EHN 410 - Group 7 - AES Encryption

1.0

Mohamed Ameen Omar (u16055323)
Douglas Healy (u16018100)
Llewellyn Moyse (u15100708)

Generated by Doxygen 1.8.13

Contents

Chapter 1

README

EHN 410 - Group 7

Group members:

- Mohamed Ameen Omar (u16055323)
- Douglas Healy (u16018100)
- Llewellyn Moyse (u15100708)

To run Encryption Platform

1. Open a Linux Terminal.
2. Navigate to the Encyption Platform Directory.
3. Run the "make" command.
4. An executable called "encyptionPlatform" will be created.
5. Use "./encyptionPlatform" to run the encryption platform. (if no input parameters are specified, a help menu will be displayed)
6. A list of input parameter and default values:

Parameter	Description	Default Value
-h	Print out the help menu	
-f or –filename	Path to an ASCII file to encrypt/decrypt	None
-F	Path to a hex file to encrypt/decrypt	None
-k or –key	The encryption key	128 bit NULL key
-K	The encryption key (in hex)	None
-i or –IV	The Initialisation vector	128 bit NULL IV
-I	The Initialisation vector (in hex)	None
-p or –plaintext	Plaintext to encrypt	None
-P	Plaintext to encrypt (in hex)	None
-c or –ciphertext	Ciphertext to decrypt	None
-C	Ciphertext to decrypt (in hex)	None
-m or –mode	Mode of encryption/decryption (ecb, cbc or cfb)	ecb
-v or –verbose	Print verbose output	Flag not set
-e or –encrypt	Encrypt plaintext or file	Flag set
-d or –decrypt	Decrypt ciphertext or file	Flag not set
-T	Set if input text is a hex string	Flag not set

File Encryption Example

```
./encryptionPlatform -e -f <path/unencrypted_file.ext> -k <key> -i <iv>
./encryptionPlatform -e -F <path/unencrypted_hex_file.ext> -K <hex_key> -I <hex_iv>
```

File Encryption Example

```
./encryptionPlatform -d -f <path/encrypted_file.ext> -k <key> -i <iv>
./encryptionPlatform -d -F <path/encrypted_hex_file.ext> -F <hex_key> -I <hex_iv>
```

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

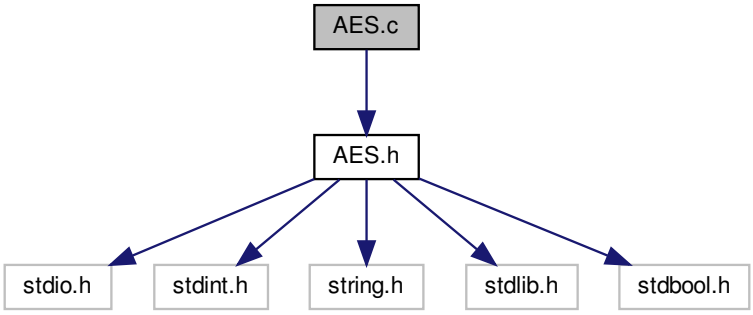
File Documentation

3.1 AES.c File Reference

AES encryption and decryption module implementation file. This file contains the implementation of the functions used for AES encryption and decryption. Input must be ASCII and not hex. The functions implemented in this file, perform the AES encryption and decryption on a single block of size dictated by the variable AES_BLOCK_SIZE.

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

Include dependency graph for AES.c:



Functions

- int getNumRounds (int keyLength)

  getNumRounds - Function to return the number of rounds of AES encryption and decryption based off of the length of the key given in
- unsigned char getSBoxValue (unsigned char index)

  getSBoxValue - Function to return the sBox value passed in as a parameter
- unsigned char getInvSBox (unsigned char index)

  getInvSBox - Function to return the inverse sBox value passed in as a parameter
- unsigned char getRconValue (unsigned char num)

getRconValue - Function to return the Rcon value for the index passed in as a parameter

- int getPaddedKeyLength (int currentKeyLength)

  getPaddedKeyLength - Function to return a valid key length (in bytes) based off of the current key length passed in as
- unsigned char * AESEncrypt (unsigned char *plainText, unsigned char *key, int plainTextLength, int key↩ Length)

  AESEncrypt - Function to encrypt a single block of plaintext passed in as parameter plainText using AES encryption, for 128, 192 and 256 bit keys. Validates the keylength and returns the corresponding ciphertext. The caller of the function must ensure that the returned ciphertext pointer is freed. The ciphertext returned is always 16 bytes and the plainText must be 16 bytes or less. Makes use of zero padding. All input must be in ASCII and NOT hex.
- unsigned char * AESDecrypt (unsigned char *cipherText, unsigned char *key, int cipherTextLength, int key↩ Length)

  AESDecrypt - Function to decrypt a single block of ciphertext passed in as parameter cipherText using AES decryption, for 128, 192 and 256 bit keys. Validates the keylength and returns the corresponding plaintext. The caller of the function must ensure that the returned plaintext pointer is freed. The plaintext returned is always 16 bytes and the plainText must be 16 bytes or less. Makes use of zero padding. All input must be in ASCII and NOT hex.
- unsigned char * RijndaelKeySchedule (unsigned char *originalKey, int keyLength)

  RijndaelKeySchedule - Function that performs the Rijndael key scheduling for AES encryption. Takes in the original key passed in as parameter.
- void KeyScheduleCore (unsigned char *word, int wordLength, int rConIterationVal)

  KeyScheduleCore - Function that performs the key schedule core for the Rijndael Key Schedule. Performs a single rotate left of the word passed in as.
- void SingleRotateLeft (unsigned char *word, int wordLength)

  SingleRotateLeft - Function to rotate the array passed in as a paramter.
- void printStateArray (uint8_t stateArray[4][4])

  printStateArray - Function to print the state array to the terminal in hex format.
- void AddRoundKey (unsigned char state[4][4], unsigned char key[4][4])

  AddRoundKey - Function that performs the Bitwise XOR between state and key as per AES encryption.
- void mixColumns (unsigned char state[4][4])

  mixColumns - Function that performs the MixColumns step of AES as specified by AES encryption.
- void invMixColumns (unsigned char state[4][4])

  invMixColumns - Function that does the inverse of the Mix Column Step for AES Encryption. Performs the gallois field multiplication and the required XOR to the state passed in as a paramter
- unsigned char galloisFieldMult (unsigned char a, unsigned char b)

  galloisFieldMult - Function to perform the Galois field multiplication operation required for the inverse mix columns and the mix columns operation of the AES encryption and decryption processes. Returns the result of the multiplication.
- void subBytes (unsigned char state[4][4])

  subBytes - Function that performs the sub byte operation where each value is replaced by the s box value
- void invSubBytes (unsigned char state[4][4])

  invSubBytes - Function that performs the inverse of Function subBytes
- void ShiftRows (unsigned char state[4][4], int wordLength)

  ShiftRows - Function to shift the state array according to the AES encryption standard for 128 - bits blocks.
- void invShiftRows (unsigned char state[4][4], int wordLength)

  invShiftRows - Function to shift the state array Inverse according to the AES encryption standard for 128 - bits blocks
- void SingleRotateRight (unsigned char *word, int wordLength)

  SingleRotateRight - Function to rotate the array passed in as a paramter.
- void getRoundKey (unsigned char *expandedKey, unsigned char *roundKey, int roundNum)

  getRoundKey - Function to extract the correct sub-key to use for the appropriate round specified by
- void constructStateArray (unsigned char *flatArray, unsigned char stateArray[ ][4])

  constructStateArray - Function to convert the state array from a flat 1D array to a multidimensional array.
- uint8_t hexToInt (char ch)

  hexToInt - Function that converts a given hex value into an integer.
- uint8_t hexToAscii (char ch1, char ch2)

  hexToAscii - Function that converts a given hex value to its ASCII equivalent.

- void hexToAsciiString (char *hexString, char *asciiString, int hexStringLength)

  hexToAsciiString - Function that converts a given string of hex values into its ASCII equivalent. A hex string contains hex chars and is "encoded" in ascii In order to encrypt it, it must be converted to the equivalent ascii plain text string plaintext string is half the size of hex, since two hex chars = 1 ascii char if hex string is "4A" it will be converted to "J" in ascii which will have a hex representation of "4a" The original hex string converted to hex staright or printed in hex straight rather will print or have the value "0x34", "0x31" BASICALLY THE HEX STRING FF IS INTERPRETED AS THE CHARS FF, whereas when using this function we intend it to be "J", ie the char "J"
- unsigned char * asciiToHexString (unsigned char *asciiString, unsigned char *hexString, size_t asciiString↩ Len)

  Function name: asciiToHexString - convert an ascii String to an ascii string.
- void validateNumRounds (int numRounds, int keyLength)

  validateNumRounds - Function that validates the number of rounds that have been passed in by the
- void validatePlainTextLength (size_t plainTextLength)

  validatePlainTextLength - Function that validates the length of the plaintext. The validation is done against the AE↩ S_BLOCK_SIZE value
- void validateCipherTextLength (int cipherTextLength)

  validateCipherTextLength - Function that validates the length of the ciphertext. The validation is done against the AES_BLOCK_SIZE value
- void printAESBlock (unsigned char *block)

  printAESBlock - Function to print a single block in hex format to the terminal.
- int fileNameDirIndex (char *fileName, int fileNameLength)

  Returns the last index of '/' in a given path, otherwise returns -1 if no '/' is found.
- void stripDirectory (char *fileName, char *extractedFileName, char *extractedFilePath, int fileNameLength, int slashIndex)

  Removes path from the provided path to a file and returns only the file name.
- void getOutputFileName (int type, char *fileName, char *outputFileName, char *mode)

  Get the output file name from all the parameters passed in.
- uint8_t isFileTxt (unsigned char *fileName)

  isFileTxt - Function to determine if the file passed in as a paramter
- unsigned char * keyHexToAscii (unsigned char *hexKey, int keyLength)

  keyHexToAscii - Function to convert a key from a Hex string passed in as a paramter
- unsigned char * IVHexToAscii (unsigned char *hexIV, int IVLength)

  IVHexToAscii - Function to convert a initialization vector from a Hex string passed in as a paramter.
- unsigned char * XORBlocks (unsigned char *block1, unsigned char *block2, int length)

  XORBlocks - Function to XOR two blocks of length.

Variables

- const size_t AES_BLOCK_SIZE = 16

  Variable- const size_t AES_BLOCK_SIZE. Used to dictate the length in bytes of a single AES block used for encryption and decryption. Set to 16 bytes for a single block.
- size_t VERBOSE = 0

  Variable- size_t VERBOSE Used to dictate whether verbose output is printed to the terminal or not. If 0, does not print verbose. If 1, prints verbose.
- const unsigned char sbox [256]

  const unsigned char sbox. Lookup table for the sbox values used during AES Encryption.
- const unsigned char invSBox [256]

  const unsigned char invSBox. Lookup table for the inverse sbox values used during AES Decryption.
- const unsigned char Rcon [255]

  const unsigned char Rcon. Lookup table for the Rcon values used during Rijndael Key Schedule during the AES Encryption and Decryption.

3.1.1 Detailed Description

AES encryption and decryption module implementation file. This file contains the implementation of the functions used for AES encryption and decryption. Input must be ASCII and not hex. The functions implemented in this file, perform the AES encryption and decryption on a single block of size dictated by the variable AES_BLOCK_SIZE.

Authors

Mohamed Ameen Omar (u16055323)
Douglas Healy (u16018100)
Llewellyn Moyse (u15100708)

Version

0.1

Date

2019-03-20

Copyright

Copyright (c) 2019

3.1.2 Function Documentation

3.1.2.1 AddRoundKey()

```
void AddRoundKey (
            unsigned char state[4][4],
            unsigned char key[4][4] )
```

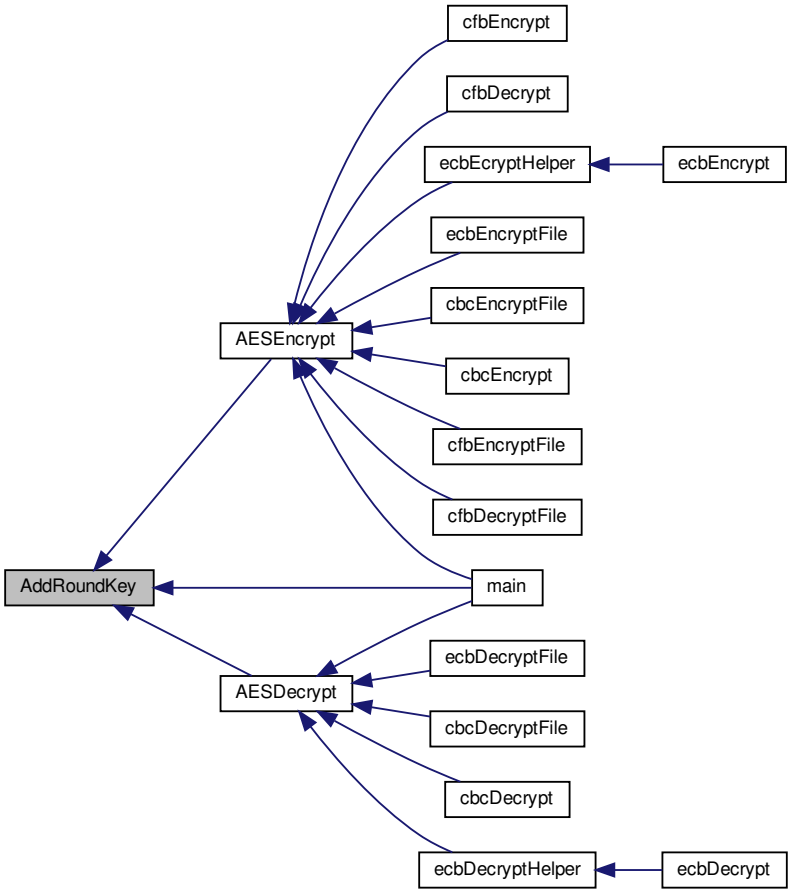
AddRoundKey - Function that performs the Bitwise XOR between state and key as per AES encryption.

Parameters

state	- unsigned char - is the current state of the ciphertext or plaintext during AES encryption or decryption
key	- unsigned char - sub key to be added for the current round to the current state vector

Definition at line 688 of file AES.c.

Here is the caller graph for this function:



3.1.2.2 AESDecrypt()

```
unsigned char* AESDecrypt (
            unsigned char * cipherText,
            unsigned char * key,
            int cipherTextLength,
            int keyLength )
```

AESDecrypt - Function to decrypt a single block of ciphertext passed in as parameter cipherText using AES decryption, for 128, 192 and 256 bit keys. Validates the keylength and returns the corresponding plaintext. The caller of the function must ensure that the returned plaintext pointer is freed. The plaintext returned is always 16 bytes and the plainText must be 16 bytes or less. Makes use of zero padding. All input must be in ASCII and NOT hex.

Parameters

char	- unsigned char* cipherText - pointer to the ciphertext that needs to be decrypted using AES decryption.

Parameters

char	- unsigned char∗ key - reference to the key that must be used for AES decryption.
cipherTextLength	- length of the ciphertext in
cipherText	to be decrypted.
keyLength	- length of the key passed in as
key	used for the AES decryption.

Returns

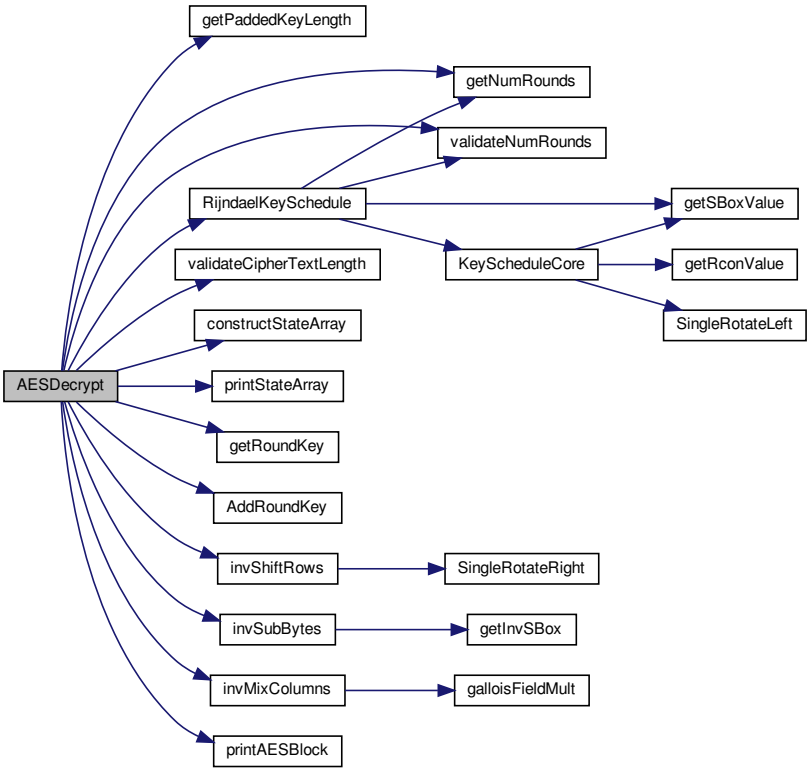
unsigned∗ char - Plaintext resulting from the decryption of the ciphertext passed in as

Parameters

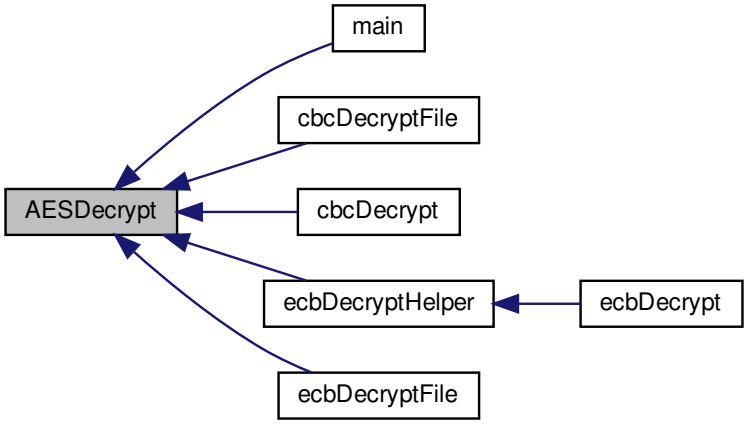
cipherText.	

Definition at line 399 of file AES.c.

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.2.3 AESEncrypt()

```
unsigned char* AESEncrypt (
            unsigned char * plainText,
            unsigned char * key,
            int plainTextLength,
            int keyLength )
```

AESEncrypt - Function to encrypt a single block of plaintext passed in as parameter plainText using AES encryption, for 128, 192 and 256 bit keys. Validates the keylength and returns the corresponding ciphertext. The caller of the function must ensure that the returned ciphertext pointer is freed. The ciphertext returned is always 16 bytes and the plainText must be 16 bytes or less. Makes use of zero padding. All input must be in ASCII and NOT hex.

Parameters

char	- unsigned char* plainText - pointer to the plaintext that needs to be encrypted using AES encryption.
char	- unsigned char* key - reference to the key that must be used for AES encryption.
plainTextLength	- length of the plaintext in
plainText	to be encrypted.
keyLength	- length of the key passed in as
key	used for the AES encryption.

Returns

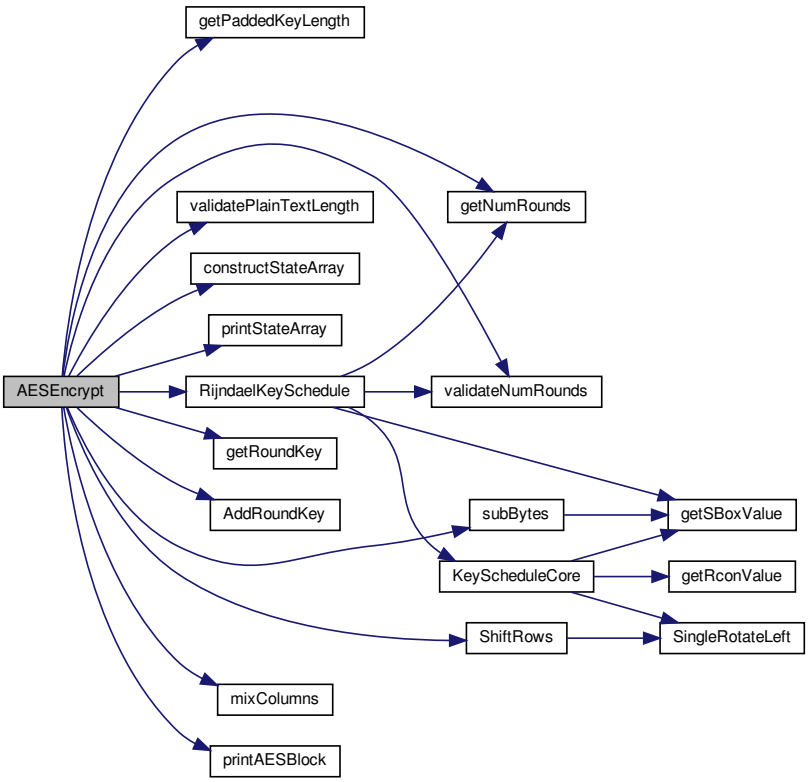
unsigned* char - Ciphertext resulting from the encryption of the plaintext passed in as

Parameters

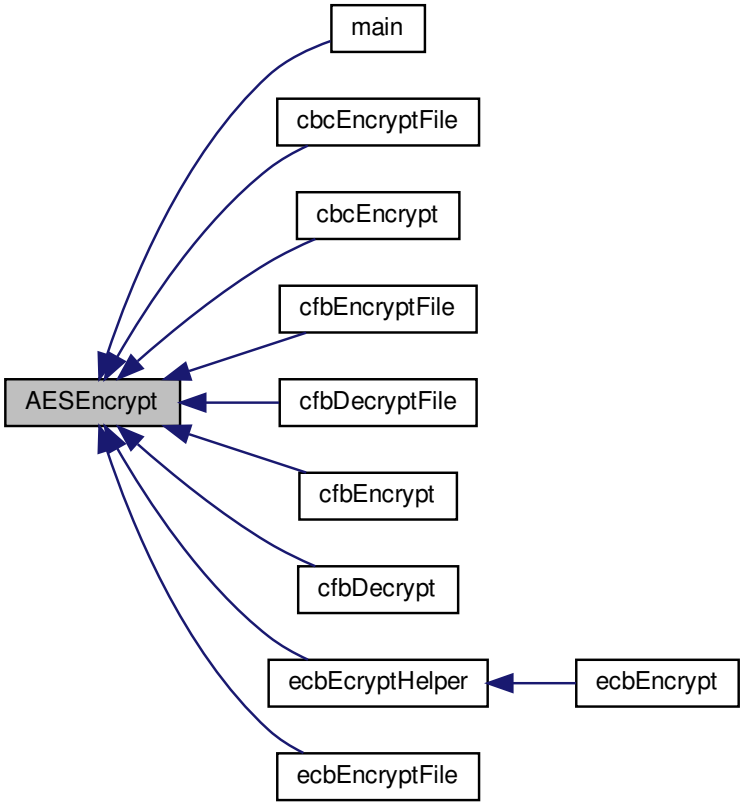
plainText.	

Definition at line 198 of file AES.c.

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.2.4 asciiToHexString()

```
unsigned char* asciiToHexString (
            unsigned char * asciiString,
            unsigned char * hexString,
            size_t asciiStringLen )
```

Function name: asciiToHexString - convert an ascii String to an ascii string.

Parameters

asciiString	- unsigned char* pointing to the ASCII String to be converted.
hexString	- unsigned char* pointing to a memory where the converted Hex string should be stored.
asciiStringLen	- size_t containing the length of the ASCII String to be converted.

Returns

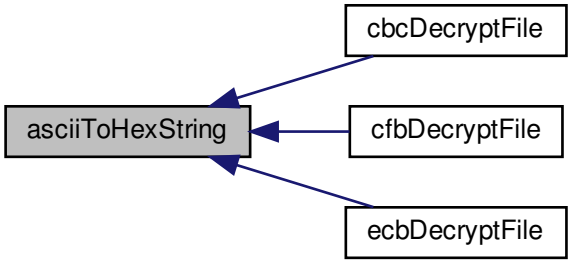
unsigned char* asciiToHexString - pointer to the converted Hex String, pointing to the same memory location as

Parameters

hexString.	

Definition at line 971 of file AES.c.

Here is the caller graph for this function:



3.1.2.5 constructStateArray()

```
void constructStateArray (
            unsigned char * flatArray,
            unsigned char stateArray[][4] )
```

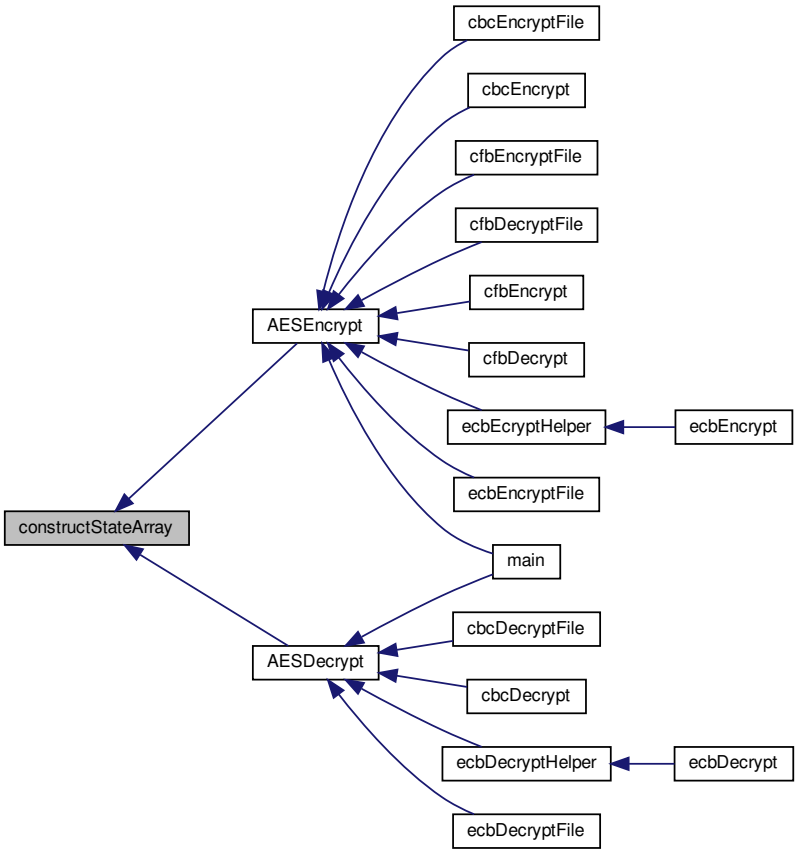
constructStateArray - Function to convert the state array from a flat 1D array to a multidimensional array.

Parameters

char	flatArray -the 1D array to be converted.
stateArray	- the multidimensional array to which to copy the flat array elements to.

Definition at line 895 of file AES.c.

Here is the caller graph for this function:



3.1.2.6 fileNameDirIndex()

```
int fileNameDirIndex (
            char * fileName,
            int fileNameLength )
```

Returns the last index of '/' in a given path, otherwise returns -1 if no '/' is found.

Parameters

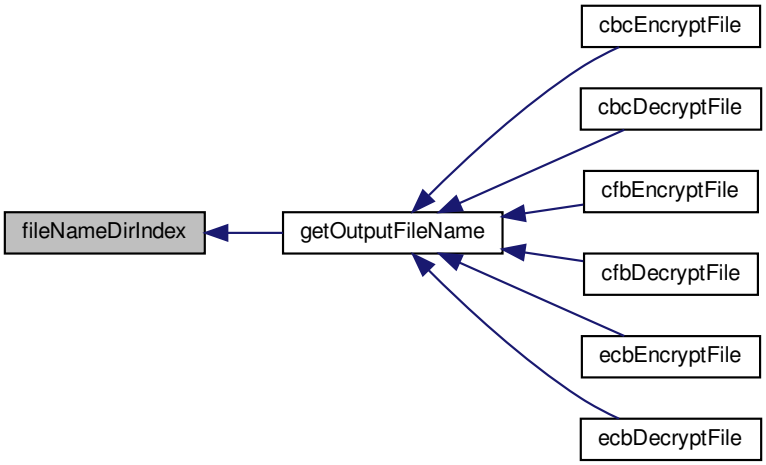
fileName	The path to a file
fileNameLength	The length of the provided file

Returns

int Index of the last '/' in the path, else -1 if no '/' was found

Definition at line 1043 of file AES.c.

Here is the caller graph for this function:



3.1.2.7 galloisFieldMult()

```
unsigned char galloisFieldMult (
            unsigned char a,
            unsigned char b )
```

galloisFieldMult - Function to perform the Galois field multiplication operation required for the inverse mix columns and the mix columns operation of the AES encryption and decryption processes. Returns the result of the multiplication.

Parameters

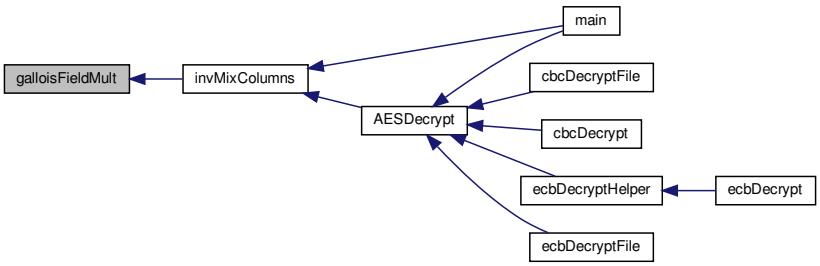
a	- first character to perform Galois field multiplication.
b	- second character to perform Galois field multiplication.

Returns

unsigned char - Result of the Galois field multiplication.

Definition at line 764 of file AES.c.

Here is the caller graph for this function:



3.1.2.8 getInvSBox()

```
unsigned char getInvSBox (
            unsigned char index )
```

getInvSBox - Function to return the inverse sBox value passed in as a parameter

Parameters

index.	Requires the original value required in hex.
index	- unsigned char - hexadecimal representation of the index for which the inverse SBox value is required.

Returns

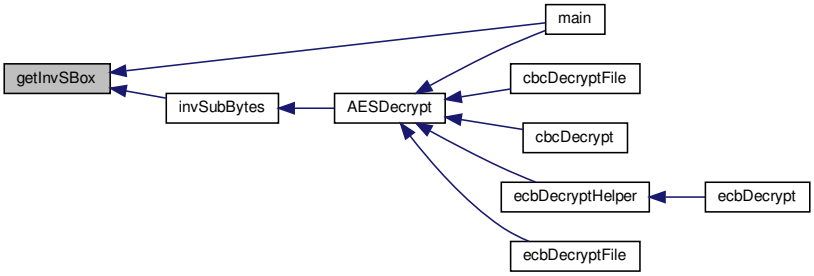
unsigned char - inverse sBox value for the paramter

Parameters

index.	

Definition at line 146 of file AES.c.

Here is the caller graph for this function:



3.1.2.9 getNumRounds()

```
int getNumRounds (
            int keyLength )
```

getNumRounds - Function to return the number of rounds of AES encryption and decryption based off of the length of the key given in

Parameters

keyLength.	
keyLength	- int - indicates the length of the key

Returns

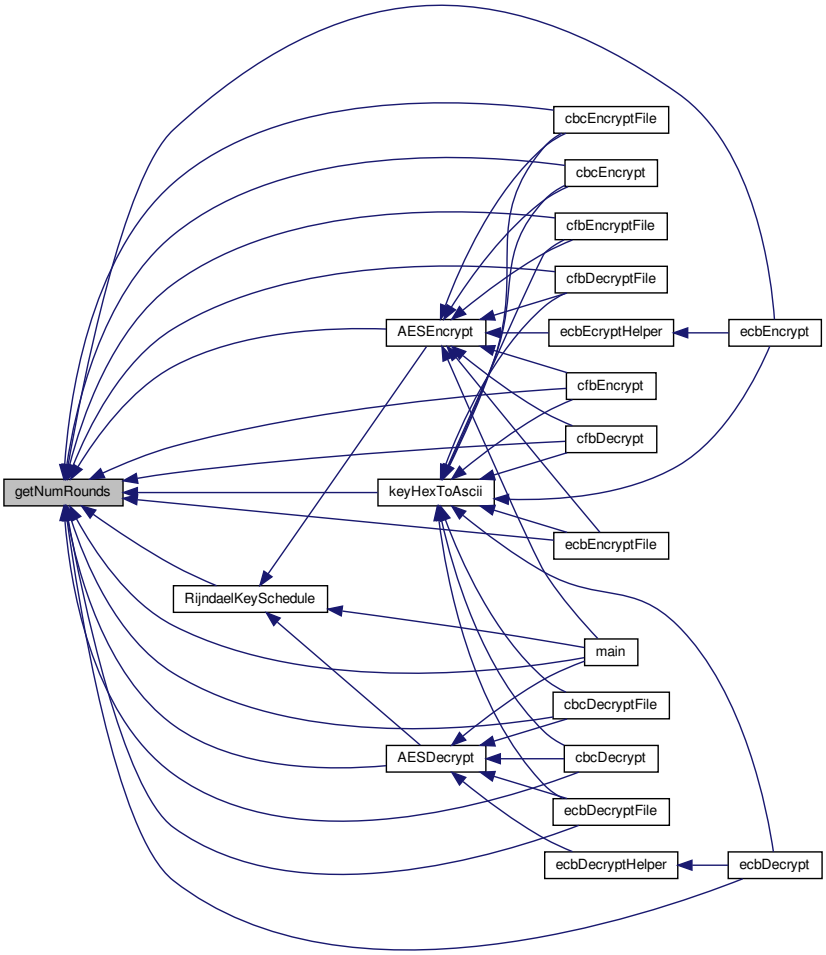
int - the number of rounds based off of the length of the key passed in the parameter

Parameters

keyLength.	If the length of the key is not valid, returns -1.

Definition at line 114 of file AES.c.

Here is the caller graph for this function:



3.1.2.10 getOutputFileName()

```
void getOutputFileName (
            int type,
            char * fileName,
            char * outputFileName,
            char * mode )
```

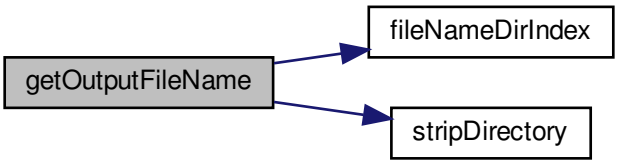
Get the output file name from all the parameters passed in.

Parameters

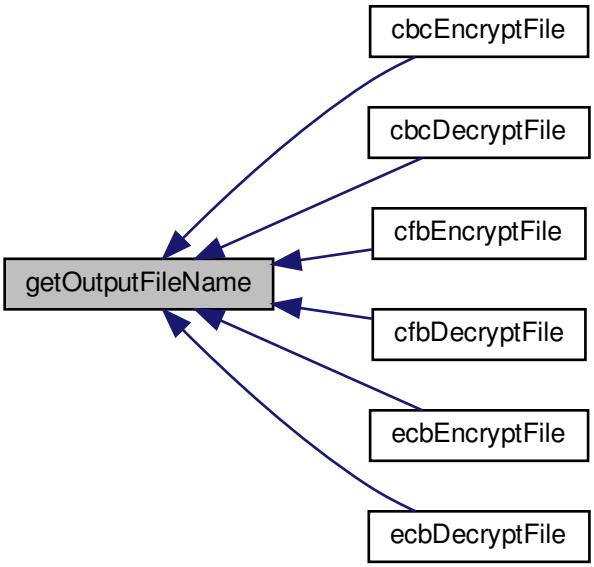
type	0 - Encrypt, 1 - Decrypt
fileName	The name of the input file
outputFileName	The name of the output file
mode	Chipher mode to be used (ECB, CBC, CFB)

Definition at line 1086 of file AES.c.

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.2.11 getPaddedKeyLength()

```
int getPaddedKeyLength (
            int currentKeyLength )
```

getPaddedKeyLength - Function to return a valid key length (in bytes) based off of the current key length passed in as

Parameters

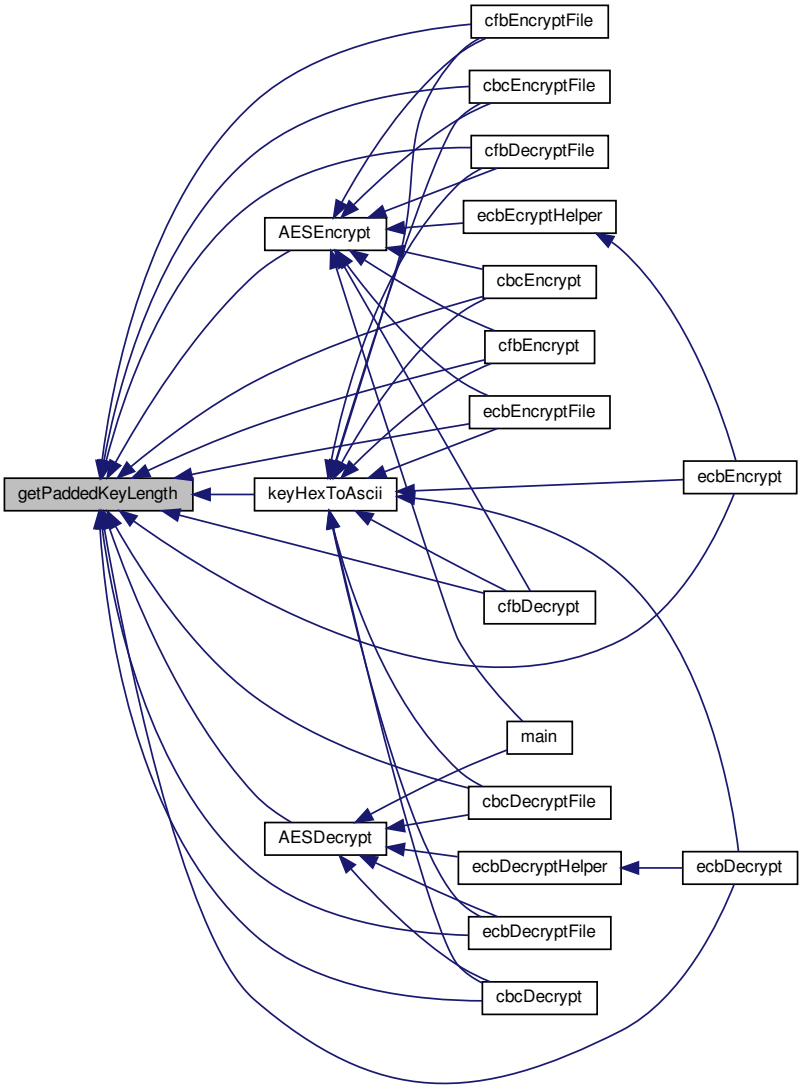
currentKeyLength.	Corresponds to minimum and maximum key length required for AES encryption and decryption. The key will then be padded to the length of the value returned from this function. If the keylength is less than 16, will return 16. If greater than 16, but less than 24, will return 24. If greater than 32, will return -1.
currentKeyLength	- int - current key length in bytes, to be padded to the return value

Returns

int - the length in bytes that the key should be padded to.

Definition at line 172 of file AES.c.

Here is the caller graph for this function:



3.1.2.12 getRconValue()

```
unsigned char getRconValue (
            unsigned char num )
```

getRconValue - Function to return the Rcon value for the index passed in as a parameter

Parameters

num.	Requires the original value required in hex.
index	- unsigned char - hexadecimal representation of the number for which the Rcon value is required during the key schedule.

Returns

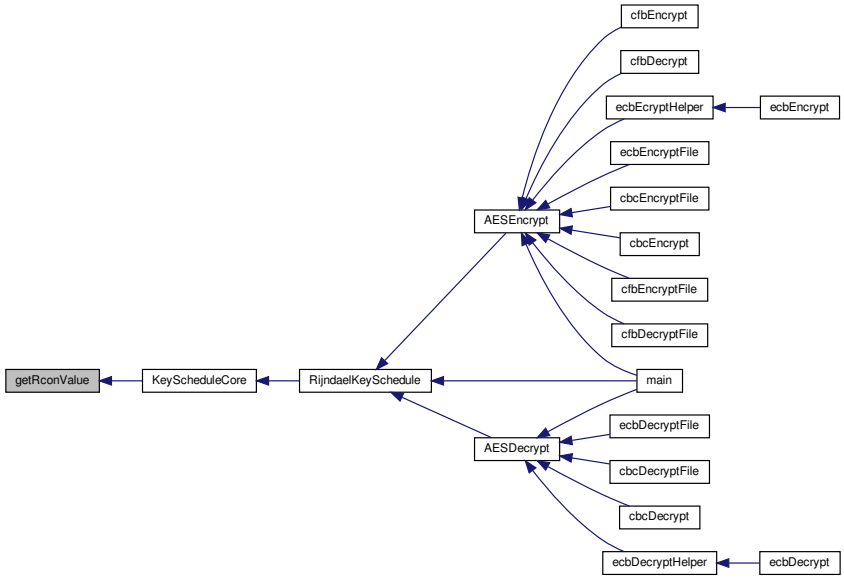
unsigned char - rCon value for the paramter

Parameters

num.	

Definition at line 158 of file AES.c.

Here is the caller graph for this function:



3.1.2.13 getRoundKey()

```
void getRoundKey (
            unsigned char * expandedKey,
            unsigned char * roundKey,
            int roundNum )
```

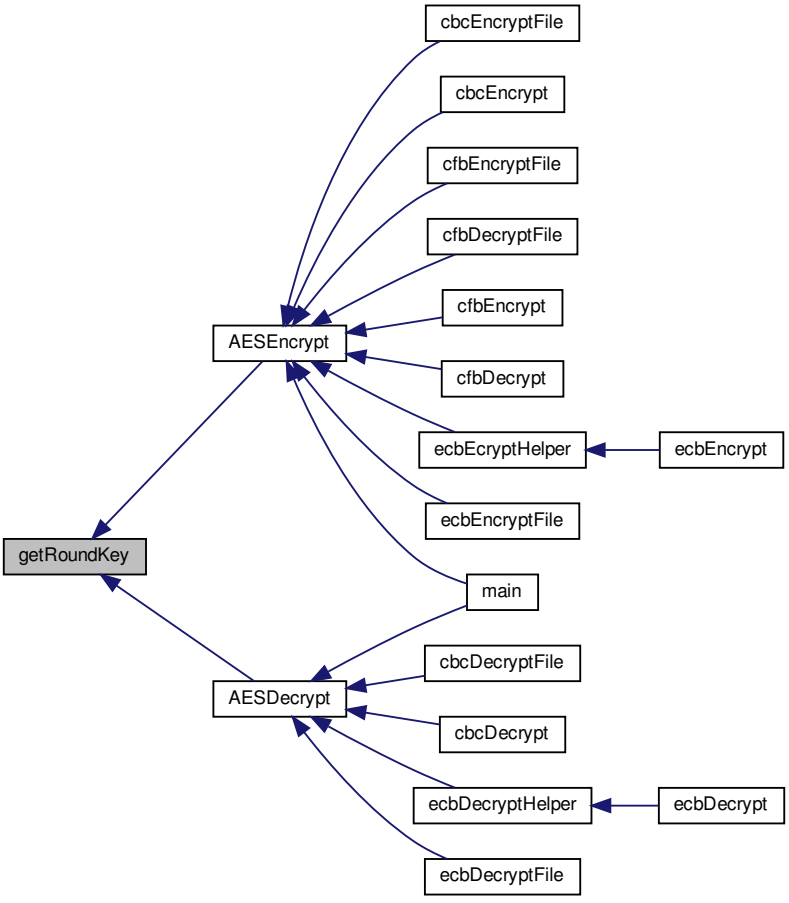
getRoundKey - Function to extract the correct sub-key to use for the appropriate round specified by

Parameters

roundNum.	Copies the sub-key from the expanded key in
expandedKey	to
roundKey.	
char	- expandedKey - The expanded key from which to extract the sub-key.
char	- roundKey - memory to which to copy the sub-key.
roundNum	- int - the round number for which the sub-key is required.

Definition at line 881 of file AES.c.

Here is the caller graph for this function:



3.1.2.14 getSBoxValue()

```
unsigned char getSBoxValue (
            unsigned char index )
```

getSBoxValue - Function to return the sBox value passed in as a parameter

Parameters

index.	Requires the original value required in hex.
index	- unsigned char - hexadecimal representation of the index for which the SBox value is required.

Returns

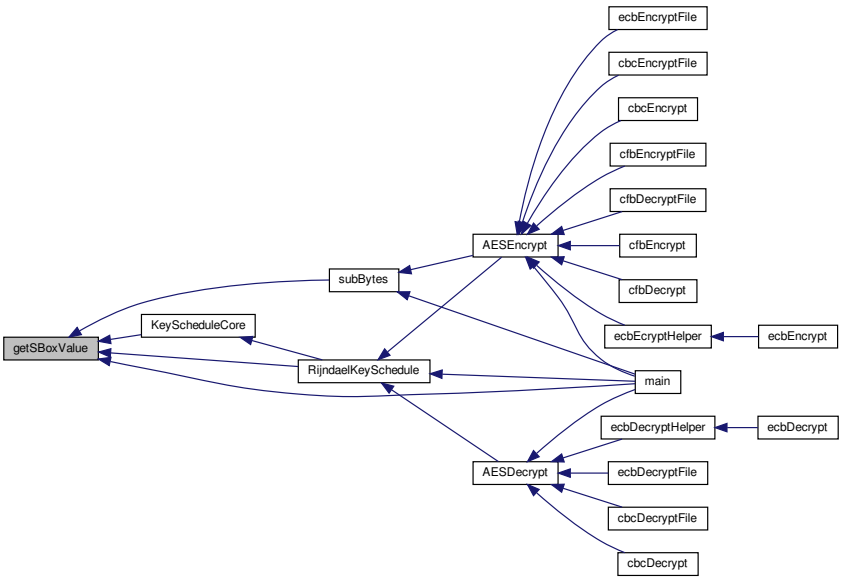
unsigned char - sBox value for the paramter

Parameters

index.	

Definition at line 134 of file AES.c.

Here is the caller graph for this function:



3.1.2.15 hexToAscii()

```
uint8_t hexToAscii (
             char ch1,
             char ch2 )
```

hexToAscii - Function that converts a given hex value to its ASCII equivalent.

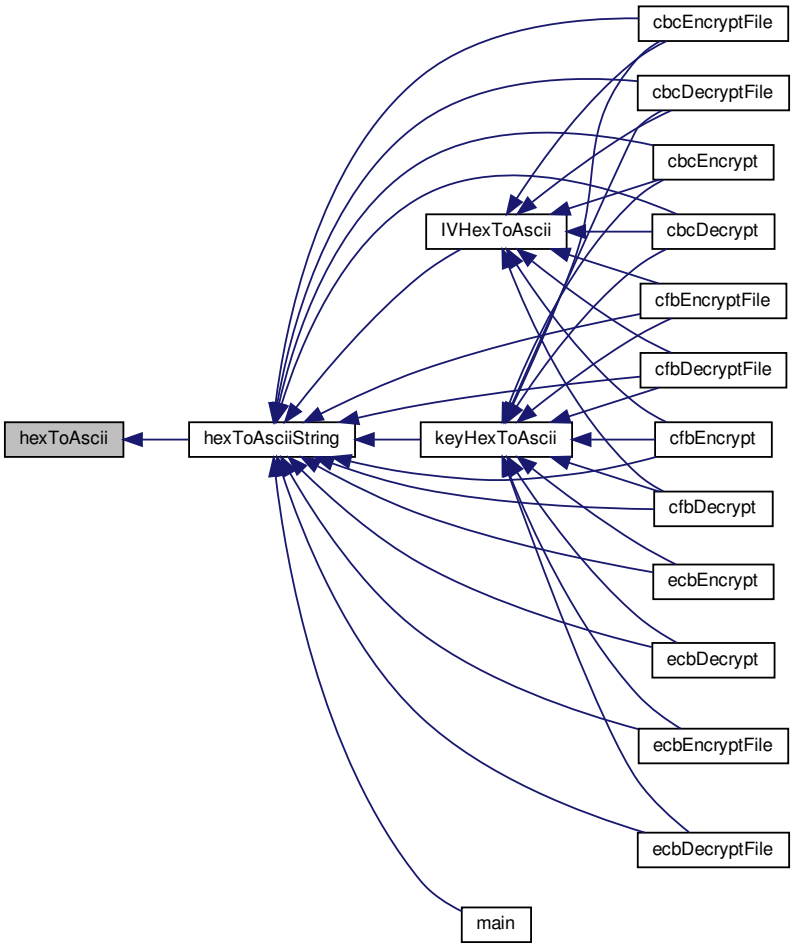
Parameters

ch1	- char value of the first hex value.
ch2	- char value of the second hex value.

Definition at line 928 of file AES.c.

Here is the call graph for this function:

Here is the caller graph for this function:



3.1.2.16 hexToAsciiString()

```
void hexToAsciiString (
            char * hexString,
            char * asciiString,
            int hexStringLength )
```

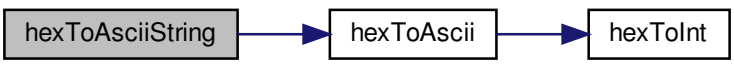
hexToAsciiString - Function that converts a given string of hex values into its ASCII equivalent. A hex string contains hex chars and is "encoded" in ascii In order to encrypt it, it must be converted to the equivalent ascii plain text string plaintext string is half the size of hex, since two hex chars = 1 ascii char if hex string is "4A" it will be converted to "J" in ascii which will have a hex representation of "4a" The original hex string converted to hex staright or printed in hex straight rather will print or have the value "0x34", "0x31" BASICALLY THE HEX STRING FF IS INTERPRETED AS THE CHARS FF, whereas when using this function we intend it to be "J", ie the char "J"

Parameters

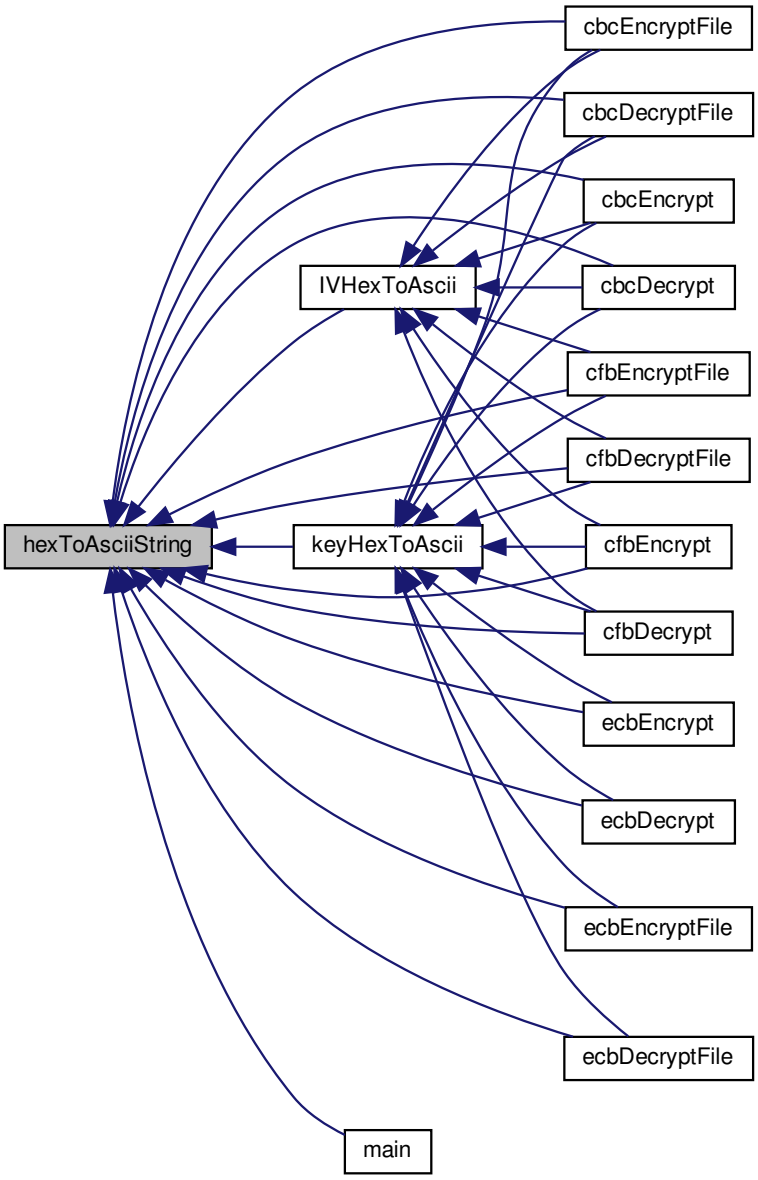
char∗	hexString - The string of hex values to be converted.
char∗	asciiString - The output of the converted hex string.
int	hexStringLength - The length of parameter hexString.

Definition at line 948 of file AES.c.

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.2.17 hexToInt()

```
uint8_t hexToInt (
            char ch )
```

hexToInt - Function that converts a given hex value into an integer.

Parameters

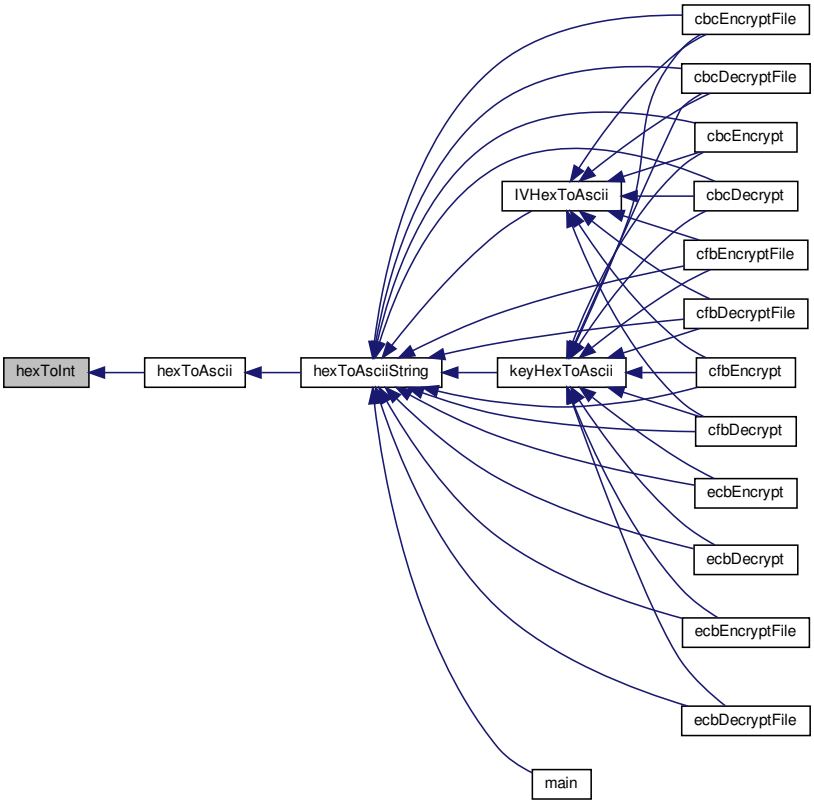
ch	- hex value that wil be converted to int.

Returns

uint8_t the converted int value.

Definition at line 909 of file AES.c.

Here is the caller graph for this function:



3.1.2.18 invMixColumns()

```
void invMixColumns (
            unsigned char state[4][4] )
```

invMixColumns - Function that does the inverse of the Mix Column Step for AES Encryption. Performs the gallois field multiplication and the required XOR to the state passed in as a paramter

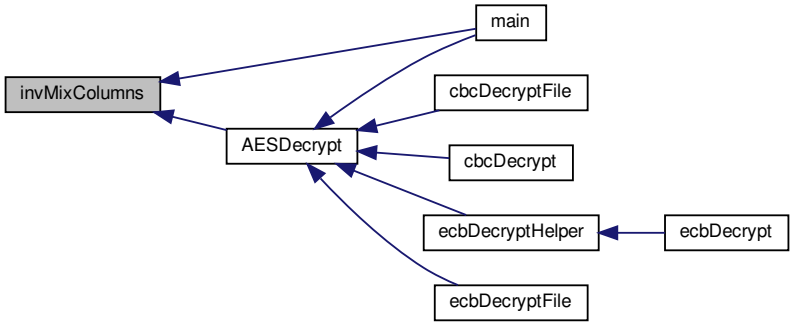
Parameters

state.	
state	- unsigned char - is the current state of the ciphertext or plaintext during AES encryption or decryption.

Definition at line 740 of file AES.c.

Here is the call graph for this function:

Here is the caller graph for this function:



3.1.2.19 invShiftRows()

```
void invShiftRows (
            unsigned char state[4][4],
            int wordLength )
```

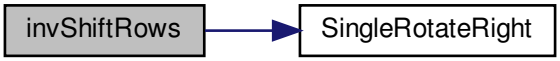
invShiftRows - Function to shift the state array Inverse according to the AES encryption standard for 128 - bits blocks

Parameters

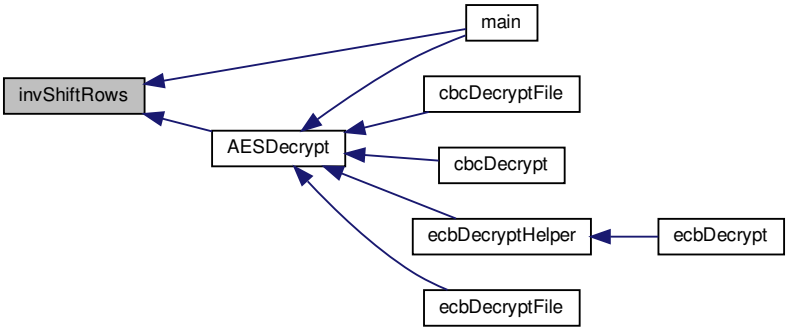
state	- unsigned char - is the current state of the ciphertext or plaintext during AES encryption or decryption

Definition at line 835 of file AES.c.

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.2.20 invSubBytes()

```
void invSubBytes (
            unsigned char state[4][4] )
```

invSubBytes - Function that performs the inverse of Function subBytes

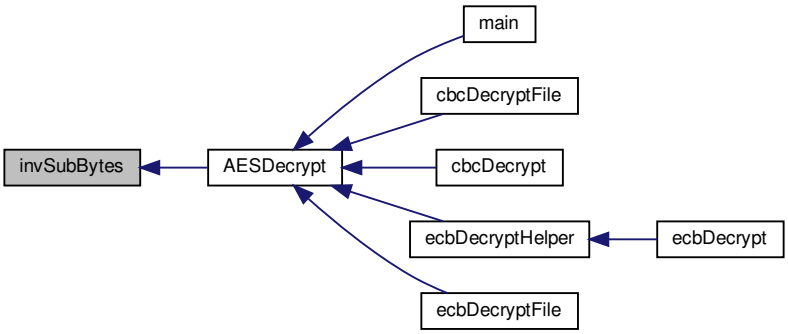
Parameters

state	- unsigned char - is the current state of the ciphertext or plaintext during AES encryption or decryption

Definition at line 802 of file AES.c.

Here is the call graph for this function:

Here is the caller graph for this function:



3.1.2.21 isFileTxt()

```
uint8_t isFileTxt (
            unsigned char * fileName )
```

isFileTxt - Function to determine if the file passed in as a paramter

Parameters

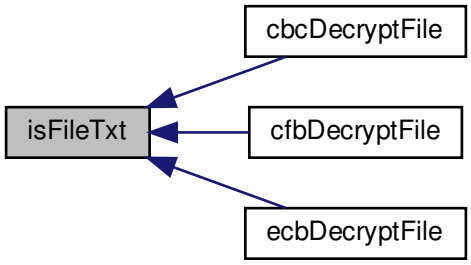
filename	is a text file with extension .txt or not. Returns a 1 if it is and a 0 if it isn't.
fileName	- unsigned char* fileName - path to file to determine if the file is a text file or not.

Returns

uint8_t - boolean indicating if it is a text file or not. (0 is not a text file, 1 is a text file)

Definition at line 1156 of file AES.c.

Here is the caller graph for this function:



3.1.2.22 IVHexToAscii()

```
unsigned char* IVHexToAscii (
            unsigned char * hexIV,
            int IVLength )
```

IVHexToAscii - Function to convert a initialization vector from a Hex string passed in as a paramter.

Parameters

hexIV	to an ascii string. User must free the returned pointer to memory allocated. Returns the Ascii equivalent. The caller must free the pointer returned.
char	- unsigned char* hexIV - hex representation
IVLength	- length of the hex representation of the IV passed in as paramter
hexIV.	

Returns

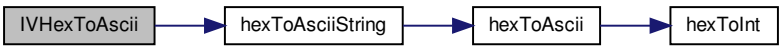
unsigned* - the ASCII representation of the hex IV passed in as parameter

Parameters

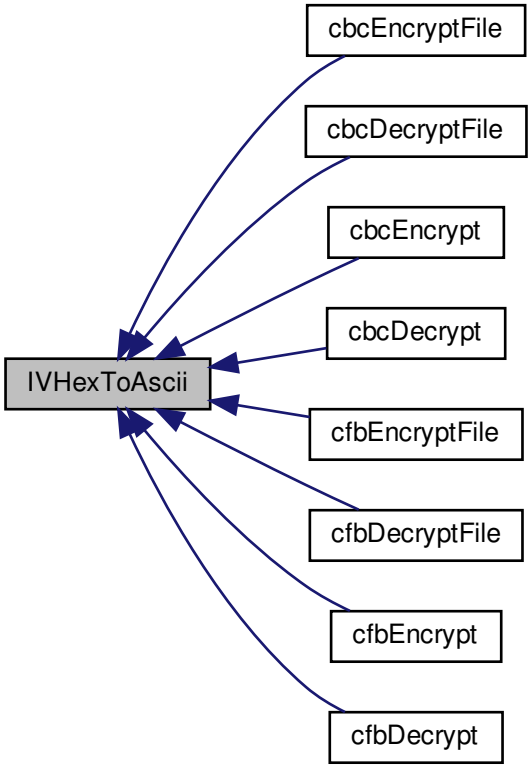
hex↩ IV.	

Definition at line 1205 of file AES.c.

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.2.23 keyHexToAscii()

```
unsigned char* keyHexToAscii (
            unsigned char * hexKey,
            int keyLength )
```

keyHexToAscii - Function to convert a key from a Hex string passed in as a paramter

Parameters

hexKey	to an ascii string. User must free the returned pointer to memory allocated. Returns the Ascii equivalent. The caller must free the pointer returned.
char	- unsigned char∗ hexKey - hex representation of the key to be converted to ASCII.
keyLength	- length of the hex representation of the key passed in as paramter
hexKey.	

Returns

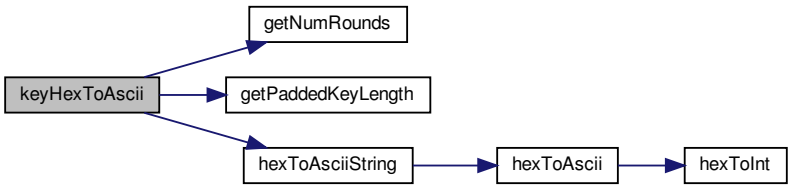
unsigned∗ - the ASCII representation of the hex key passed in as parameter

Parameters

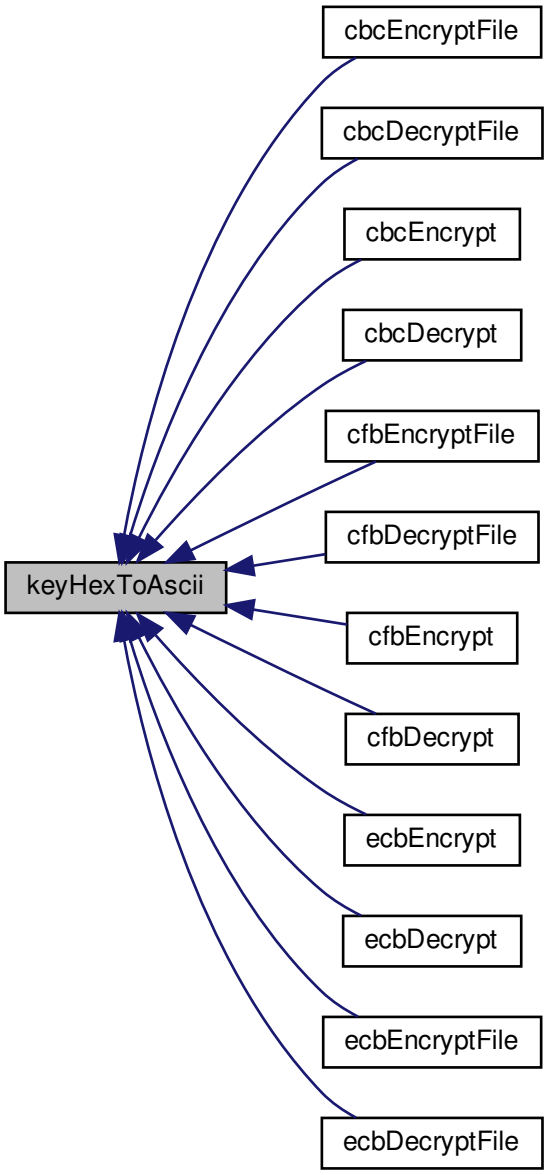
hexKey.	

Definition at line 1178 of file AES.c.

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.2.24 KeyScheduleCore()

```
void KeyScheduleCore (
            unsigned char * word,
            int wordLength,
            int rConIterationVal )
```

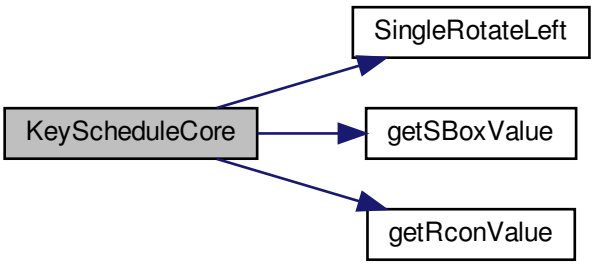
KeyScheduleCore - Function that performs the key schedule core for the Rijndael Key Schedule. Performs a single rotate left of the word passed in as.

Parameters

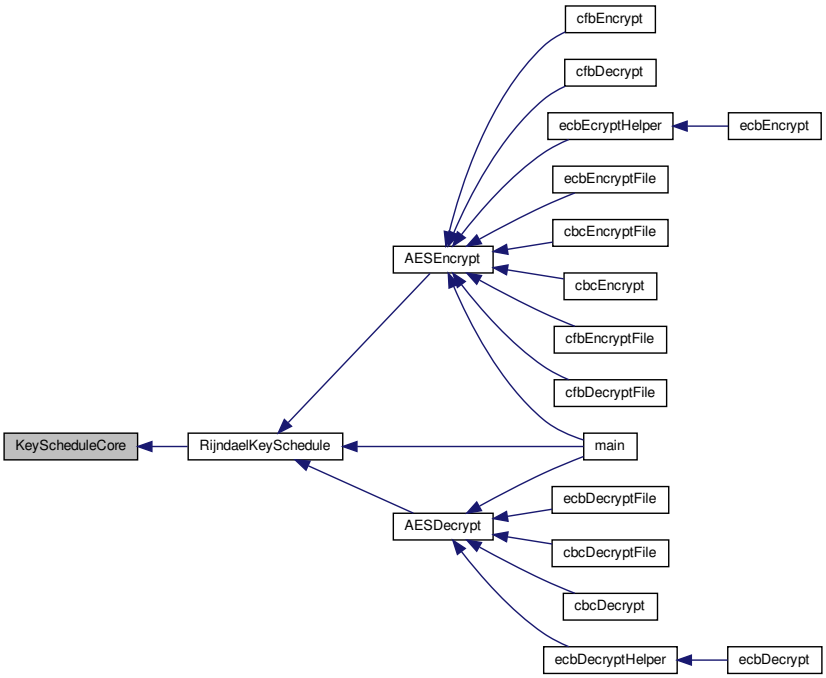
word	and applies the required s-box substituion and rcon XOR.
char	- unsigned char∗ word - pointer to the word onto which the key schedule core should be operated.
wordLength	- length of the word passed in as a parameter
word.	
rConIterationVal	- the iteration value to be used for the rcon XOR.

Definition at line 631 of file AES.c.

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.2.25 mixColumns()

```
void mixColumns (
            unsigned char state[4][4] )
```

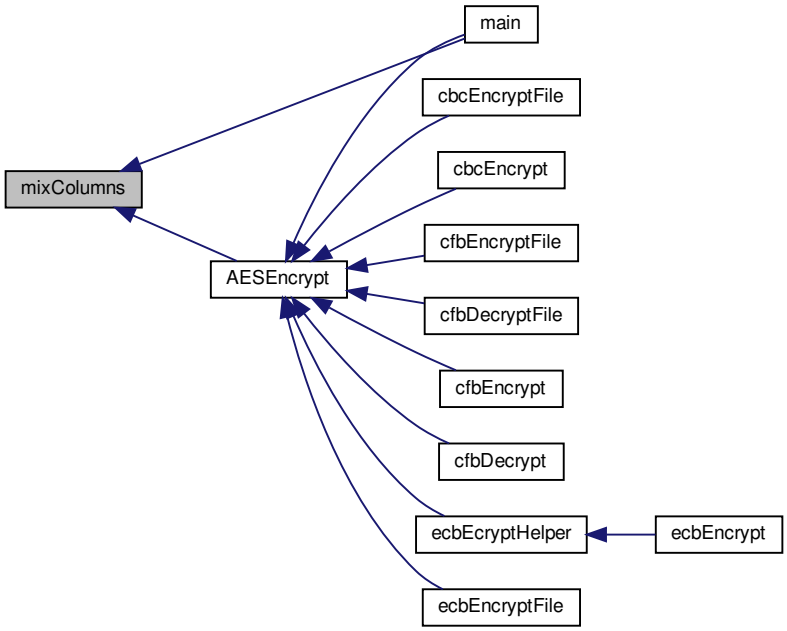
mixColumns - Function that performs the MixColumns step of AES as specified by AES encryption.

Parameters

state	- unsigned char - is the current state of the ciphertext or plaintext during AES encryption or decryption

Definition at line 702 of file AES.c.

Here is the caller graph for this function:



3.1.2.26 printAESBlock()

```
void printAESBlock (
            unsigned char * block )
```

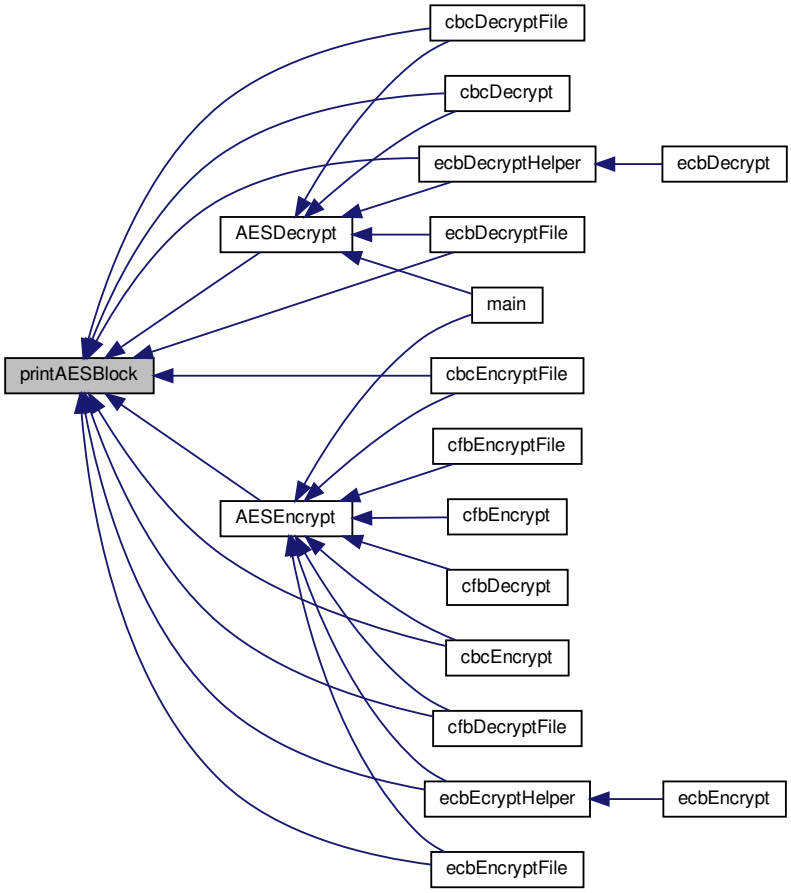
printAESBlock - Function to print a single block in hex format to the terminal.

Parameters

block	- block to be printed.

Definition at line 1028 of file AES.c.

Here is the caller graph for this function:



3.1.2.27 printStateArray()

```
void printStateArray (
            uint8_t stateArray[4][4] )
```

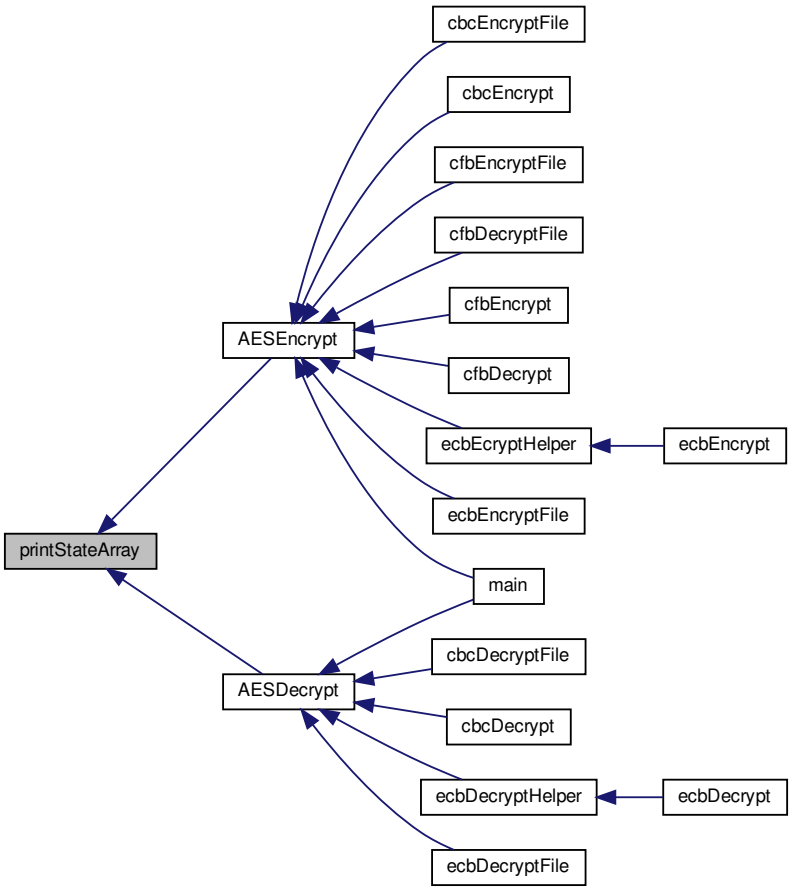
printStateArray - Function to print the state array to the terminal in hex format.

Parameters

stateArray	- the state array that should be printed to the terminal.

Definition at line 673 of file AES.c.

Here is the caller graph for this function:



3.1.2.28 RijndaelKeySchedule()

```
unsigned char* RijndaelKeySchedule (
            unsigned char * originalKey,
            int keyLength )
```

RijndaelKeySchedule - Function that performs the Rijndael key scheduling for AES encryption. Takes in the original key passed in as parameter.

Parameters

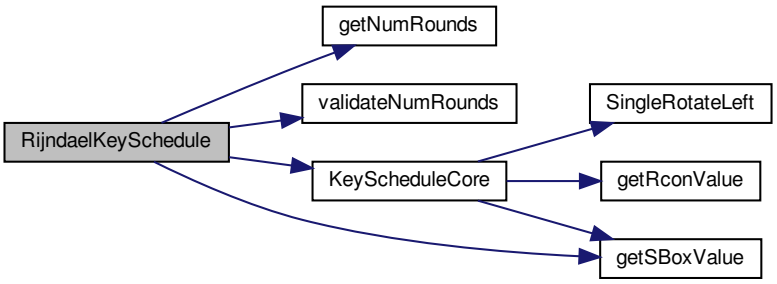
originalKey	and the length of the original key given as parameter. The caller must free the memory allocated and returned.
originalKey	- unsigned char * - An unsigned char pointer to the original key.
keyLength	- int - length of originalKey passed in as a parameter
originalKey	

Returns

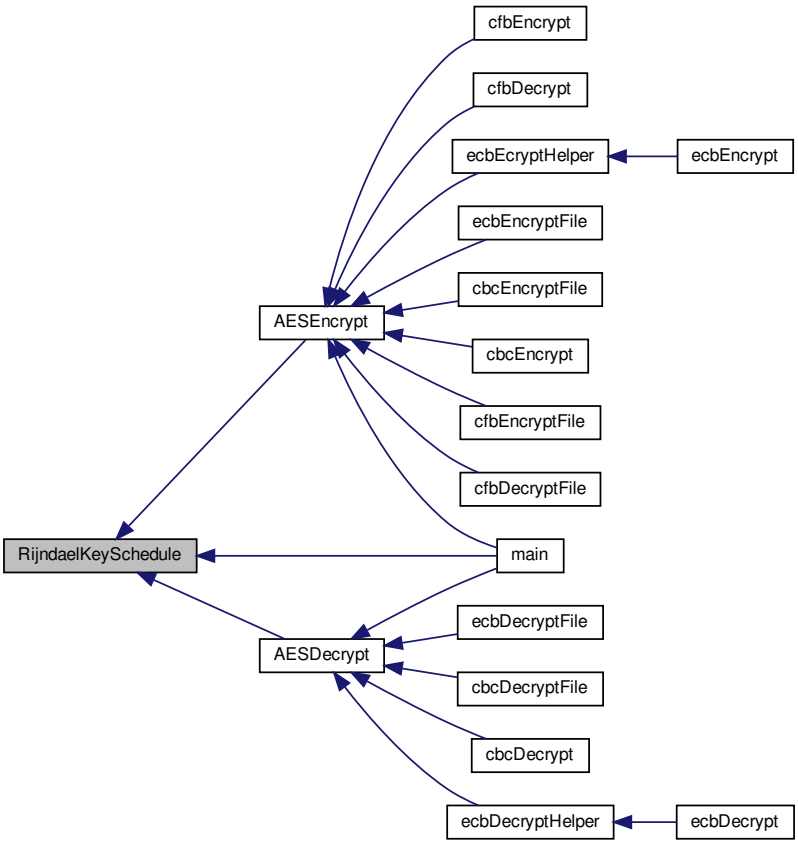
expandedKey - The key that has been expanded.

Definition at line 577 of file AES.c.

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.2.29 ShiftRows()

```
void ShiftRows (
            unsigned char state[4][4],
            int wordLength )
```

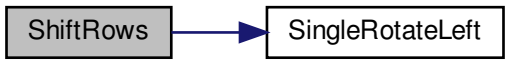
ShiftRows - Function to shift the state array according to the AES encryption standard for 128 - bits blocks.

Parameters

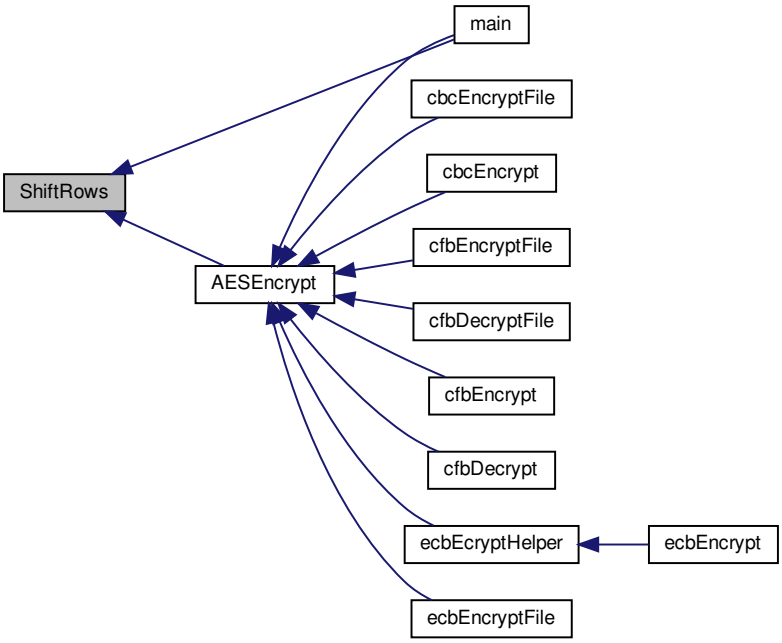
state	- unsigned char - is the current state of the ciphertext or plaintext during AES encryption or decryption

Definition at line 815 of file AES.c.

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.2.30 SingleRotateLeft()

```
void SingleRotateLeft (
            unsigned char * word,
            int wordLength )
```

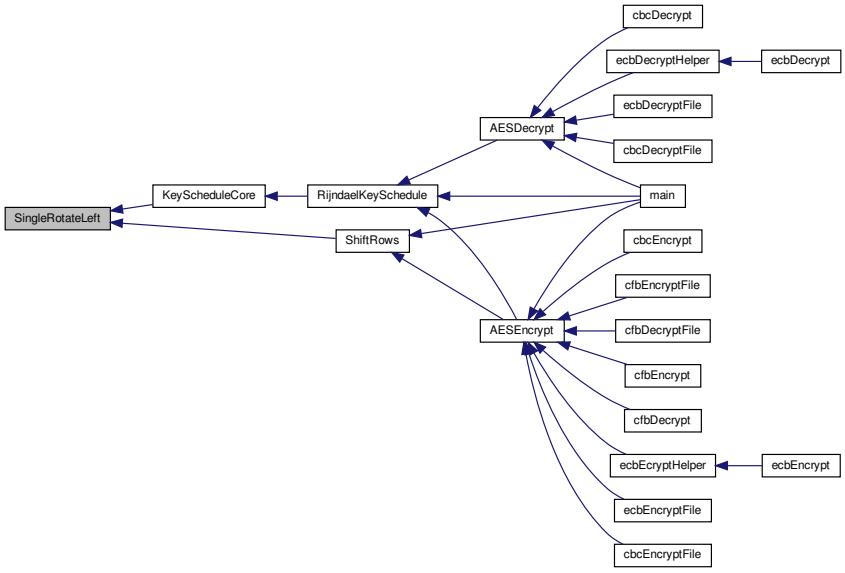
SingleRotateLeft - Function to rotate the array passed in as a paramter.

Parameters

word,a	single time left (8 bits to the left), with the left most element becoming the right most element. As such: rotate(1d2c3a4f) = 2c3a4f1d.
word	- unsigned char ∗word - the array/word to be left rotated by 8 bits.
wordLength	- int - length of the parameter
word.	

Definition at line 655 of file AES.c.

Here is the caller graph for this function:



3.1.2.31 SingleRotateRight()

```
void SingleRotateRight (
            unsigned char * word,
            int wordLength )
```

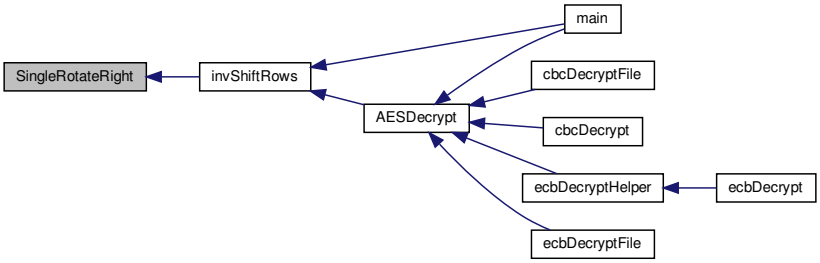
SingleRotateRight - Function to rotate the array passed in as a paramter.

Parameters

word,a	single time right (8 bits to the right), with the right most element becoming the left most element. As such: rotate(1d2c3a4f) = 4f1d2c3a.
word	- unsigned char ∗word - the array/word to be right rotated by 8 bits.
wordLength	- int - length of the parameter
word.	

Definition at line 857 of file AES.c.

Here is the caller graph for this function:



3.1.2.32 stripDirectory()

```
void stripDirectory (
            char * fileName,
            char * extractedFileName,
            char * extractedFilePath,
            int fileNameLength,
            int slashIndex )
```

Removes path from the provided path to a file and returns only the file name.

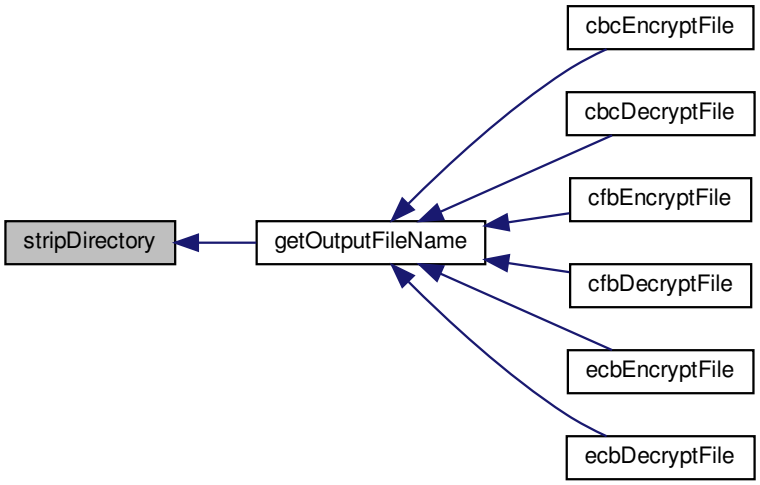
stripDirectory - Function that removes path from the provided path to a file and returns only the file name

Parameters

fileName	The path to a specified file
extractedFileName	The name of the file within the provided path to a file
extractedFilePath	The path to file, excluding the file name
fileNameLength	The length of the paramter
fileName	
slashIndex	The index of the last '/' in the original file path passed in as a paramter
fileName	

Definition at line 1063 of file AES.c.

Here is the caller graph for this function:



3.1.2.33 subBytes()

```
void subBytes (
            unsigned char state[4][4] )
```

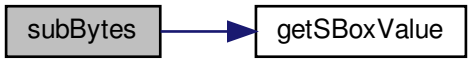
subBytes - Function that performs the sub byte operation where each value is replaced by the s box value

Parameters

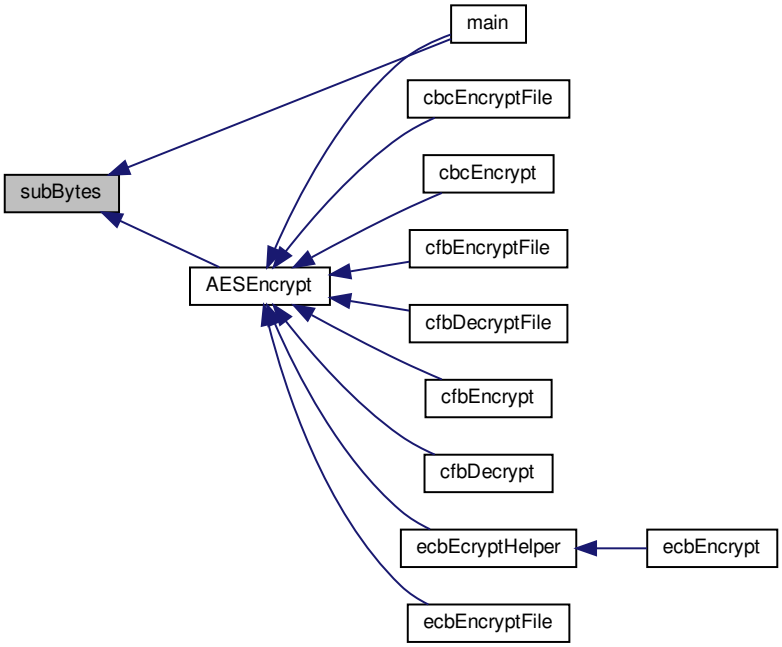
state	- unsigned char - is the current state of the ciphertext or plaintext during AES encryption or decryption.

Definition at line 789 of file AES.c.

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.2.34 validateCipherTextLength()

```
void validateCipherTextLength (
            int cipherTextLength )
```

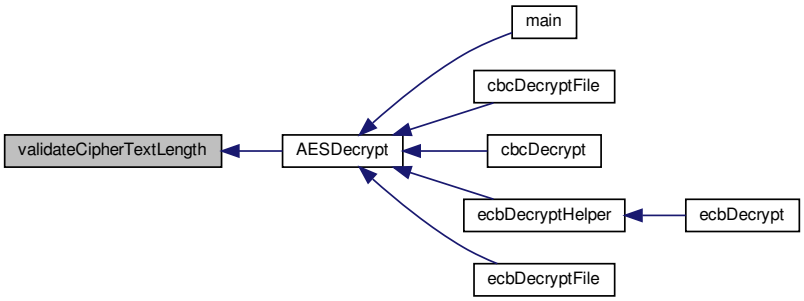
validateCipherTextLength - Function that validates the length of the ciphertext. The validation is done against the AES_BLOCK_SIZE value

Parameters

cipherTextLength	- int - The length of the cipher text as an integer value

Definition at line 1015 of file AES.c.

Here is the caller graph for this function:



3.1.2.35 validateNumRounds()

```
void validateNumRounds (
            int numRounds,
            int keyLength )
```

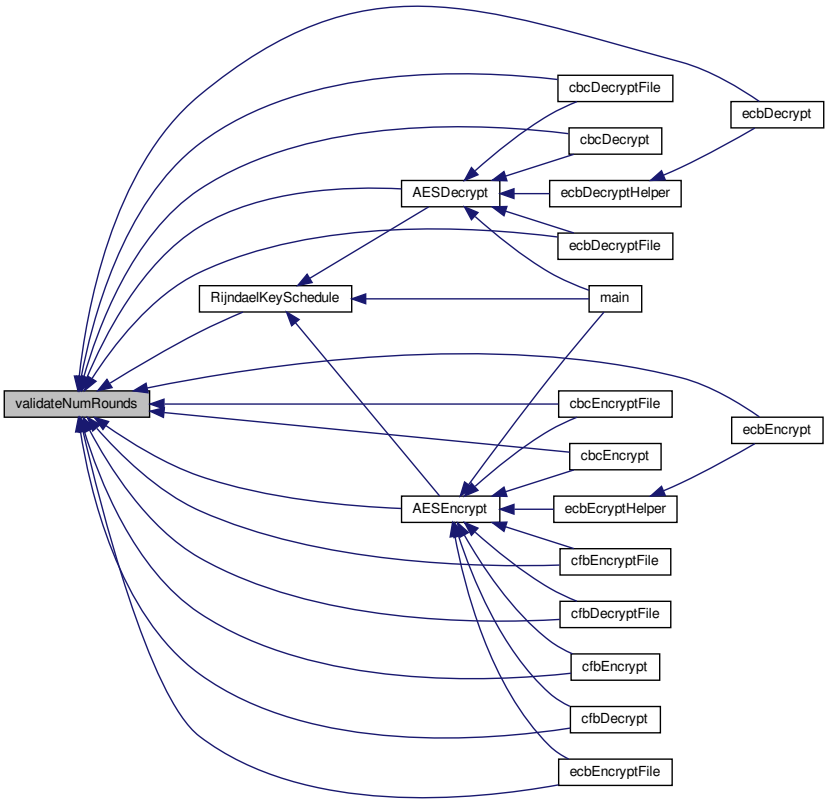
validateNumRounds - Function that validates the number of rounds that have been passed in by the

Parameters

numRounds.	Upon invalid validation, relevent error information will be printed to terminal and the program will exit with an EXIT_FAILURE flag.
numRounds	- int - Integer value of the number rounds

Definition at line 989 of file AES.c.

Here is the caller graph for this function:



3.1.2.36 validatePlainTextLength()

```
void validatePlainTextLength (
            size_t plainTextLength )
```

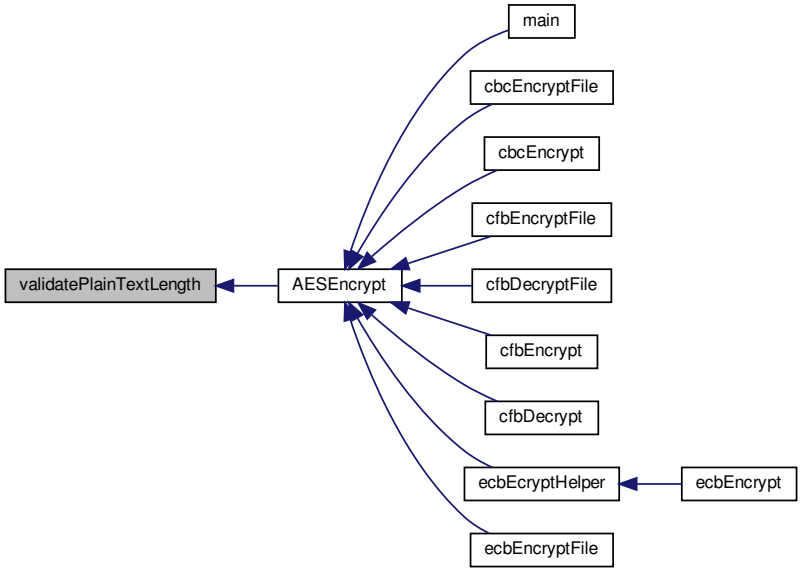
validatePlainTextLength - Function that validates the length of the plaintext. The validation is done against the AES_BLOCK_SIZE value

Parameters

plainTextLength	- int - The length of the plaintext text as an integer value

Definition at line 1002 of file AES.c.

Here is the caller graph for this function:



3.1.2.37 XORBlocks()

```
unsigned char* XORBlocks (
            unsigned char * block1,
            unsigned char * block2,
            int length )
```

XORBlocks - Function to XOR two blocks of length.

Parameters

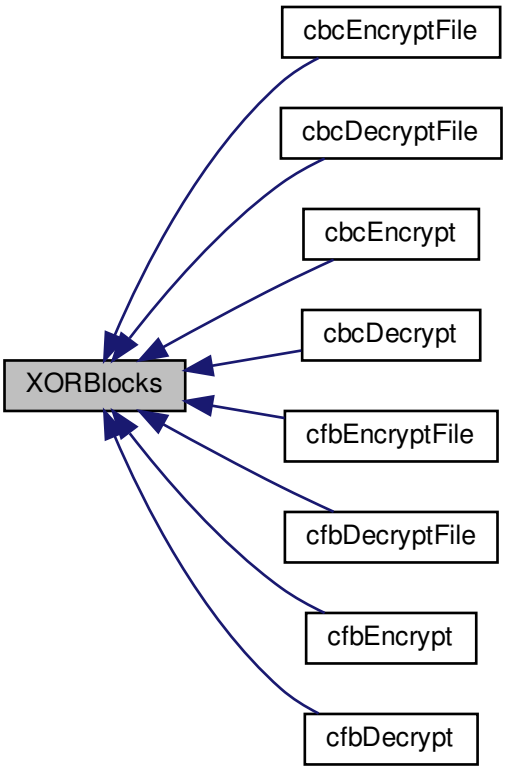
length	and retuns the XOR'd result. User must free the memory returned.
char	- block1 - First block to be XOR'd.
char	- block2 - Second block to be XOR'd.
length	- length of the blocks to be XOR'd.

Returns

unsigned* - Result of the XOR.

Definition at line 1228 of file AES.c.

Here is the caller graph for this function:



3.1.3 Variable Documentation

3.1.3.1 AES_BLOCK_SIZE

```
const size_t AES_BLOCK_SIZE = 16
```

Variable- const size_t AES_BLOCK_SIZE. Used to dictate the length in bytes of a single AES block used for encryption and decryption. Set to 16 bytes for a single block.

Variable - AES_BLOCK_SIZE - specifies the length per AES block - 16 bytes.

Definition at line 24 of file AES.c.

3.1.3.2 invSBox

```
const unsigned char invSBox[256]
```

Initial value:

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

const unsigned char invSBox. Lookup table for the inverse sbox values used during AES Decryption.

Definition at line 60 of file AES.c.

3.1.3.3 Rcon

```
const unsigned char Rcon[255]
```

Initial value:

```
= {
    0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8,
    0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3,
    0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f,
    0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d,
    0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab,
    0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25,
    0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d,
    0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01,
    0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d,
    0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa,
    0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a,
    0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02,
    0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a,
    0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef,
    0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94,
    0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04,
    0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f,
    0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5,
    0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33,
    0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb }
```

const unsigned char Rcon. Lookup table for the Rcon values used during Rijndael Key Schedule during the AES Encryption and Decryption.

Definition at line 83 of file AES.c.

3.1.3.4 sbox

```
const unsigned char sbox[256]
```

Initial value:

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

const unsigned char sbox. Lookup table for the sbox values used during AES Encryption.

Definition at line 37 of file AES.c.

3.1.3.5 VERBOSE

```
size_t VERBOSE = 0
```

Variable- size_t VERBOSE Used to dictate whether verbose output is printed to the terminal or not. If 0, does not print verbose. If 1, prints verbose.

Variable - VERBOSE - specifies if verbose output should be printed or not.

Definition at line 31 of file AES.c.

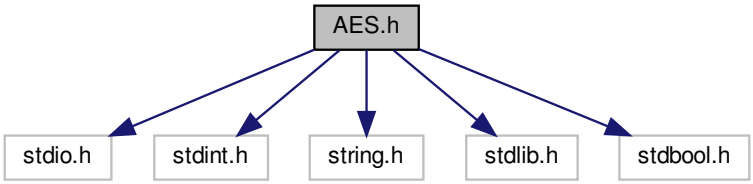
3.2 AES.h File Reference

AES encryption and decryption module header file. This file contains the function headers for the functions used for AES encryption and decryption. Input must be ASCII and not hex. The functions implemented in this file, perform the AES encryption and decryption on a single block of size dictated by the variable AES_BLOCK_SIZE.

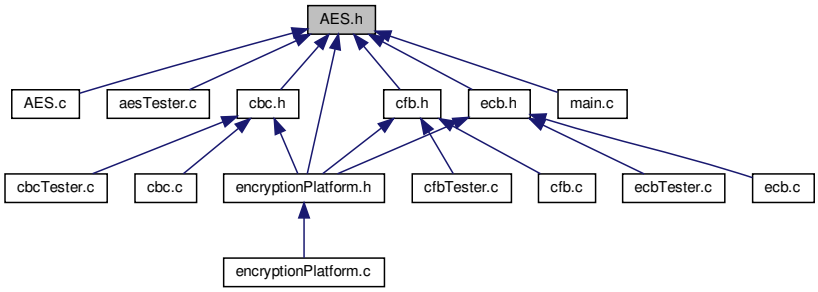
```
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include <stdlib.h>
```

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

Include dependency graph for AES.h:



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



Functions

- int getNumRounds (int)

  getNumRounds - Function to return the number of rounds of AES encryption and decryption based off of the length of the key given in

- unsigned char * AESEncrypt (unsigned char *, unsigned char *, int, int)

  AESEncrypt - Function to encrypt a single block of plaintext passed in as parameter plainText using AES encryption, for 128, 192 and 256 bit keys. Validates the keylength and returns the corresponding ciphertext. The caller of the function must ensure that the returned ciphertext pointer is freed. The ciphertext returned is always 16 bytes and the plainText must be 16 bytes or less. Makes use of zero padding. All input must be in ASCII and NOT hex.

- unsigned char * AESDecrypt (unsigned char *, unsigned char *, int, int)

  AESDecrypt - Function to decrypt a single block of ciphertext passed in as parameter cipherText using AES decryption, for 128, 192 and 256 bit keys. Validates the keylength and returns the corresponding plaintext. The caller of the function must ensure that the returned plaintext pointer is freed. The plaintext returned is always 16 bytes and the plainText must be 16 bytes or less. Makes use of zero padding. All input must be in ASCII and NOT hex.

- unsigned char * RijndaelKeySchedule (unsigned char *, int)

  RijndaelKeySchedule - Function that performs the Rijndael key scheduling for AES encryption. Takes in the original key passed in as parameter.

- void printStateArray (uint8_t[4][4])

  printStateArray - Function to print the state array to the terminal in hex format.

- unsigned char getSBoxValue (unsigned char)

  getSBoxValue - Function to return the sBox value passed in as a parameter

- unsigned char getInvSBox (unsigned char)

  getInvSBox - Function to return the inverse sBox value passed in as a parameter
- unsigned char getRconValue (unsigned char)

  getRconValue - Function to return the Rcon value for the index passed in as a parameter
- void SingleRotateLeft (unsigned char ∗, int)

  SingleRotateLeft - Function to rotate the array passed in as a paramter.
- void KeyScheduleCore (unsigned char ∗, int, int)

  KeyScheduleCore - Function that performs the key schedule core for the Rijndael Key Schedule. Performs a single rotate left of the word passed in as.
- void ShiftRows (unsigned char[4][4], int)

  ShiftRows - Function to shift the state array according to the AES encryption standard for 128 - bits blocks.
- void AddRoundKey (unsigned char[4][4], unsigned char[4][4])

  AddRoundKey - Function that performs the Bitwise XOR between state and key as per AES encryption.
- void subBytes (unsigned char[4][4])

  subBytes - Function that performs the sub byte operation where each value is replaced by the s box value
- void mixColumns (unsigned char[4][4])

  mixColumns - Function that performs the MixColumns step of AES as specified by AES encryption.
- void invSubBytes (unsigned char[4][4])

  invSubBytes - Function that performs the inverse of Function subBytes
- void invShiftRows (unsigned char[4][4], int)

  invShiftRows - Function to shift the state array Inverse according to the AES encryption standard for 128 - bits blocks
- void SingleRotateRight (unsigned char ∗, int)

  SingleRotateRight - Function to rotate the array passed in as a paramter.
- int getPaddedKeyLength (int)

  getPaddedKeyLength - Function to return a valid key length (in bytes) based off of the current key length passed in as
- void invMixColumns (unsigned char[4][4])

  invMixColumns - Function that does the inverse of the Mix Column Step for AES Encryption. Performs the gallois field multiplication and the required XOR to the state passed in as a paramter
- unsigned char galloisFieldMult (unsigned char, unsigned char)

  galloisFieldMult - Function to perform the Galois field multiplication operation required for the inverse mix columns and the mix columns operation of the AES encryption and decryption processes. Returns the result of the multiplication.
- void getRoundKey (unsigned char ∗, unsigned char ∗, int)

  getRoundKey - Function to extract the correct sub-key to use for the appropriate round specified by
- void constructStateArray (unsigned char ∗, unsigned char[ ][4])

  constructStateArray - Function to convert the state array from a flat 1D array to a multidimensional array.
- uint8_t hexToInt (char c)

  hexToInt - Function that converts a given hex value into an integer.
- uint8_t hexToAscii (char c, char d)

  hexToAscii - Function that converts a given hex value to its ASCII equivalent.
- void hexToAsciiString (char ∗str, char ∗done, int)

  hexToAsciiString - Function that converts a given string of hex values into its ASCII equivalent. A hex string contains hex chars and is "encoded" in ascii In order to encrypt it, it must be converted to the equivalent ascii plain text string plaintext string is half the size of hex, since two hex chars = 1 ascii char if hex string is "4A" it will be converted to "J" in ascii which will have a hex representation of "4a" The original hex string converted to hex staright or printed in hex straight rather will print or have the value "0x34", "0x31" BASICALLY THE HEX STRING FF IS INTERPRETED AS THE CHARS FF, whereas when using this function we intend it to be "J", ie the char "J"
- unsigned char ∗ asciiToHexString (unsigned char ∗asciiString, unsigned char ∗hexString, size_t asciiString↩ Len)

  Function name: asciiToHexString - convert an ascii String to an ascii string.
- void validateNumRounds (int numRounds, int keyLength)

  validateNumRounds - Function that validates the number of rounds that have been passed in by the
- void validatePlainTextLength (size_t plainTextLength)

validatePlainTextLength - Function that validates the length of the plaintext. The validation is done against the AE↩ S_BLOCK_SIZE value

- void validateCipherTextLength (int cipherTextLength)

  validateCipherTextLength - Function that validates the length of the ciphertext. The validation is done against the AES_BLOCK_SIZE value
- void printAESBlock (unsigned char ∗temp)

  printAESBlock - Function to print a single block in hex format to the terminal.
- int fileNameDirIndex (char ∗fileName, int fileNameLength)

  Returns the last index of '/' in a given path, otherwise returns -1 if no '/' is found.
- void stripDirectory (char ∗fileName, char ∗extractedFileName, char ∗extractedFilePath, int fileNameLength, int slashIndex)

  stripDirectory - Function that removes path from the provided path to a file and returns only the file name
- void getOutputFileName (int type, char ∗fileName, char ∗outputFileName, char ∗)

  Get the output file name from all the parameters passed in.
- uint8_t isFileTxt (unsigned char ∗fileName)

  isFileTxt - Function to determine if the file passed in as a paramter
- unsigned char ∗ keyHexToAscii (unsigned char ∗hexKey, int keyLength)

  keyHexToAscii - Function to convert a key from a Hex string passed in as a paramter
- unsigned char ∗ IVHexToAscii (unsigned char ∗hexIV, int IVLength)

  IVHexToAscii - Function to convert a initialization vector from a Hex string passed in as a paramter.
- unsigned char ∗ XORBlocks (unsigned char ∗block1, unsigned char ∗block2, int length)

  XORBlocks - Function to XOR two blocks of length.

Variables

- size_t VERBOSE

  Variable- size_t VERBOSE Used to dictate whether verbose output is printed to the terminal or not. If 0, does not print verbose. If 1, prints verbose.
- const size_t AES_BLOCK_SIZE

  Variable- const size_t AES_BLOCK_SIZE. Used to dictate the length in bytes of a single AES block used for encryption and decryption. Set to 16 bytes for a single block.
- const unsigned char invSBox [256]

  const unsigned char invSBox. Lookup table for the inverse sbox values used during AES Decryption.
- const unsigned char sbox [256]

  const unsigned char sbox. Lookup table for the sbox values used during AES Encryption.
- const unsigned char Rcon [255]

  const unsigned char Rcon. Lookup table for the Rcon values used during Rijndael Key Schedule during the AES Encryption and Decryption.

3.2.1 Detailed Description

AES encryption and decryption module header file. This file contains the function headers for the functions used for AES encryption and decryption. Input must be ASCII and not hex. The functions implemented in this file, perform the AES encryption and decryption on a single block of size dictated by the variable AES_BLOCK_SIZE.

Authors

Mohamed Ameen Omar (u16055323)
Douglas Healy (u16018100)
Llewellyn Moyse (u15100708)

Version

0.1

Date

2019-03-20

Copyright

3.2.2 Function Documentation

3.2.2.1 AddRoundKey()

```
void AddRoundKey (
            unsigned char state[4][4],
            unsigned char key[4][4] )
```

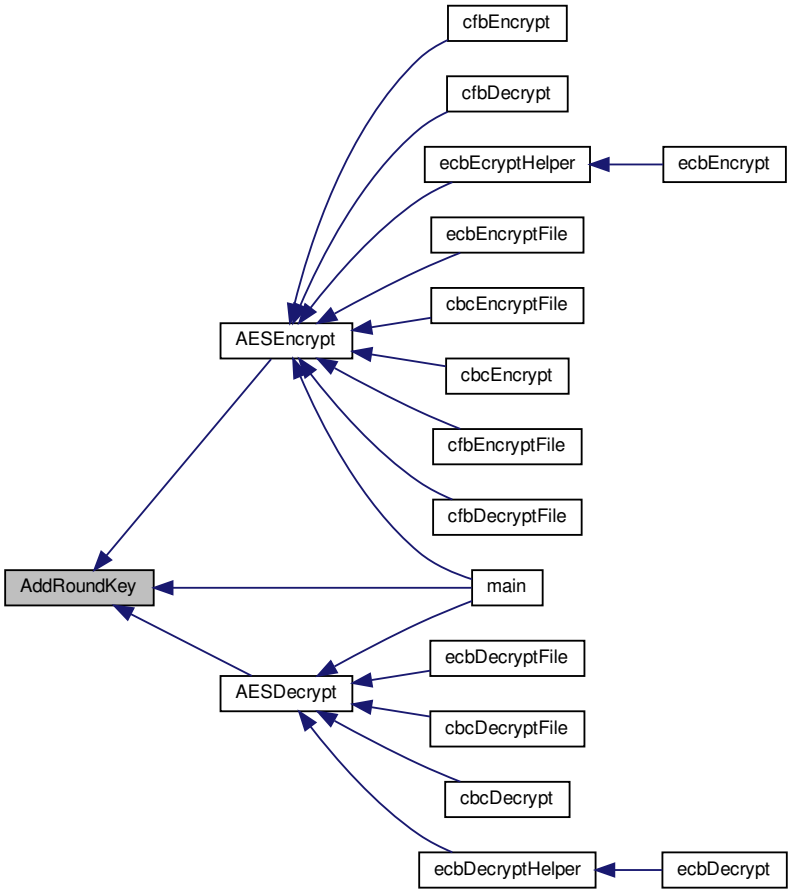
AddRoundKey - Function that performs the Bitwise XOR between state and key as per AES encryption.

Parameters

state	- unsigned char - is the current state of the ciphertext or plaintext during AES encryption or decryption
key	- unsigned char - sub key to be added for the current round to the current state vector

Definition at line 688 of file AES.c.

Here is the caller graph for this function:



3.2.2.2 AESDecrypt()

```
unsigned char* AESDecrypt (
            unsigned char * cipherText,
            unsigned char * key,
            int cipherTextLength,
            int keyLength )
```

AESDecrypt - Function to decrypt a single block of ciphertext passed in as parameter cipherText using AES decryption, for 128, 192 and 256 bit keys. Validates the keylength and returns the corresponding plaintext. The caller of the function must ensure that the returned plaintext pointer is freed. The plaintext returned is always 16 bytes and the plainText must be 16 bytes or less. Makes use of zero padding. All input must be in ASCII and NOT hex.

Parameters

char	- unsigned char* cipherText - pointer to the ciphertext that needs to be decrypted using AES decryption.

Parameters

char	- unsigned char∗ key - reference to the key that must be used for AES decryption.
cipherTextLength	- length of the ciphertext in
cipherText	to be decrypted.
keyLength	- length of the key passed in as
key	used for the AES decryption.

Returns

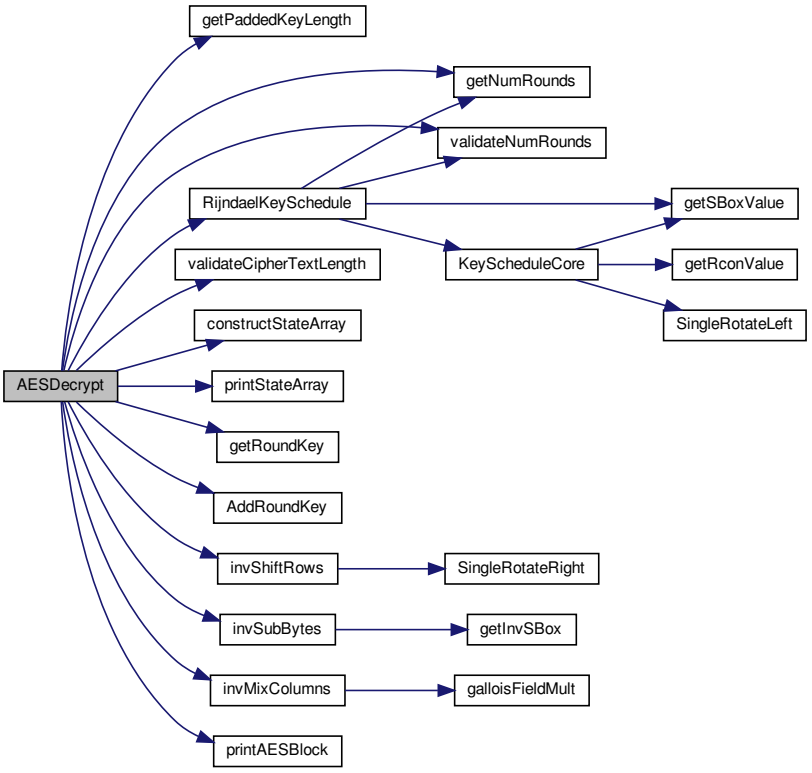
unsigned∗ char - Plaintext resulting from the decryption of the ciphertext passed in as

Parameters

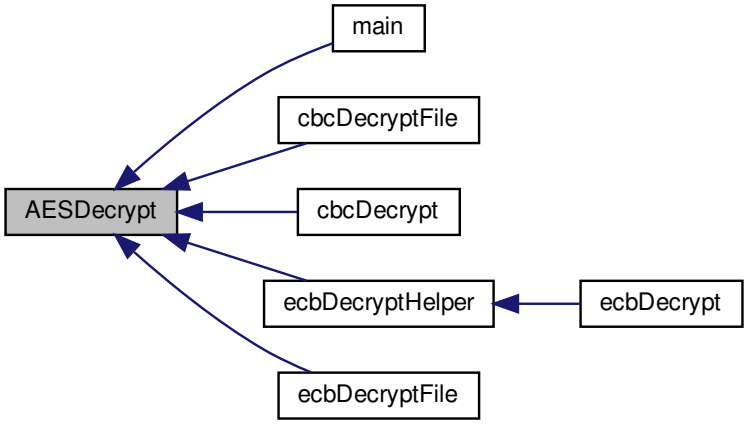
cipherText.	

Definition at line 399 of file AES.c.

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.2.3 AESEncrypt()

```
unsigned char* AESEncrypt (
            unsigned char * plainText,
            unsigned char * key,
            int plainTextLength,
            int keyLength )
```

AESEncrypt - Function to encrypt a single block of plaintext passed in as parameter plainText using AES encryption, for 128, 192 and 256 bit keys. Validates the keylength and returns the corresponding ciphertext. The caller of the function must ensure that the returned ciphertext pointer is freed. The ciphertext returned is always 16 bytes and the plainText must be 16 bytes or less. Makes use of zero padding. All input must be in ASCII and NOT hex.

Parameters

char	- unsigned char* plainText - pointer to the plaintext that needs to be encrypted using AES encryption.
char	- unsigned char* key - reference to the key that must be used for AES encryption.
plainTextLength	- length of the plaintext in
plainText	to be encrypted.
keyLength	- length of the key passed in as
key	used for the AES encryption.

Returns

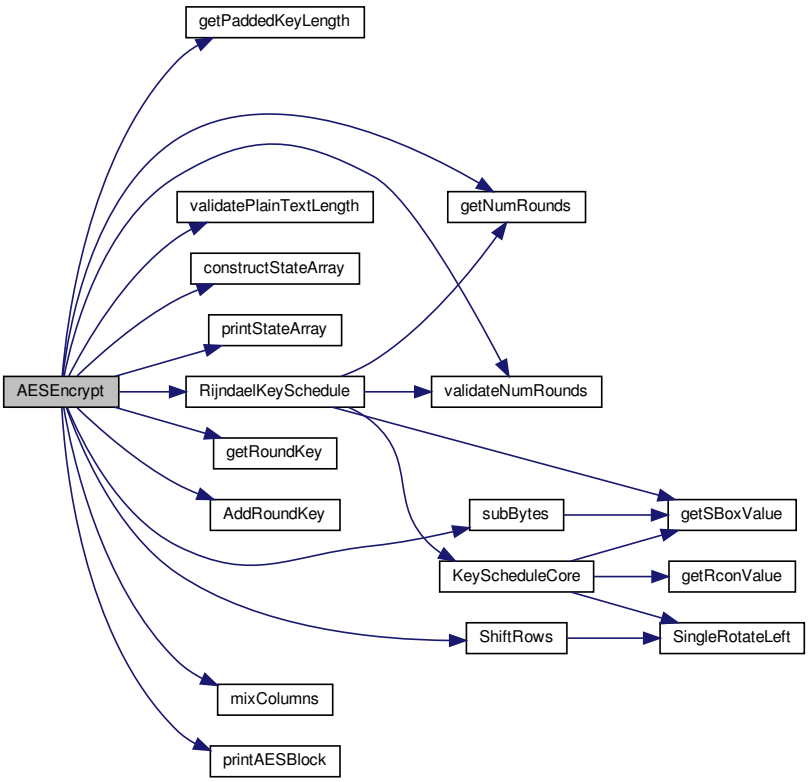
unsigned* char - Ciphertext resulting from the encryption of the plaintext passed in as

Parameters

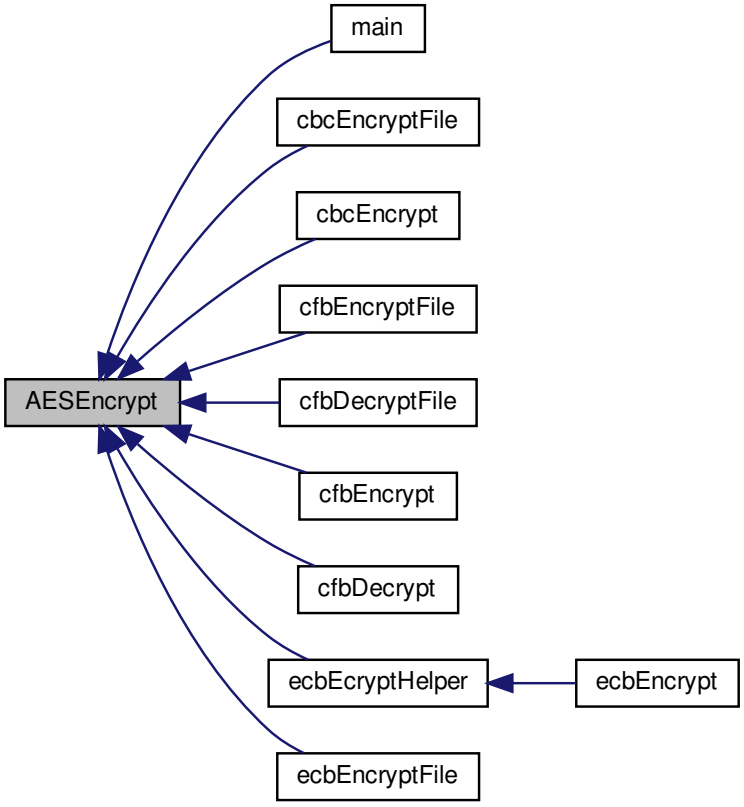
plainText.	

Definition at line 198 of file AES.c.

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.2.4 asciiToHexString()

```
unsigned char* asciiToHexString (
            unsigned char * asciiString,
            unsigned char * hexString,
            size_t asciiStringLen )
```

Function name: asciiToHexString - convert an ascii String to an ascii string.

Parameters

asciiString	- unsigned char* pointing to the ASCII String to be converted.
hexString	- unsigned char* pointing to a memory where the converted Hex string should be stored.
asciiStringLen	- size_t containing the length of the ASCII String to be converted.

Returns

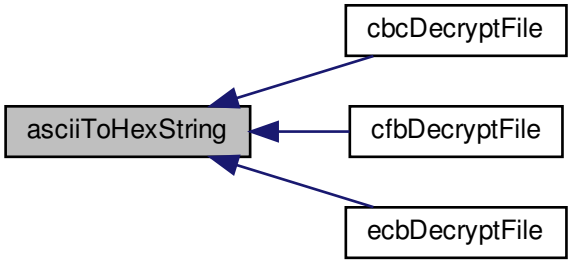
unsigned char* asciiToHexString - pointer to the converted Hex String, pointing to the same memory location as

Parameters

hexString.	

Definition at line 971 of file AES.c.

Here is the caller graph for this function:



3.2.2.5 constructStateArray()

```
void constructStateArray (
            unsigned char * flatArray,
            unsigned char stateArray[][4] )
```

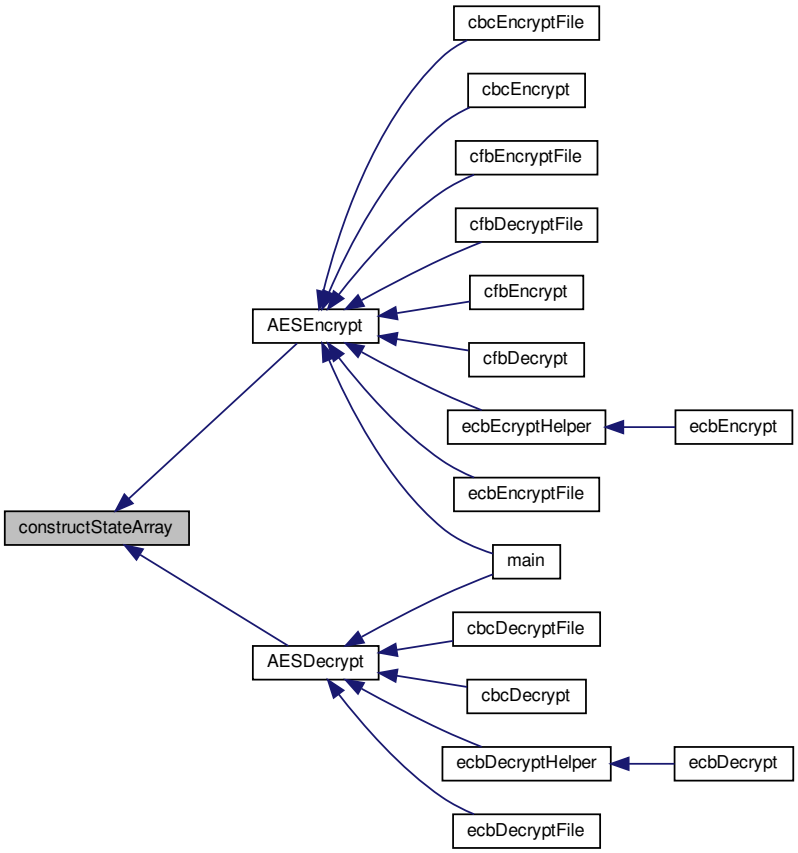
constructStateArray - Function to convert the state array from a flat 1D array to a multidimensional array.

Parameters

char	flatArray -the 1D array to be converted.
stateArray	- the multidimensional array to which to copy the flat array elements to.

Definition at line 895 of file AES.c.

Here is the caller graph for this function:



3.2.2.6 fileNameDirIndex()

```
int fileNameDirIndex (
            char * fileName,
            int fileNameLength )
```

Returns the last index of '/' in a given path, otherwise returns -1 if no '/' is found.

Parameters

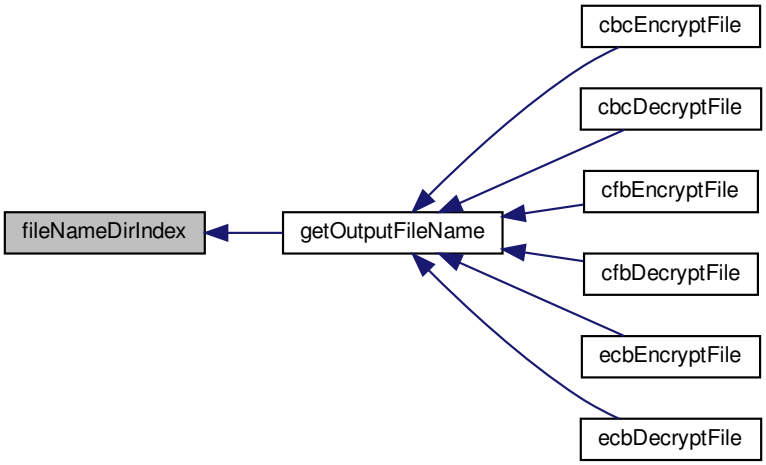
fileName	The path to a file
fileNameLength	The length of the provided file

Returns

int Index of the last '/' in the path, else -1 if no '/' was found

Definition at line 1043 of file AES.c.

Here is the caller graph for this function:



3.2.2.7 galloisFieldMult()

```
unsigned char galloisFieldMult (
            unsigned char a,
            unsigned char b )
```

galloisFieldMult - Function to perform the Galois field multiplication operation required for the inverse mix columns and the mix columns operation of the AES encryption and decryption processes. Returns the result of the multiplication.

Parameters

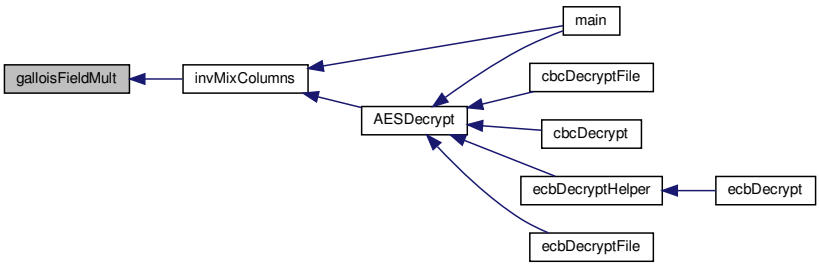
a	- first character to perform Galois field multiplication.
b	- second character to perform Galois field multiplication.

Returns

unsigned char - Result of the Galois field multiplication.

Definition at line 764 of file AES.c.

Here is the caller graph for this function:



3.2.2.8 getInvSBox()

```
unsigned char getInvSBox (
            unsigned char index )
```

getInvSBox - Function to return the inverse sBox value passed in as a parameter

Parameters

index.	Requires the original value required in hex.
index	- unsigned char - hexadecimal representation of the index for which the inverse SBox value is required.

Returns

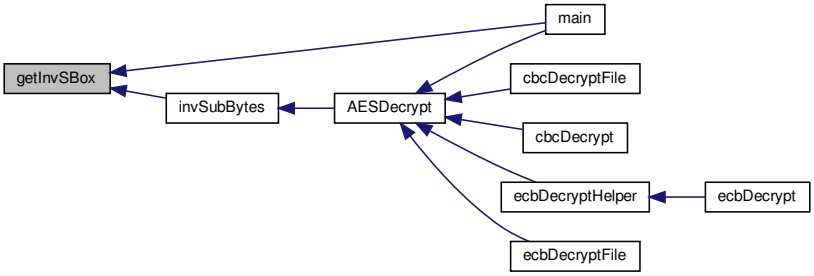
unsigned char - inverse sBox value for the paramter

Parameters

index.	

Definition at line 146 of file AES.c.

Here is the caller graph for this function:



3.2.2.9 getNumRounds()

```
int getNumRounds (
            int keyLength )
```

getNumRounds - Function to return the number of rounds of AES encryption and decryption based off of the length of the key given in

Parameters

keyLength.	
keyLength	- int - indicates the length of the key

Returns

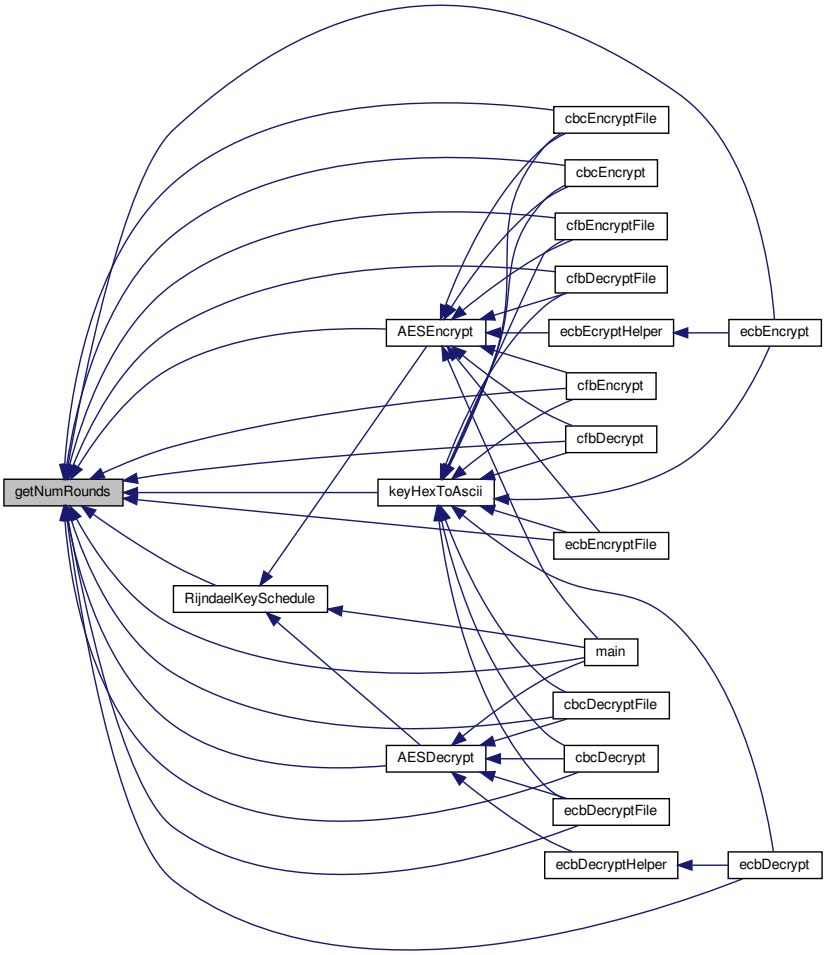
int - the number of rounds based off of the length of the key passed in the parameter

Parameters

keyLength.	If the length of the key is not valid, returns -1.

Definition at line 114 of file AES.c.

Here is the caller graph for this function:



3.2.2.10 getOutputFileName()

```
void getOutputFileName (
            int type,
            char * fileName,
            char * outputFileName,
            char * mode )
```

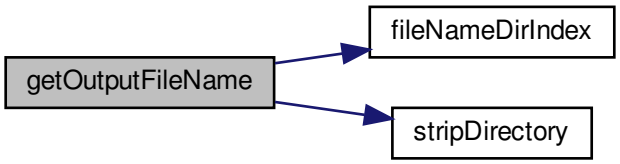
Get the output file name from all the parameters passed in.

Parameters

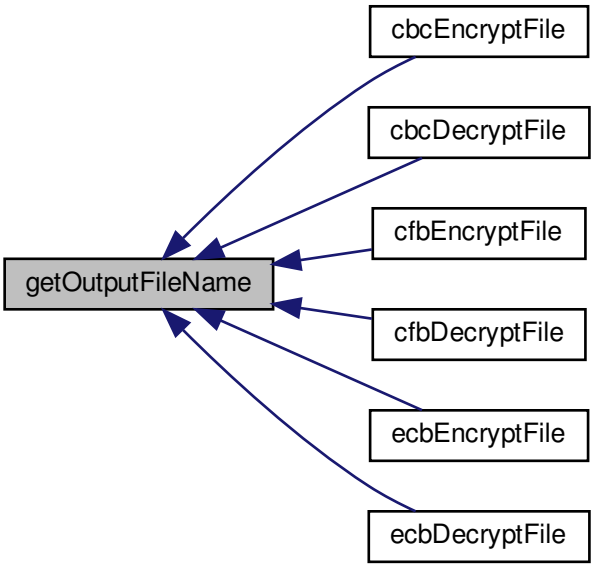
type	0 - Encrypt, 1 - Decrypt
fileName	The name of the input file
outputFileName	The name of the output file
mode	Chipher mode to be used (ECB, CBC, CFB)

Definition at line 1086 of file AES.c.

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.2.11 getPaddedKeyLength()

```
int getPaddedKeyLength (
            int currentKeyLength )
```

getPaddedKeyLength - Function to return a valid key length (in bytes) based off of the current key length passed in as

Parameters

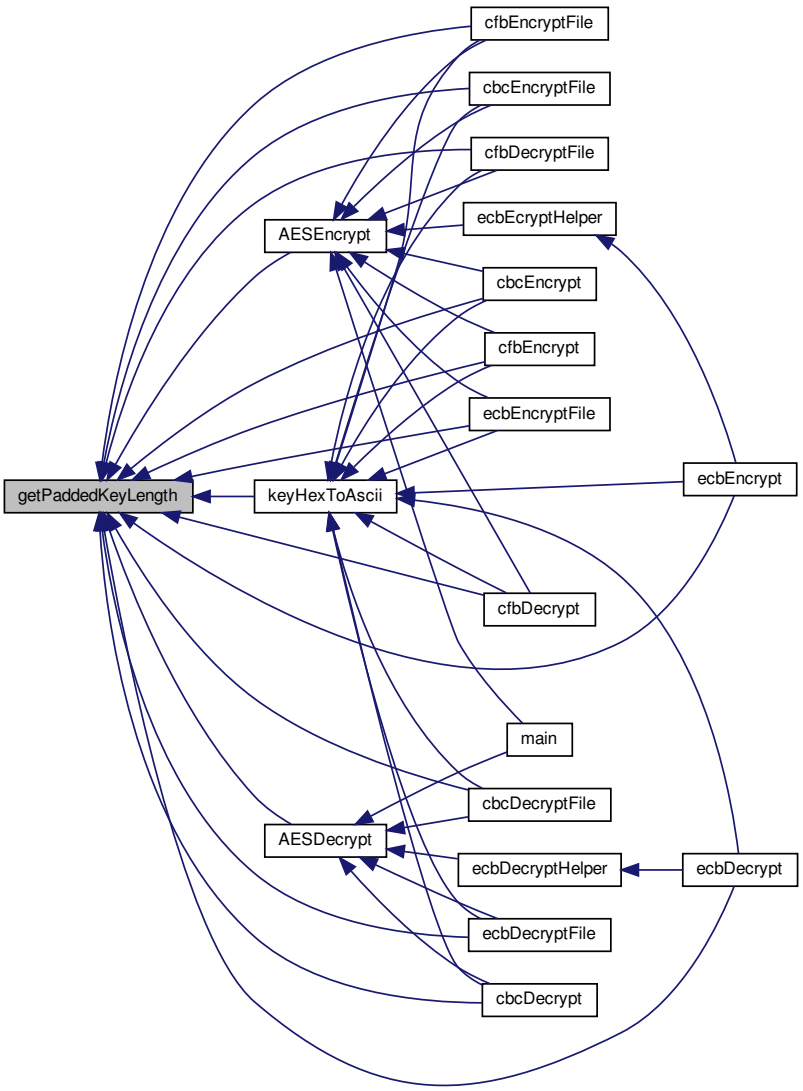
currentKeyLength.	Corresponds to minimum and maximum key length required for AES encryption and decryption. The key will then be padded to the length of the value returned from this function. If the keylength is less than 16, will return 16. If greater than 16, but less than 24, will return 24. If greater than 32, will return -1.
currentKeyLength	- int - current key length in bytes, to be padded to the return value

Returns

int - the length in bytes that the key should be padded to.

Definition at line 172 of file AES.c.

Here is the caller graph for this function:



3.2.2.12 getRconValue()

```
unsigned char getRconValue (
            unsigned char num )
```

getRconValue - Function to return the Rcon value for the index passed in as a parameter

Parameters

num.	Requires the original value required in hex.
index	- unsigned char - hexadecimal representation of the number for which the Rcon value is required during the key schedule.

Returns

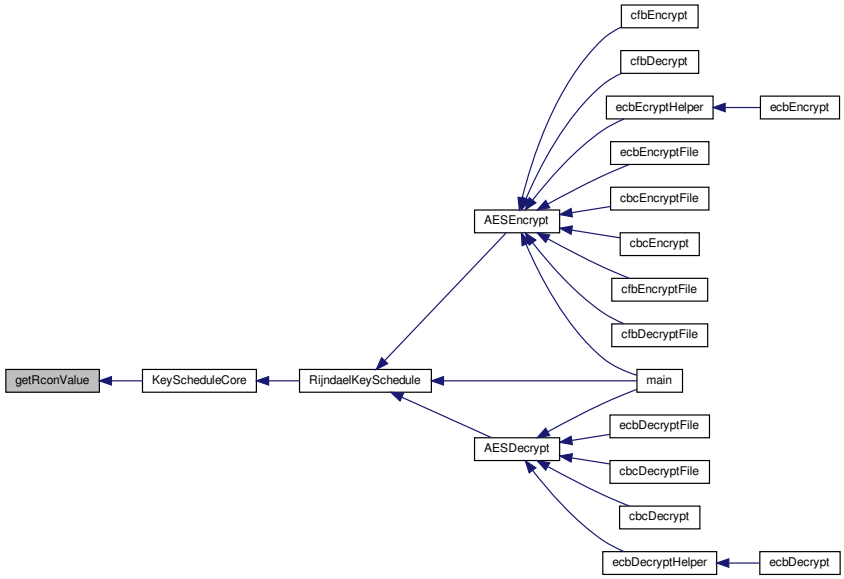
unsigned char - rCon value for the paramter

Parameters

num.	

Definition at line 158 of file AES.c.

Here is the caller graph for this function:



3.2.2.13 getRoundKey()

```
void getRoundKey (
            unsigned char * expandedKey,
            unsigned char * roundKey,
            int roundNum )
```

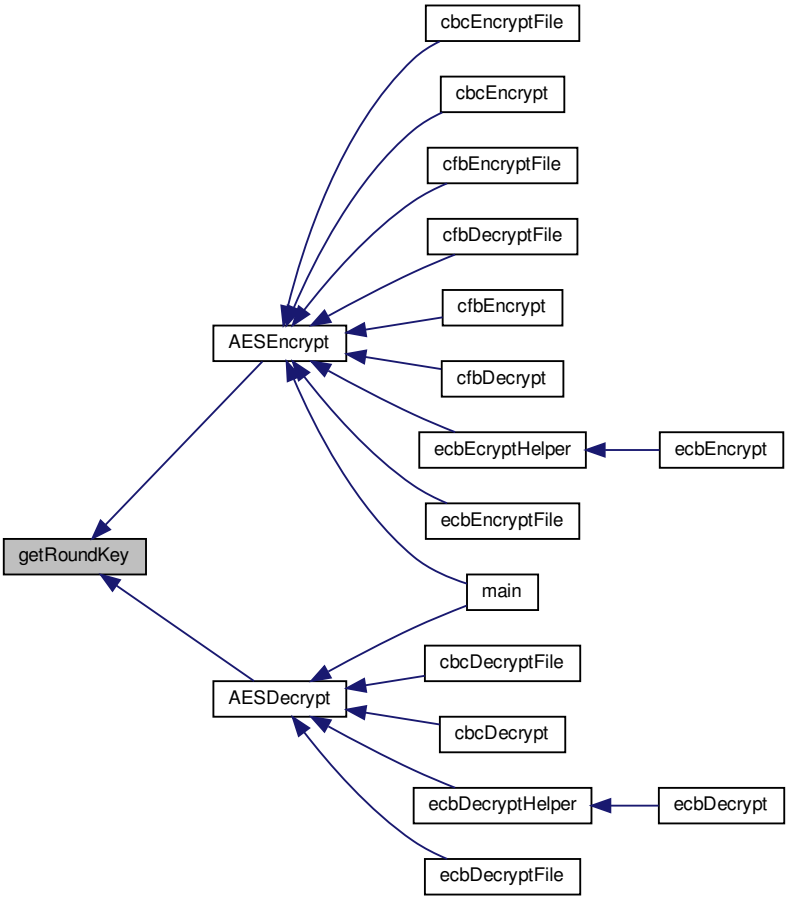
getRoundKey - Function to extract the correct sub-key to use for the appropriate round specified by

Parameters

roundNum.	Copies the sub-key from the expanded key in
expandedKey	to
roundKey.	
char	- expandedKey - The expanded key from which to extract the sub-key.
char	- roundKey - memory to which to copy the sub-key.
roundNum	- int - the round number for which the sub-key is required.

Definition at line 881 of file AES.c.

Here is the caller graph for this function:



3.2.2.14 getSBoxValue()

```
unsigned char getSBoxValue (
            unsigned char index )
```

getSBoxValue - Function to return the sBox value passed in as a parameter

Parameters

index.	Requires the original value required in hex.
index	- unsigned char - hexadecimal representation of the index for which the SBox value is required.

Returns

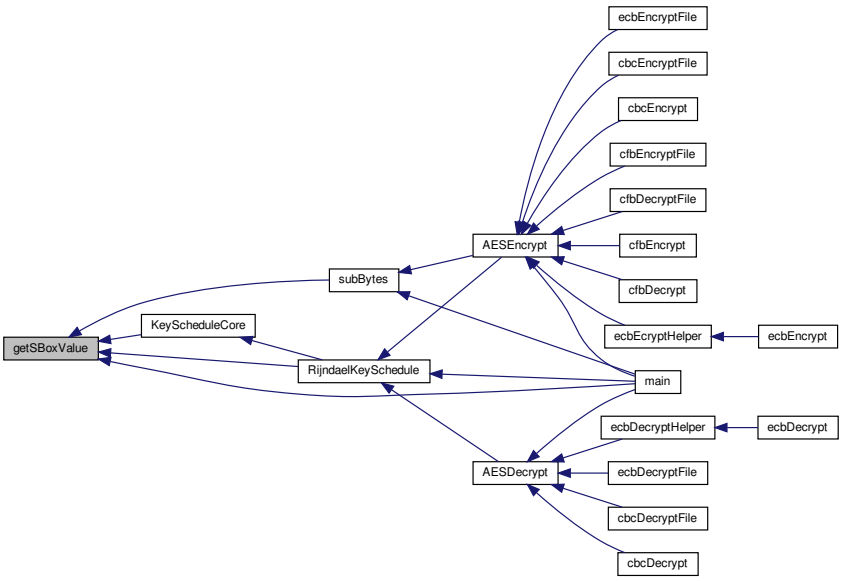
unsigned char - sBox value for the paramter

Parameters

index.	

Definition at line 134 of file AES.c.

Here is the caller graph for this function:



3.2.2.15 hexToAscii()

```
uint8_t hexToAscii (
            char ch1,
            char ch2 )
```

hexToAscii - Function that converts a given hex value to its ASCII equivalent.

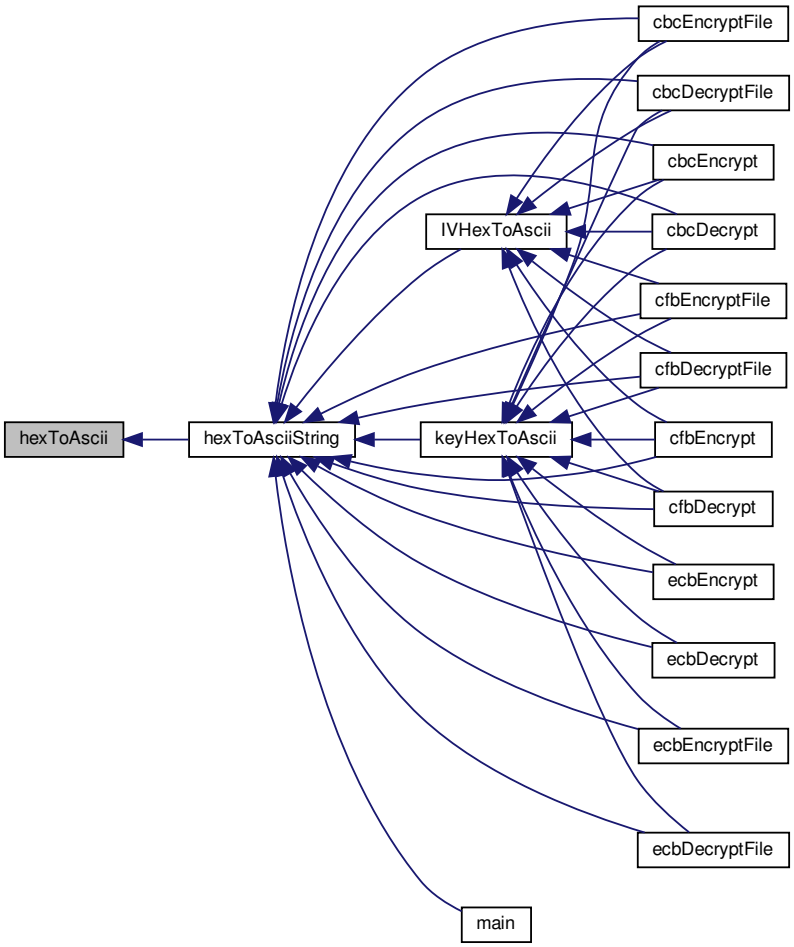
Parameters

ch1	- char value of the first hex value.
ch2	- char value of the second hex value.

Definition at line 928 of file AES.c.

Here is the call graph for this function:

Here is the caller graph for this function:



3.2.2.16 hexToAsciiString()

```
void hexToAsciiString (
            char * hexString,
            char * asciiString,
            int hexStringLength )
```

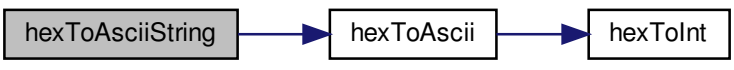
hexToAsciiString - Function that converts a given string of hex values into its ASCII equivalent. A hex string contains hex chars and is "encoded" in ascii In order to encrypt it, it must be converted to the equivalent ascii plain text string plaintext string is half the size of hex, since two hex chars = 1 ascii char if hex string is "4A" it will be converted to "J" in ascii which will have a hex representation of "4a" The original hex string converted to hex staright or printed in hex straight rather will print or have the value "0x34", "0x31" BASICALLY THE HEX STRING FF IS INTERPRETED AS THE CHARS FF, whereas when using this function we intend it to be "J", ie the char "J"

Parameters

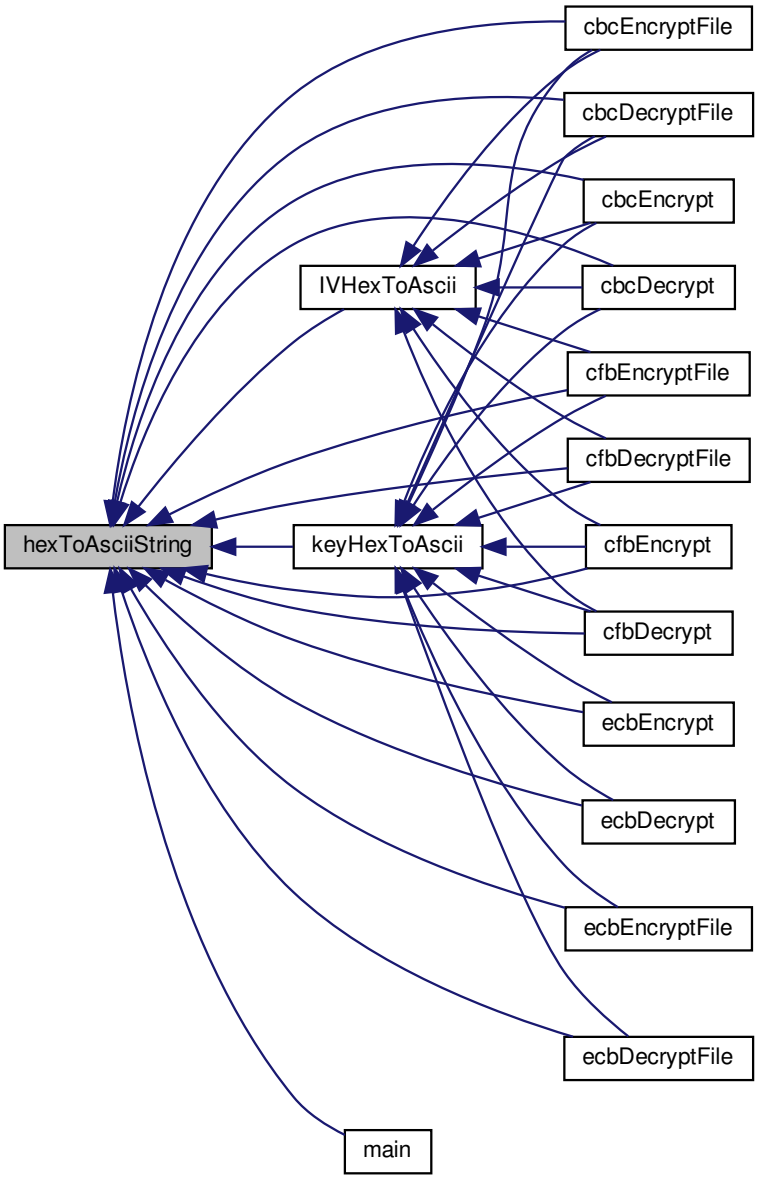
char∗	hexString - The string of hex values to be converted.
char∗	asciiString - The output of the converted hex string.
int	hexStringLength - The length of parameter hexString.

Definition at line 948 of file AES.c.

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.2.17 hexToInt()

```
uint8_t hexToInt (
            char ch )
```

hexToInt - Function that converts a given hex value into an integer.

Parameters

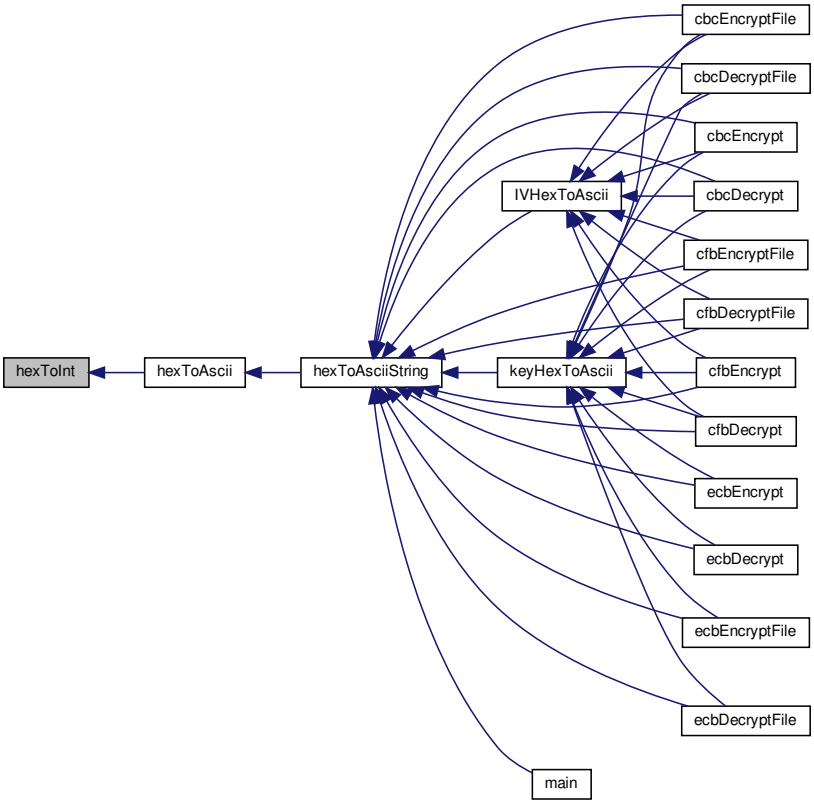
ch	- hex value that wil be converted to int.

Returns

uint8_t the converted int value.

Definition at line 909 of file AES.c.

Here is the caller graph for this function:



3.2.2.18 invMixColumns()

```
void invMixColumns (
            unsigned char state[4][4] )
```

invMixColumns - Function that does the inverse of the Mix Column Step for AES Encryption. Performs the gallois field multiplication and the required XOR to the state passed in as a paramter

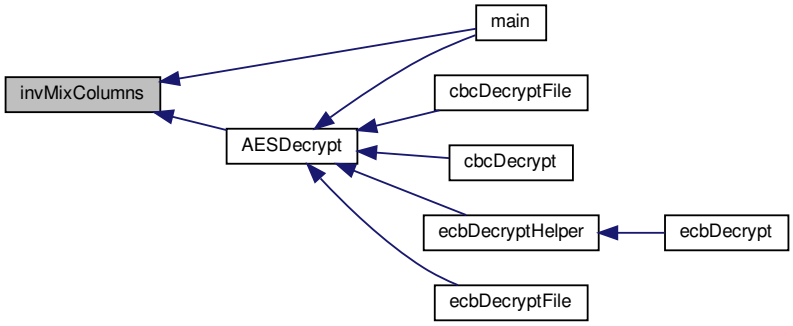
Parameters

state.	
state	- unsigned char - is the current state of the ciphertext or plaintext during AES encryption or decryption.

Definition at line 740 of file AES.c.

Here is the call graph for this function:

Here is the caller graph for this function:



3.2.2.19 invShiftRows()

```
void invShiftRows (
            unsigned char state[4][4],
            int wordLength )
```

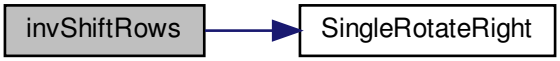
invShiftRows - Function to shift the state array Inverse according to the AES encryption standard for 128 - bits blocks

Parameters

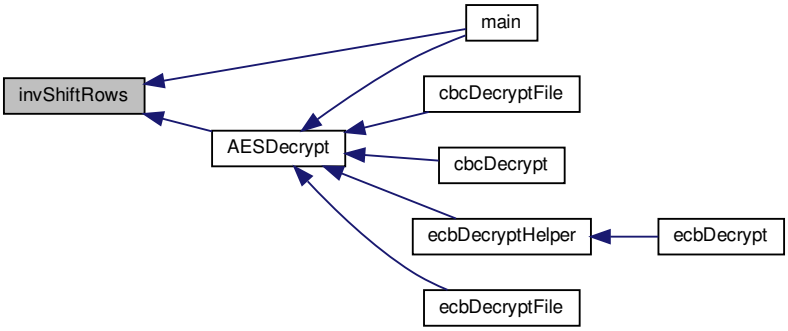
state	- unsigned char - is the current state of the ciphertext or plaintext during AES encryption or decryption

Definition at line 835 of file AES.c.

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.2.20 invSubBytes()

```
void invSubBytes (
            unsigned char state[4][4] )
```

invSubBytes - Function that performs the inverse of Function subBytes

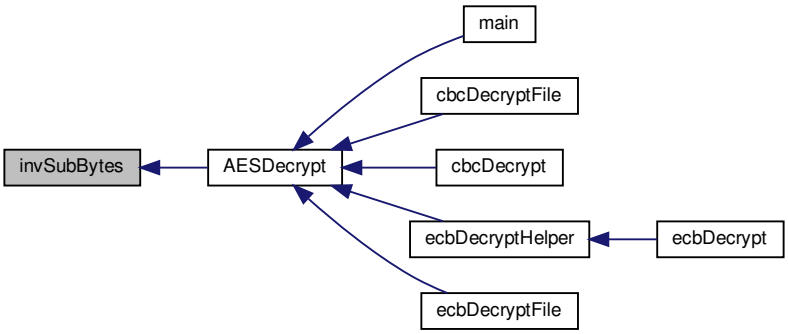
Parameters

state	- unsigned char - is the current state of the ciphertext or plaintext during AES encryption or decryption

Definition at line 802 of file AES.c.

Here is the call graph for this function:

Here is the caller graph for this function:



3.2.2.21 isFileTxt()

```
uint8_t isFileTxt (
            unsigned char * fileName )
```

isFileTxt - Function to determine if the file passed in as a paramter

Parameters

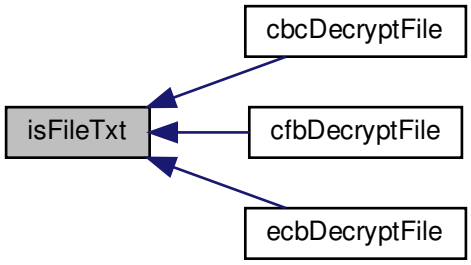
filename	is a text file with extension .txt or not. Returns a 1 if it is and a 0 if it isn't.
fileName	- unsigned char* fileName - path to file to determine if the file is a text file or not.

Returns

uint8_t - boolean indicating if it is a text file or not. (0 is not a text file, 1 is a text file)

Definition at line 1156 of file AES.c.

Here is the caller graph for this function:



3.2.2.22 IVHexToAscii()

```
unsigned char* IVHexToAscii (
            unsigned char * hexIV,
            int IVLength )
```

IVHexToAscii - Function to convert a initialization vector from a Hex string passed in as a paramter.

Parameters

hexIV	to an ascii string. User must free the returned pointer to memory allocated. Returns the Ascii equivalent. The caller must free the pointer returned.
char	- unsigned char* hexIV - hex representation
IVLength	- length of the hex representation of the IV passed in as paramter
hexIV.	

Returns

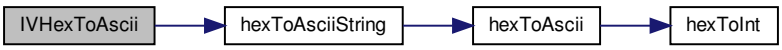
unsigned* - the ASCII representation of the hex IV passed in as parameter

Parameters

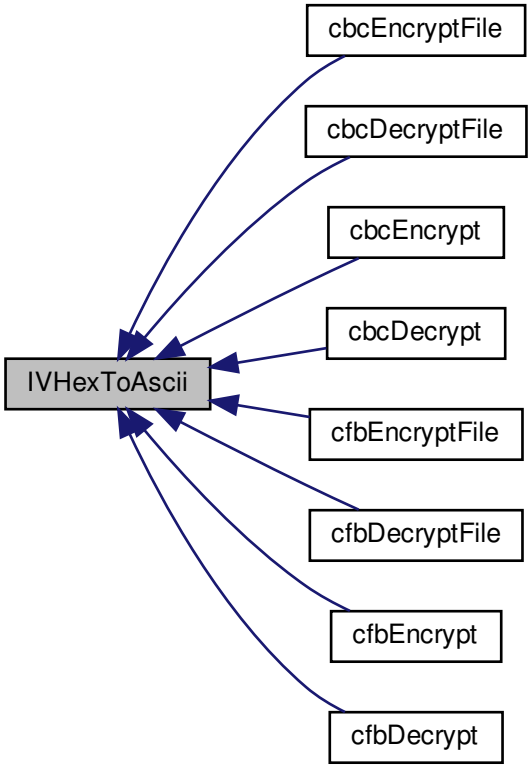
hex↩ IV.	

Definition at line 1205 of file AES.c.

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.2.23 keyHexToAscii()

```
unsigned char* keyHexToAscii (
            unsigned char * hexKey,
            int keyLength )
```

keyHexToAscii - Function to convert a key from a Hex string passed in as a paramter

Parameters

hexKey	to an ascii string. User must free the returned pointer to memory allocated. Returns the Ascii equivalent. The caller must free the pointer returned.
char	- unsigned char∗ hexKey - hex representation of the key to be converted to ASCII.
keyLength	- length of the hex representation of the key passed in as paramter
hexKey.	

Returns

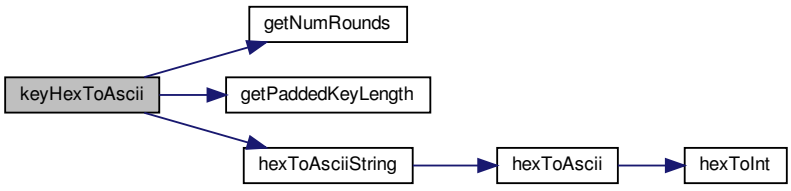
unsigned∗ - the ASCII representation of the hex key passed in as parameter

Parameters

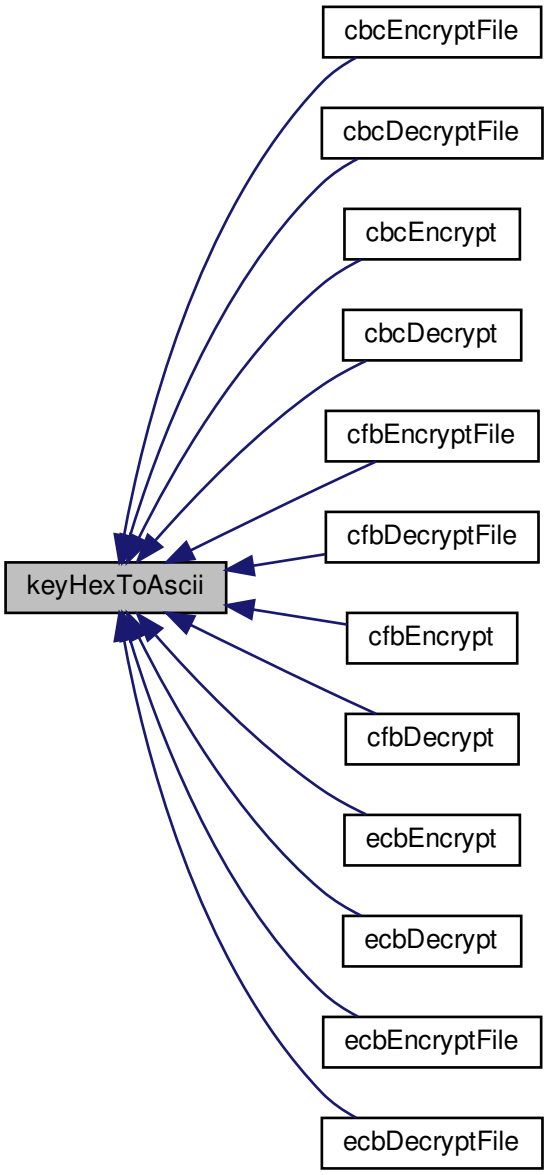
hexKey.	

Definition at line 1178 of file AES.c.

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.2.24 KeyScheduleCore()

```
void KeyScheduleCore (
            unsigned char * word,
            int wordLength,
            int rConIterationVal )
```

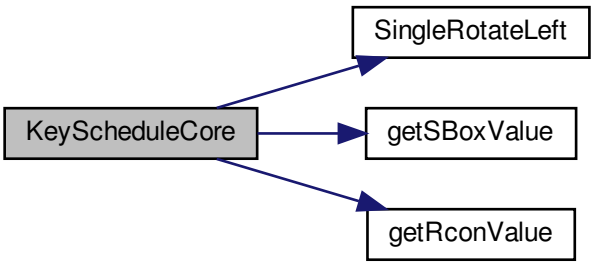
KeyScheduleCore - Function that performs the key schedule core for the Rijndael Key Schedule. Performs a single rotate left of the word passed in as.

Parameters

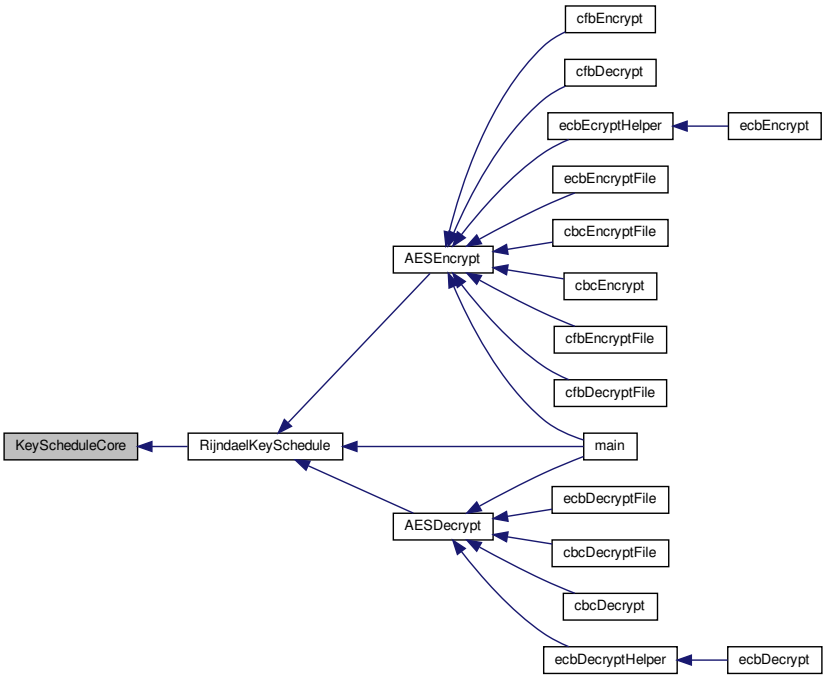
word	and applies the required s-box substituion and rcon XOR.
char	- unsigned char* word - pointer to the word onto which the key schedule core should be operated.
wordLength	- length of the word passed in as a parameter
word.	
rConIterationVal	- the iteration value to be used for the rcon XOR.

Definition at line 631 of file AES.c.

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.2.25 mixColumns()

```
void mixColumns (
            unsigned char state[4][4] )
```

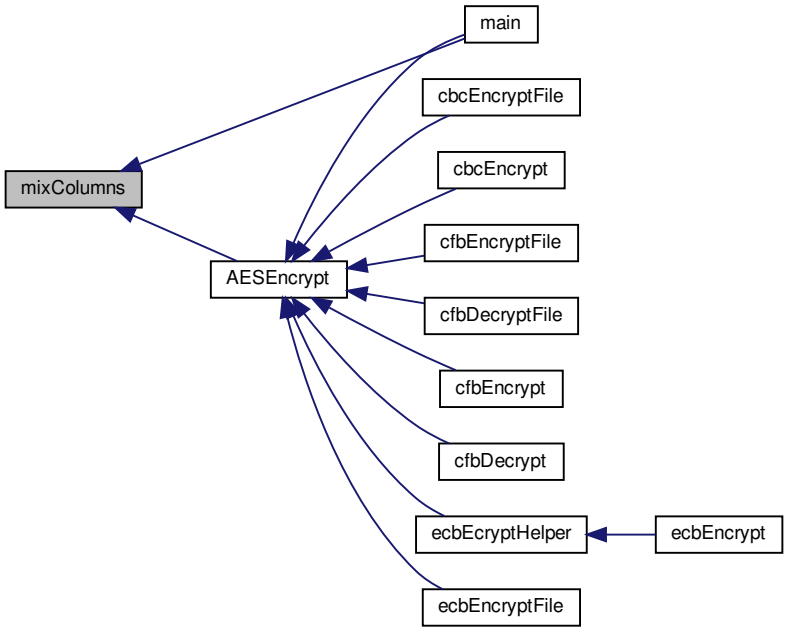
mixColumns - Function that performs the MixColumns step of AES as specified by AES encryption.

Parameters

state	- unsigned char - is the current state of the ciphertext or plaintext during AES encryption or decryption

Definition at line 702 of file AES.c.

Here is the caller graph for this function:



3.2.2.26 printAESBlock()

```
void printAESBlock (
            unsigned char * block )
```

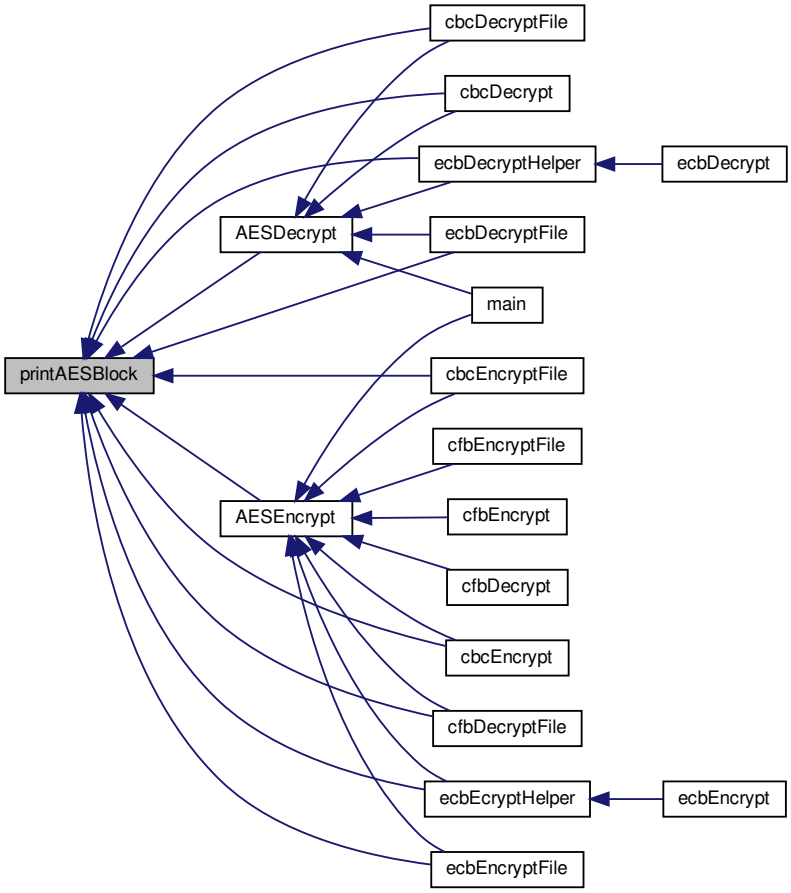
printAESBlock - Function to print a single block in hex format to the terminal.

Parameters

block	- block to be printed.

Definition at line 1028 of file AES.c.

Here is the caller graph for this function:



3.2.2.27 printStateArray()

```
void printStateArray (
            uint8_t stateArray[4][4] )
```

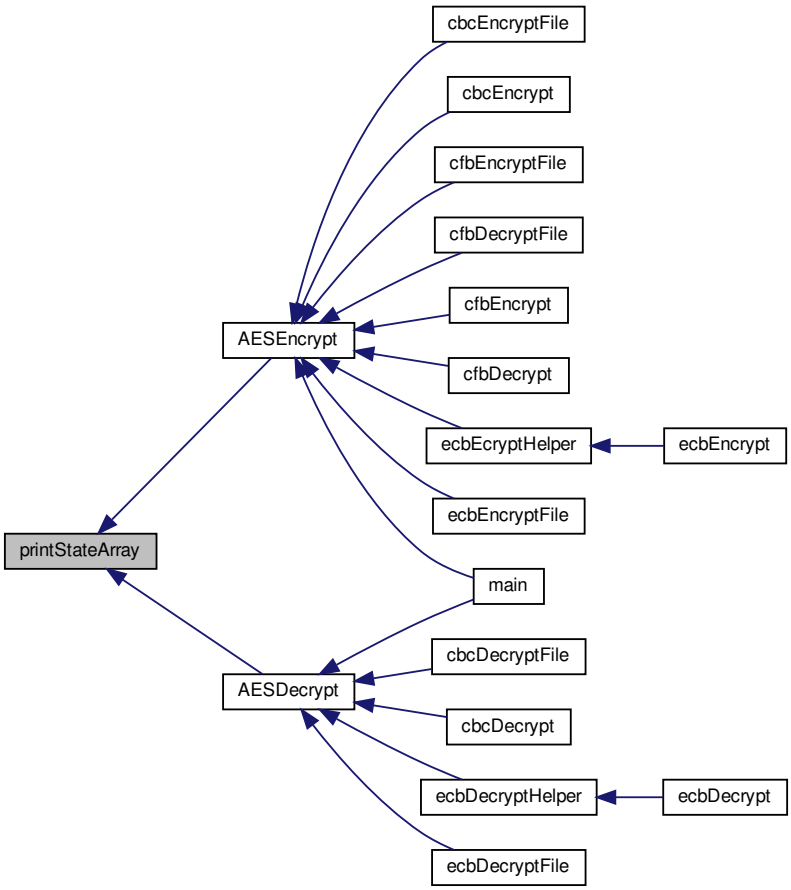
printStateArray - Function to print the state array to the terminal in hex format.

Parameters

stateArray	- the state array that should be printed to the terminal.

Definition at line 673 of file AES.c.

Here is the caller graph for this function:



3.2.2.28 RijndaelKeySchedule()

```
unsigned char* RijndaelKeySchedule (
            unsigned char * originalKey,
            int keyLength )
```

RijndaelKeySchedule - Function that performs the Rijndael key scheduling for AES encryption. Takes in the original key passed in as parameter.

Parameters

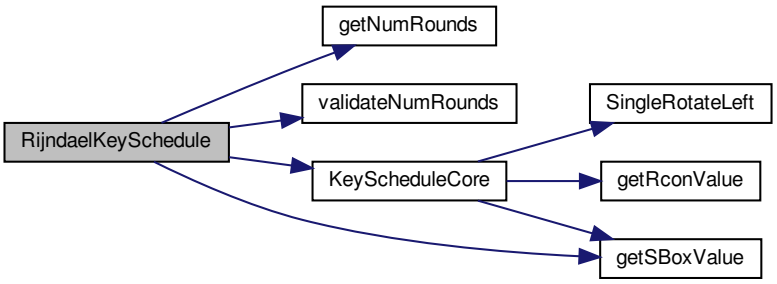
originalKey	and the length of the original key given as parameter. The caller must free the memory allocated and returned.
originalKey	- unsigned char * - An unsigned char pointer to the original key.
keyLength	- int - length of originalKey passed in as a parameter
originalKey	

Returns

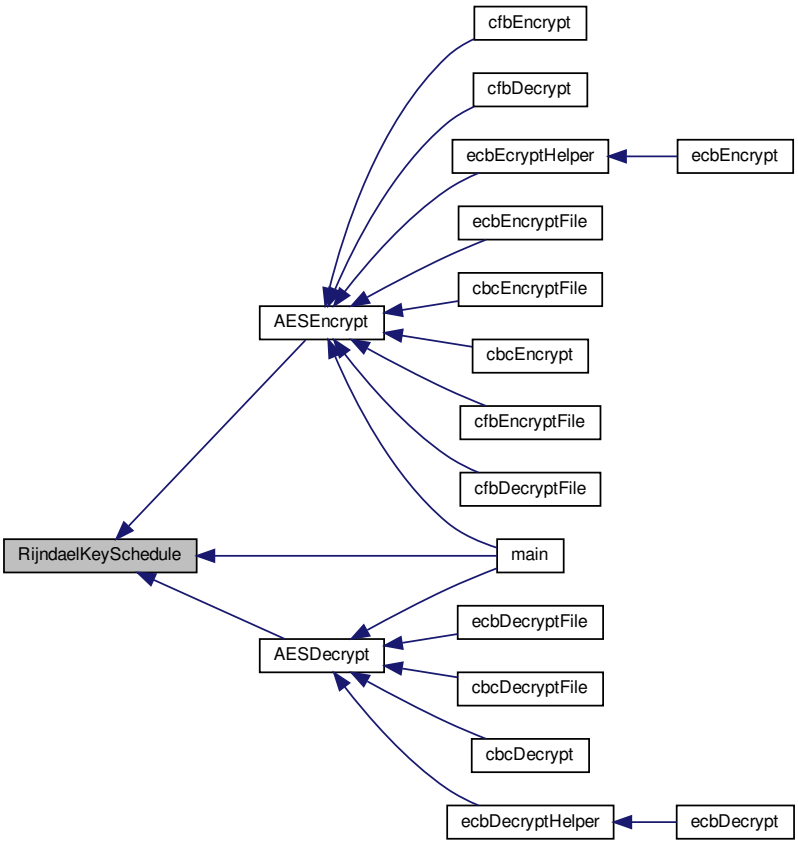
expandedKey - The key that has been expanded.

Definition at line 577 of file AES.c.

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.2.29 ShiftRows()

```
void ShiftRows (
            unsigned char state[4][4],
            int wordLength )
```

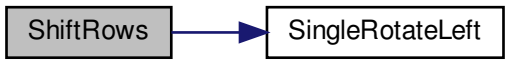
ShiftRows - Function to shift the state array according to the AES encryption standard for 128 - bits blocks.

Parameters

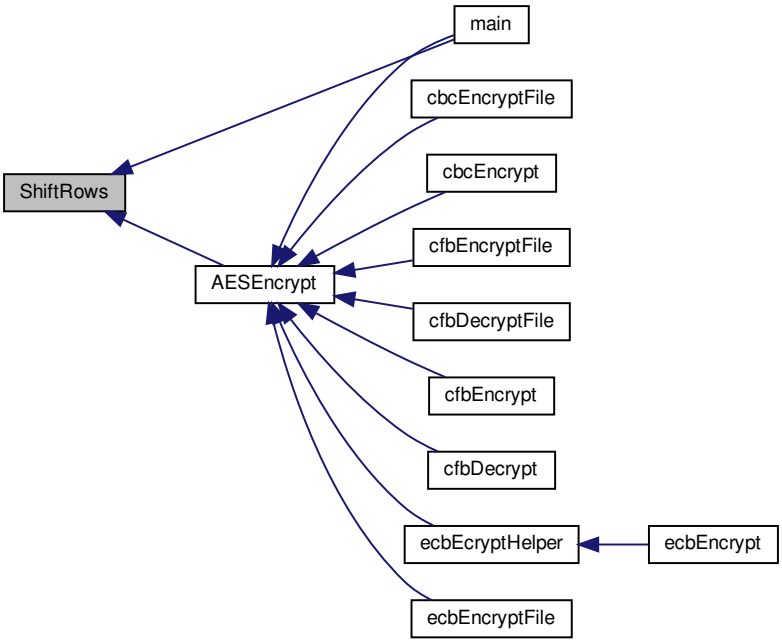
state	- unsigned char - is the current state of the ciphertext or plaintext during AES encryption or decryption

Definition at line 815 of file AES.c.

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.2.30 SingleRotateLeft()

```
void SingleRotateLeft (
            unsigned char * word,
            int wordLength )
```

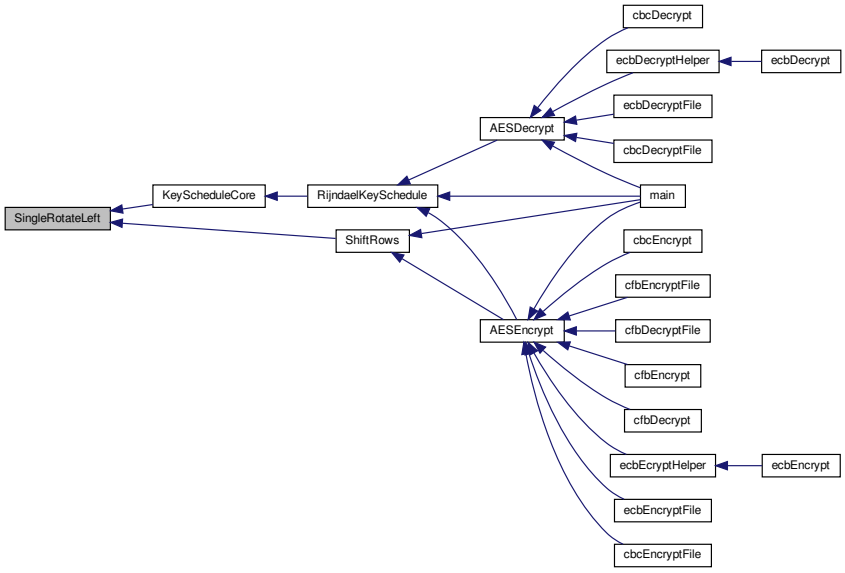
SingleRotateLeft - Function to rotate the array passed in as a paramter.

Parameters

word,a	single time left (8 bits to the left), with the left most element becoming the right most element. As such: rotate(1d2c3a4f) = 2c3a4f1d.
word	- unsigned char ∗word - the array/word to be left rotated by 8 bits.
wordLength	- int - length of the parameter
word.	

Definition at line 655 of file AES.c.

Here is the caller graph for this function:



3.2.2.31 SingleRotateRight()

```
void SingleRotateRight (
            unsigned char * word,
            int wordLength )
```

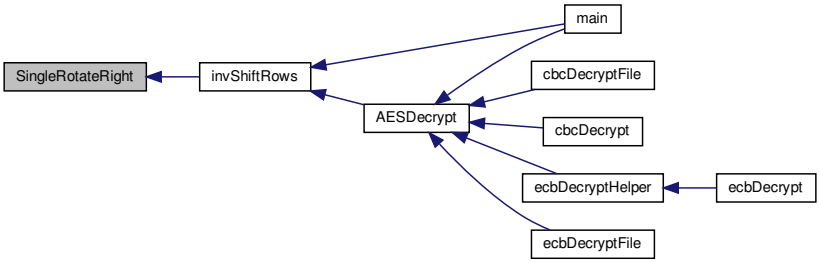
SingleRotateRight - Function to rotate the array passed in as a paramter.

Parameters

word,a	single time right (8 bits to the right), with the right most element becoming the left most element. As such: rotate(1d2c3a4f) = 4f1d2c3a.
word	- unsigned char ∗word - the array/word to be right rotated by 8 bits.
wordLength	- int - length of the parameter
word.	

Definition at line 857 of file AES.c.

Here is the caller graph for this function:



3.2.2.32 stripDirectory()

```
void stripDirectory (
            char * fileName,
            char * extractedFileName,
            char * extractedFilePath,
            int fileNameLength,
            int slashIndex )
```

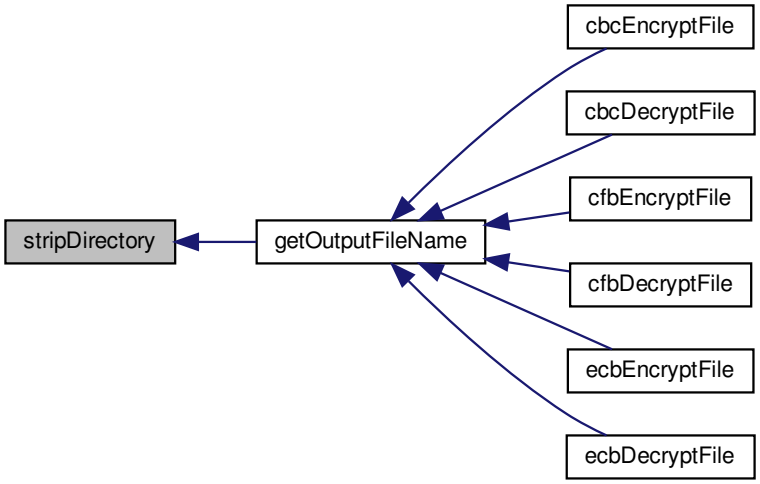
stripDirectory - Function that removes path from the provided path to a file and returns only the file name

Parameters

fileName	The path to a specified file
extractedFileName	The name of the file within the provided path to a file
extractedFilePath	The path to file, excluding the file name
fileNameLength	The length of the paramter
fileName	
slashIndex	The index of the last '/' in the original file path passed in as a paramter
fileName	stripDirectory - Function that removes path from the provided path to a file and returns only the file name
fileName	The path to a specified file
extractedFileName	The name of the file within the provided path to a file
extractedFilePath	The path to file, excluding the file name
fileNameLength	The length of the paramter
fileName	
slashIndex	The index of the last '/' in the original file path passed in as a paramter
fileName	

Definition at line 1063 of file AES.c.

Here is the caller graph for this function:



3.2.2.33 subBytes()

```
void subBytes (
            unsigned char state[4][4] )
```

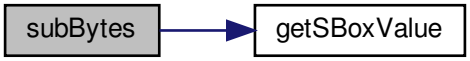
subBytes - Function that performs the sub byte operation where each value is replaced by the s box value

Parameters

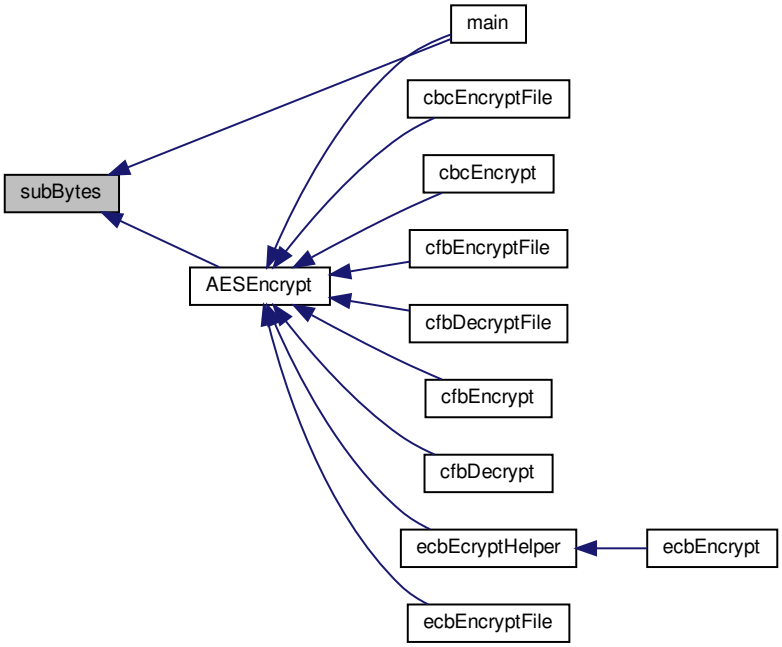
state	- unsigned char - is the current state of the ciphertext or plaintext during AES encryption or decryption.

Definition at line 789 of file AES.c.

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.2.34 validateCipherTextLength()

```
void validateCipherTextLength (
            int cipherTextLength )
```

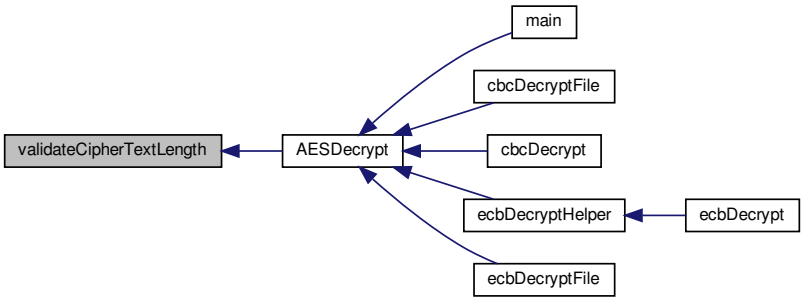
validateCipherTextLength - Function that validates the length of the ciphertext. The validation is done against the AES_BLOCK_SIZE value

Parameters

cipherTextLength	- int - The length of the cipher text as an integer value

Definition at line 1015 of file AES.c.

Here is the caller graph for this function:



3.2.2.35 validateNumRounds()

```
void validateNumRounds (
            int numRounds,
            int keyLength )
```

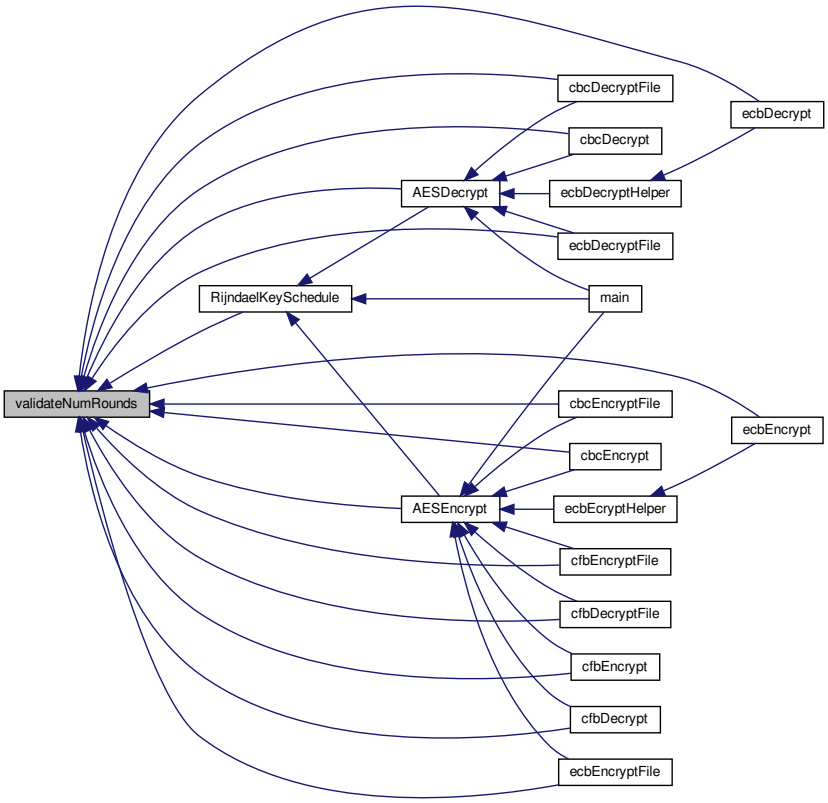
validateNumRounds - Function that validates the number of rounds that have been passed in by the

Parameters

numRounds.	Upon invalid validation, relevent error information will be printed to terminal and the program will exit with an EXIT_FAILURE flag.
numRounds	- int - Integer value of the number rounds

Definition at line 989 of file AES.c.

Here is the caller graph for this function:



3.2.2.36 validatePlainTextLength()

```
void validatePlainTextLength (
            size_t plainTextLength )
```

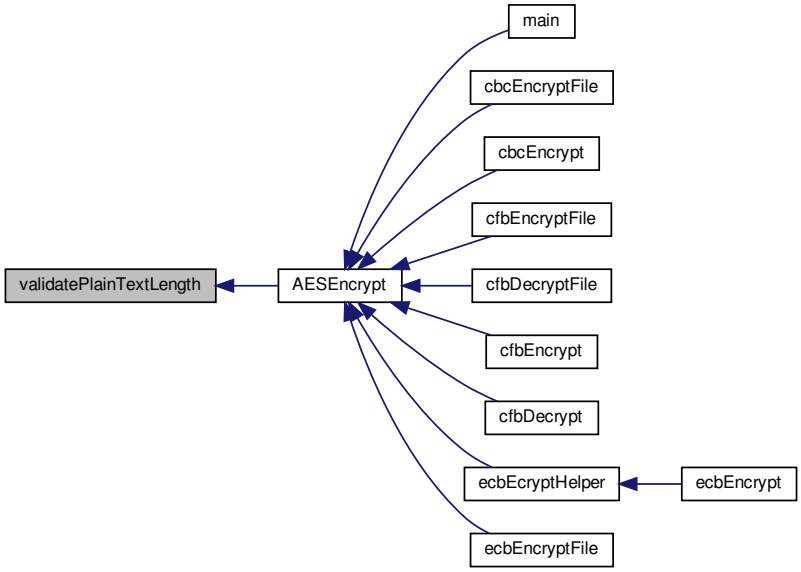
validatePlainTextLength - Function that validates the length of the plaintext. The validation is done against the AES_BLOCK_SIZE value

Parameters

plainTextLength	- int - The length of the plaintext text as an integer value

Definition at line 1002 of file AES.c.

Here is the caller graph for this function:



3.2.2.37 XORBlocks()

```
unsigned char* XORBlocks (
            unsigned char * block1,
            unsigned char * block2,
            int length )
```

XORBlocks - Function to XOR two blocks of length.

Parameters

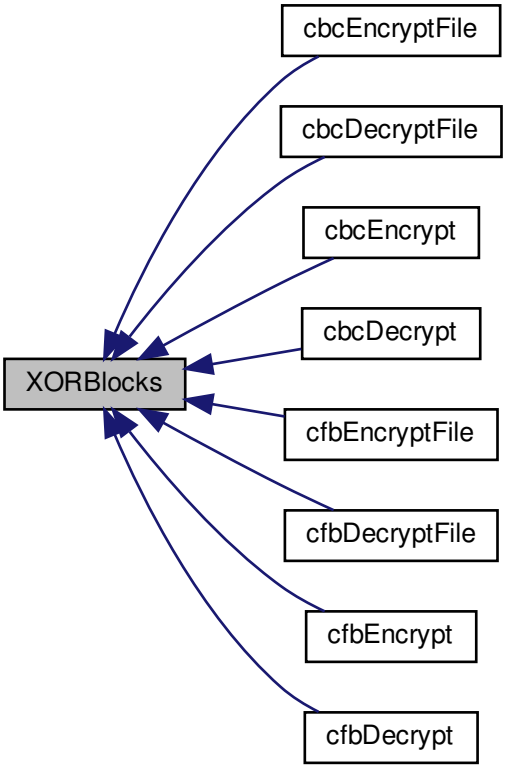
length	and retuns the XOR'd result. User must free the memory returned.
char	- block1 - First block to be XOR'd.
char	- block2 - Second block to be XOR'd.
length	- length of the blocks to be XOR'd.

Returns

unsigned* - Result of the XOR.

Definition at line 1228 of file AES.c.

Here is the caller graph for this function:

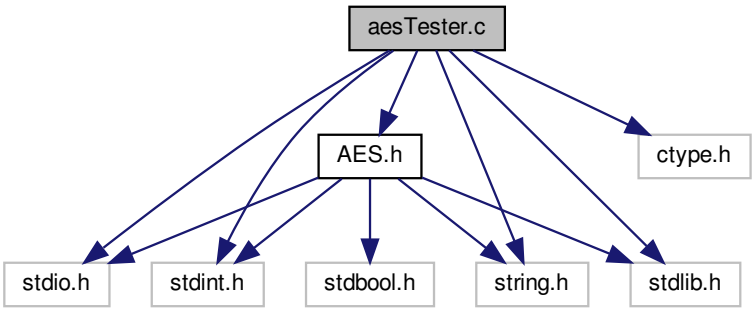


3.3 aesTester.c File Reference

Main file.

```
#include "stdio.h"
#include "AES.h"
#include <stdint.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
```

Include dependency graph for aesTester.c:



Functions

- int main (int argc, char ∗argv[ ])

Variables

- const unsigned char invSBox [256]

  const unsigned char invSBox. Lookup table for the inverse sbox values used during AES Decryption.
- const unsigned char sbox [256]

  const unsigned char sbox. Lookup table for the sbox values used during AES Encryption.
- const unsigned char Rcon [255]

  const unsigned char Rcon. Lookup table for the Rcon values used during Rijndael Key Schedule during the AES Encryption and Decryption.

3.3.1 Detailed Description

Main file.

Authors

Mohamed Ameen Omar (u16055323)
Douglas Healy (u16018100)
Llewellyn Moyse (u15100708)

Version

0.1

Date

2019-03-19

Copyright

3.3.2 Function Documentation

3.3.2.1 main()

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

Result Should BE: d4 e0 b8 1e bf b4 41 27 5d 52 11 98 30 ae f1 e5

Result Should BE: e9 cb 3d af 31 32 2e 09 7d 2c 89 07 b5 72 5f 94

Result Should BE: *20 B7 EF 8F 45 F9 B7 92 F9 8F 92 31 8F B7 4D 31*

Result Should BE: 00 3C 6E 47 1F 4E 22 74 0E 08 1B 31 54 59 0B 1A

Result should be: 4a a8 b3 7a 6c 47 d8 c7 5b cf 6 29 7b 3a 3d 93

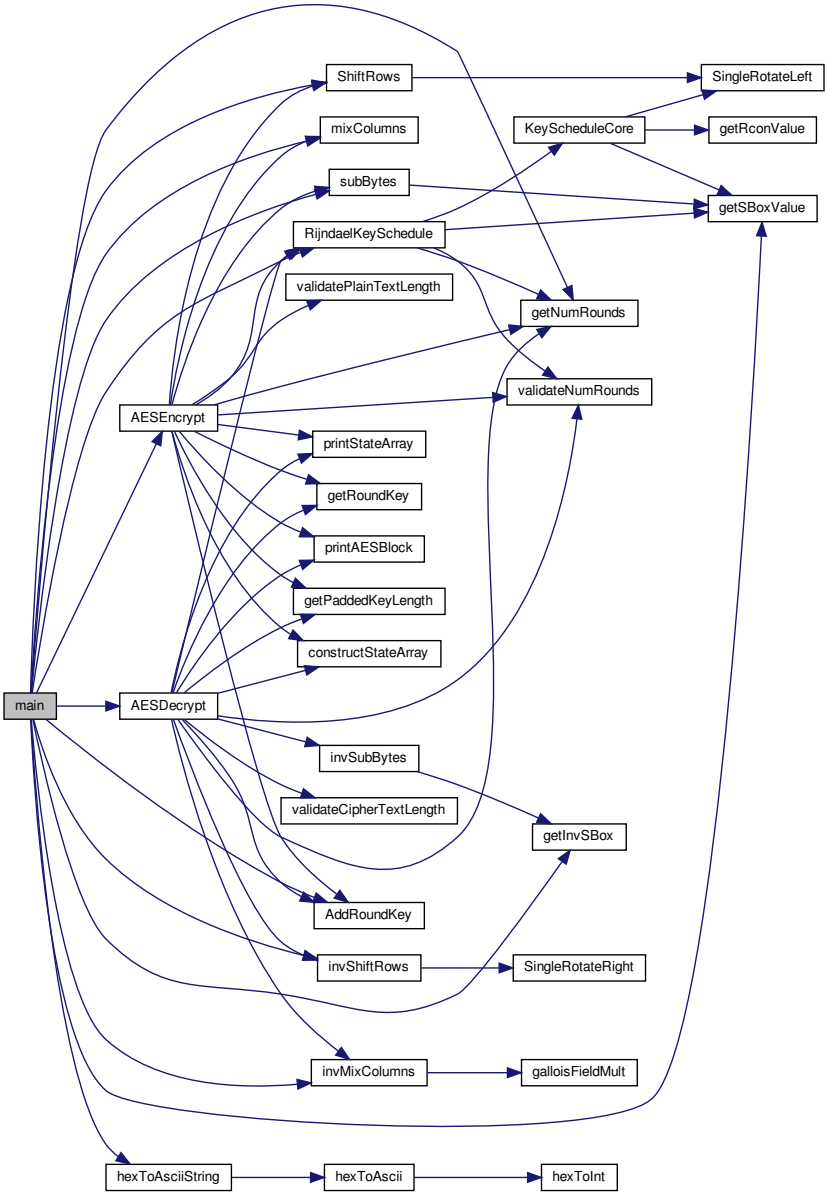
Result Should BE: { 0xd4,0xe0,0xb8,0x1e, 0x27,0xbf,0xb4,0x41, 0x11,0x98,0x5d,0x52, 0xae,0xf1,0xe5,0x30};

Result Should BE: {0xe9,0xcb,0x3d,0xaf, 0x09,0x31,0x32,0x2e, 0x89,0x07,0x7d,0x2c, 0x72,0x5f,0x94,0xb5};

Result should be: { 0x74,0x20,0x61,0x73, 0x68,0x69,0x20,0x74, 0x69,0x73,0x74,0x2e, 0x73,0x20,0x65,0x2e };

Definition at line 32 of file aesTester.c.

Here is the call graph for this function:

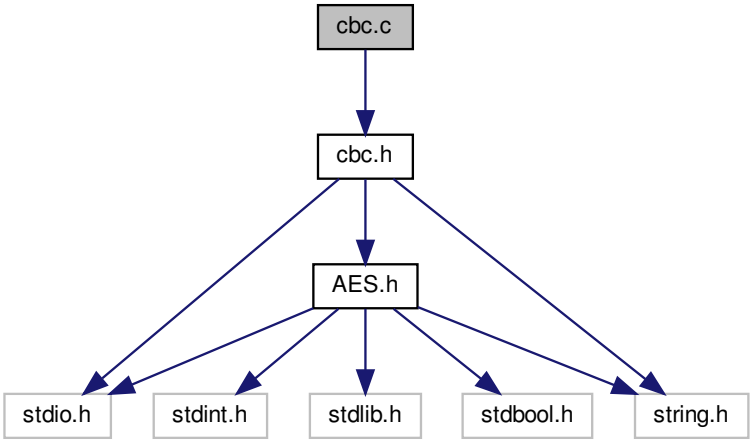


3.4 cbc.c File Reference

Cipher Block Chaining (CBC) - AES Implementation file This file contains the implementation of the functions used for the CBC mode of AES encryption. This system supports both file and user input encryption, as hex or ascii input. If the user inputs data to be encrypted or decrypted, the result will be printed to the terminal, whereas if the user specifies a file to be encrypted or decrypted, a new file will be created and the result will be written to the file. The CBC Encryption platform encrypts and decrypts blocks 16 bytes at a time, using 0 padding. The IV is limited to 16 bytes and the key is limited to 32 bytes as per the AES encryption standard.

```
#include "cbc.h"
```

Include dependency graph for cbc.c:



Functions

- void cbcEncryptFile (unsigned char ∗fileName, unsigned char ∗key, unsigned char ∗initializationVector, int keyLength, int initializationVectorLength, int isTextHex, int isKeyHex, int isIvHex)

  cbcEncryptFile - Function to encrypt the file with name
- void cbcDecryptFile (unsigned char ∗fileName, unsigned char ∗key, unsigned char ∗initializationVector, int keyLength, int initializationVectorLength, int isTextHex, int isKeyHex, int isIvHex)

  cbcDecryptFile - Function to decrypt the file with name
- void cbcEncrypt (unsigned char ∗plainText, unsigned char ∗key, unsigned char ∗initializationVector, int plain↩ TextLength, int keyLength, int initializationVectorLength, int isTextHex, int isKeyHex, int isIvHex)

  cbcEncrypt - Function to encrypt the user input pointed to by
- void cbcDecrypt (unsigned char ∗cipherText, unsigned char ∗key, unsigned char ∗initializationVector, int cipherTextLength, int keyLength, int initializationVectorLength, int isTextHex, int isKeyHex, int isIvHex)

  cbcDecrypt - Function to decrypt the user input pointed to by

3.4.1 Detailed Description

Cipher Block Chaining (CBC) - AES Implementation file This file contains the implementation of the functions used for the CBC mode of AES encryption. This system supports both file and user input encryption, as hex or ascii input. If the user inputs data to be encrypted or decrypted, the result will be printed to the terminal, whereas if the user specifies a file to be encrypted or decrypted, a new file will be created and the result will be written to the file. The CBC Encryption platform encrypts and decrypts blocks 16 bytes at a time, using 0 padding. The IV is limited to 16 bytes and the key is limited to 32 bytes as per the AES encryption standard.

Authors

Mohamed Ameen Omar (u16055323)
Douglas Healy (u16018100)
Llewellyn Moyse (u15100708)

Version

0.1

Date

2019-03-28

Copyright

3.4.2 Function Documentation

3.4.2.1 cbcDecrypt()

```
void cbcDecrypt (
            unsigned char * cipherText,
            unsigned char * key,
            unsigned char * initializationVector,
            int cipherTextLength,
            int keyLength,
            int initializationVectorLength,
            int isTextHex,
            int isKeyHex,
            int isIvHex )
```

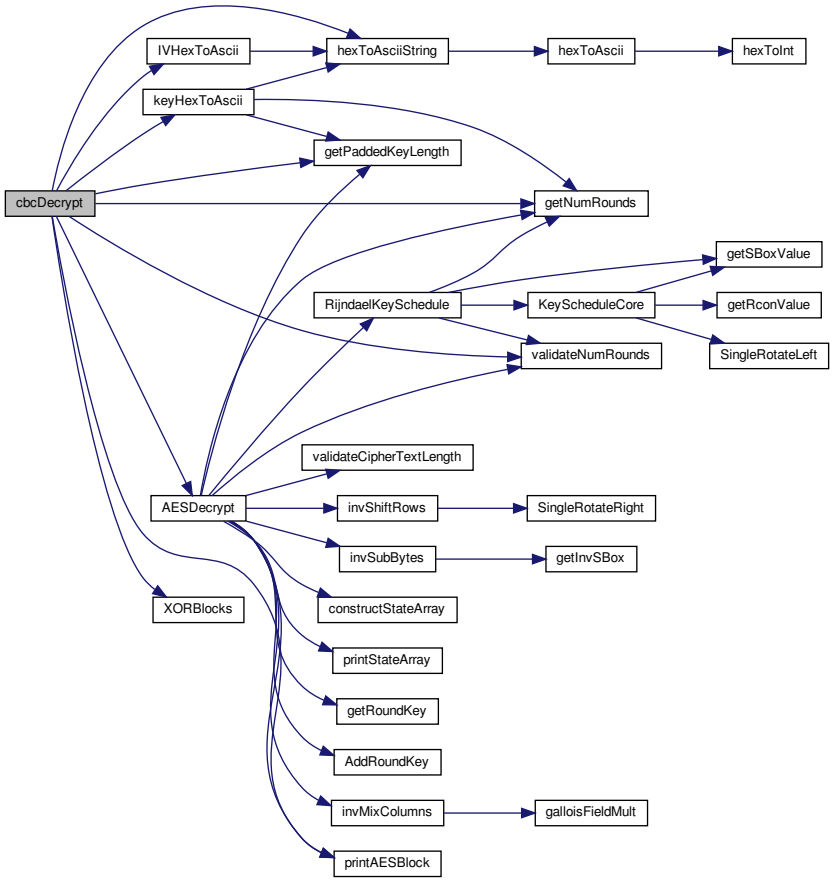
cbcDecrypt - Function to decrypt the user input pointed to by

Parameters

cipherText	and print decrypted result in hex to terminal. Performs decryption using the cbc mode prints the result to the terminal for each block in hex. If any input is hex, it will convert it to ascii, perform decryption and print it in hex. Makes use of zero padding.
char	- unsigned char* cipherText - the user input to be decrypted.
char	- unsigned char* key - the key to use for decryption.
char	- unsigned char* initializationVector - the initialization vector to use for cbc decryption.
cipherTextLength	- int - the length of the ciphertext to be encrypted in
cipherText.	
keyLength	- int - the length of the key specified in
key.	
initializationVectorLength	- int - the length of the key specified in
initializationVector.	
isTextHex	- int - boolean used to signify whether the file pointed to by
fileName	is a hexString or ASCII string. (1 = file is a hexString)
isKeyHex	- int - boolean used to signify whether the key pointed to by
key	is a hexString or ASCII string. (1 = file is a hexString)
isIvHex	- int - boolean used to signify whether the IV pointed to by
initializationVector	is a hexString or ASCII string. (1 = file is a hexString)

Definition at line 648 of file cbc.c.

Here is the call graph for this function:



3.4.2.2 cbcDecryptFile()

```
void cbcDecryptFile (
            unsigned char * fileName,
            unsigned char * key,
            unsigned char * initializationVector,
            int keyLength,
            int initializationVectorLength,
            int isTextHex,
            int isKeyHex,
            int isIvHex )
```

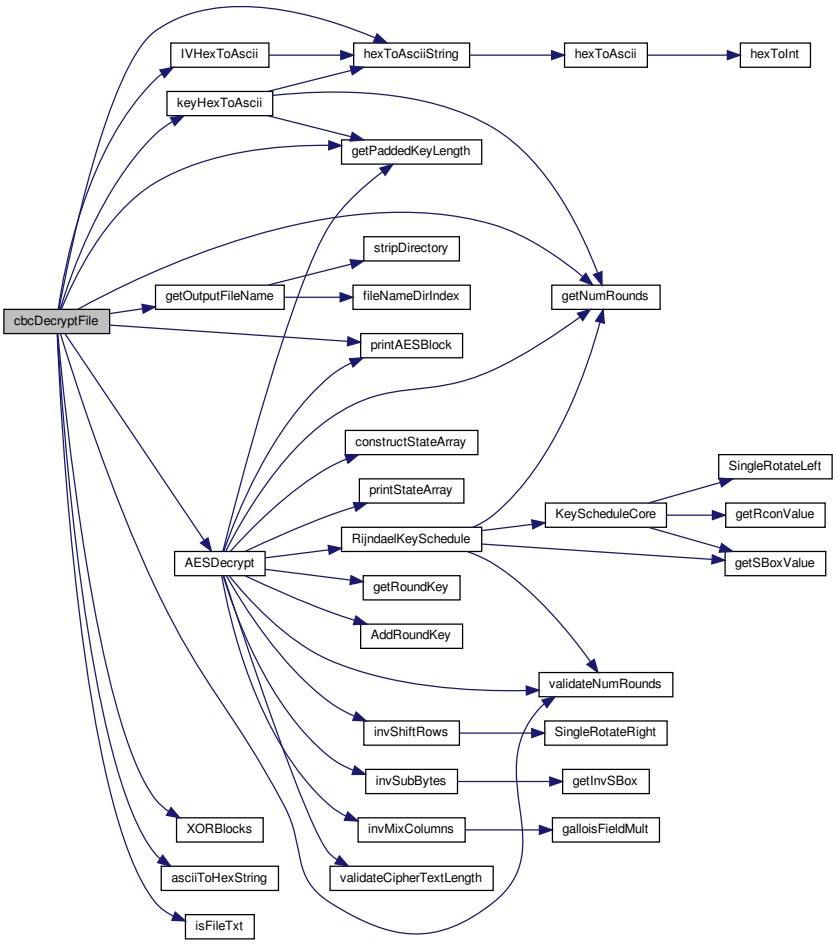
cbcDecryptFile - Function to decrypt the file with name

Parameters

fileName	and write the decrypted version to file with cbcDecrypted appended to the original filename. Performs decryption using the cbc mode and writes the result to a file. If any input is hex, it will convert it to ascii, perform decryption and write it back to the file in the same format as the input. That is if the input file was a hexString, the decrypted file will also contain a hex string. All terminal output, however, will be hex. Makes use of zero padding.
char	- unsigned char* fileName - the path to the file to be decrypted
char	- unsigned char* key - the key to use for decryption.
char	- unsigned char* initializationVector - the initialization vector to use for cbc decryption
keyLength	- int - the length of the key specified in
key.	
initializationVectorLength	- int - the length of the key specified in
initializationVector.	
isTextHex	- int - boolean used to signify whether the file pointed to by
fileName	is a hexString or ASCII string. (1 = file is a hexString)
isKeyHex	- int - boolean used to signify whether the key pointed to by
key	is a hexString or ASCII string. (1 = file is a hexString)
isIvHex	- int - boolean used to signify whether the IV pointed to by
initializationVector	is a hexString or ASCII string. (1 = file is a hexString)

Definition at line 263 of file cbc.c.

Here is the call graph for this function:



3.4.2.3 cbcEncrypt()

```
void cbcEncrypt (
            unsigned char * plainText,
            unsigned char * key,
            unsigned char * initializationVector,
            int plainTextLength,
            int keyLength,
            int initializationVectorLength,
            int isTextHex,
            int isKeyHex,
            int isIvHex )
```

cbcEncrypt - Function to encrypt the user input pointed to by

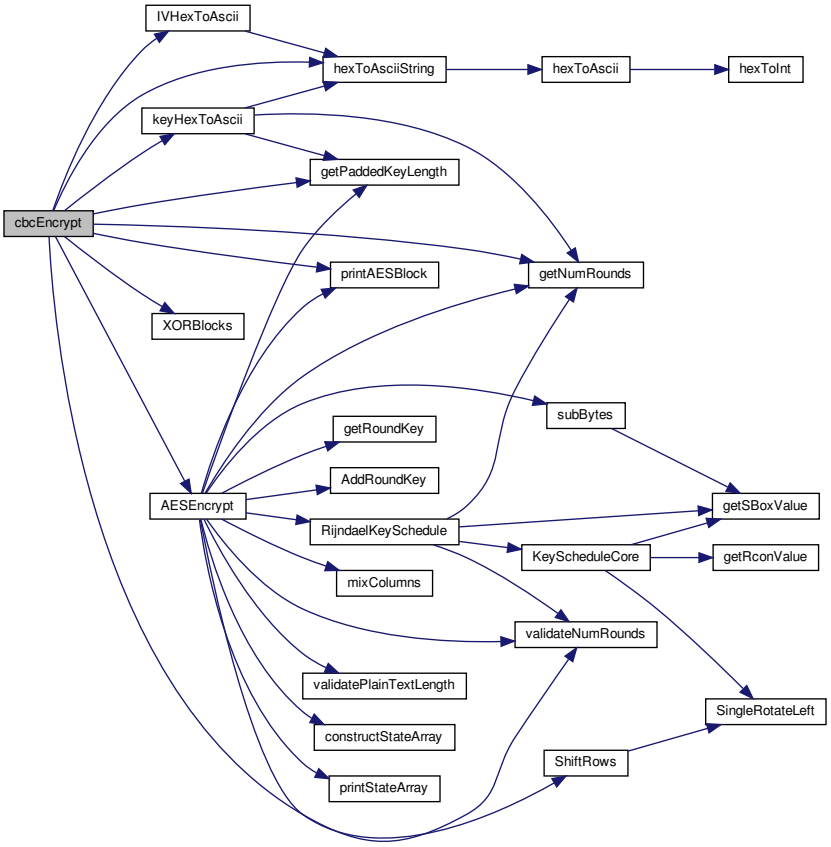
Parameters

plainText	and print encrypted result in hex to terminal. Performs encryption using the cbc mode prints the result to the terminal for each block in hex. If any input is hex, it will convert it to ascii, perform encryption and print it in hex. Makes use of zero padding.
char	- unsigned char∗ plainText - the user input to be encrypted.
char	- unsigned char∗ key - the key to use for encryption.
char	- unsigned char∗ initializationVector - the initialization vector to use for cbc encryption
plainTextLength	- - int - the length of the plaintext to be encrypted in
plainText.	
keyLength	- int - the length of the key specified in
key.	
initializationVectorLength	- int - the length of the key specified in
initializationVector.	
isTextHex	- int - boolean used to signify whether the file pointed to by
fileName	is a hexString or ASCII string. (1 = file is a hexString)
isKeyHex	- int - boolean used to signify whether the key pointed to by
key	is a hexString or ASCII string. (1 = file is a hexString)
isIvHex	- int - boolean used to signify whether the IV pointed to by
initializationVector	is a hexString or ASCII string. (1 = file is a hexString)

Process for CBC encrypt file: Read from file, if hex, convert if not do nothing. Store read converted in plaintextblock Pad the converted plaintextblock and store in paddedPlaintext Store IV and previous ciphertext in placeholderblock XOR paddedPlaintext and placeholder - store in intermediate Encrypt intermediate - store in cipherTextBlock Write to the file Free memory, read again and check that the read buffer length (amount read from the file iss not 0)

Definition at line 492 of file cbc.c.

Here is the call graph for this function:



3.4.2.4 cbcEncryptFile()

```
void cbcEncryptFile (
            unsigned char * fileName,
            unsigned char * key,
            unsigned char * initializationVector,
            int keyLength,
            int initializationVectorLength,
            int isTextHex,
            int isKeyHex,
            int isIvHex )
```

cbcEncryptFile - Function to encrypt the file with name

Parameters

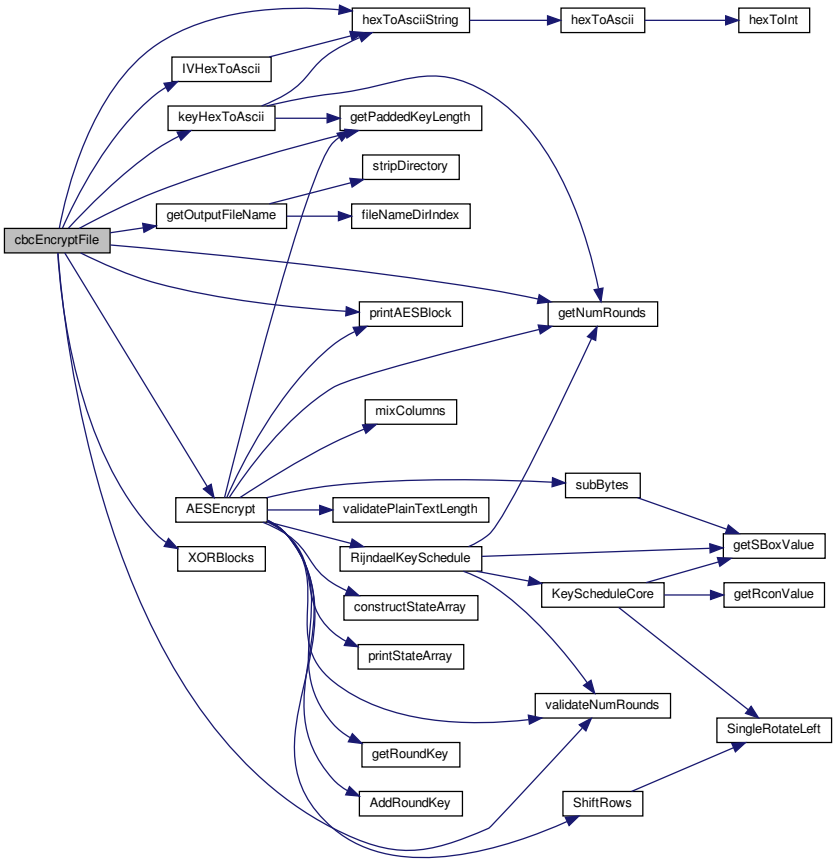
fileName	and write the encrypted version to file with cbcEncrypted appended to the original filename. Performs encryption using the cbc mode and writes the result to a file. If any input is hex, it will convert it to ascii, perform encryption and write it back as ASCII. All terminal output, however, will be hex. Makes use of zero padding.

Parameters

char	- unsigned char* fileName - the path to the file to be encrypted
char	- unsigned char* key - the key to use for encryption.
char	- unsigned char* initializationVector - the initialization vector to use for cbc encryption
keyLength	- int - the length of the key specified in
key.	
initializationVectorLength	- int - the length of the key specified in
initializationVector.	
isTextHex	- int - boolean used to signify whether the file pointed to by
fileName	is a hexString or ASCII string. (1 = file is a hexString)
isKeyHex	- int - boolean used to signify whether the key pointed to by
key	is a hexString or ASCII string. (1 = file is a hexString)
isIvHex	- int - boolean used to signify whether the IV pointed to by
initializationVector	is a hexString or ASCII string. (1 = file is a hexString)

Definition at line 38 of file cbc.c.

Here is the call graph for this function:

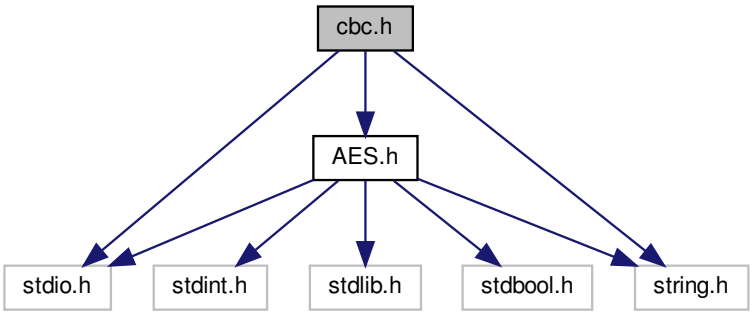


3.5 cbc.h File Reference

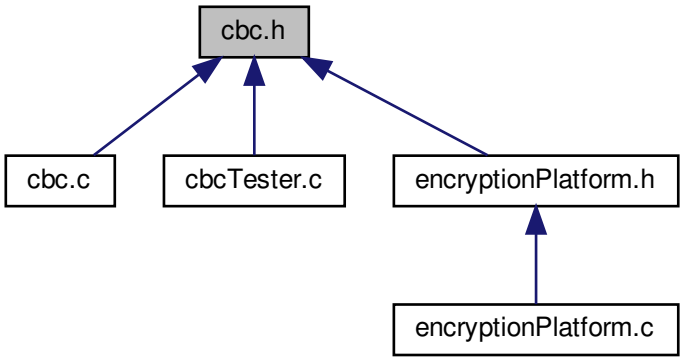
Cipher Block Chaining (CBC) - AES Header file This file contains the function headers of the functions used for the CBC mode of AES encryption. This system supports both file and user input encryption, as hex or ascii input. If the user inputs data to be encrypted or decrypted, the result will be printed to the terminal, whereas if the user specifies a file to be encrypted or decrypted, a new file will be created and the result will be written to the file. The CBC Encryption platform encrypts and decrypts blocks 16 bytes at a time, using 0 padding. The IV is limited to 16 bytes and the key is limited to 32 bytes as per the AES encryption standard.

```
#include "AES.h"
#include <stdio.h>
#include <string.h>
```

Include dependency graph for cbc.h:



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



Functions

- void cbcEncryptFile (unsigned char ∗fileName, unsigned char ∗key, unsigned char ∗initializationVector, int keyLength, int initializationVectorLength, int isTextHex, int isKeyHex, int isIvHex)

cbcEncryptFile - Function to encrypt the file with name

- void cbcDecryptFile (unsigned char *fileName, unsigned char *key, unsigned char *initializationVector, int keyLength, int initializationVectorLength, int isTextHex, int isKeyHex, int isIvHex)

  cbcDecryptFile - Function to decrypt the file with name
- void cbcEncrypt (unsigned char *plainText, unsigned char *key, unsigned char *initializationVector, int plainTextLength, int keyLength, int initializationVectorLength, int isTextHex, int isKeyHex, int isIvHex)

  cbcEncrypt - Function to encrypt the user input pointed to by
- void cbcDecrypt (unsigned char *cipherText, unsigned char *key, unsigned char *initializationVector, int cipherTextLength, int keyLength, int initializationVectorLength, int isTextHex, int isKeyHex, int isIvHex)

  cbcDecrypt - Function to decrypt the user input pointed to by

Variables

- size_t VERBOSE

  Variable- size_t VERBOSE Used to dictate whether verbose output is printed to the terminal or not. If 0, does not print verbose. If 1, prints verbose.
- const size_t AES_BLOCK_SIZE

  Variable- const size_t AES_BLOCK_SIZE. Used to dictate the length in bytes of a single AES block used for encryption and decryption. Set to 16 bytes for a single block.

3.5.1 Detailed Description

Cipher Block Chaining (CBC) - AES Header file This file contains the function headers of the functions used for the CBC mode of AES encryption. This system supports both file and user input encryption, as hex or ascii input. If the user inputs data to be encrypted or decrypted, the result will be printed to the terminal, whereas if the user specifies a file to be encrypted or decrypted, a new file will be created and the result will be written to the file. The CBC Encryption platform encrypts and decrypts blocks 16 bytes at a time, using 0 padding. The IV is limited to 16 bytes and the key is limited to 32 bytes as per the AES encryption standard.

Authors

Mohamed Ameen Omar (u16055323)
Douglas Healy (u16018100)
Llewellyn Moyse (u15100708)

Version

0.1

Date

2019-03-28

Copyright

3.5.2 Function Documentation

3.5.2.1 cbcDecrypt()

```
void cbcDecrypt (
            unsigned char * cipherText,
            unsigned char * key,
            unsigned char * initializationVector,
            int cipherTextLength,
            int keyLength,
            int initializationVectorLength,
            int isTextHex,
            int isKeyHex,
            int isIvHex )
```

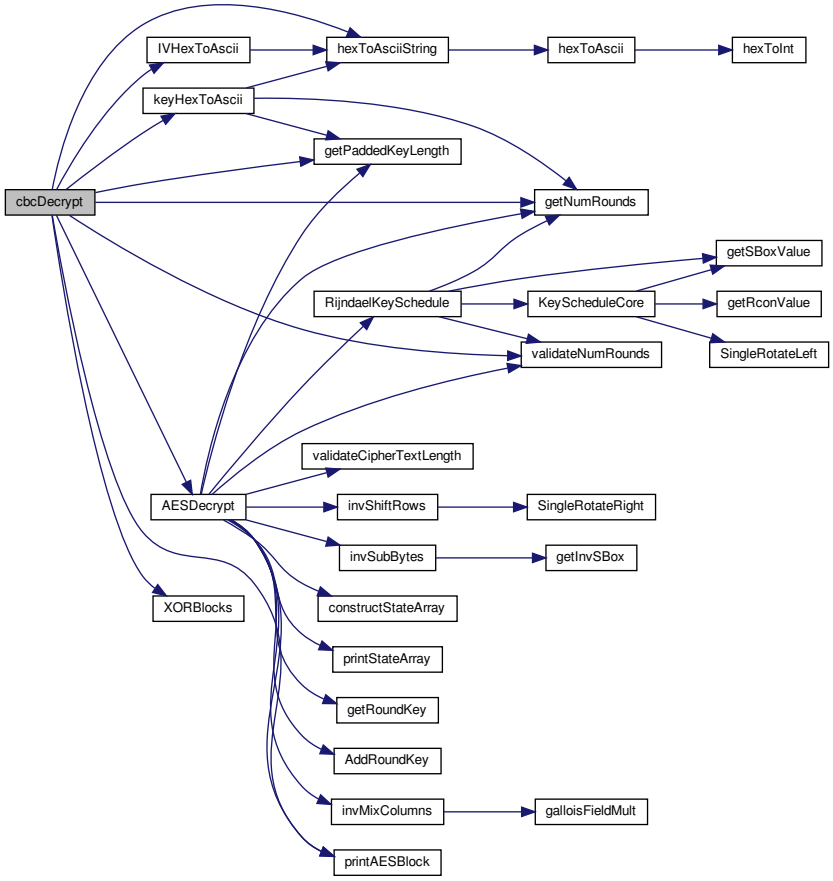
cbcDecrypt - Function to decrypt the user input pointed to by

Parameters

cipherText	and print decrypted result in hex to terminal. Performs decryption using the cbc mode prints the result to the terminal for each block in hex. If any input is hex, it will convert it to ascii, perform decryption and print it in hex. Makes use of zero padding.
char	- unsigned char∗ cipherText - the user input to be decrypted.
char	- unsigned char∗ key - the key to use for decryption.
char	- unsigned char∗ initializationVector - the initialization vector to use for cbc decryption.
cipherTextLength	- int - the length of the ciphertext to be encrypted in
cipherText.	
keyLength	- int - the length of the key specified in
key.	
initializationVectorLength	- int - the length of the key specified in
initializationVector.	
isTextHex	- int - boolean used to signify whether the file pointed to by
fileName	is a hexString or ASCII string. (1 = file is a hexString)
isKeyHex	- int - boolean used to signify whether the key pointed to by
key	is a hexString or ASCII string. (1 = file is a hexString)
isIvHex	- int - boolean used to signify whether the IV pointed to by
initializationVector	is a hexString or ASCII string. (1 = file is a hexString)

Definition at line 648 of file cbc.c.

Here is the call graph for this function:



3.5.2.2 cbcDecryptFile()

```
void cbcDecryptFile (
            unsigned char * fileName,
            unsigned char * key,
            unsigned char * initializationVector,
            int keyLength,
            int initializationVectorLength,
            int isTextHex,
            int isKeyHex,
            int isIvHex )
```

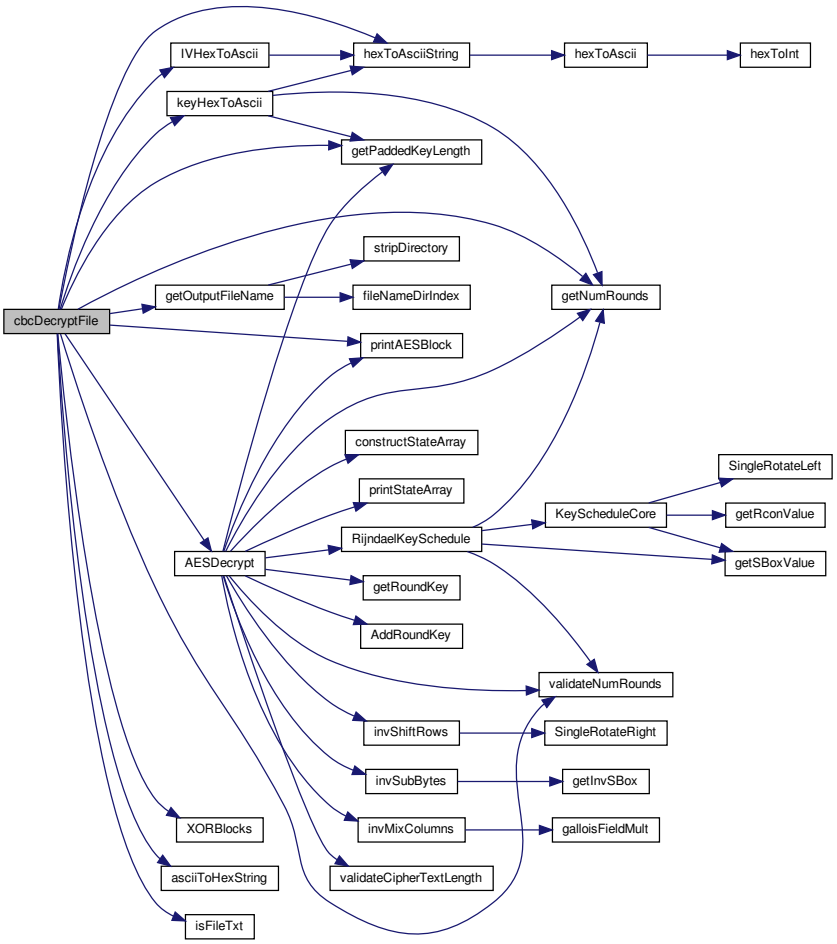
cbcDecryptFile - Function to decrypt the file with name

Parameters

fileName	and write the decrypted version to file with cbcDecrypted appended to the original filename. Performs decryption using the cbc mode and writes the result to a file. If any input is hex, it will convert it to ascii, perform decryption and write it back to the file in the same format as the input. That is if the input file was a hexString, the decrypted file will also contain a hex string. All terminal output, however, will be hex. Makes use of zero padding.
char	- unsigned char* fileName - the path to the file to be decrypted
char	- unsigned char* key - the key to use for decryption.
char	- unsigned char* initializationVector - the initialization vector to use for cbc decryption
keyLength	- int - the length of the key specified in
key.	
initializationVectorLength	- int - the length of the key specified in
initializationVector.	
isTextHex	- int - boolean used to signify whether the file pointed to by
fileName	is a hexString or ASCII string. (1 = file is a hexString)
isKeyHex	- int - boolean used to signify whether the key pointed to by
key	is a hexString or ASCII string. (1 = file is a hexString)
isIvHex	- int - boolean used to signify whether the IV pointed to by
initializationVector	is a hexString or ASCII string. (1 = file is a hexString)

Definition at line 263 of file cbc.c.

Here is the call graph for this function:



3.5.2.3 cbcEncrypt()

```
void cbcEncrypt (
            unsigned char * plainText,
            unsigned char * key,
            unsigned char * initializationVector,
            int plainTextLength,
            int keyLength,
            int initializationVectorLength,
            int isTextHex,
            int isKeyHex,
            int isIvHex )
```

cbcEncrypt - Function to encrypt the user input pointed to by

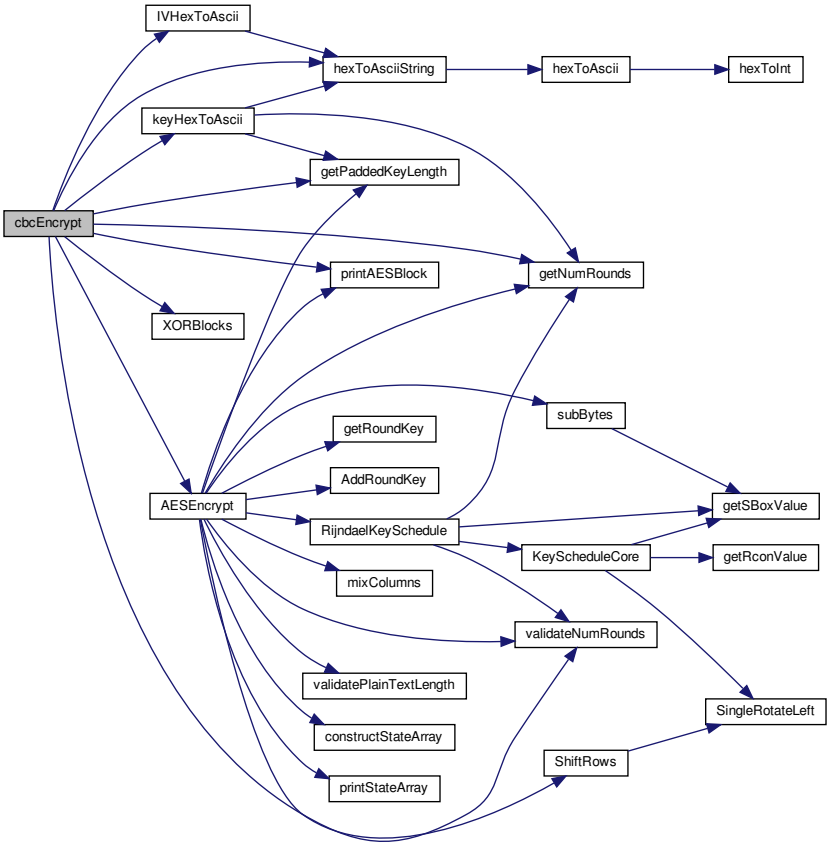
Parameters

plainText	and print encrypted result in hex to terminal. Performs encryption using the cbc mode prints the result to the terminal for each block in hex. If any input is hex, it will convert it to ascii, perform encryption and print it in hex. Makes use of zero padding.
char	- unsigned char∗ plainText - the user input to be encrypted.
char	- unsigned char∗ key - the key to use for encryption.
char	- unsigned char∗ initializationVector - the initialization vector to use for cbc encryption
plainTextLength	- - int - the length of the plaintext to be encrypted in
plainText.	
keyLength	- int - the length of the key specified in
key.	
initializationVectorLength	- int - the length of the key specified in
initializationVector.	
isTextHex	- int - boolean used to signify whether the file pointed to by
fileName	is a hexString or ASCII string. (1 = file is a hexString)
isKeyHex	- int - boolean used to signify whether the key pointed to by
key	is a hexString or ASCII string. (1 = file is a hexString)
isIvHex	- int - boolean used to signify whether the IV pointed to by
initializationVector	is a hexString or ASCII string. (1 = file is a hexString)

Process for CBC encrypt file: Read from file, if hex, convert if not do nothing. Store read converted in plaintextblock Pad the converted plaintextblock and store in paddedPlaintext Store IV and previous ciphertext in placeholderblock XOR paddedPlaintext and placeholder - store in intermediate Encrypt intermediate - store in cipherTextBlock Write to the file Free memory, read again and check that the read buffer length (amount read from the file iss not 0)

Definition at line 492 of file cbc.c.

Here is the call graph for this function:



3.5.2.4 cbcEncryptFile()

```
void cbcEncryptFile (
            unsigned char * fileName,
            unsigned char * key,
            unsigned char * initializationVector,
            int keyLength,
            int initializationVectorLength,
            int isTextHex,
            int isKeyHex,
            int isIvHex )
```

cbcEncryptFile - Function to encrypt the file with name

Parameters

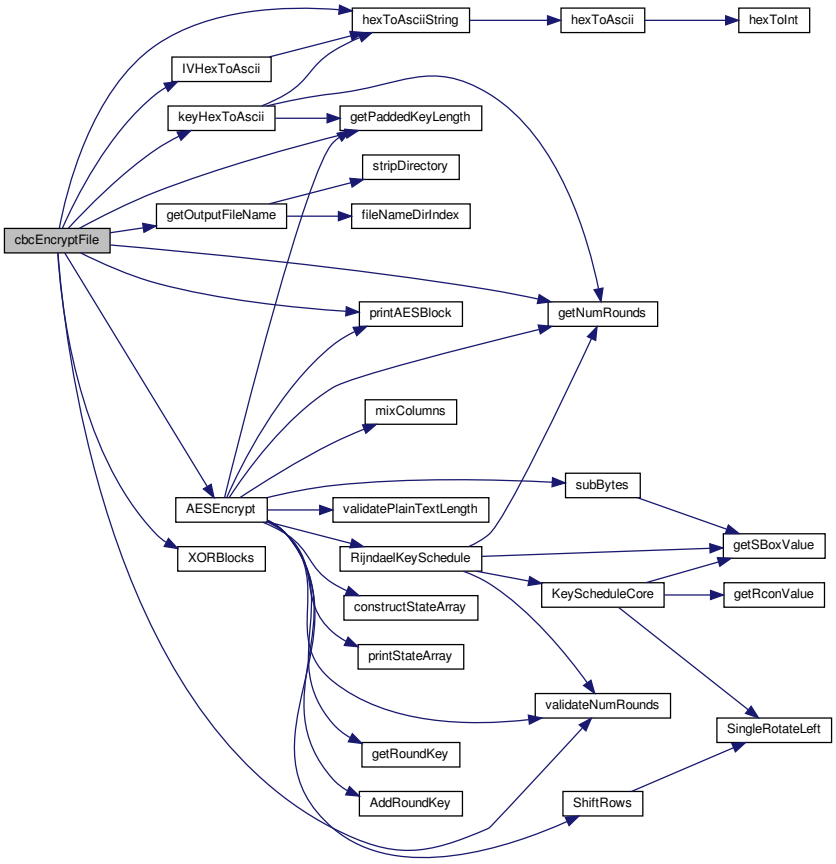
fileName	and write the encrypted version to file with cbcEncrypted appended to the original filename. Performs encryption using the cbc mode and writes the result to a file. If any input is hex, it will convert it to ascii, perform encryption and write it back as ASCII. All terminal output, however, will be hex. Makes use of zero padding.

Parameters

char	- unsigned char* fileName - the path to the file to be encrypted
char	- unsigned char* key - the key to use for encryption.
char	- unsigned char* initializationVector - the initialization vector to use for cbc encryption
keyLength	- int - the length of the key specified in
key.	
initializationVectorLength	- int - the length of the key specified in
initializationVector.	
isTextHex	- int - boolean used to signify whether the file pointed to by
fileName	is a hexString or ASCII string. (1 = file is a hexString)
isKeyHex	- int - boolean used to signify whether the key pointed to by
key	is a hexString or ASCII string. (1 = file is a hexString)
isIvHex	- int - boolean used to signify whether the IV pointed to by
initializationVector	is a hexString or ASCII string. (1 = file is a hexString)

Definition at line 38 of file cbc.c.

Here is the call graph for this function:

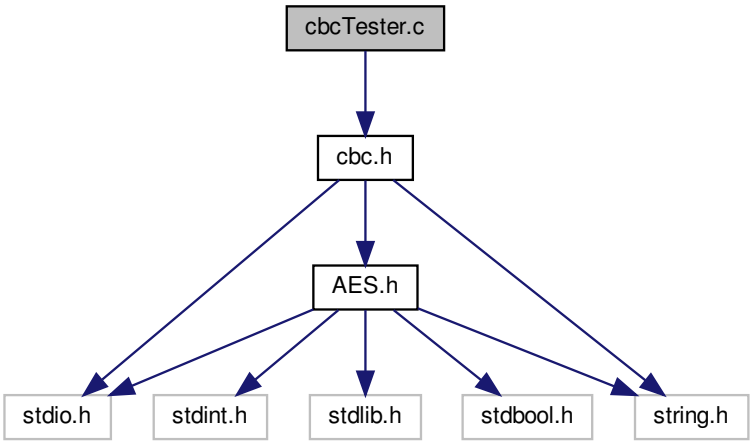


3.6 cbcTester.c File Reference

Main file.

```
#include "cbc.h"
```

Include dependency graph for cbcTester.c:



Functions

- int **main** (int argc, char ∗argv[ ])

3.6.1 Detailed Description

Main file.

Authors

Mohamed Ameen Omar (u16055323)
Douglas Healy (u16018100)
Llewellyn Moyse (u15100708)

Version

0.1

Date

2019-03-19

Copyright

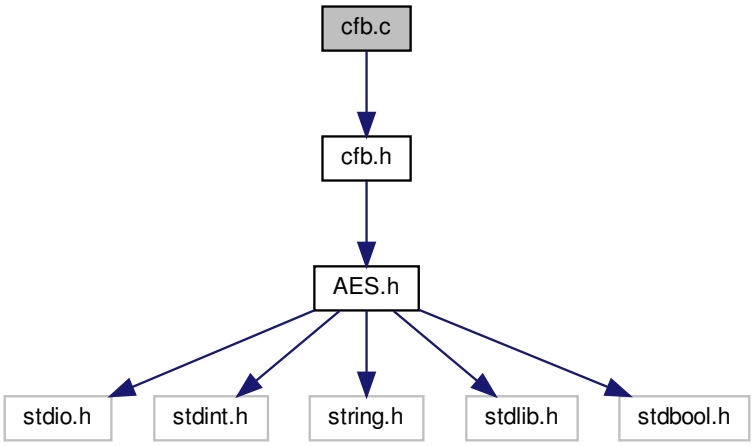
Copyright (c) 2019

3.7 cfb.c File Reference

Cipher Feedback (CFB) - AES implementation file This file contains the implementation of the functions used for the CFB mode of AES encryption. This system supports both file and user input encryption, as hex or ascii input. If the user inputs data to be encrypted or decrypted, the result will be printed to the terminal, whereas if the user specifies a file to be encrypted or decrypted, a new file will be created and the result will be written to the file. The CFB Encryption platform encrypts and decrypts blocks 16 bytes at a time, using 0 padding.

```
#include "cfb.h"
```

Include dependency graph for cfb.c:



Functions

- void cfbEncryptFile (unsigned char ∗fileName, unsigned char ∗key, unsigned char ∗initializationVector, int keyLength, int initializationVectorLength, int isTextHex, int isKeyHex, int isIvHex)

  cfbEncryptFile - Function to encrypt the file with name
- void cfbDecryptFile (unsigned char ∗fileName, unsigned char ∗key, unsigned char ∗initializationVector, int keyLength, int initializationVectorLength, int isTextHex, int isKeyHex, int isIvHex)

  cfbDecryptFile - Function to decrypt the file with name
- void cfbEncrypt (unsigned char ∗plainText, unsigned char ∗key, unsigned char ∗initializationVector, int plain↩ TextLength, int keyLength, int initializationVectorLength, int isTextHex, int isKeyHex, int isIvHex)

  cfbEncrypt - Function to encrypt the user input pointed to by
- void cfbDecrypt (unsigned char ∗cipherText, unsigned char ∗key, unsigned char ∗initializationVector, int cipherTextLength, int keyLength, int initializationVectorLength, int isTextHex, int isKeyHex, int isIvHex)

  cfbDecrypt - Function to decrypt the user input pointed to by

Variables

- size_t const shiftRegLength = 16

  Variable - size_t const shiftRegLength - used to specify the length of the shift register.
- size_t const streamSize = 16

  Variable - size_t const streamSize - used to speciffy the length of the stream per encryption round.

3.7.1 Detailed Description

Cipher Feedback (CFB) - AES implementation file This file contains the implementation of the functions used for the CFB mode of AES encryption. This system supports both file and user input encryption, as hex or ascii input. If the user inputs data to be encrypted or decrypted, the result will be printed to the terminal, whereas if the user specifies a file to be encrypted or decrypted, a new file will be created and the result will be written to the file. The CFB Encryption platform encrypts and decrypts blocks 16 bytes at a time, using 0 padding.

Authors

Mohamed Ameen Omar (u16055323)
Douglas Healy (u16018100)
Llewellyn Moyse (u15100708)

Version

0.1

Date

2019-03-28

Copyright

Copyright (c) 2019

3.7.2 Function Documentation

3.7.2.1 cfbDecrypt()

```
void cfbDecrypt (
            unsigned char * cipherText,
            unsigned char * key,
            unsigned char * initializationVector,
            int cipherTextLength,
            int keyLength,
            int initializationVectorLength,
            int isTextHex,
            int isKeyHex,
            int isIvHex )
```

cfbDecrypt - Function to decrypt the user input pointed to by

Parameters

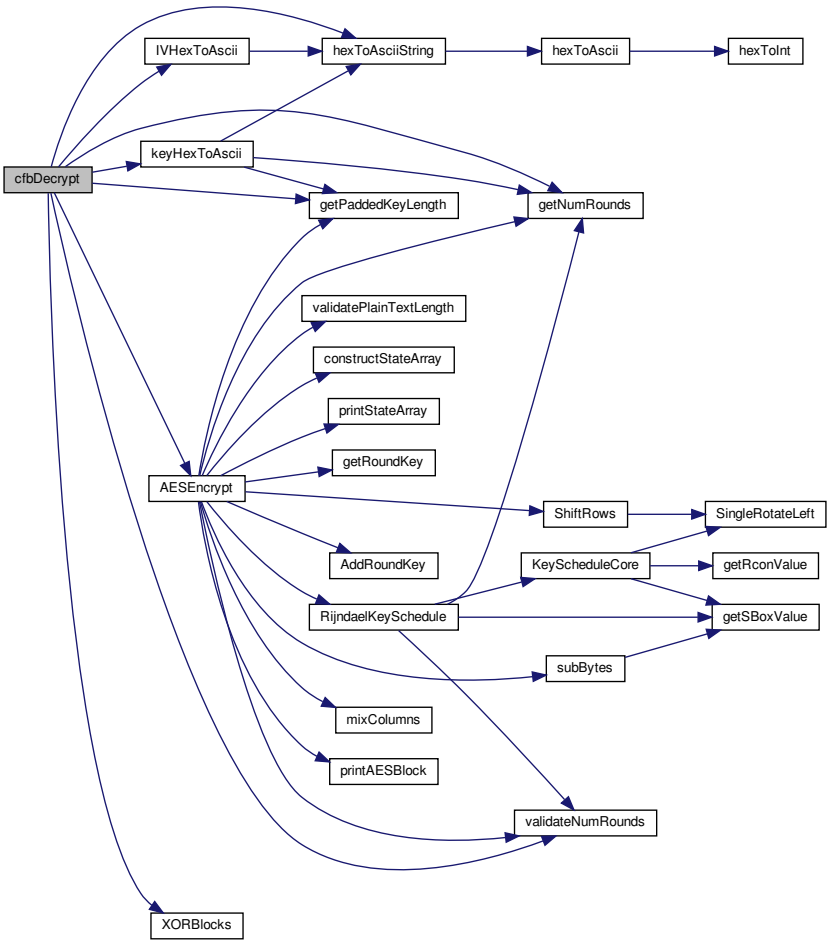
cipherText	and print decrypted result in hex to terminal. Performs decryption using the cfb mode prints the result to the terminal for each block in hex. If any input is hex, it will convert it to ascii, perform decryption and print it in hex. Makes use of zero padding.
char	- unsigned char* cipherText - the user input to be decrypted.

Parameters

char	- unsigned char∗ key - the key to use for decryption.
char	- unsigned char∗ initializationVector - the initialization vector to use for cfb decryption.
cipherTextLength	- int - the length of the ciphertext to be encrypted in
cipherText.	
keyLength	- int - the length of the key specified in
key.	
initializationVectorLength	- int - the length of the key specified in
initializationVector.	
isTextHex	- int - boolean used to signify whether the file pointed to by
fileName	is a hexString or ASCII string. (1 = file is a hexString)
isKeyHex	- int - boolean used to signify whether the key pointed to by
key	is a hexString or ASCII string. (1 = file is a hexString)
isIvHex	- int - boolean used to signify whether the IV pointed to by
initializationVector	is a hexString or ASCII string. (1 = file is a hexString)

Definition at line 550 of file cfb.c.

Here is the call graph for this function:



3.7.2.2 cfbDecryptFile()

```
void cfbDecryptFile (
            unsigned char * fileName,
            unsigned char * key,
            unsigned char * initializationVector,
            int keyLength,
            int initializationVectorLength,
            int isTextHex,
            int isKeyHex,
            int isIvHex )
```

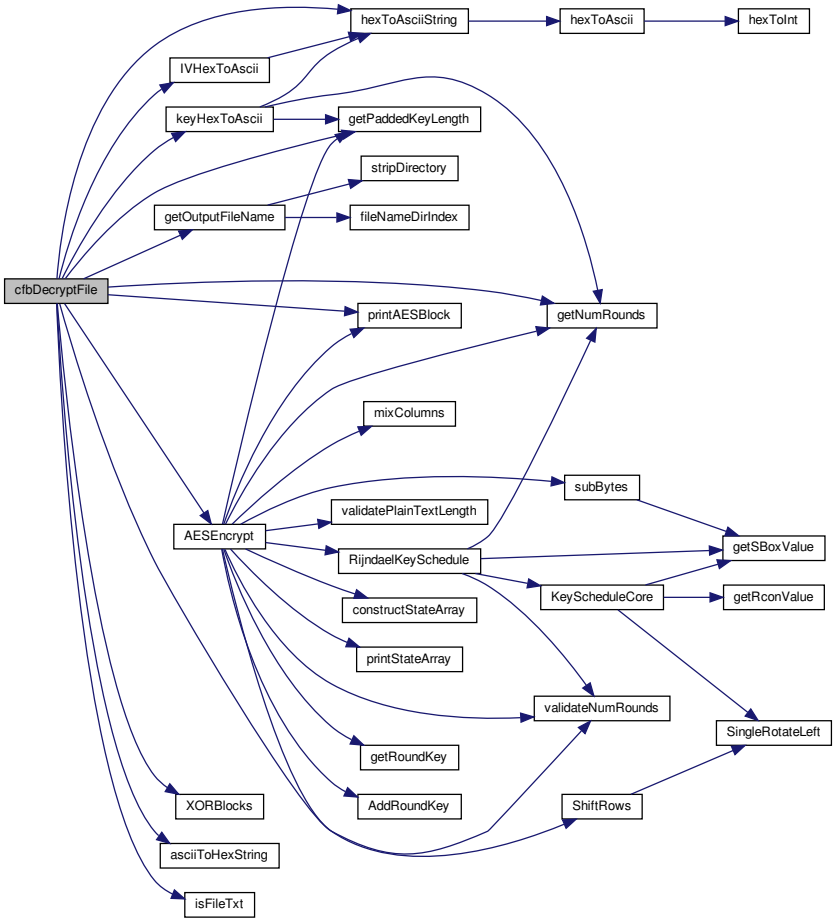
cfbDecryptFile - Function to decrypt the file with name

Parameters

fileName	and write the decrypted version to file with cfbDecrypted appended to the original filename. Performs decryption using the cfb mode and writes the result to a file. If any input is hex, it will convert it to ascii, perform decryption and write it back to the file in the same format as the input. That is if the input file was a hexString, the decrypted file will also contain a hex string. All terminal output, however, will be hex. Makes use of zero padding.
char	- unsigned char∗ fileName - the path to the file to be decrypted
char	- unsigned char∗ key - the key to use for decryption.
char	- unsigned char∗ initializationVector - the initialization vector to use for cfb decryption
keyLength	- int - the length of the key specified in
key.	
initializationVectorLength	- int - the length of the key specified in
initializationVector.	
isTextHex	- int - boolean used to signify whether the file pointed to by
fileName	is a hexString or ASCII string. (1 = file is a hexString)
isKeyHex	- int - boolean used to signify whether the key pointed to by
key	is a hexString or ASCII string. (1 = file is a hexString)
isIvHex	- int - boolean used to signify whether the IV pointed to by
initializationVector	is a hexString or ASCII string. (1 = file is a hexString)

Definition at line 221 of file cfb.c.

Here is the call graph for this function:



3.7.2.3 cfbEncrypt()

```
void cfbEncrypt (
            unsigned char * plainText,
            unsigned char * key,
            unsigned char * initializationVector,
            int plainTextLength,
            int keyLength,
            int initializationVectorLength,
            int isTextHex,
            int isKeyHex,
            int isIvHex )
```

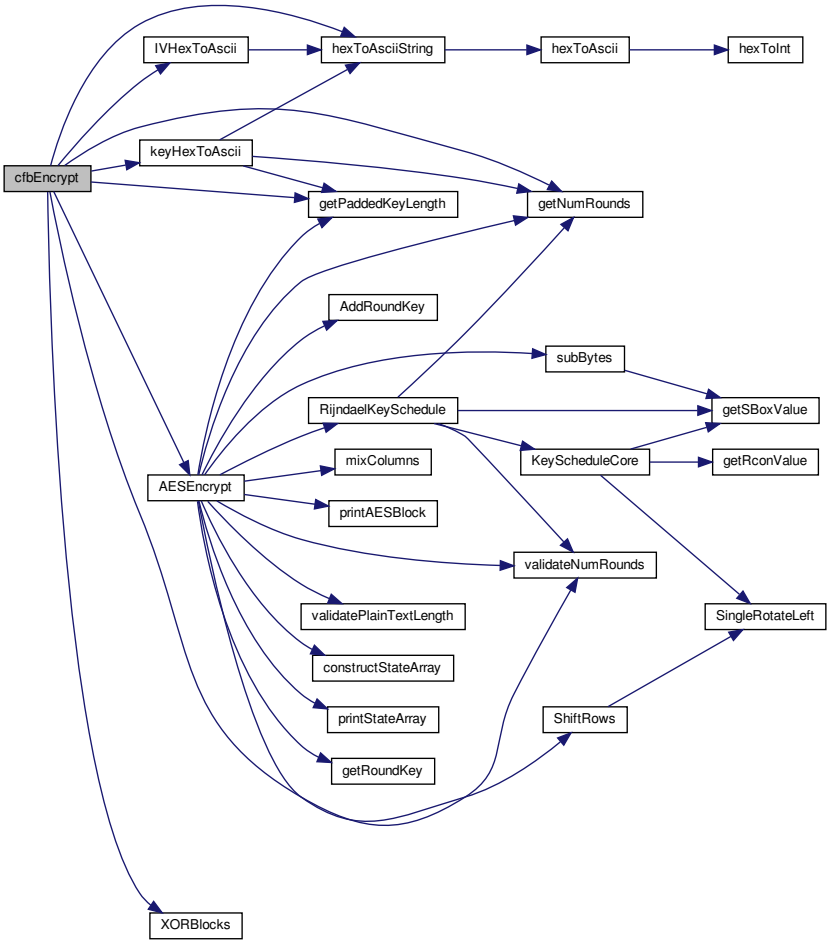
cfbEncrypt - Function to encrypt the user input pointed to by

Parameters

plainText	and print encrypted result in hex to terminal. Performs encryption using the cfb mode prints the result to the terminal for each block in hex. If any input is hex, it will convert it to ascii, perform encryption and print it in hex. Makes use of zero padding.
char	- unsigned char∗ plainText - the user input to be encrypted.
char	- unsigned char∗ key - the key to use for encryption.
char	- unsigned char∗ initializationVector - the initialization vector to use for cfb encryption
plainTextLength	- - int - the length of the plaintext to be encrypted in
plainText.	
keyLength	- int - the length of the key specified in
key.	
initializationVectorLength	- int - the length of the key specified in
initializationVector.	
isTextHex	- int - boolean used to signify whether the file pointed to by
fileName	is a hexString or ASCII string. (1 = file is a hexString)
isKeyHex	- int - boolean used to signify whether the key pointed to by
key	is a hexString or ASCII string. (1 = file is a hexString)
isIvHex	- int - boolean used to signify whether the IV pointed to by
initializationVector	is a hexString or ASCII string. (1 = file is a hexString)

Definition at line 405 of file cfb.c.

Here is the call graph for this function:



3.7.2.4 cfbEncryptFile()

```
void cfbEncryptFile (
            unsigned char * fileName,
            unsigned char * key,
            unsigned char * initializationVector,
            int keyLength,
            int initializationVectorLength,
            int isTextHex,
            int isKeyHex,
            int isIvHex )
```

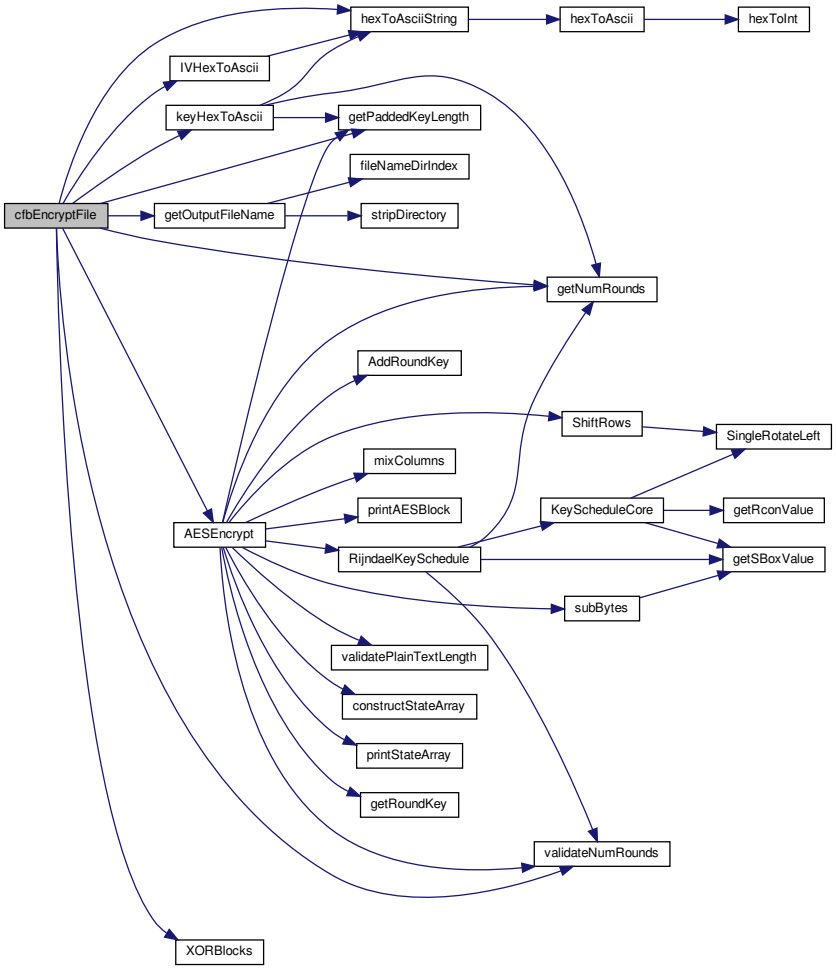
cfbEncryptFile - Function to encrypt the file with name

Parameters

fileName	and write the encrypted version to file with cfbEncrypted appended to the original filename. Performs encryption using the cfb mode and writes the result to a file. If any input is hex, it will convert it to ascii, perform encryption and write it back as ASCII. All terminal output, however, will be hex. Makes use of zero padding.
char	- unsigned char∗ fileName - the path to the file to be encrypted
char	- unsigned char∗ key - the key to use for encryption.
char	- unsigned char∗ initializationVector - the initialization vector to use for cfb encryption
keyLength	- int - the length of the key specified in
key.	
initializationVectorLength	- int - the length of the key specified in
initializationVector.	
isTextHex	- int - boolean used to signify whether the file pointed to by
fileName	is a hexString or ASCII string. (1 = file is a hexString)
isKeyHex	- int - boolean used to signify whether the key pointed to by
key	is a hexString or ASCII string. (1 = file is a hexString)
isIvHex	- int - boolean used to signify whether the IV pointed to by
initializationVector	is a hexString or ASCII string. (1 = file is a hexString)

Definition at line 46 of file cfb.c.

Here is the call graph for this function:

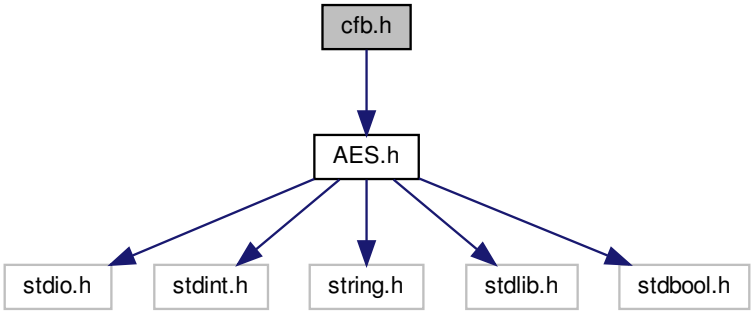


3.8 cfb.h File Reference

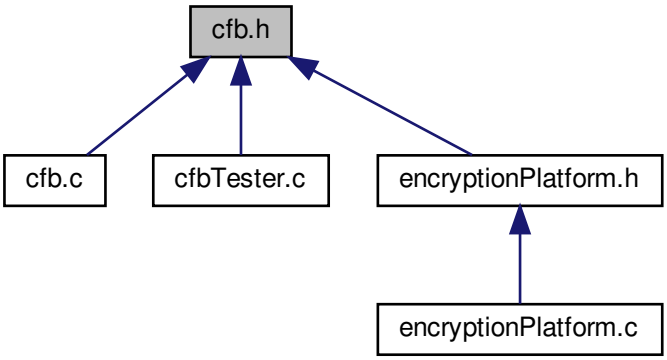
Cipher Feedback (CFB) - AES header file This file contains the function headers of the functions used for the CFB mode of AES encryption. This system supports both file and user input encryption, as hex or ascii input. If the user inputs data to be encrypted or decrypted, the result will be printed to the terminal, whereas if the user specifies a file to be encrypted or decrypted, a new file will be created and the result will be written to the file. The CFB Encryption platform encrypts and decrypts blocks 16 bytes at a time, using 0 padding.

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

Include dependency graph for cfb.h:



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



Functions

- void cfbEncryptFile (unsigned char ∗fileName, unsigned char ∗key, unsigned char ∗initializationVector, int keyLength, int initializationVectorLength, int isTextHex, int isKeyHex, int isIvHex)

  cfbEncryptFile - Function to encrypt the file with name
- void cfbDecryptFile (unsigned char ∗fileName, unsigned char ∗key, unsigned char ∗initializationVector, int keyLength, int initializationVectorLength, int isTextHex, int isKeyHex, int isIvHex)

  cfbDecryptFile - Function to decrypt the file with name
- void cfbEncrypt (unsigned char ∗plainText, unsigned char ∗key, unsigned char ∗initializationVector, int plain↩ TextLength, int keyLength, int initializationVectorLength, int isTextHex, int isKeyHex, int isIvHex)

  cfbEncrypt - Function to encrypt the user input pointed to by
- void cfbDecrypt (unsigned char ∗cipherText, unsigned char ∗key, unsigned char ∗initializationVector, int cipherTextLength, int keyLength, int initializationVectorLength, int isTextHex, int isKeyHex, int isIvHex)

  cfbDecrypt - Function to decrypt the user input pointed to by

Variables

- size_t VERBOSE

  Variable - VERBOSE - specifies if verbose output should be printed or not.
- const size_t AES_BLOCK_SIZE

  Variable - AES_BLOCK_SIZE - specifies the length per AES block - 16 bytes.
- const size_t shiftRegLength

  Variable - size_t const shiftRegLength - used to specify the length of the shift register.
- const size_t streamSize

  Variable - size_t const streamSize - used to speciffy the length of the stream per encryption round.

3.8.1 Detailed Description

Cipher Feedback (CFB) - AES header file This file contains the function headers of the functions used for the CFB mode of AES encryption. This system supports both file and user input encryption, as hex or ascii input. If the user inputs data to be encrypted or decrypted, the result will be printed to the terminal, whereas if the user specifies a file to be encrypted or decrypted, a new file will be created and the result will be written to the file. The CFB Encryption platform encrypts and decrypts blocks 16 bytes at a time, using 0 padding.

Authors

Mohamed Ameen Omar (u16055323)
Douglas Healy (u16018100)
Llewellyn Moyse (u15100708)

Version

0.1

Date

2019-03-28

Copyright

3.8.2 Function Documentation

3.8.2.1 cfbDecrypt()

```
void cfbDecrypt (
            unsigned char * cipherText,
            unsigned char * key,
            unsigned char * initializationVector,
            int cipherTextLength,
            int keyLength,
            int initializationVectorLength,
            int isTextHex,
            int isKeyHex,
            int isIvHex )
```

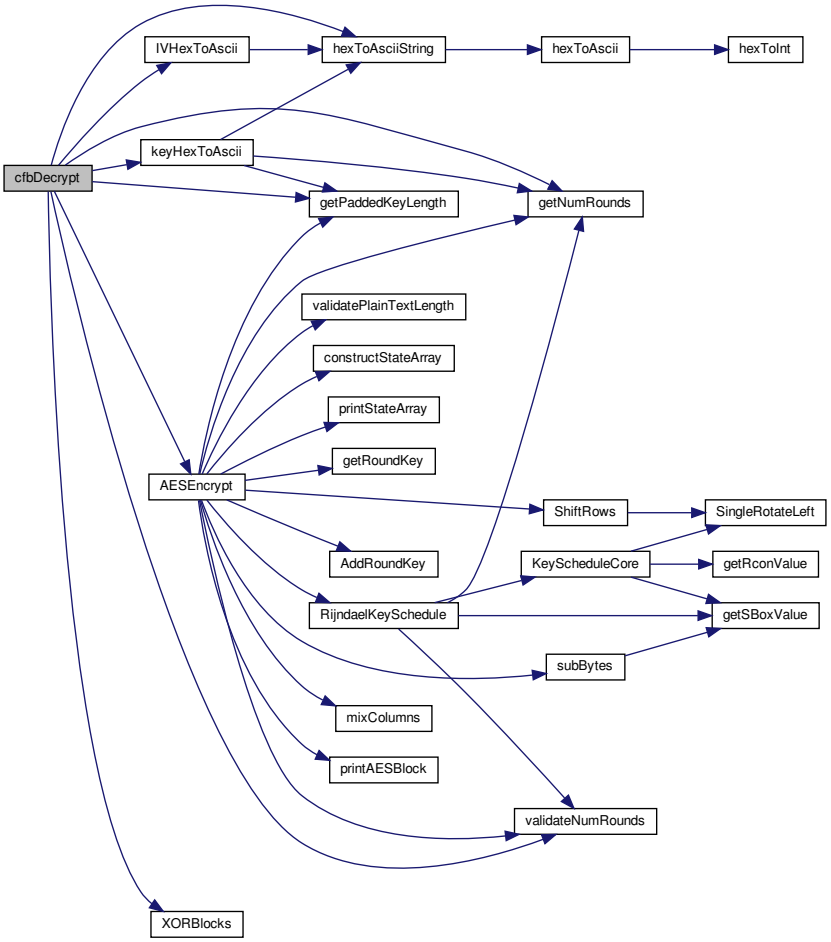
cfbDecrypt - Function to decrypt the user input pointed to by

Parameters

cipherText	and print decrypted result in hex to terminal. Performs decryption using the cfb mode prints the result to the terminal for each block in hex. If any input is hex, it will convert it to ascii, perform decryption and print it in hex. Makes use of zero padding.
char	- unsigned char∗ cipherText - the user input to be decrypted.
char	- unsigned char∗ key - the key to use for decryption.
char	- unsigned char∗ initializationVector - the initialization vector to use for cfb decryption.
cipherTextLength	- int - the length of the ciphertext to be encrypted in
cipherText.	
keyLength	- int - the length of the key specified in
key.	
initializationVectorLength	- int - the length of the key specified in
initializationVector.	
isTextHex	- int - boolean used to signify whether the file pointed to by
fileName	is a hexString or ASCII string. (1 = file is a hexString)
isKeyHex	- int - boolean used to signify whether the key pointed to by
key	is a hexString or ASCII string. (1 = file is a hexString)
isIvHex	- int - boolean used to signify whether the IV pointed to by
initializationVector	is a hexString or ASCII string. (1 = file is a hexString)

Definition at line 550 of file cfb.c.

Here is the call graph for this function:



3.8.2.2 cfbDecryptFile()

```
void cfbDecryptFile (
            unsigned char * fileName,
            unsigned char * key,
            unsigned char * initializationVector,
            int keyLength,
            int initializationVectorLength,
            int isTextHex,
            int isKeyHex,
            int isIvHex )
```

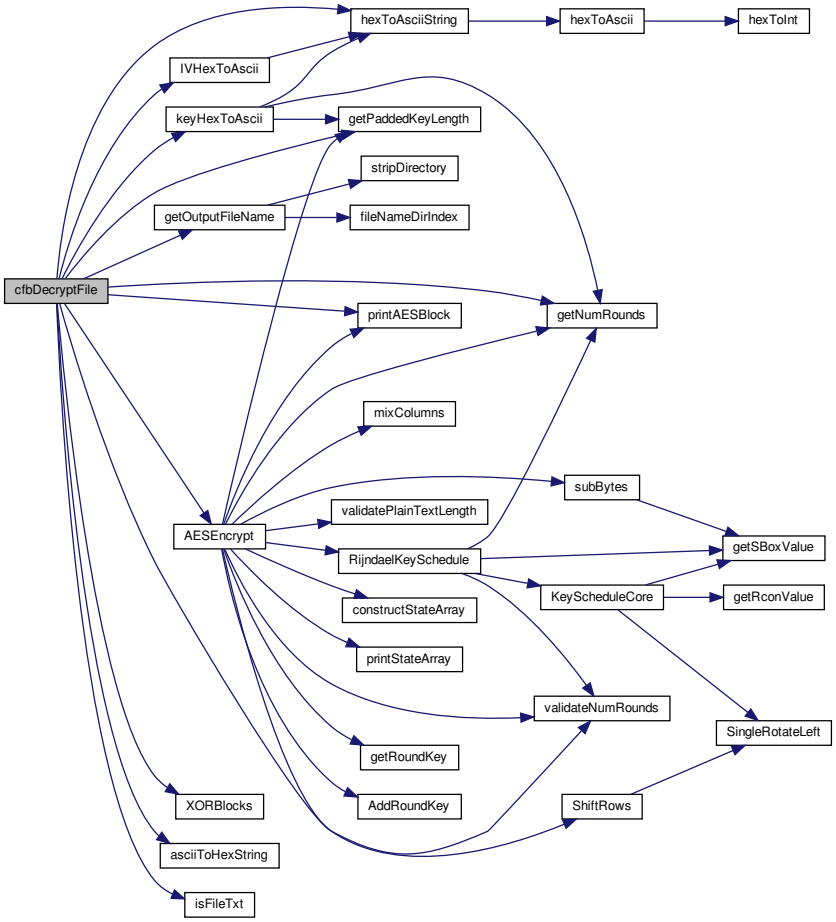
cfbDecryptFile - Function to decrypt the file with name

Parameters

fileName	and write the decrypted version to file with cfbDecrypted appended to the original filename. Performs decryption using the cfb mode and writes the result to a file. If any input is hex, it will convert it to ascii, perform decryption and write it back to the file in the same format as the input. That is if the input file was a hexString, the decrypted file will also contain a hex string. All terminal output, however, will be hex. Makes use of zero padding.
char	- unsigned char* fileName - the path to the file to be decrypted
char	- unsigned char* key - the key to use for decryption.
char	- unsigned char* initializationVector - the initialization vector to use for cfb decryption
keyLength	- int - the length of the key specified in
key.	
initializationVectorLength	- int - the length of the key specified in
initializationVector.	
isTextHex	- int - boolean used to signify whether the file pointed to by
fileName	is a hexString or ASCII string. (1 = file is a hexString)
isKeyHex	- int - boolean used to signify whether the key pointed to by
key	is a hexString or ASCII string. (1 = file is a hexString)
isIvHex	- int - boolean used to signify whether the IV pointed to by
initializationVector	is a hexString or ASCII string. (1 = file is a hexString)

Definition at line 221 of file cfb.c.

Here is the call graph for this function:



3.8.2.3 cfbEncrypt()

```
void cfbEncrypt (
            unsigned char * plainText,
            unsigned char * key,
            unsigned char * initializationVector,
            int plainTextLength,
            int keyLength,
            int initializationVectorLength,
            int isTextHex,
            int isKeyHex,
            int isIvHex )
```

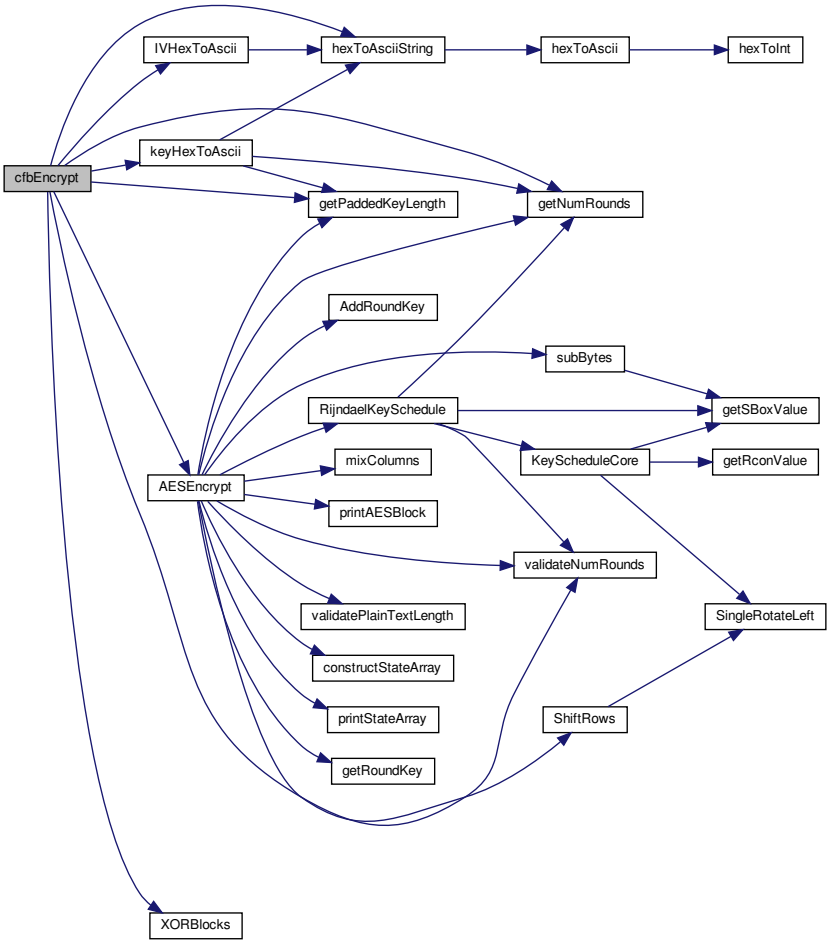
cfbEncrypt - Function to encrypt the user input pointed to by

Parameters

plainText	and print encrypted result in hex to terminal. Performs encryption using the cfb mode prints the result to the terminal for each block in hex. If any input is hex, it will convert it to ascii, perform encryption and print it in hex. Makes use of zero padding.
char	- unsigned char∗ plainText - the user input to be encrypted.
char	- unsigned char∗ key - the key to use for encryption.
char	- unsigned char∗ initializationVector - the initialization vector to use for cfb encryption
plainTextLength	- - int - the length of the plaintext to be encrypted in
plainText.	
keyLength	- int - the length of the key specified in
key.	
initializationVectorLength	- int - the length of the key specified in
initializationVector.	
isTextHex	- int - boolean used to signify whether the file pointed to by
fileName	is a hexString or ASCII string. (1 = file is a hexString)
isKeyHex	- int - boolean used to signify whether the key pointed to by
key	is a hexString or ASCII string. (1 = file is a hexString)
isIvHex	- int - boolean used to signify whether the IV pointed to by
initializationVector	is a hexString or ASCII string. (1 = file is a hexString)

Definition at line 405 of file cfb.c.

Here is the call graph for this function:



3.8.2.4 cfbEncryptFile()

```
void cfbEncryptFile (
            unsigned char * fileName,
            unsigned char * key,
            unsigned char * initializationVector,
            int keyLength,
            int initializationVectorLength,
            int isTextHex,
            int isKeyHex,
            int isIvHex )
```

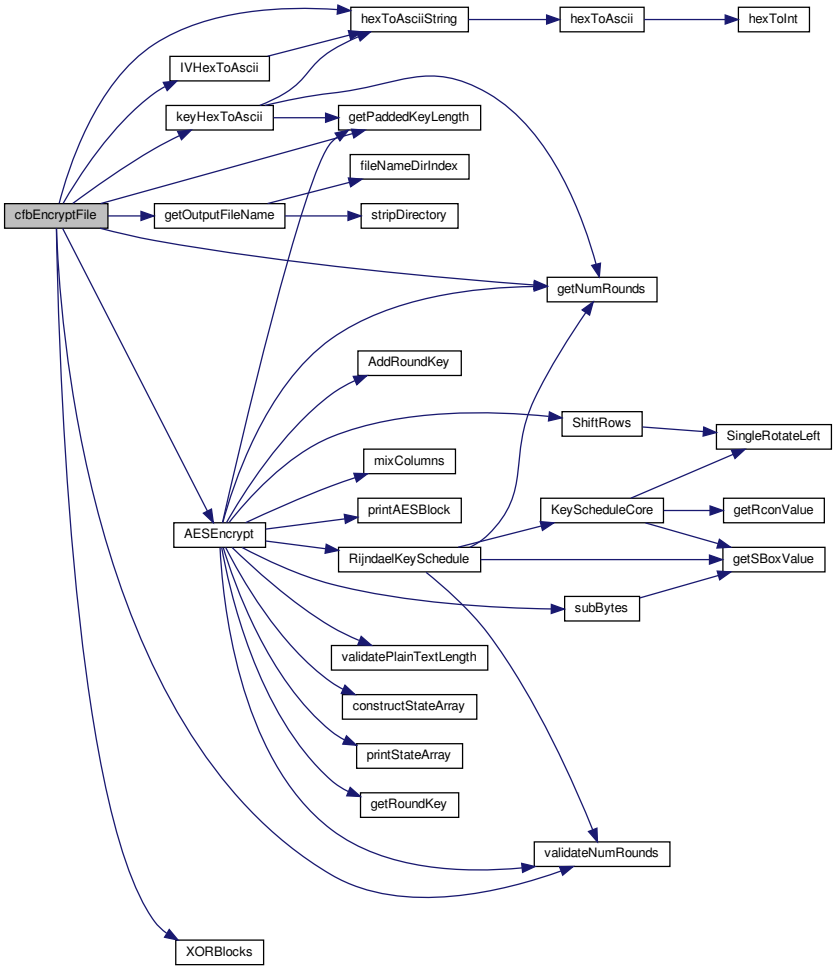
cfbEncryptFile - Function to encrypt the file with name

Parameters

fileName	and write the encrypted version to file with cfbEncrypted appended to the original filename. Performs encryption using the cfb mode and writes the result to a file. If any input is hex, it will convert it to ascii, perform encryption and write it back as ASCII. All terminal output, however, will be hex. Makes use of zero padding.
char	- unsigned char∗ fileName - the path to the file to be encrypted
char	- unsigned char∗ key - the key to use for encryption.
char	- unsigned char∗ initializationVector - the initialization vector to use for cfb encryption
keyLength	- int - the length of the key specified in
key.	
initializationVectorLength	- int - the length of the key specified in
initializationVector.	
isTextHex	- int - boolean used to signify whether the file pointed to by
fileName	is a hexString or ASCII string. (1 = file is a hexString)
isKeyHex	- int - boolean used to signify whether the key pointed to by
key	is a hexString or ASCII string. (1 = file is a hexString)
isIvHex	- int - boolean used to signify whether the IV pointed to by
initializationVector	is a hexString or ASCII string. (1 = file is a hexString)

Definition at line 46 of file cfb.c.

Here is the call graph for this function:



3.8.3 Variable Documentation

3.8.3.1 AES_BLOCK_SIZE

```
const size_t AES_BLOCK_SIZE
```

Variable - AES_BLOCK_SIZE - specifies the length per AES block - 16 bytes.

Variable - AES_BLOCK_SIZE - specifies the length per AES block - 16 bytes.

Definition at line 24 of file AES.c.

3.8.3.2 VERBOSE

```
size_t VERBOSE
```

Variable - VERBOSE - specifies if verbose output should be printed or not.

Variable - VERBOSE - specifies if verbose output should be printed or not.

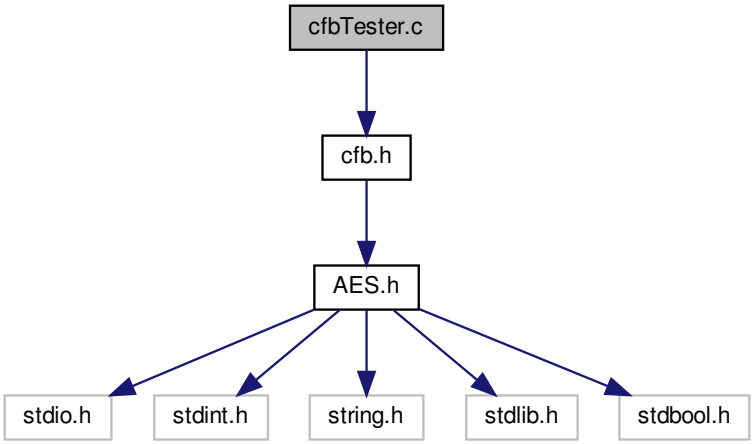
Definition at line 31 of file AES.c.

3.9 cfbTester.c File Reference

Main file.

```
#include "cfb.h"
```

Include dependency graph for cfbTester.c:



Functions

- int **main** (int argc, char ∗argv[ ])

3.9.1 Detailed Description

Main file.

Authors

Mohamed Ameen Omar (u16055323)
Douglas Healy (u16018100)
Llewellyn Moyse (u15100708)

Version

0.1

Date

2019-03-19

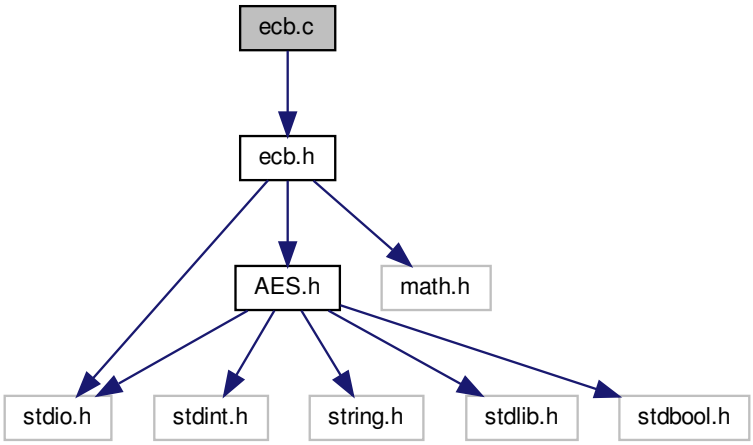
Copyright

3.10 ecb.c File Reference

Electronic code book (ECB) - AES Implementation file This file contains the implementation of the functions used for the ECB mode of AES encryption. This system supports both file and user input encryption, as hex or ascii input. If the user inputs data to be encrypted or decrypted, the result will be printed to the terminal, whereas if the user specifies a file to be encrypted or decrypted, a new file will be created and the result will be written to the file. The ECB Encryption platform encrypts and decrypts blocks 16 bytes at a time, using 0 padding.

```
#include "ecb.h"
```

Include dependency graph for ecb.c:



Functions

- void ecbEncrypt (unsigned char *plainText, unsigned char *key, int plainTextLength, int keyLength, int is↩ TextHex, int isKeyHex)

  ecbEncrypt - Function to encrypt the user input pointed to by
- void ecbDecrypt (unsigned char *cipherText, unsigned char *key, int cipherTextLength, int keyLength, int isTextHex, int isKeyHex)

  ecbDecrypt - Function to decrypt the user input pointed to by

- void ecbEcryptHelper (unsigned char ∗plainText, unsigned char ∗key, int plainTextLength, int keyLength)
  ecbEcryptHelper - Helper function used to encrypt the plaintext pointed to by
- void ecbDecryptHelper (unsigned char ∗cipherText, unsigned char ∗key, int cipherTextLength, int keyLength)
  ecbDecryptHelper - Helper function used to decrypt the ciphertext pointed to by
- void ecbEncryptFile (unsigned char ∗fileName, unsigned char ∗key, int keyLength, int isTextHex, int isKeyHex)
  ecbEncryptFile - Function to encrypt the file with name
- void ecbDecryptFile (unsigned char ∗fileName, unsigned char ∗key, int keyLength, int isTextHex, int isKeyHex)
  ecbDecryptFile - Function to encrypt the file with name

3.10.1 Detailed Description

Electronic code book (ECB) - AES Implementation file This file contains the implementation of the functions used for the ECB mode of AES encryption. This system supports both file and user input encryption, as hex or ascii input. If the user inputs data to be encrypted or decrypted, the result will be printed to the terminal, whereas if the user specifies a file to be encrypted or decrypted, a new file will be created and the result will be written to the file. The ECB Encryption platform encrypts and decrypts blocks 16 bytes at a time, using 0 padding.

Authors

Mohamed Ameen Omar (u16055323)
Douglas Healy (u16018100)
Llewellyn Moyse (u15100708)

Version

0.1

Date

2019-04-17

Copyright

3.10.2 Function Documentation

3.10.2.1 ecbDecrypt()

```
void ecbDecrypt (
            unsigned char * cipherText,
            unsigned char * key,
            int cipherTextLength,
            int keyLength,
            int isTextHex,
            int isKeyHex )
```

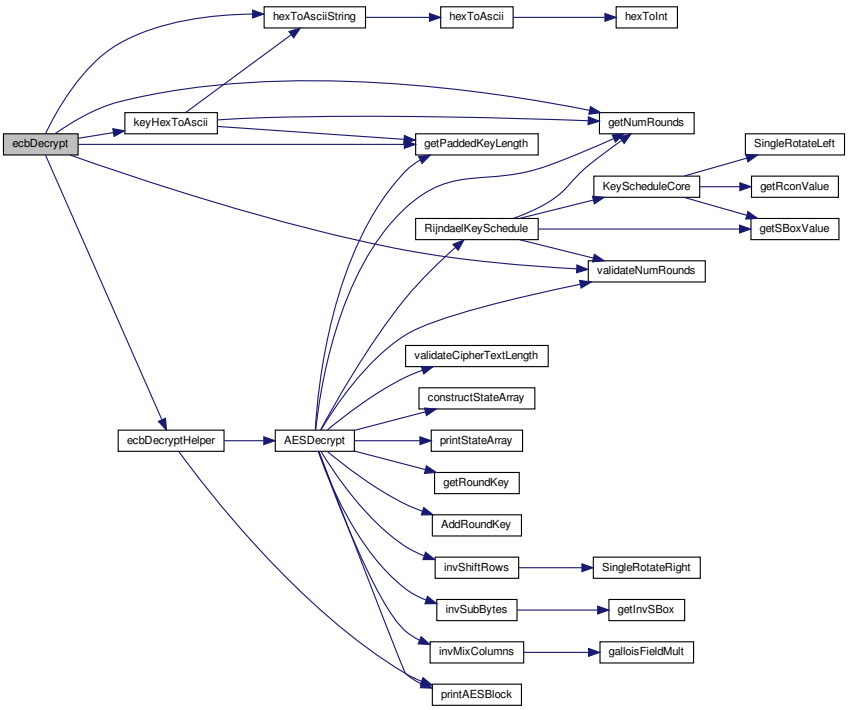
ecbDecrypt - Function to decrypt the user input pointed to by

Parameters

cipherText	and print decrypted result in hex to terminal. Performs decryption using the ECB mode prints the result to the terminal for each block in hex. If any input is hex, it will convert it to ascii, perform decryption and print it in hex. Makes use of zero padding.
char	- unsigned char* cipherText - the user input to be decrypted.
char	- unsigned char* key - the key to use for decryption.
cipherTextLength	- int - the length of the ciphertext to be encrypted in
cipherText.	
keyLength	- int - the length of the key specified in
key.	
isTextHex	- int - boolean used to signify whether the file pointed to by
fileName	is a hexString or ASCII string. (1 = file is a hexString)
isKeyHex	- int - boolean used to signify whether the key pointed to by
key	is a hexString or ASCII string. (1 = file is a hexString)

Definition at line 116 of file ecb.c.

Here is the call graph for this function:



3.10.2.2 ecbDecryptFile()

```
void ecbDecryptFile (
            unsigned char * fileName,
```

```
unsigned char * key,
int keyLength,
int isTextHex,
int isKeyHex )
```

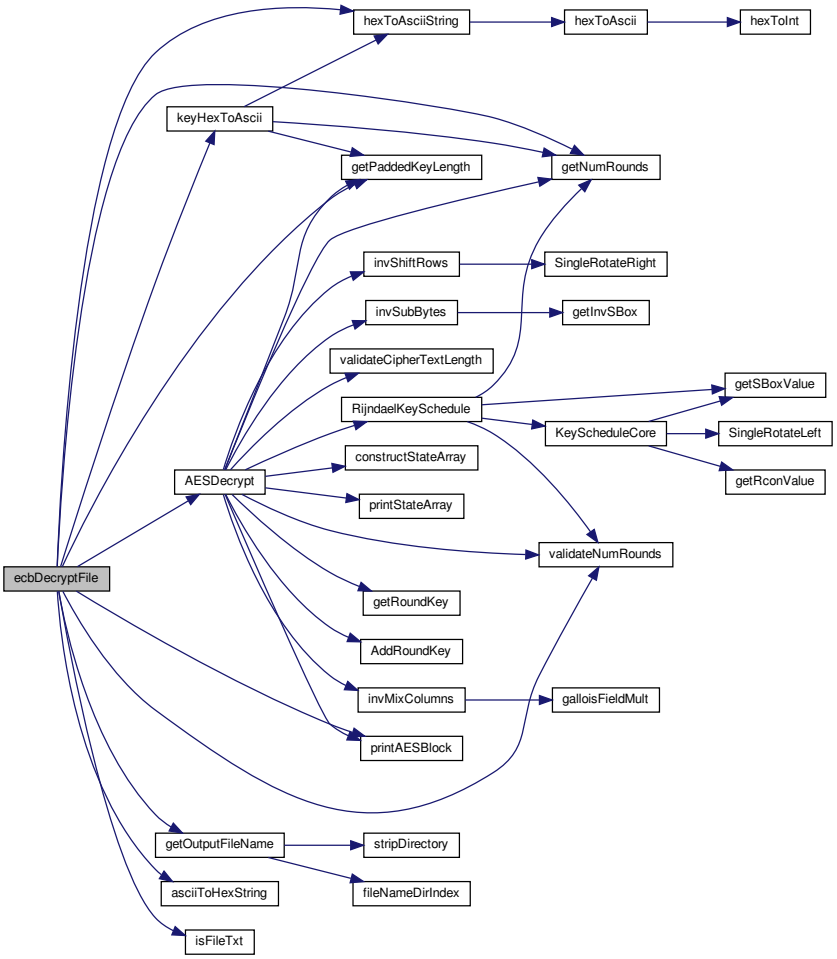
ecbDecryptFile - Function to encrypt the file with name

Parameters

fileName	and write the decrypted version to file with ecbDecrypted appended to the original filename. Performs decryption using the ecb mode and writes the result to a file. If any input is hex, it will convert it to ascii, perform encryption and write it back as ASCII. All terminal output, however, will be hex. Makes use of zero padding.
char	- unsigned char∗ fileName - the path to the file to be decrypted.
char	- unsigned char∗ key - the key to use for decryption.
keyLength	- int - the length of the key specified in
key.	
isTextHex	- int - boolean used to signify whether the file pointed to by
fileName	is a hexString or ASCII string. (1 = file is a hexString)
isKeyHex	- int - boolean used to signify whether the key pointed to by
key	is a hexString or ASCII string. (1 = file is a hexString)

Definition at line 370 of file ecb.c.

Here is the call graph for this function:



3.10.2.3 ecbDecryptHelper()

```
void ecbDecryptHelper (
            unsigned char * cipherText,
            unsigned char * key,
            int cipherTextLength,
            int keyLength )
```

ecbDecryptHelper - Helper function used to decrypt the ciphertext pointed to by

Parameters

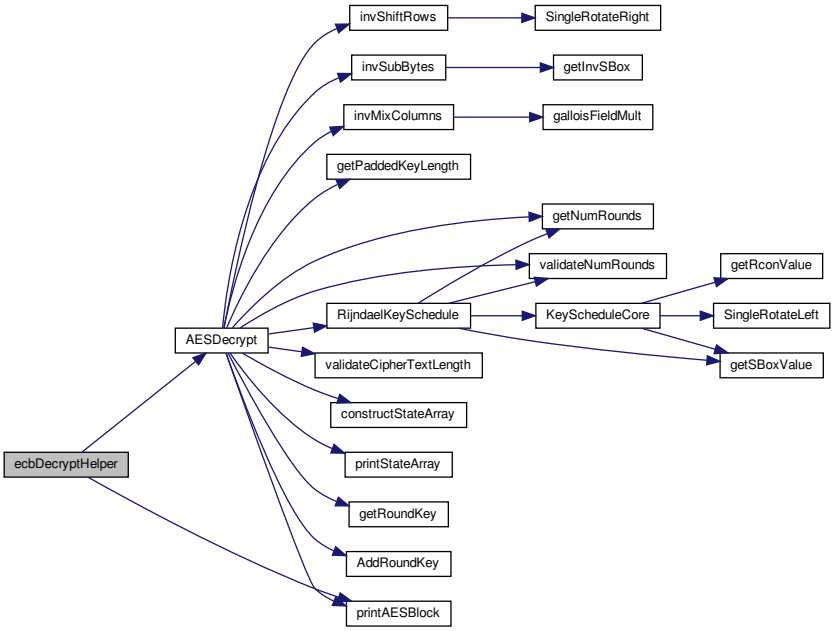
cipherText	using ECB mode of decryption and output the result to the terminal. Decrypts a single block of 16 bytes using AES decryption.
char	- unsigned char* cipherText - the block input to be encrypted.
char	- unsigned char* key - the key to use for decryption.

Parameters

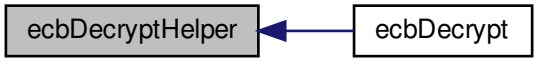
cipherTextLength	- int - the length of the ciphertext to be encrypted in
cipherText.	
keyLength	- int - the length of the key specified in
key.	

Definition at line 202 of file ecb.c.

Here is the call graph for this function:



Here is the caller graph for this function:



3.10.2.4 ecbEcryptHelper()

```
void ecbEcryptHelper (
            unsigned char * plainText,
            unsigned char * key,
            int plainTextLength,
            int keyLength )
```

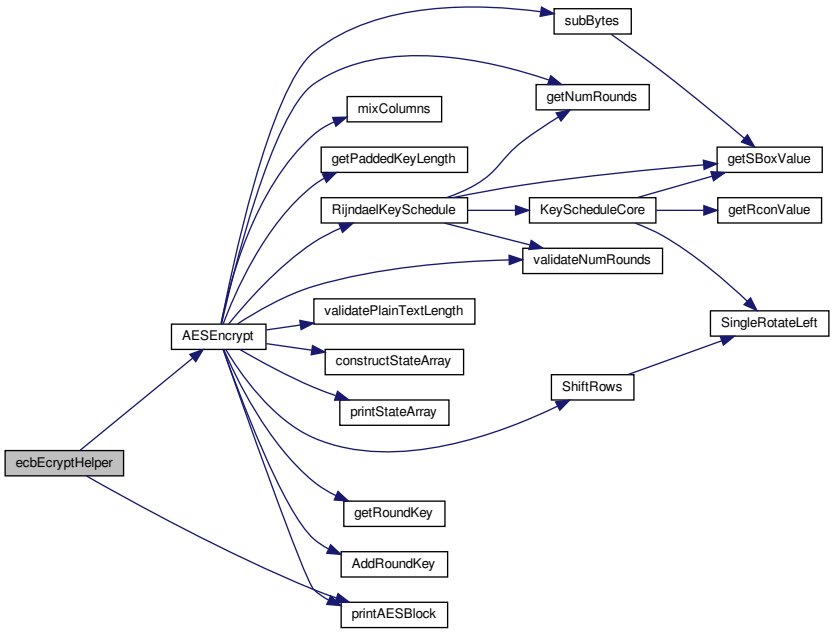
ecbEcryptHelper - Helper function used to encrypt the plaintext pointed to by

Parameters

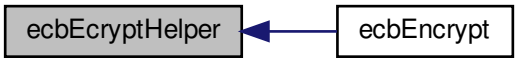
plainText	using ECB mode of encryption and output the result to the terminal. Encrypts a single block of 16 bytes using AES encryption.
char	- unsigned char* plainText - the block input to be encrypted.
char	- unsigned char* key - the key to use for encryption.
plainTextLength	- int - the length of the plaintext to be encrypted in
plainText.	
keyLength	- int - the length of the key specified in
key.	

Definition at line 187 of file ecb.c.

Here is the call graph for this function:



Here is the caller graph for this function:



3.10.2.5 ecbEncrypt()

```
void ecbEncrypt (
             unsigned char * plainText,
             unsigned char * key,
             int plainTextLength,
             int keyLength,
             int isTextHex,
             int isKeyHex )
```

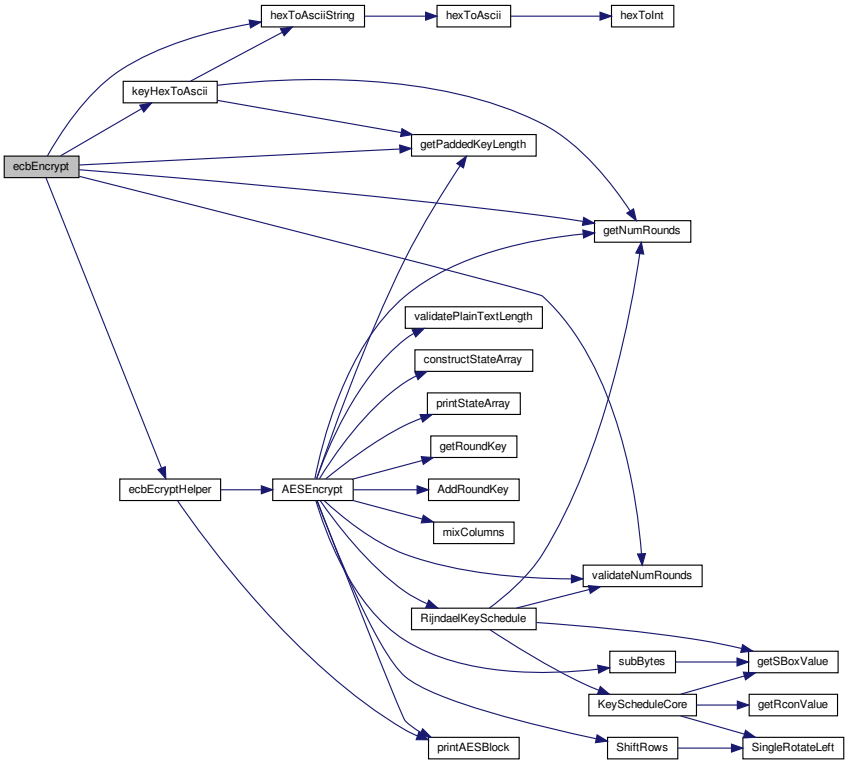
ecbEncrypt - Function to encrypt the user input pointed to by

Parameters

plainText	and print encrypted result in hex to terminal. Performs encryption using the ECB mode prints the result to the terminal for each block in hex. If any input is hex, it will convert it to ascii, perform encryption and print it in hex. Makes use of zero padding.
char	- unsigned char∗ plainText - the user input to be encrypted.
char	- unsigned char∗ key - the key to use for encryption.
plainTextLength	- int - the length of the plaintext to be encrypted in
plainText.	
keyLength	- int - the length of the key specified in
key.	
isTextHex	- int - boolean used to signify whether the file pointed to by
fileName	is a hexString or ASCII string. (1 = file is a hexString)
isKeyHex	- int - boolean used to signify whether the key pointed to by
key	is a hexString or ASCII string. (1 = file is a hexString)

Definition at line 34 of file ecb.c.

Here is the call graph for this function:



3.10.2.6 ecbEncryptFile()

```
void ecbEncryptFile (
            unsigned char * fileName,
            unsigned char * key,
            int keyLength,
            int isTextHex,
            int isKeyHex )
```

ecbEncryptFile - Function to encrypt the file with name

Parameters

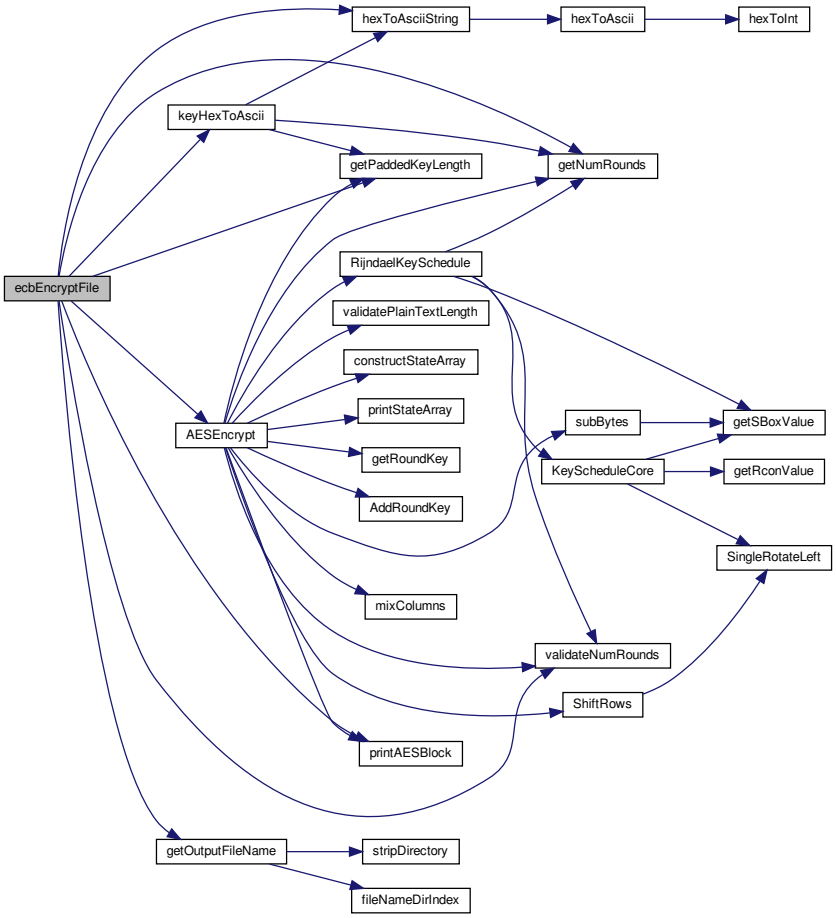
fileName	and write the encrypted version to file with ecbEncrypted appended to the original filename. Performs encryption using the ecb mode and writes the result to a file. If any input is hex, it will convert it to ascii, perform encryption and write it back as ASCII. All terminal output, however, will be hex. Makes use of zero padding.
char	- unsigned char* fileName - the path to the file to be encrypted.
char	- unsigned char* key - the key to use for encryption.
keyLength	- int - the length of the key specified in
key.	
isTextHex	- int - boolean used to signify whether the file pointed to by

Parameters

fileName	is a hexString or ASCII string. (1 = file is a hexString)
isKeyHex	- int - boolean used to signify whether the key pointed to by
key	is a hexString or ASCII string. (1 = file is a hexString)

Definition at line 221 of file ecb.c.

Here is the call graph for this function:



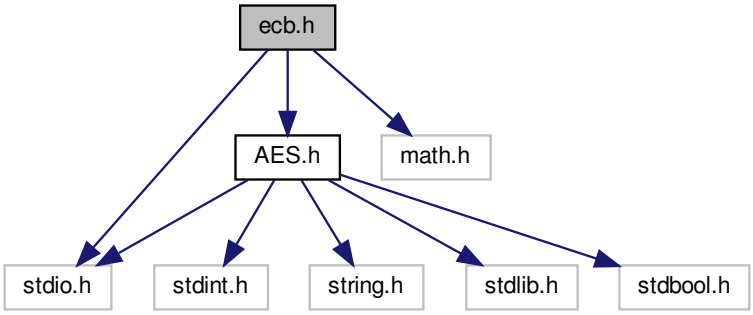
3.11 ecb.h File Reference

Electronic code book (ECB) - AES header file This file contains the function headers of the functions used for the ECB mode of AES encryption. This system supports both file and user input encryption, as hex or ascii input. If the user inputs data to be encrypted or decrypted, the result will be printed to the terminal, whereas if the user specifies a file to be encrypted or decrypted, a new file will be created and the result will be written to the file. The ECB Encryption platform encrypts and decrypts blocks 16 bytes at a time, using 0 padding.

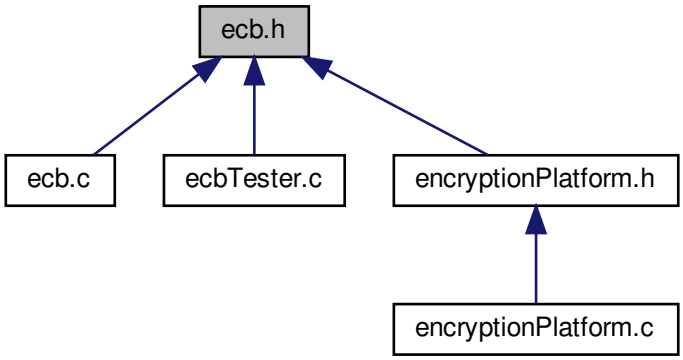
```
#include "AES.h"
#include "stdio.h"
```

```
#include "math.h"
```

Include dependency graph for ecb.h:



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



Functions

- void ecbEncrypt (unsigned char ∗plainText, unsigned char ∗key, int plainTextLength, int keyLength, int is↩ TextHex, int isKeyHex)

  ecbEncrypt - Function to encrypt the user input pointed to by
- void ecbDecrypt (unsigned char ∗cipherText, unsigned char ∗key, int cipherTextLength, int keyLength, int isTextHex, int isKeyHex)

  ecbDecrypt - Function to decrypt the user input pointed to by
- void ecbEcryptHelper (unsigned char ∗plainText, unsigned char ∗key, int plainTextLength, int keyLength)

  ecbEcryptHelper - Helper function used to encrypt the plaintext pointed to by
- void ecbDecryptHelper (unsigned char ∗cipherText, unsigned char ∗key, int cipherTextLength, int keyLength)

  ecbDecryptHelper - Helper function used to decrypt the ciphertext pointed to by
- void ecbEncryptFile (unsigned char ∗fileName, unsigned char ∗key, int keyLength, int isTextHex, int isKeyHex)

ecbEncryptFile - Function to encrypt the file with name

- void ecbDecryptFile (unsigned char *fileName, unsigned char *key, int keyLength, int isTextHex, int isKeyHex)

  ecbDecryptFile - Function to encrypt the file with name

3.11.1 Detailed Description

Electronic code book (ECB) - AES header file This file contains the function headers of the functions used for the ECB mode of AES encryption. This system supports both file and user input encryption, as hex or ascii input. If the user inputs data to be encrypted or decrypted, the result will be printed to the terminal, whereas if the user specifies a file to be encrypted or decrypted, a new file will be created and the result will be written to the file. The ECB Encryption platform encrypts and decrypts blocks 16 bytes at a time, using 0 padding.

Authors

Mohamed Ameen Omar (u16055323)
Douglas Healy (u16018100)
Llewellyn Moyse (u15100708)

Version

0.1

Date

2019-04-17

Copyright

3.11.2 Function Documentation

3.11.2.1 ecbDecrypt()

```
void ecbDecrypt (
            unsigned char * cipherText,
            unsigned char * key,
            int cipherTextLength,
            int keyLength,
            int isTextHex,
            int isKeyHex )
```

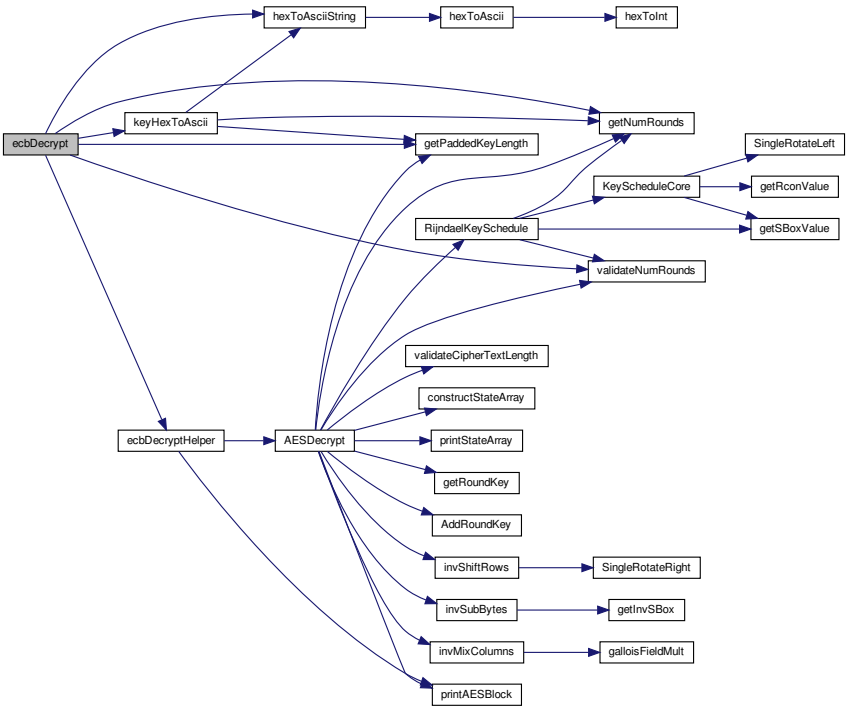
ecbDecrypt - Function to decrypt the user input pointed to by

Parameters

cipherText	and print decrypted result in hex to terminal. Performs decryption using the ECB mode prints the result to the terminal for each block in hex. If any input is hex, it will convert it to ascii, perform decryption and print it in hex. Makes use of zero padding.
char	- unsigned char* cipherText - the user input to be decrypted.
char	- unsigned char* key - the key to use for decryption.
cipherTextLength	- int - the length of the ciphertext to be encrypted in
cipherText.	
keyLength	- int - the length of the key specified in
key.	
isTextHex	- int - boolean used to signify whether the file pointed to by
fileName	is a hexString or ASCII string. (1 = file is a hexString)
isKeyHex	- int - boolean used to signify whether the key pointed to by
key	is a hexString or ASCII string. (1 = file is a hexString)

Definition at line 116 of file ecb.c.

Here is the call graph for this function:



3.11.2.2 ecbDecryptFile()

```
void ecbDecryptFile (
            unsigned char * fileName,
```

```
unsigned char * key,
int keyLength,
int isTextHex,
int isKeyHex )
```

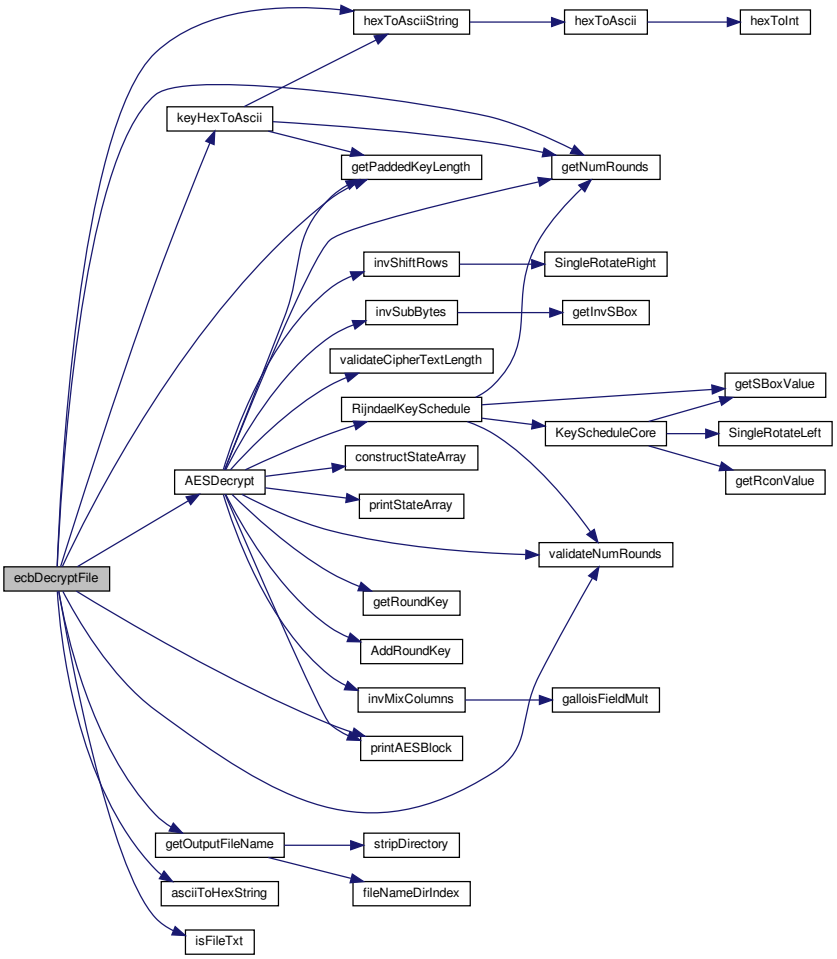
ecbDecryptFile - Function to encrypt the file with name

Parameters

fileName	and write the decrypted version to file with ecbDecrypted appended to the original filename. Performs decryption using the ecb mode and writes the result to a file. If any input is hex, it will convert it to ascii, perform encryption and write it back as ASCII. All terminal output, however, will be hex. Makes use of zero padding.
char	- unsigned char* fileName - the path to the file to be decrypted.
char	- unsigned char* key - the key to use for decryption.
keyLength	- int - the length of the key specified in
key.	
isTextHex	- int - boolean used to signify whether the file pointed to by
fileName	is a hexString or ASCII string. (1 = file is a hexString)
isKeyHex	- int - boolean used to signify whether the key pointed to by
key	is a hexString or ASCII string. (1 = file is a hexString)

Definition at line 370 of file ecb.c.

Here is the call graph for this function:



3.11.2.3 ecbDecryptHelper()

```
void ecbDecryptHelper (
            unsigned char * cipherText,
            unsigned char * key,
            int cipherTextLength,
            int keyLength )
```

ecbDecryptHelper - Helper function used to decrypt the ciphertext pointed to by

Parameters

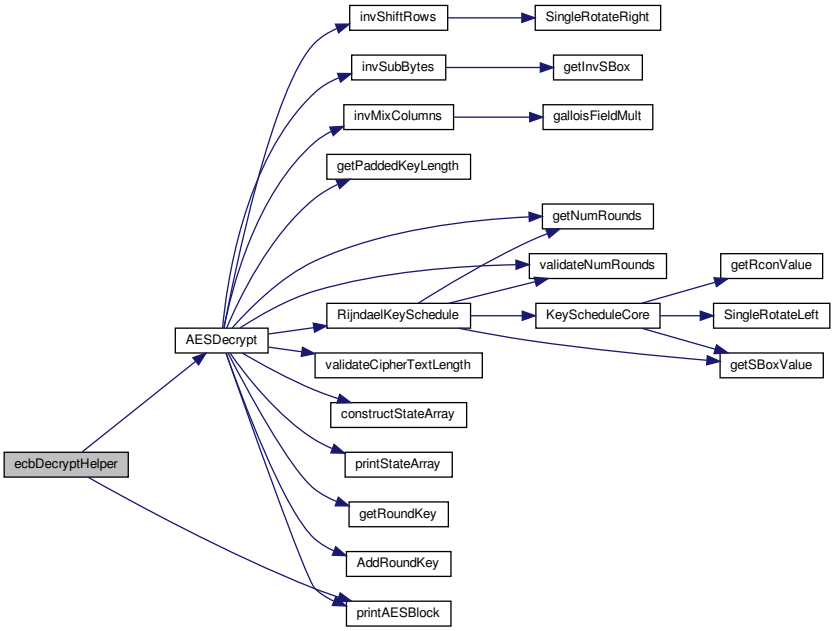
cipherText	using ECB mode of decryption and output the result to the terminal. Decrypts a single block of 16 bytes using AES decryption.
char	- unsigned char* cipherText - the block input to be encrypted.
char	- unsigned char* key - the key to use for decryption.

Parameters

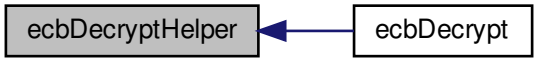
cipherTextLength	- int - the length of the ciphertext to be encrypted in
cipherText.	
keyLength	- int - the length of the key specified in
key.	

Definition at line 202 of file ecb.c.

Here is the call graph for this function:



Here is the caller graph for this function:



3.11.2.4 ecbEcryptHelper()

```
void ecbEcryptHelper (
            unsigned char * plainText,
            unsigned char * key,
            int plainTextLength,
            int keyLength )
```

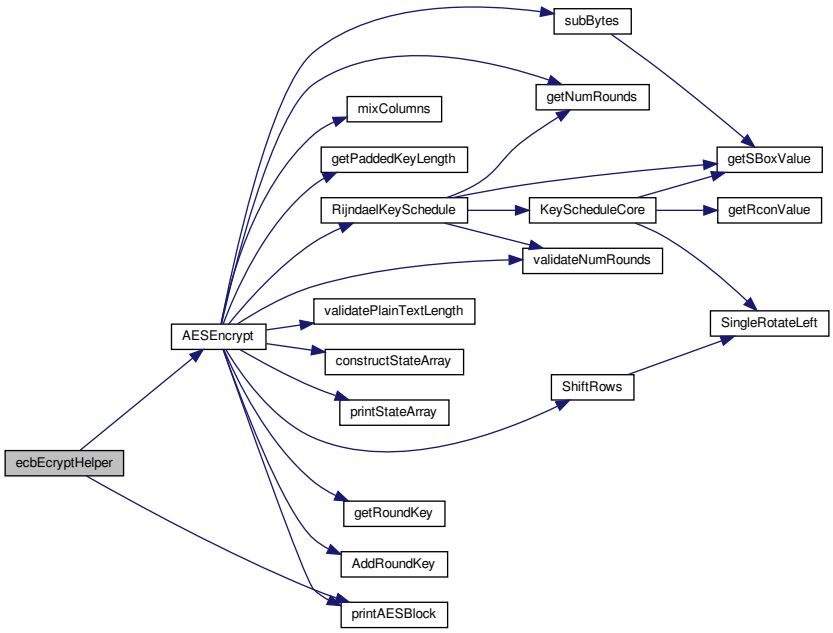
ecbEcryptHelper - Helper function used to encrypt the plaintext pointed to by

Parameters

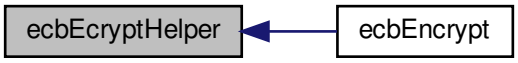
plainText	using ECB mode of encryption and output the result to the terminal. Encrypts a single block of 16 bytes using AES encryption.
char	- unsigned char* plainText - the block input to be encrypted.
char	- unsigned char* key - the key to use for encryption.
plainTextLength	- int - the length of the plaintext to be encrypted in
plainText.	
keyLength	- int - the length of the key specified in
key.	

Definition at line 187 of file ecb.c.

Here is the call graph for this function:



Here is the caller graph for this function:



3.11.2.5 ecbEncrypt()

```
void ecbEncrypt (
             unsigned char * plainText,
             unsigned char * key,
             int plainTextLength,
             int keyLength,
             int isTextHex,
             int isKeyHex )
```

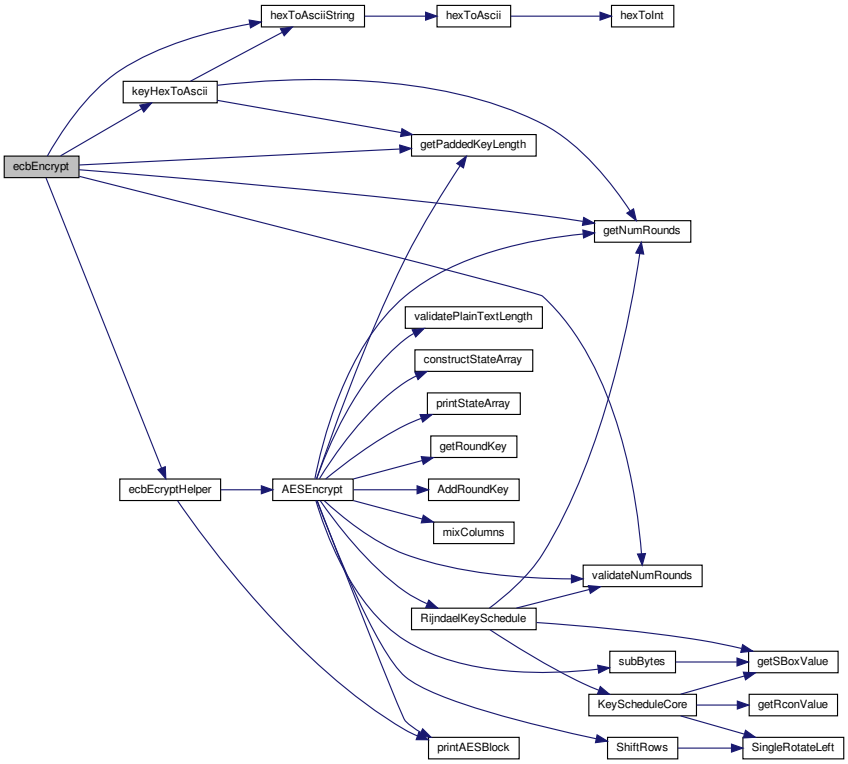
ecbEncrypt - Function to encrypt the user input pointed to by

Parameters

plainText	and print encrypted result in hex to terminal. Performs encryption using the ECB mode prints the result to the terminal for each block in hex. If any input is hex, it will convert it to ascii, perform encryption and print it in hex. Makes use of zero padding.
char	- unsigned char∗ plainText - the user input to be encrypted.
char	- unsigned char∗ key - the key to use for encryption.
plainTextLength	- int - the length of the plaintext to be encrypted in
plainText.	
keyLength	- int - the length of the key specified in
key.	
isTextHex	- int - boolean used to signify whether the file pointed to by
fileName	is a hexString or ASCII string. (1 = file is a hexString)
isKeyHex	- int - boolean used to signify whether the key pointed to by
key	is a hexString or ASCII string. (1 = file is a hexString)

Definition at line 34 of file ecb.c.

Here is the call graph for this function:



3.11.2.6 ecbEncryptFile()

```
void ecbEncryptFile (
            unsigned char * fileName,
            unsigned char * key,
            int keyLength,
            int isTextHex,
            int isKeyHex )
```

ecbEncryptFile - Function to encrypt the file with name

Parameters

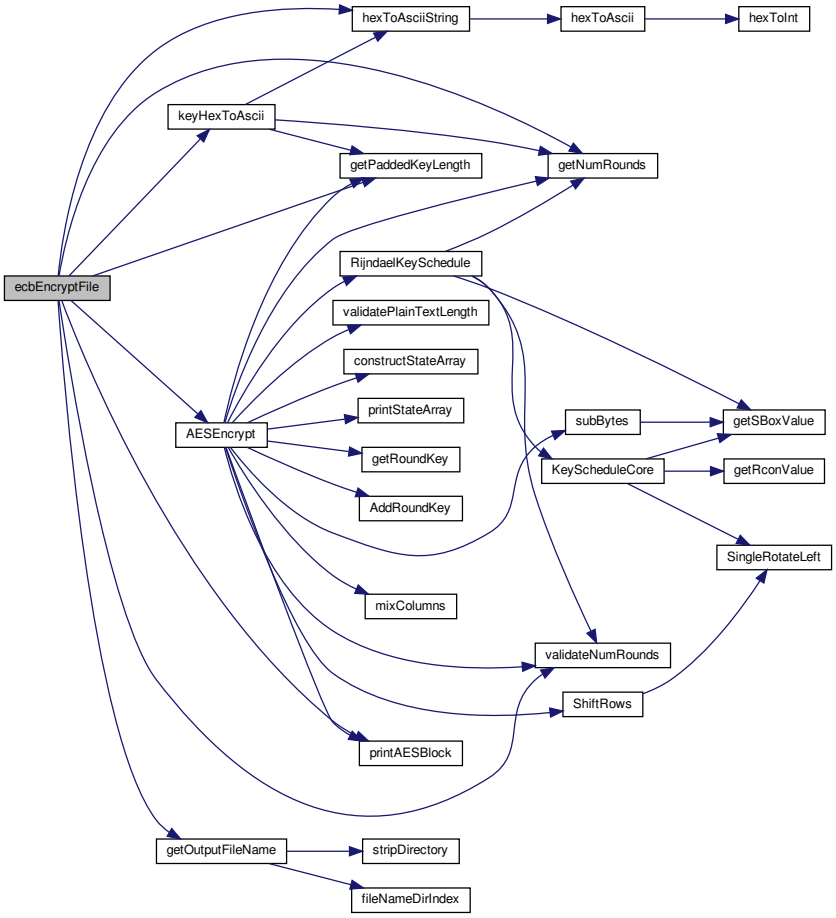
fileName	and write the encrypted version to file with ecbEncrypted appended to the original filename. Performs encryption using the ecb mode and writes the result to a file. If any input is hex, it will convert it to ascii, perform encryption and write it back as ASCII. All terminal output, however, will be hex. Makes use of zero padding.
char	- unsigned char* fileName - the path to the file to be encrypted.
char	- unsigned char* key - the key to use for encryption.
keyLength	- int - the length of the key specified in
key.	
isTextHex	- int - boolean used to signify whether the file pointed to by

Parameters

fileName	is a hexString or ASCII string. (1 = file is a hexString)
isKeyHex	- int - boolean used to signify whether the key pointed to by
key	is a hexString or ASCII string. (1 = file is a hexString)

Definition at line 221 of file ecb.c.

Here is the call graph for this function:

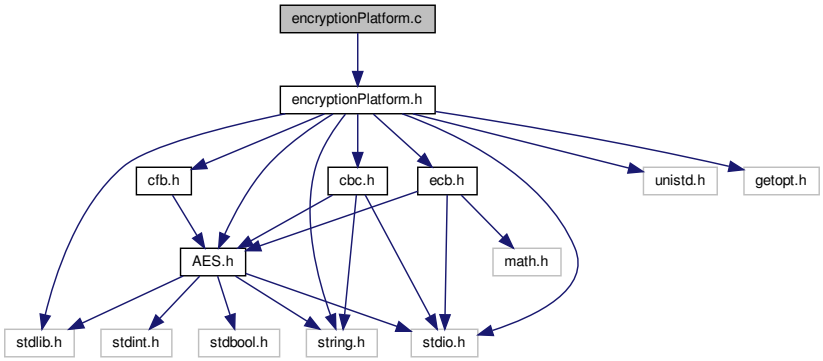


3.12 encryptionPlatform.c File Reference

Implementation file for the cbc and cfb encryption platform. All functions, to specify key, IV, type of encrytion mode, file path or string controlled by commandline parameters.

```
#include "encryptionPlatform.h"
```

Include dependency graph for encryptionPlatform.c:



Functions

- int **main** (int argc, char ∗argv[ ])
- void printHelp ()

  Prints a usage menu to be shown to a user when entering the 'help' command line parameter or when parameters are entered incorrectly.
- void freeMemory (unsigned char ∗key, unsigned char ∗IV, int freeKey, int freeIV)

  Free up memory allocated to stored key and IV.

3.12.1 Detailed Description

Implementation file for the cbc and cfb encryption platform. All functions, to specify key, IV, type of encrytion mode, file path or string controlled by commandline parameters.

Authors

Mohamed Ameen Omar (u16055323)
Douglas Healy (u16018100)
Llewellyn Moyse (u15100708)

Version

0.1

Date

2019-04-03

Copyright

3.12.2 Function Documentation

3.12.2.1 freeMemory()

```
void freeMemory (
            unsigned char * key,
            unsigned char * IV,
            int freeKey,
            int freeIV )
```

Free up memory allocated to stored key and IV.

Parameters

char	Pointer to a location in memory containing the key
char	Pointer to a location in memory containing the IV
freeKey	Boolean value indicating whether or not the IV memory should be freed
freeIV	Boolean value indicating whether or not the Key memory should be freed

Definition at line 262 of file encryptionPlatform.c.

3.12.2.2 printHelp()

```
void printHelp ( )
```

Prints a usage menu to be shown to a user when entering the 'help' command line parameter or when parameters are entered incorrectly.

Returns

Definition at line 232 of file encryptionPlatform.c.

Index