EHN Group 12 Practical 2

Generated by Doxygen 1.8.13

Contents

Chapter 1

EHN 410 Group 12 Practical 2

This software was developed by EHN 410 group 12 and is a tool that can be used to encrypt or decrypt data using the super-secure Advanced Encryption Standard (AES) algorithm implemented from first principles.

The main features are:

- AES 128, AES 192 and AES 256 support.
- Cipher Block Chaining support.
- Cipher Feedback support for a stream of 8, 64 and 128-bits.
- Text and file input/output support.
- Step-by-step verbose mode.

Compilation

Use the standard **gcc** compiler available on most builds of **Linux**.

This command should be run in a terminal window in the same folder as **AES.c** and **AES.h**:

```
$ gcc AES.c -o AES
```

Only standard libraries are used, so no packages need to be installed separately.

Compilation was tested on **Linux Ubuntu 18.04.4 LTS**.

Usage

A command in the following format should be run in a terminal window in the same folder where the executable is located:

```
$ ./AES -arg1 value1 -arg2 value2...
```

The following arguments should then be given in this order:

```
-e (encryption), or
-d (decryption)

-cbc <len> (Cipher Block Chaining, <len> either 128, 192 or 256), or
-cfb <len> (Cipher Feedback, <len> either 128, 192 or 256)

-t <text to encrypt in ASCII or text to decrypt in HEX>, or
-fi <input file> and
-fo <output file>

-key <password in ASCII>

-iv <initialization vector in ASCII>

-streamlen <len> (length of the CFB stream if '-cfb' is given, either 8, 64 or 128)

-h help (will show this message)

-verbose (will show all steps in the AES process)
```

These arguments **are** required:

- The operation (**-e** or **-d**).
- The chaining mode (**-cbc** or **-cfb**) and the corresponding AES width.
- The input (**-t** or **-fi**).
- The user key (**-key**).

These arguments **are not** required:

- The output file (**-fo**) (default value of "encrypted.enc" or "decrypted.txt" will be used if not specified).
- The initialization vector (**-iv**) (will be set to all zeroes if not specified).
- The CFB stream length (**-streamlen**) (will be set to 128-bits if not specified).
- The help screen (**-h**).
- The verbose mode (**-verbose**).

Attention: please take special note of the following:

- Remember to add **"double quotes"** to **ASCII** inputs if **spaces** are present in the string.
  If this is **not** done, only the **first word** in the string will be processed.
- The expected input length for the **-key** argument is **16** characters for AES 128, **24** characters for AES 192 and **32** characters for AES256.
  If an ASCII string with **less** characters are given, the key will be **padded with zeroes** at the end. If an ASCII string with **more** characters are given, the **trailing characters** will be **discarded**.
- The expected input length for the **-iv** argument is **16** characters.
  The **same** rules for the **-key** argument apply here.

Example usage

Example 1

The following command will **encrypt** a **file** called **"input.txt"** (in the same folder) using **AES 128** in **Cipher Block Chaining** mode:

```
$ ./AES -e -cbc 128 -fi "input.txt" -fo "encrypted.enc" -key "Very strong password" -iv "Initialization vecto
```

The following output is expected:

```
Encryption selected
AES128 with CBC selected
Plaintext file input: "input.txt"
Key (ASCII): "Very strong pass"
Initialization Vector (ASCII): "Initialization v"


Encryption in process...

Encrypted file output: "CBC output/encrypted.enc"
```

The file "encrypted.enc" can be found in the folder "CBC output" located in the same folder as the executable.

If the output folder does not exist, the program will attempt to create it.

If the program does not have sufficient permissions to create folders, the file will be found in the same folder as the executable.

Example 2

The following command will **decrypt** a **file** called **"encrypted.jpg"** using **AES 192** in **Cipher Feedback** mode with a stream length of **64-bits**:

```
$ ./AES -d -cfb 192 -fi "encrypted.jpg" -fo "image.jpg" -key "Very strong password" -iv "Initialization vecto
```

The following output is expected:

```
Decryption selected
AES192 with CFB selected
Encrypted file input: "encrypted.jpg"
Key (ASCII): "Very strong password"
Initialization Vector (ASCII): "Initialization v"
64-bit CFB selected


Decryption in process...

Plaintext file output: "CFB output/image.jpg"
```

The file "image.jpg" can be found in the folder "CFB output" located in the same folder as the executable.

The same conditions mentioned in Example 1 apply here.

Example 3

The following command will **encrypt** the **ASCII** string **"Text to encrypt"** using **AES 256** in **Cipher Block Chaining** mode:

```
$ ./AES -e -cbc 256 -t "Text to encrypt" -key "Very strong password" -iv "Initialization vector"
```

The following output is expected:

```
 Encryption selected
 AES256 with CBC selected
 Plaintext message (ASCII): "Text to encrypt"
 Key (ASCII): "Very strong password"
 Initialization Vector (ASCII): "Initialization v"


 Encryption in process...

 Encrypted (HEX):
 CCBD19AB3022404EFDC9804AD802936B
```

Example 4

The following command will **decrypt** the **HEX** string **C7D3CAAFEE6137** using **AES 128** in **Cipher Feedback** mode with a stream length of **8-bits**:

```
$ ./AES -d -cfb 128 -t C7D3CAAFEE6137 -key "Very strong password" -iv "Initialization vector" -streamlen 8
```

The following output is expected:

```
Decryption selected
AES128 with CFB selected
Encrypted message (HEX): C7D3CAAFEE6137
Key (ASCII): "Very strong pass"
Initialization Vector (ASCII): "Initialization v"
8-bit CFB selected


Decryption in process...

Decrypted (ASCII):
"Success"
```

Example 5

The following command will **encrypt** the **ASCII** string **"Test"** using **AES 128** in **Cipher Block Chaining** mode with **verbose output**:

```
$ ./AES -e -cbc 128 -t "Verbose" -key "Very strong password" -verbose
```

The following output is expected:

```
Encryption selected
AES128 with CBC selected
Plaintext message (ASCII): "Test"
Key (ASCII): "Very strong pass"

Verbose mode activated
All steps in the AES process will now be shown

The initialization vector was not set, setting to all zeroes


Encryption in process...




********Block 1:********

~~~~AES encrypt input block:~~~~
54 00 00 00
65 00 00 00
73 00 00 00
74 00 00 00

Add round key (initial):
02 20 6F 70
00 73 6E 61
01 74 67 73
0D 72 20 73


----Round 1:----
Substitute bytes step:
77 B7 A8 51
63 8F 9F EF
7C 92 85 8F
D7 40 B7 8F

Shift rows step:
77 B7 A8 51
8F 9F EF 63
85 8F 7C 92
8F D7 40 B7

Mix columns step:
6E 97 5D 22
69 CF A9 8D
63 4F 7F CF
96 67 F0 77

Add round key step:
D6 0F AA A5
83 56 5E 1B
9E C6 91 52
BE 3D 8A 7E


----Round 2:----
Substitute bytes step:
F6 76 AC 06
EC B1 58 AF
0B B4 81 00
AE 27 7E F3

Shift rows step:
F6 76 AC 06
B1 58 AF EC
81 00 0B B4
F3 AE 27 7E

Mix columns step:
4D AA 85 E9
E4 68 D3 7C
```

```
50 C7 7C 1B
CC 85 05 AE

Add round key step:
67 18 C0 2B
50 45 09 30
AC B2 E7 1D
F3 E0 1A B8
```

∼output omitted∼

```
----Last round:----
Substitute bytes step:
AF 7B 8E 07
0B FB AD B3
FD D5 E1 E5
50 58 0B B1

Shift rows step:
AF 7B 8E 07
FB AD B3 0B
E1 E5 FD D5
B1 50 58 0B

No mix columns step in the last round

Add round key step:
2C 4F 1C 96
7F 20 BF BA
54 A3 56 71
76 ED F5 6B



********Expanded key:********
56 65 72 79 20 73 74 72 6F 6E 67 20 70 61 73 73
B8 EA FD 28 98 99 89 5A F7 F7 EE 7A 87 96 9D 09
2A B4 FC 3F B2 2D 75 65 45 DA 9B 1F C2 4C 06 16
07 DB BB 1A B5 F6 CE 7F F0 2C 55 60 32 60 53 76
DF 36 83 39 6A C0 4D 46 9A EC 18 26 A8 8C 4B 50
AB 85 D0 FB C1 45 9D BD 5B A9 85 9B F3 25 CE CB
B4 0E CF F6 75 4B 52 4B 2E E2 D7 D0 DD C7 19 1B
32 DA 60 37 47 91 32 7C 69 73 E5 AC B4 B4 FC B7
3F 6A C9 BA 78 FB FB C6 11 88 1E 6A A5 3C E2 DD
CF F2 08 BC B7 09 F3 7A A6 81 ED 10 03 BD 0F CD
83 84 B5 C7 34 8D 46 BD 92 0C AB AD 91 B1 A4 60

Encrypted (HEX):
2C7F54764F20A3ED1CBF56F596BA716B
```

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

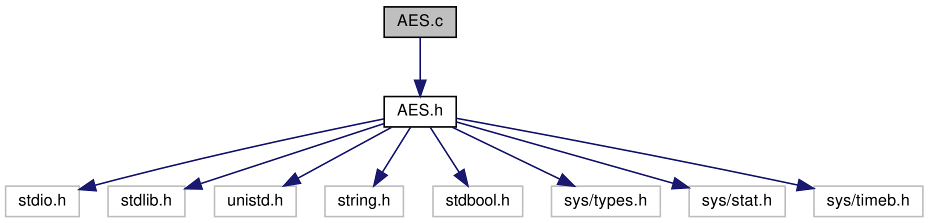
Chapter 3

File Documentation

3.1 AES.c File Reference

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

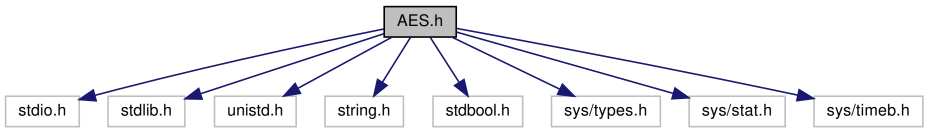
Include dependency graph for AES.c:



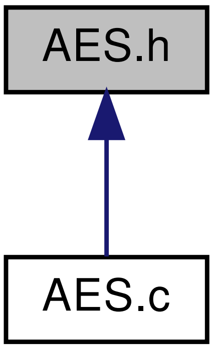
3.2 AES.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <stdbool.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/timeb.h>
```

Include dependency graph for AES.h:



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



Macros

- #define MAX_REQ_LEN 104857600

  The maximum length of an input to be hadled.
- #define VERBOSE 1

  Activate or deactivate verbose mode capabilities.
- #define AES128 0
- #define AES192 1
- #define AES256 2
- #define AES128_ROUNDS 10
- #define AES192_ROUNDS 12
- #define AES256_ROUNDS 14
- #define AES128_KEY_SIZE 176
- #define AES192_KEY_SIZE 208
- #define AES256_KEY_SIZE 240
- #define AES128_USER_KEY_SIZE 16
- #define AES192_USER_KEY_SIZE 24
- #define AES256_USER_KEY_SIZE 32
- #define AES128_EXPANSION 10
- #define AES192_EXPANSION 8
- #define AES256_EXPANSION 7
- #define AES128_SUB_EXPANSION 3
- #define AES192_SUB_EXPANSION 5
- #define AES256_SUB_EXPANSION 7
- #define CFB8 1
- #define CFB64 8
- #define CFB128 16

Functions

- int main (int argc, char ∗argv[ ])
- void char_blockify (unsigned char message[ ], int current_block[4][4], int start_pos)
- void int_blockify (int message[16], int current_block[4][4])
- void print_word (int word[ ], int length)
- void print_block (int current_block[4][4])
- void print_expanded_key (int width, int expanded_key[ ])
- void print_hex_string (unsigned char hex_string[ ], int message_len)

- void write_to_file (char filename[ ], unsigned char message[ ], int message_len)
- void char_unblockify (unsigned char message[ ], int current_block[4][4], int start_pos)
- void AES_word_rotate (int word[ ], int length, int rotations, bool inverse)
- int AES_s_box_transform (int input, bool inverse)
- void AES_key_scheduler (int word[4], int rcon)
- int AES_exp_2 (int previous)
- void AES_key_expansion (int width, int expanded_key[ ], int user_key[ ])
- void AES_sub_bytes (int current_block[4][4], bool inverse)
- void AES_shift_rows (int current_block[4][4], bool inverse)
- int AES_dot_product (int a, int b)
- void AES_mix_cols (int current_block[4][4], bool inverse)
- void AES_add_round_key (int current_block[4][4], int expanded_key[ ], int key_index)
- bool AES_encrypt (int width, int current_block[4][4], int expanded_key[ ])
- bool AES_decrypt (int width, int current_block[4][4], int expanded_key[ ])
- bool CBC_encrypt (int width, unsigned char message[ ], int message_len, int IV[16], int user_key[ ])
- bool CBC_decrypt (int width, unsigned char message[ ], int message_len, int IV[16], int user_key[ ])
- bool CFB_encrypt (int width, unsigned char message[ ], int message_len, int CFB_len, int IV[16], int user_↩ key[ ])
- bool CFB_decrypt (int width, unsigned char message[ ], int message_len, int CF_Blen, int IV[16], int user_↩ key[ ])
- bool AES_encrypt_verbose (int width, int current_block[4][4], int expanded_key[ ])
- bool AES_decrypt_verbose (int width, int current_block[4][4], int expanded_key[ ])
- bool CBC_encrypt_verbose (int width, unsigned char message[ ], int message_len, int IV[16], int user_key[ ])
- bool CBC_decrypt_verbose (int width, unsigned char message[ ], int message_len, int IV[16], int user_key[ ])
- bool CFB_encrypt_verbose (int width, unsigned char message[ ], int message_len, int CFB_len, int IV[16], int user_key[ ])
- bool CFB_decrypt_verbose (int width, unsigned char message[ ], int message_len, int CF_Blen, int IV[16], int user_key[ ])
- int hex_convert (char hex_string[ ], int length)
- void test_functionality ()

Variables

- const int S_BOX [2][16][16]

  Provides a one-to-one mapping for the non-linear substitution of a byte.
- const int PRIME_MATRIX [2][4][4]

  Used for the transformation of a column in the mix columns operation.

3.2.1 Macro Definition Documentation

3.2.1.1 AES128

```
#define AES128 0
```

3.2.1.2 AES128_EXPANSION

```
#define AES128_EXPANSION 10
```

3.2.1.3 AES128_KEY_SIZE

```
#define AES128_KEY_SIZE 176
```

3.2.1.4 AES128_ROUNDS

```
#define AES128_ROUNDS 10
```

3.2.1.5 AES128_SUB_EXPANSION

```
#define AES128_SUB_EXPANSION 3
```

3.2.1.6 AES128_USER_KEY_SIZE

```
#define AES128_USER_KEY_SIZE 16
```

3.2.1.7 AES192

```
#define AES192 1
```

3.2.1.8 AES192_EXPANSION

```
#define AES192_EXPANSION 8
```

3.2.1.9 AES192_KEY_SIZE

```
#define AES192_KEY_SIZE 208
```

3.2.1.10 AES192_ROUNDS

```
#define AES192_ROUNDS 12
```

3.2.1.11 AES192_SUB_EXPANSION

```
#define AES192_SUB_EXPANSION 5
```

3.2.1.12 AES192_USER_KEY_SIZE

```
#define AES192_USER_KEY_SIZE 24
```

3.2.1.13 AES256

```
#define AES256 2
```

3.2.1.14 AES256_EXPANSION

```
#define AES256_EXPANSION 7
```

3.2.1.15 AES256_KEY_SIZE

```
#define AES256_KEY_SIZE 240
```

3.2.1.16 AES256_ROUNDS

```
#define AES256_ROUNDS 14
```

3.2.1.17 AES256_SUB_EXPANSION

```
#define AES256_SUB_EXPANSION 7
```

3.2.1.18 AES256_USER_KEY_SIZE

```
#define AES256_USER_KEY_SIZE 32
```

3.2.1.19 CFB128

```
#define CFB128 16
```

3.2.1.20 CFB64

```
#define CFB64 8
```

3.2.1.21 CFB8

```
#define CFB8 1
```

3.2.1.22 MAX_REQ_LEN

```
#define MAX_REQ_LEN 104857600
```

The maximum length of an input to be hadled.

3.2.1.23 VERBOSE

```
#define VERBOSE 1
```

Activate or deactivate verbose mode capabilities.

3.2.2 Function Documentation

3.2.2.1 AES_add_round_key()

```
void AES_add_round_key (
             int current_block[4][4],
             int expanded_key[],
             int key_index )
```

XOR a block with the expanded key at a certain index

Parameters

current_block	The block to which the round key should be added, also the output.
expanded_key	The expanded key to use.
key_index	The index in the key to start from.

3.2.2.2 AES_decrypt()

```
bool AES_decrypt (
             int width,
             int current_block[4][4],
             int expanded_key[] )
```

The AES decryption algorithm.

Parameters

width	Use the macros AES128, AES192 or AES256 to select which width to use.
current_block	The block to be decrypted, also the output.
expanded_key	The expanded key to be used.

Returns

Successful execution.

3.2.2.3 AES_decrypt_verbose()

```
bool AES_decrypt_verbose (
             int width,
             int current_block[4][4],
             int expanded_key[] )
```

3.2.2.4 AES_dot_product()

```
int AES_dot_product (
             int a,
             int b )
```

Finite field multiplication.

Parameters

a	The first value.
b	The second value.

Returns

The result of the dot product.

3.2.2.5 AES_encrypt()

```
bool AES_encrypt (
            int width,
            int current_block[4][4],
            int expanded_key[] )
```

The AES encryption algorithm.

Parameters

width	Use the macros AES128, AES192 or AES256 to select which width to use.
current_block	The block to be encrypted, also the output.
expanded_key	The expanded key to be used.

Returns

Successful execution.

3.2.2.6 AES_encrypt_verbose()

```
bool AES_encrypt_verbose (
            int width,
            int current_block[4][4],
            int expanded_key[] )
```

3.2.2.7 AES_exp_2()

```
int AES_exp_2 (
            int previous )
```

Exponentiation of 2, double the previous value except when 0x80 and max value of 0xFF.

Parameters

previous	The value to be used exponentiated.

Returns

The exponentiated value.

3.2.2.8 AES_key_expansion()

```
void AES_key_expansion (
             int width,
             int expanded_key[ ],
             int user_key[ ] )
```

Main key expansion function.

Parameters

width	Use the macros AES128, AES192 or AES256 to select which width to use.
expanded_key	The expanded key output, the correct length array (AESxxx_KEY_SIZE + 32) must exist and be passed in here. Allocate more space since AES_key_expansion deliberately writes out of bounds.
user_key	The user key to be expanded.

3.2.2.9 AES_key_scheduler()

```
void AES_key_scheduler (
             int word[4],
             int rcon )
```

Core key operation, transform of previous 4 bytes.

Parameters

word	The bytes to be transformed, also the output.
rcon	The round constant to be used.

3.2.2.10 AES_mix_cols()

```
void AES_mix_cols (
             int current_block[4][4],
             bool inverse )
```

Perform the dot product of the block and the prime matrix.

Parameters

current_block	The block to be used in the dot product, also the output.
inverse	Perform the inverse dot product if true.

3.2.2.11 AES_s_box_transform()

```
int AES_s_box_transform (
             int input,
             bool inverse )
```

Divide value up into its MSB and LSB Nibble and return the s_box value.

Parameters

input	The value to be transformed.
inverse	Perform the inverse transform if true.

Returns

The transformed value.

3.2.2.12 AES_shift_rows()

```
void AES_shift_rows (
             int current_block[4][4],
             bool inverse )
```

The AES row shifting function.

Parameters

current_block	The block to be shifted, also the output.
inverse	Perform the inverse shift if true.

3.2.2.13 AES_sub_bytes()

```
void AES_sub_bytes (
             int current_block[4][4],
             bool inverse )
```

Substitute a block through the S-transform.

Parameters

current_block	The block to be transformed, also the output.
inverse	Perform the inverse transform if true.

3.2.2.14 AES_word_rotate()

```
void AES_word_rotate (
             int word[],
             int length,
             int rotations,
             bool inverse )
```

Shift last items in an array to the front or vice-versa.

Parameters

word	The array to be rotated, also the output.
length	The length of the word.
rotations	The number of rotations to perform.
inverse	Rotate in the opposite direction if true.

3.2.2.15 CBC_decrypt()

```
bool CBC_decrypt (
             int width,
             unsigned char message[],
             int message_len,
             int IV[16],
             int user_key[] )
```

The Cipher Block Chaining decryption algorithm.

Parameters

width	Use the macros AES128, AES192 or AES256 to select which width to use.
message	The message to be decrypted, also the output.
message_len	The length of the message.
IV	The initialization vector to be used.
user_key	The user key to be used.

Returns

Successful execution.

3.2.2.16 CBC_decrypt_verbose()

```
bool CBC_decrypt_verbose (
             int width,
             unsigned char message[],
             int message_len,
             int IV[16],
             int user_key[] )
```

3.2.2.17 CBC_encrypt()

```
bool CBC_encrypt (
             int width,
             unsigned char message[],
             int message_len,
             int IV[16],
             int user_key[] )
```

The Cipher Block Chaining encryption algorithm.

Parameters

width	Use the macros AES128, AES192 or AES256 to select which width to use.
message	The message to be encrypted, also the output.
message_len	The length of the message.
IV	The initialization vector to be used.
user_key	The user key to be used.

Returns

Successful execution.

3.2.2.18 CBC_encrypt_verbose()

```
bool CBC_encrypt_verbose (
             int width,
             unsigned char message[],
             int message_len,
             int IV[16],
             int user_key[] )
```

3.2.2.19 CFB_decrypt()

```
bool CFB_decrypt (
             int width,
             unsigned char message[],
             int message_len,
             int CF_Blen,
             int IV[16],
             int user_key[] )
```

The Cipher Feedback decryption algorithm.

Parameters

width	Use the macros AES128, AES192 or AES256 to select which width to use.
message	The stream to be decrypted, also the output.
message_len	The length of the message.
CFB_len	The length of the chain to use.
IV	The initialization vector to be used.
user_key	The user key to be used.

Returns

Successful execution.

3.2.2.20 CFB_decrypt_verbose()

```
bool CFB_decrypt_verbose (
             int width,
             unsigned char message[],
             int message_len,
             int CF_Blen,
             int IV[16],
             int user_key[] )
```

3.2.2.21 CFB_encrypt()

```
bool CFB_encrypt (
             int width,
             unsigned char message[],
             int message_len,
             int CFB_len,
             int IV[16],
             int user_key[] )
```

The Cipher Feedback encryption algorithm.

Parameters

width	Use the macros AES128, AES192 or AES256 to select which width to use.
message	The stream to be encrypted, also the output.
message_len	The length of the message.
CFB_len	The length of the chain to use.
IV	The initialization vector to be used.
user_key	The user key to be used.

Returns

Successful execution.

3.2.2.22 CFB_encrypt_verbose()

```
bool CFB_encrypt_verbose (
             int width,
             unsigned char message[],
             int message_len,
             int CFB_len,
             int IV[16],
             int user_key[] )
```

3.2.2.23 char_blockify()

```
void char_blockify (
             unsigned char message[],
             int current_block[4][4],
             int start_pos )
```

Convert a char array to 4x4 block of hex.

Parameters

message	A c-string containing the message to be converted.
current_block	The output as a 4x4 integer array.
start_pos	The position from which to start the conversion in the string.

3.2.2.24 char_unblockify()

```
void char_unblockify (
             unsigned char message[],
             int current_block[4][4],
             int start_pos )
```

Convert block back to c-string.

Parameters

message	The output array, must exist before being passed in.
current_block	The block to be converted.
start_pos	The position to start converting in the output.

3.2.2.25 hex_convert()

```
int hex_convert (
            char hex_string[],
            int length )
```

3.2.2.26 int_blockify()

```
void int_blockify (
            int message[16],
            int current_block[4][4] )
```

Convert an integer array to 4x4 block of hex.

Parameters

message	An integer array containing the values to be converted.
current_block	The output as a 4x4 integer array.

3.2.2.27 main()

```
int main (
            int argc,
            char * argv[] )
```

The main function. Arguments as described in the README is passed to this function. This function then uses the arguments to either encrypt or decrypt some input.

Parameters

argc	The number of arguments passed.
argv	The arguments as C-strings.

Returns

Successful execution.

3.2.2.28 print_block()

```
void print_block (
            int current_block[4][4] )
```

Output a 4x4 block to the terminal as a block of hex.

Parameters

current_block	The block to be printed.

3.2.2.29 print_expanded_key()

```
void print_expanded_key (
            int width,
            int expanded_key[] )
```

Output the expanded key in rows of 16.

Parameters

width	Use the macros AES128, AES192 or AES256 to select which width to use.
expanded_key	The expanded key to print.

3.2.2.30 print_hex_string()

```
void print_hex_string (
            unsigned char hex_string[],
            int message_len )
```

3.2.2.31 print_word()

```
void print_word (
            int word[],
            int length )
```

Output a word to the terminal.

Parameters

word	The word to be printed.
length	The length of the word.

3.2.2.32 test_functionality()

```
void test_functionality ( )
```

3.2.2.33 write_to_file()

```
void write_to_file (
             char filename[],
             unsigned char message[],
             int message_len )
```

3.2.3 Variable Documentation

3.2.3.1 PRIME_MATRIX

```
const int PRIME_MATRIX[2][4][4]
```

Initial value:

```
= {{{2, 3, 1, 1},
                                     {1, 2, 3, 1},
                                     {1, 1, 2, 3},
                                     {3, 1, 1, 2}},
                                    {{14, 11, 13,  9},
                                     { 9, 14, 11, 13},
                                     {13,  9, 14, 11},
                                     {11, 13,  9, 14}}}
```

Used for the transformation of a column in the mix columns operation.

3.2.3.2 S_BOX

```
const int S_BOX[2][16][16]
```

Provides a one-to-one mapping for the non-linear substitution of a byte.

3.3 README.md File Reference

Index