumberOfUsers();
00024         bool checkPermission(uint8_t permissionLevel, ESP8266WebServer *server);
00025
00026     private:
00027         String* _parseLine(String line);
00028         String _users[MAX_NUMBER_USERS*USER_INFO_LENGTH];
00029         uint8_t _numberUsers;
00030         CbcEncryptor cryptor;
00031 };
00032 #endif
```

5.35 README.md File Reference

Chapter 6

Example Documentation

6.1 aes.ino

For Arduino

Updated: spaniakos 2015

This is an example of how to use [AES](#) in CBC mode easily. The text and keys can be either in HEX or String format.

6.2 aes.cpp

For Raspberry pi

Updated: spaniakos 2015

This is an example of how to use [AES](#) in CBC mode easily. The text and keys can be either in HEX or String format.

6.3 test_vectors.ino

For Arduino

Updated: spaniakos 2015

This is an example of monte carlo test vectors, in order to justify the effectiveness of the algorithm. plus is a classical approach to the [AES](#) encryption library with out the easy to use add-on modifications.

6.4 test_vectors.cpp

For Raspberry pi

Updated: spaniakos 2015

This is an example of monte carlo test vectors, in order to justify the effectiveness of the algorithm. plus is a classical approach to the [AES](#) encryption library with out the easy to use add-on modifications.

Index

- ADDRESS_LENGTH
 - BufferOverflow.h, [48](#)
- AES, [9](#)
 - AES, [10](#)
 - calc_size_n_pad, [10](#)
 - cbc_decrypt, [10](#), [11](#)
 - cbc_encrypt, [11](#)
 - CheckPad, [12](#)
 - clean, [12](#)
 - copy_n_bytes, [12](#)
 - decrypt, [13](#)
 - do_aes_decrypt, [13](#), [14](#)
 - do_aes_encrypt, [15](#)
 - encrypt, [16](#)
 - get_IV, [16](#)
 - get_pad, [16](#)
 - get_size, [17](#)
 - iv_inc, [17](#)
 - padPlaintext, [17](#)
 - printArray, [18](#)
 - set_IV, [18](#)
 - set_key, [20](#)
 - set_size, [20](#)
- AES.cpp
 - d2, [30](#)
 - DPOLY, [30](#)
 - f2, [30](#)
 - WPOLY, [30](#)
- AES_config.h
 - FAILURE, [41](#)
 - KEY_SCHEDULE_BYTES, [41](#)
 - N_BLOCK, [41](#)
 - N_COL, [42](#)
 - N_MAX_ROUNDS, [42](#)
 - N_ROW, [42](#)
 - SUCCESS, [42](#)
- ARG_DEBUG_OFF
 - SerialCommandExecuter.h, [84](#)
- ARG_DEBUG_ON
 - SerialCommandExecuter.h, [84](#)
- ARG_GPG_DECRYPT
 - SerialCommandExecuter.h, [84](#)
- ARG_GPG_ENCRYPT
 - SerialCommandExecuter.h, [85](#)
- ARG_HELP_LONG
 - SerialCommandExecuter.h, [85](#)
- ARG_HELP_SHORT
 - SerialCommandExecuter.h, [85](#)
- ARG_HOSTNAME_DEFAULT
 - SerialCommandExecuter.h, [85](#)
- ARG_HOSTNAME_SET
 - SerialCommandExecuter.h, [85](#)
- ARG_LS_FILE_1_1
 - SerialCommandExecuter.h, [85](#)
- ARG_LS_FILE_1_2
 - SerialCommandExecuter.h, [86](#)
- ARG_LS_FILE_2_1
 - SerialCommandExecuter.h, [86](#)
- ARG_LS_FILE_2_2
 - SerialCommandExecuter.h, [86](#)
- brightness
 - hackableEspDevice.ino, [65](#)
- BufferOverflow, [21](#)
 - BufferOverflow, [21](#)
 - ls, [21](#)
 - objectDump, [21](#)
 - runCProgram, [22](#)
 - vi, [22](#)
- BufferOverflow.h
 - ADDRESS_LENGTH, [48](#)
 - MAX_NUM_CHARS, [49](#)
 - OVERFLOW_BEGIN, [49](#)
 - OVERFLOW_LENGTH, [49](#)
 - RETURN_ADDRESS, [49](#)
- calc_size_n_pad
 - AES, [10](#)
- cbc_decrypt
 - AES, [10](#), [11](#)
- cbc_encrypt
 - AES, [11](#)
- CbcEncryptor, [22](#)
 - CbcEncryptor, [23](#)
 - decryptFile, [23](#)
 - decryptLine, [24](#)
 - encryptFile, [24](#)
 - encryptLine, [24](#)
 - setKey, [25](#)
- checkEepromCommit
 - HostnameWrite.cpp, [72](#)
 - HostnameWrite.h, [75](#)
- CheckPad
 - AES, [12](#)
- checkPermission
 - UserHandler, [27](#)
- clean
 - AES, [12](#)
- cliExecuter

- hackableEspDevice.ino, [65](#)
- COMMAND_DEBUG
 - SerialCommandExecuter.h, [86](#)
- COMMAND_GPG
 - SerialCommandExecuter.h, [86](#)
- COMMAND_HELP
 - SerialCommandExecuter.h, [86](#)
- COMMAND_HOSTNAME
 - SerialCommandExecuter.h, [87](#)
- COMMAND_KEYS
 - SerialCommandExecuter.h, [87](#)
- COMMAND_LS
 - SerialCommandExecuter.h, [87](#)
- COMMAND_OBJDUMP
 - SerialCommandExecuter.h, [87](#)
- COMMAND_RESTART
 - SerialCommandExecuter.h, [87](#)
- COMMAND_RUN
 - SerialCommandExecuter.h, [87](#)
- COMMAND_SU
 - SerialCommandExecuter.h, [88](#)
- COMMAND_USERS
 - SerialCommandExecuter.h, [88](#)
- COMMAND_VI
 - SerialCommandExecuter.h, [88](#)
- COMMAND_WHOAMI
 - SerialCommandExecuter.h, [88](#)
- copy_n_bytes
 - AES, [12](#)
- d2
 - AES.cpp, [30](#)
- debug
 - Debugger.cpp, [56](#)
 - Debugger.h, [59](#)
- Debugger.cpp
 - debug, [56](#)
 - debugIn, [56](#)
 - getDebugEnabled, [56](#)
 - setDebugEnabled, [57](#)
- Debugger.h
 - debug, [59](#)
 - debugIn, [59](#)
 - ENABLE_DEBUG_FLAG_ADDRESS, [59](#)
 - getDebugEnabled, [60](#)
 - setDebugEnabled, [60](#)
- debugIn
 - Debugger.cpp, [56](#)
 - Debugger.h, [59](#)
- decrypt
 - AES, [13](#)
- decryptFile
 - CbcEncryptor, [23](#)
- decryptLine
 - CbcEncryptor, [24](#)
- do_aes_decrypt
 - AES, [13](#), [14](#)
- do_aes_encrypt
 - AES, [15](#)
- DPOLY
 - AES.cpp, [30](#)
- ENABLE_DEBUG_FLAG_ADDRESS
 - Debugger.h, [59](#)
- encrypt
 - AES, [16](#)
- encryptFile
 - CbcEncryptor, [24](#)
- encryptLine
 - CbcEncryptor, [24](#)
- ERROR_CMD_NOT_FOUND
 - SerialCommandExecuter.h, [88](#)
- ERROR_NO_FILE
 - SerialCommandExecuter.h, [88](#)
- ERROR_NO_FILE_DIR
 - SerialCommandExecuter.h, [89](#)
- ERROR_NO_PERMISSION
 - SerialCommandExecuter.h, [89](#)
- ERROR_NO_VALID_KEY
 - SerialCommandExecuter.h, [89](#)
- ERROR_PERM_DENIED
 - SerialCommandExecuter.h, [89](#)
- ERROR_TOO_FEW_ARGS
 - SerialCommandExecuter.h, [89](#)
- ERROR_TOO_MANY_ARGS
 - SerialCommandExecuter.h, [89](#)
- ERROR_WRONG_ARGS
 - SerialCommandExecuter.h, [90](#)
- ERROR_WRONG_PWD
 - SerialCommandExecuter.h, [90](#)
- executeCommand
 - SerialCommandExecuter, [26](#)
- f2
 - AES.cpp, [30](#)
- FAILURE
 - AES_config.h, [41](#)
- fsUploadFile
 - hackableEspDevice.ino, [66](#)
- get_IV
 - AES, [16](#)
- get_pad
 - AES, [16](#)
- get_size
 - AES, [17](#)
- getContentType
 - hackableEspDevice.ino, [63](#)
- getDebugEnabled
 - Debugger.cpp, [56](#)
 - Debugger.h, [60](#)
- getHostname
 - HostnameWrite.cpp, [72](#)
 - HostnameWrite.h, [75](#)
- getNumberOfUsers
 - UserHandler, [28](#)
- getUsers
 - UserHandler, [28](#)

- hackableEspDevice.ino
 - brightness, 65
 - cliExecuter, 65
 - fsUploadFile, 66
 - getContentType, 63
 - handleFileDownload, 63
 - handleFileRequest, 63
 - handleFileUpload, 64
 - initializeHostname, 64
 - initializeServer, 64
 - ledState, 66
 - loop, 64
 - MIN_BRIGHTNESS, 62
 - OFF, 62
 - ON, 62
 - sendToFrontend, 64
 - server, 65, 66
 - setup, 65
 - setupWifi, 65
- hackableEspDevice/AES.cpp, 29, 31
- hackableEspDevice/AES.h, 38
- hackableEspDevice/AES_config.h, 40, 43
- hackableEspDevice/BufferOverflow.cpp, 44
- hackableEspDevice/BufferOverflow.h, 47, 50
- hackableEspDevice/CbcEncryptor.cpp, 51
- hackableEspDevice/CbcEncryptor.h, 54
- hackableEspDevice/Debugger.cpp, 55, 57
- hackableEspDevice/Debugger.h, 58, 60
- hackableEspDevice/hackableEspDevice.ino, 61, 66
- hackableEspDevice/HostnameWrite.cpp, 71, 73
- hackableEspDevice/HostnameWrite.h, 74, 76
- hackableEspDevice/SerialCommandExecuter.cpp, 77
- hackableEspDevice/SerialCommandExecuter.h, 82, 91
- hackableEspDevice/StartupText.h, 92, 94
- hackableEspDevice/userHandler.cpp, 98
- hackableEspDevice/userHandler.h, 100, 101
- handleFileDownload
 - hackableEspDevice.ino, 63
- handleFileRequest
 - hackableEspDevice.ino, 63
- handleFileUpload
 - hackableEspDevice.ino, 64
- HostnameWrite.cpp
 - checkEepromCommit, 72
 - getHostname, 72
 - setEepromToNull, 72
 - writeHostname, 73
- HostnameWrite.h
 - checkEepromCommit, 75
 - getHostname, 75
 - setEepromToNull, 75
 - writeHostname, 76
- initializeHostname
 - hackableEspDevice.ino, 64
- initializeServer
 - hackableEspDevice.ino, 64
- iv_inc
 - AES, 17
- KEY_SCHEDULE_BYTES
 - AES_config.h, 41
- ledState
 - hackableEspDevice.ino, 66
- LENGTH
 - StartupText.h, 92
- loop
 - hackableEspDevice.ino, 64
- ls
 - BufferOverflow, 21
- MAX_NUM_CHARS
 - BufferOverflow.h, 49
- MAX_NUMBER_PARAMS
 - SerialCommandExecuter.h, 90
- MESS_SUPER_USER
 - SerialCommandExecuter.h, 90
- MIN_BRIGHTNESS
 - hackableEspDevice.ino, 62
- N_BLOCK
 - AES_config.h, 41
- N_COL
 - AES_config.h, 42
- N_MAX_ROUNDS
 - AES_config.h, 42
- N_ROW
 - AES_config.h, 42
- objectDump
 - BufferOverflow, 21
- OFF
 - hackableEspDevice.ino, 62
- ON
 - hackableEspDevice.ino, 62
- OVERFLOW_BEGIN
 - BufferOverflow.h, 49
- OVERFLOW_LENGTH
 - BufferOverflow.h, 49
- padPlaintext
 - AES, 17
- printArray
 - AES, 18
- printStartupText
 - StartupText.h, 93
- printStringInBytes
 - StartupText.h, 93
- README.md, 101
- RETURN_ADDRESS
 - BufferOverflow.h, 49
- runCProgram
 - BufferOverflow, 22
- sendToFrontend
 - hackableEspDevice.ino, 64
- SerialCommandExecuter, 25
 - executeCommand, 26

- SerialCommandExecuter, [26](#)
- setUsers, [26](#)
- SerialCommandExecuter.h
 - ARG_DEBUG_OFF, [84](#)
 - ARG_DEBUG_ON, [84](#)
 - ARG_GPG_DECRYPT, [84](#)
 - ARG_GPG_ENCRYPT, [85](#)
 - ARG_HELP_LONG, [85](#)
 - ARG_HELP_SHORT, [85](#)
 - ARG_HOSTNAME_DEFAULT, [85](#)
 - ARG_HOSTNAME_SET, [85](#)
 - ARG_LS_FILE_1_1, [85](#)
 - ARG_LS_FILE_1_2, [86](#)
 - ARG_LS_FILE_2_1, [86](#)
 - ARG_LS_FILE_2_2, [86](#)
 - COMMAND_DEBUG, [86](#)
 - COMMAND_GPG, [86](#)
 - COMMAND_HELP, [86](#)
 - COMMAND_HOSTNAME, [87](#)
 - COMMAND_KEYS, [87](#)
 - COMMAND_LS, [87](#)
 - COMMAND_OBJDUMP, [87](#)
 - COMMAND_RESTART, [87](#)
 - COMMAND_RUN, [87](#)
 - COMMAND_SU, [88](#)
 - COMMAND_USERS, [88](#)
 - COMMAND_VI, [88](#)
 - COMMAND_WHOAMI, [88](#)
 - ERROR_CMD_NOT_FOUND, [88](#)
 - ERROR_NO_FILE, [88](#)
 - ERROR_NO_FILE_DIR, [89](#)
 - ERROR_NO_PERMISSION, [89](#)
 - ERROR_NO_VALID_KEY, [89](#)
 - ERROR_PERM_DENIED, [89](#)
 - ERROR_TOO_FEW_ARGS, [89](#)
 - ERROR_TOO_MANY_ARGS, [89](#)
 - ERROR_WRONG_ARGS, [90](#)
 - ERROR_WRONG_PWD, [90](#)
 - MAX_NUMBER_PARAMS, [90](#)
 - MESS_SUPER_USER, [90](#)
- server
 - hackableEspDevice.ino, [65](#), [66](#)
- set_IV
 - AES, [18](#)
- set_key
 - AES, [20](#)
- set_size
 - AES, [20](#)
- setDebugEnabled
 - Debugger.cpp, [57](#)
 - Debugger.h, [60](#)
- setEepromToNull
 - HostnameWrite.cpp, [72](#)
 - HostnameWrite.h, [75](#)
- setKey
 - CbcEncryptor, [25](#)
- setup
 - hackableEspDevice.ino, [65](#)
- setupWifi
 - hackableEspDevice.ino, [65](#)
- setUsers
 - SerialCommandExecuter, [26](#)
- StartupText.h
 - LENGTH, [92](#)
 - printStartupText, [93](#)
 - printStringInBytes, [93](#)
- SUCCESS
 - AES_config.h, [42](#)
- updateUsers
 - UserHandler, [28](#)
- UserHandler, [27](#)
 - checkPermission, [27](#)
 - getNumberOfUsers, [28](#)
 - getUsers, [28](#)
 - updateUsers, [28](#)
 - UserHandler, [27](#)
- vi
 - BufferOverflow, [22](#)
- WPOLY
 - AES.cpp, [30](#)
- writeHostname
 - HostnameWrite.cpp, [73](#)
 - HostnameWrite.h, [76](#)