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

1	REA	DME			1
2	File	Index			3
	2.1	File Lis	st		3
3	File	Docum	entation		5
	3.1	AES.c	File Refere	ence	5
		3.1.1	Detailed	Description	8
		3.1.2	Function	Documentation	8
			3.1.2.1	AddRoundKey()	8
			3.1.2.2	AESDecrypt()	9
			3.1.2.3	AESEncrypt()	11
			3.1.2.4	asciiToHexString()	13
			3.1.2.5	constructStateArray()	14
			3.1.2.6	fileNameDirIndex()	15
			3.1.2.7	galloisFieldMult()	16
			3.1.2.8	getInvSBox()	17
			3.1.2.9	getNumRounds()	18
			3.1.2.10	getOutputFileName()	19
			3.1.2.11	getPaddedKeyLength()	20
			3.1.2.12	getRconValue()	22
			3.1.2.13	getRoundKey()	23
			3.1.2.14	getSBoxValue()	24
			31215	hexToAscii()	25

ii CONTENTS

		3.1.2.16	hexToAsciiString()	26
		3.1.2.17	hexToInt()	28
		3.1.2.18	invMixColumns()	29
		3.1.2.19	invShiftRows()	30
		3.1.2.20	invSubBytes()	31
		3.1.2.21	isFileTxt()	32
		3.1.2.22	IVHexToAscii()	33
		3.1.2.23	keyHexToAscii()	34
		3.1.2.24	KeyScheduleCore()	36
		3.1.2.25	mixColumns()	39
		3.1.2.26	printAESBlock()	39
		3.1.2.27	printStateArray()	40
		3.1.2.28	RijndaelKeySchedule()	41
		3.1.2.29	ShiftRows()	43
		3.1.2.30	SingleRotateLeft()	44
		3.1.2.31	SingleRotateRight()	44
		3.1.2.32	stripDirectory()	45
		3.1.2.33	subBytes()	46
		3.1.2.34	validateCipherTextLength()	47
		3.1.2.35	validateNumRounds()	48
		3.1.2.36	validatePlainTextLength()	49
		3.1.2.37	XORBlocks()	50
	3.1.3	Variable	Documentation	51
		3.1.3.1	AES_BLOCK_SIZE	51
		3.1.3.2	invSBox	52
		3.1.3.3	Rcon	52
		3.1.3.4	sbox	53
		3.1.3.5	VERBOSE	53
3.2	AES.h	File Refer	ence	53
	3.2.1	Detailed	Description	56

CONTENTS

3.2.2	Function	Documentation	57
	3.2.2.1	AddRoundKey()	57
	3.2.2.2	AESDecrypt()	58
	3.2.2.3	AESEncrypt()	60
	3.2.2.4	asciiToHexString()	62
	3.2.2.5	constructStateArray()	63
	3.2.2.6	fileNameDirIndex()	64
	3.2.2.7	galloisFieldMult()	65
	3.2.2.8	getInvSBox()	66
	3.2.2.9	getNumRounds()	67
	3.2.2.10	getOutputFileName()	68
	3.2.2.11	getPaddedKeyLength()	69
	3.2.2.12	getRconValue()	71
	3.2.2.13	getRoundKey()	72
	3.2.2.14	getSBoxValue()	73
	3.2.2.15	hexToAscii()	74
	3.2.2.16	hexToAsciiString()	75
	3.2.2.17	hexToInt()	77
	3.2.2.18	invMixColumns()	78
	3.2.2.19	invShiftRows()	79
	3.2.2.20	invSubBytes()	80
	3.2.2.21	isFileTxt()	81
	3.2.2.22	IVHexToAscii()	82
	3.2.2.23	keyHexToAscii()	83
	3.2.2.24	KeyScheduleCore()	85
	3.2.2.25	mixColumns()	88
	3.2.2.26	printAESBlock()	88
	3.2.2.27	printStateArray()	89
	3.2.2.28	RijndaelKeySchedule()	90
	3.2.2.29	ShiftRows()	92

iv CONTENTS

		3.2.2.30	SingleRotateLeft()	 93
		3.2.2.31	SingleRotateRight()	 93
		3.2.2.32	stripDirectory()	 94
		3.2.2.33	subBytes()	 95
		3.2.2.34	validateCipherTextLength()	 96
		3.2.2.35	validateNumRounds()	 97
		3.2.2.36	validatePlainTextLength()	 98
		3.2.2.37	XORBlocks()	 99
3.3	aesTes	ster.c File F	Reference	 100
	3.3.1	Detailed	Description	 101
	3.3.2	Function	Documentation	 102
		3.3.2.1	main()	 102
3.4	cbc.c F	File Refere	ence	 103
	3.4.1	Detailed	Description	 104
	3.4.2	Function	Documentation	 105
		3.4.2.1	cbcDecrypt()	 105
		3.4.2.2	cbcDecryptFile()	 106
		3.4.2.3	cbcEncrypt()	 108
		3.4.2.4	cbcEncryptFile()	 110
3.5	cbc.h F	File Refere	ence	 112
	3.5.1	Detailed	Description	 113
	3.5.2	Function	Documentation	 113
		3.5.2.1	cbcDecrypt()	 114
		3.5.2.2	cbcDecryptFile()	 115
		3.5.2.3	cbcEncrypt()	 117
		3.5.2.4	cbcEncryptFile()	 119
3.6	cbcTes	ster.c File F	Reference	 121
	3.6.1	Detailed	Description	 121
3.7	cfb.c F	ile Referer	nce	 122
	3.7.1	Detailed	Description	 123

CONTENTS

	3.7.2	Function	Documentation	. 123
		3.7.2.1	cfbDecrypt()	. 123
		3.7.2.2	cfbDecryptFile()	. 125
		3.7.2.3	cfbEncrypt()	. 126
		3.7.2.4	cfbEncryptFile()	. 128
3.8	cfb.h F	ile Referer	nce	. 130
	3.8.1	Detailed	Description	. 132
	3.8.2	Function	Documentation	. 132
		3.8.2.1	cfbDecrypt()	. 132
		3.8.2.2	cfbDecryptFile()	. 134
		3.8.2.3	cfbEncrypt()	. 136
		3.8.2.4	cfbEncryptFile()	. 138
	3.8.3	Variable I	Documentation	. 140
		3.8.3.1	AES_BLOCK_SIZE	. 140
		3.8.3.2	VERBOSE	. 141
3.9	cfbTest	er.c File R	Reference	. 141
	3.9.1	Detailed	Description	. 141
3.10	ecb.c F	ile Refere	nce	. 142
	3.10.1	Detailed	Description	. 143
	3.10.2	Function	Documentation	. 143
		3.10.2.1	ecbDecrypt()	. 143
		3.10.2.2	ecbDecryptFile()	. 144
		3.10.2.3	ecbDecryptHelper()	. 146
		3.10.2.4		
		3.10.2.5	ecbEncrypt()	. 149
		3.10.2.6	ecbEncryptFile()	. 150
3.11	ecb.h F	ile Refere	nce	. 151
	3.11.1	Detailed	Description	. 153
	3.11.2	Function	Documentation	. 153
		3.11.2.1	ecbDecrypt()	. 153
		3.11.2.2	ecbDecryptFile()	. 154
		3.11.2.3	ecbDecryptHelper()	. 156
			ecbEcryptHelper()	
			ecbEncrypt()	
		3.11.2.6	ecbEncryptFile()	. 160
3.12	encryp		m.c File Reference	
			Description	
			Documentation	
		3.12.2.1	freeMemory()	. 163
			printHelp()	
			· · · · ·	
Index				165

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:

2 README

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:

AES.c		
	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	5
AES.h	, – –	
	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	53
aesTeste		
	Main file	100
cbc.c		
	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	103
cbc.h		
	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	110
cbcTeste	16 bytes and the key is limited to 32 bytes as per the AES encryption standard	112
ODO TOSIO	Main file	121
cfb.c		
	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	122

File Index

cfb.h	
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	130
cfbTester.c	
Main file	141
ecb.c	
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	142
ecb.h	
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	151
ecbTester.c	??
encryptionPlatform.c	
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	161
encryptionPlatform.h	??
main.c	??
myTester.c	??

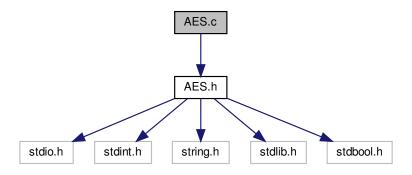
Chapter 3

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()

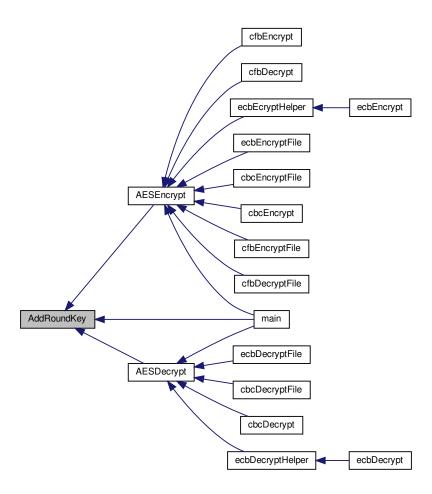
AddRoundKey - Function that performs the Bitwise XOR between state and key as per AES encryption.

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.

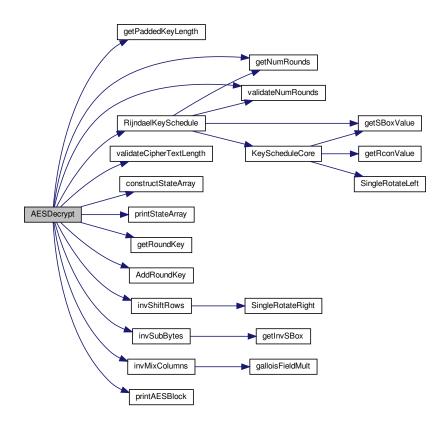
Returns

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

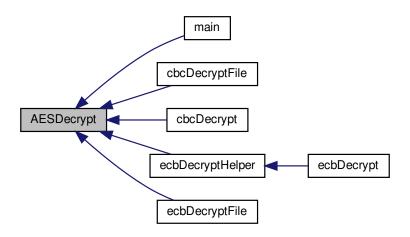
Parameters

cipherText.

Definition at line 399 of file AES.c.



Here is the 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.	

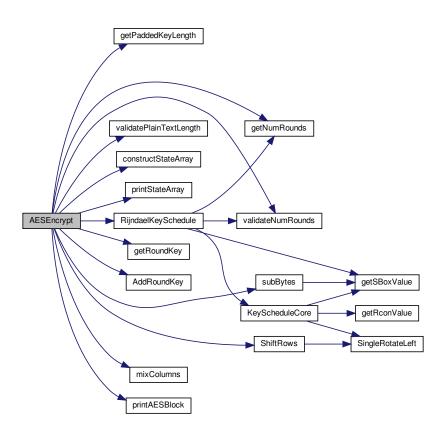
Returns

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

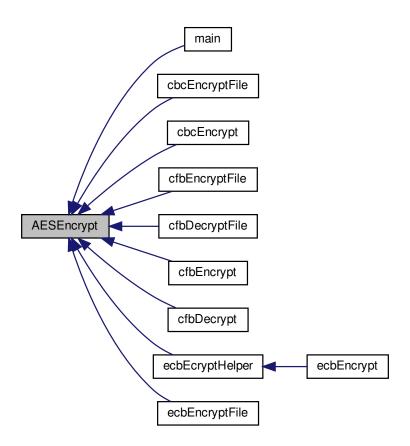
Da			

plainText.

Definition at line 198 of file AES.c.



Here is the caller graph for this function:



3.1.2.4 asciiToHexString()

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

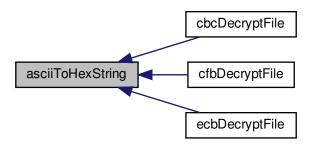
unsigned char* asciiToHexString - pointer to the converted Hex String, pointing to the same memory location as

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] )
```

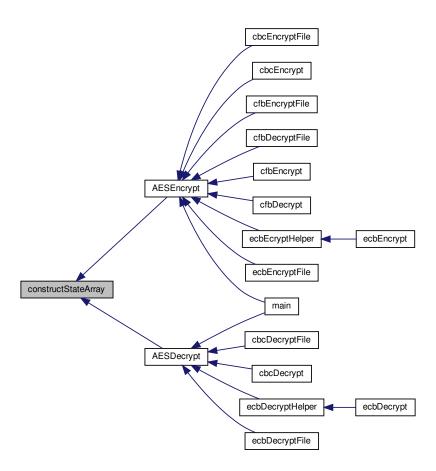
constructStateArray - Function to convert the state array from a flat 1D array to a multidimensional array.

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()

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

Parameters

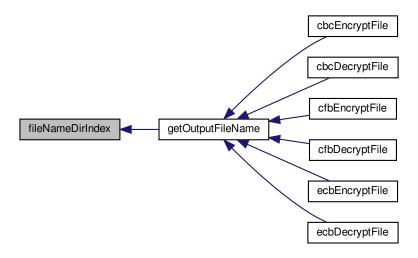
fileName	The path to a file
fileNameLength	The length of the provided file

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()

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

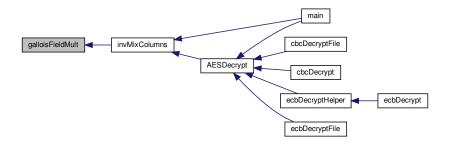
а	- first character to perform Galois field multiplication.
b	- second character to perform Galois field multiplication.

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

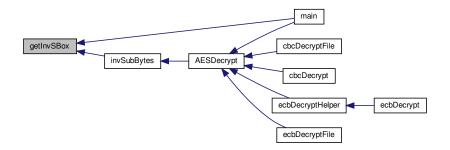
unsigned char - inverse sBox value for the paramter

Parameters

index.

Definition at line 146 of file AES.c.

Here is the caller graph for this function:



3.1.2.9 getNumRounds()

```
\label{eq:conds} \mbox{int getNumRounds (} \\ \mbox{int } \mbox{\it 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

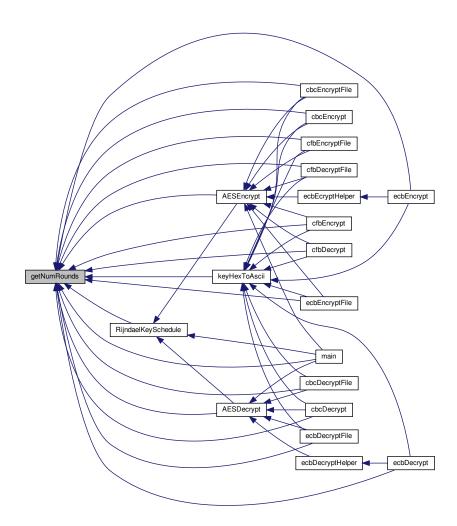
Returns

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.
------------	--

Here is the caller graph for this function:



3.1.2.10 getOutputFileName()

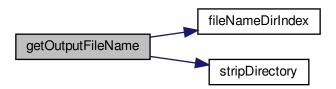
Get the output file name from all the parameters passed in.

Parameters

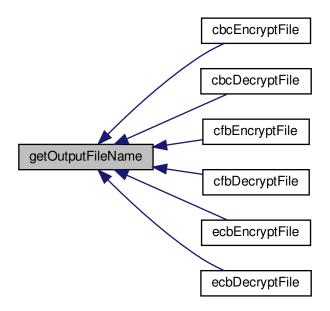
type	0 - Encrypt, 1 - Decrypt
fileName	The name of the input file
outputFileName	The name of the output file
mode	Chipher mode to be used (ECB, CBC, CFB)

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()

```
\label{lem:continuous} \mbox{int getPaddedKeyLength (} \\ \mbox{int } \mbox{currentKeyLength )}
```

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

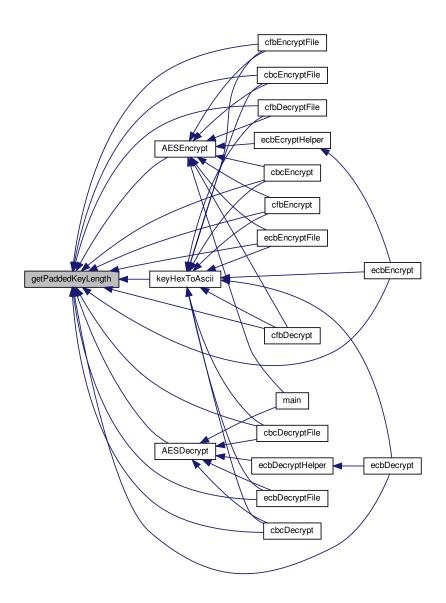
Parameters

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.



3.1.2.12 getRconValue()

```
unsigned char getRconValue ( {\tt unsigned\ char\ \it 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.

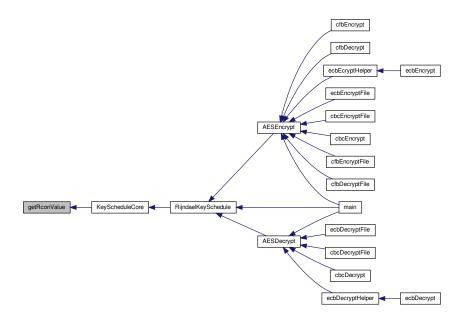
Returns

unsigned char - rCon value for the paramter

Parameters

num.

Definition at line 158 of file AES.c.



3.1.2.13 getRoundKey()

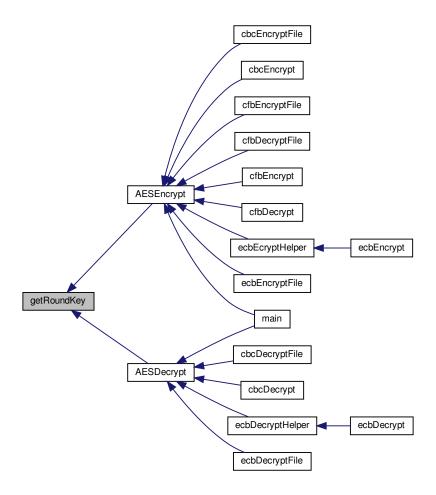
```
void getRoundKey (
          unsigned char * expandedKey,
          unsigned char * roundKey,
          int roundNum )
```

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.



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.

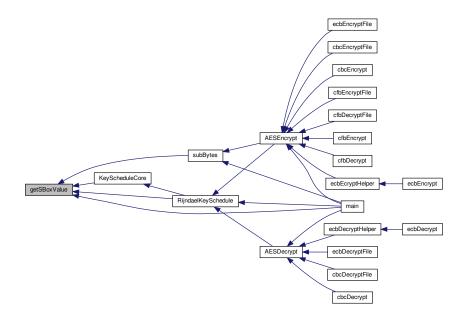
Returns

unsigned char - sBox value for the paramter

Parameters



Definition at line 134 of file AES.c.



3.1.2.15 hexToAscii()

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

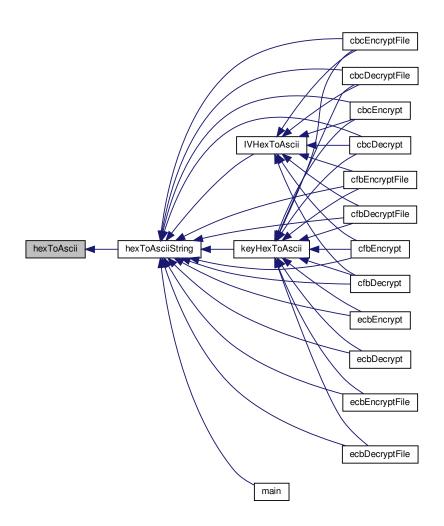
Parameters

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 caller graph for this function:



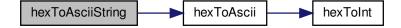
3.1.2.16 hexToAsciiString()

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"

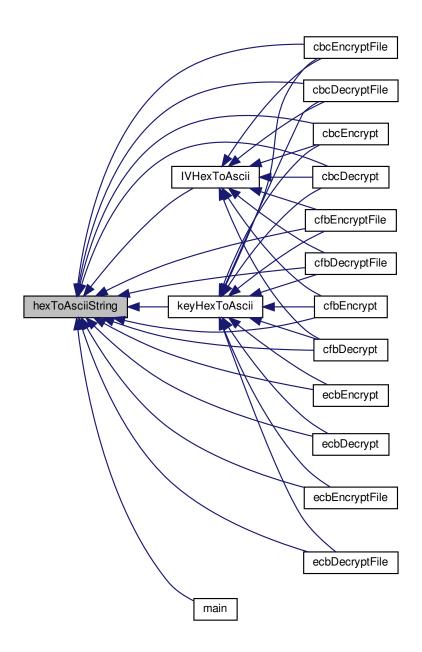
Parameters

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 caller graph for this function:



3.1.2.17 hexToInt()

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

Parameters

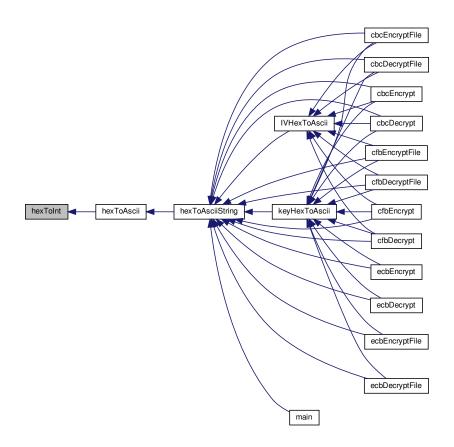
ch - hex value that wil be converted to int.

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

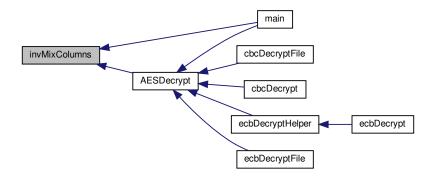
state.	
state	- unsigned char - is the current state of the ciphertext or plaintext during AES encryption or decryption.

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

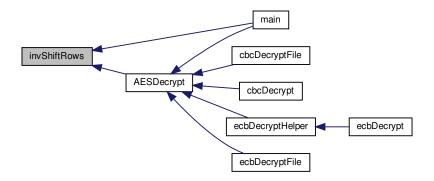
state - unsigned char - is the current state of the ciphertext or plaintext during AES encryption or decryption

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

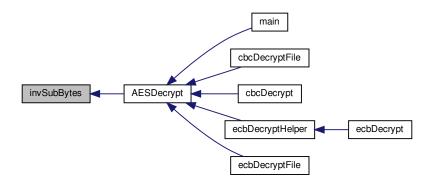
state - unsigned char - is the current state of the ciphertext or plaintext during AES encryption or decryption

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()

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

Parameters

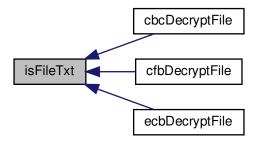
filename	is a text file with extension .txt or not. Returns a 1 if it is and a 0 if it isn't.
fileName	- unsigned char* fileName - path to file to determine if the file is a text file or not.

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

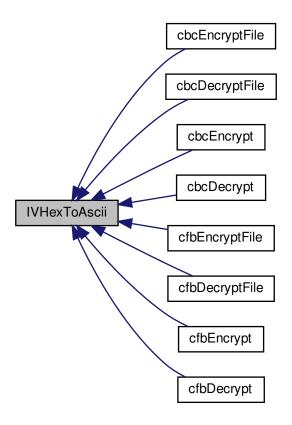


Definition at line 1205 of file AES.c.

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.	

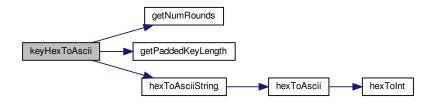
Returns

 $unsigned *- the \ ASCII \ representation \ of the \ hex \ key \ passed \ in \ as \ parameter$

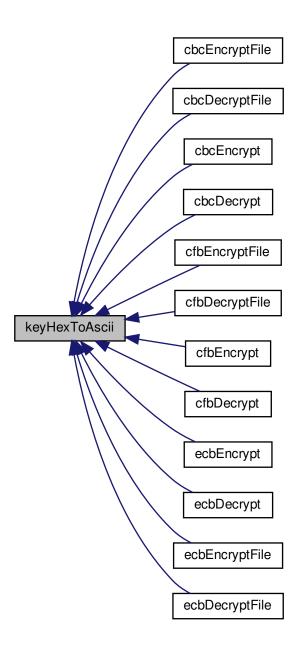
Parameters

Definition at line 1178 of file AES.c.

Here is the call graph for this function:

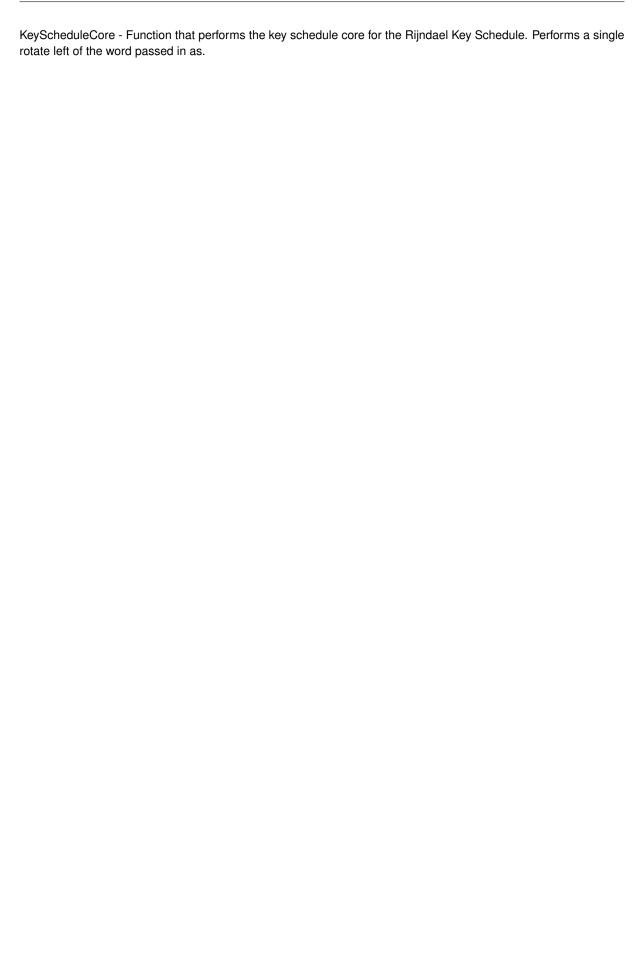


Here is the caller graph for this function:



3.1.2.24 KeyScheduleCore()

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

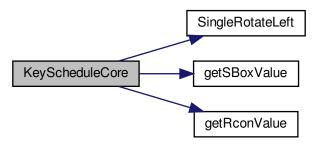


Parameters

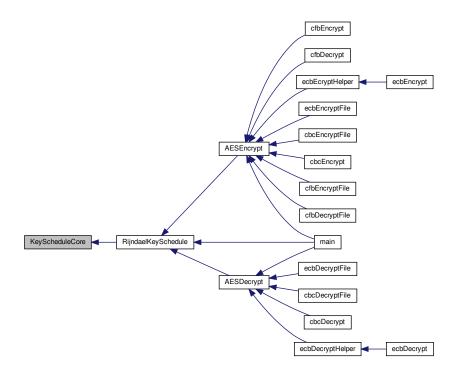
word	and applies the required s-box substituion and rcon XOR.
char	- unsigned char* word - pointer to the word onto which the key schedule core should be operated.
wordLength	- length of the word passed in as a parameter
word.	
rConIterationVal	- the iteration value to be used for the rcon XOR.

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()

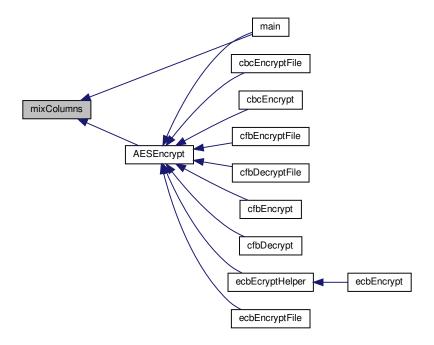
mixColumns - Function that performs the MixColumns step of AES as specified by AES encryption.

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 ( \label{eq:unsigned} \mbox{unsigned char} \ * \ block \ )
```

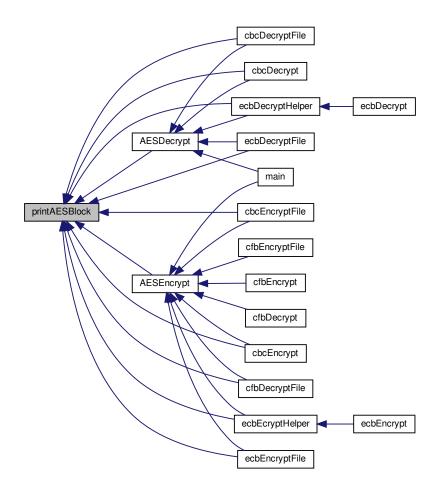
printAESBlock - Function to print a single block in hex format to the terminal.

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()

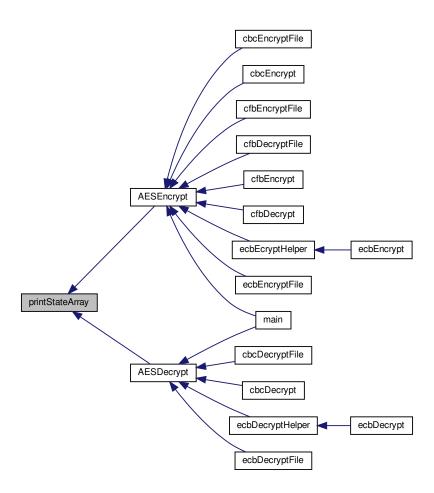
printStateArray - Function to print the state array to the terminal in hex format.

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		

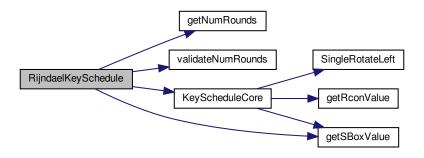
Generated by Doxygen

Returns

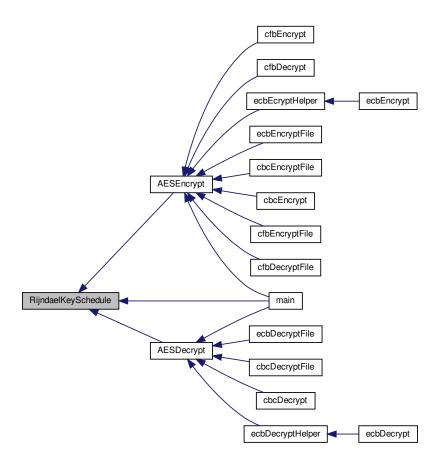
expandedKey - The key that has been expanded.

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()

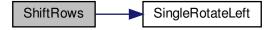
ShiftRows - Function to shift the state array according to the AES encryption standard for 128 - bits blocks.

Parameters

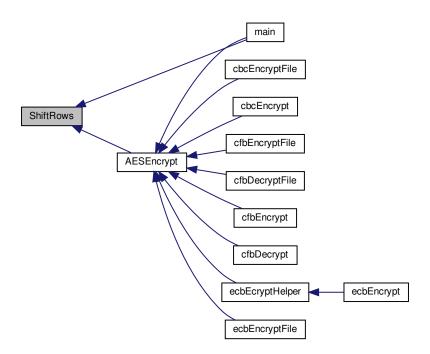
state - unsigned char - is the current state of the ciphertext or plaintext during AES encryption or decryption

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 )
```

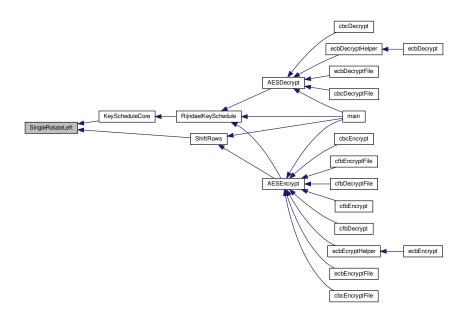
SingleRotateLeft - Function to rotate the array passed in as a paramter.

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 )
```

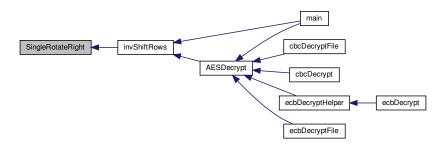
 $Single Rotate Right - Function \ to \ rotate \ the \ array \ passed \ in \ as \ a \ paramter.$

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()

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

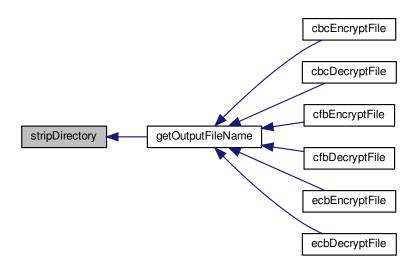
stripDirectory - Function that removes path from the provided path to a file and returns only the file name

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 ( \label{eq:unsigned char state[4][4] } unsigned char \ state[4][4] \ )
```

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

Parameters

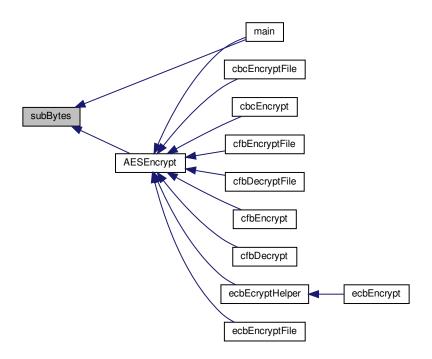
state - unsigned char - is the current state of the ciphertext or plaintext during AES encryption or decryption.

Definition at line 789 of file AES.c.

Here is the call graph for this function:



Here is the caller graph for this function:

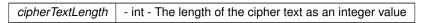


3.1.2.34 validateCipherTextLength()

```
void validateCipherTextLength ( int \ \textit{cipherTextLength} \ )
```

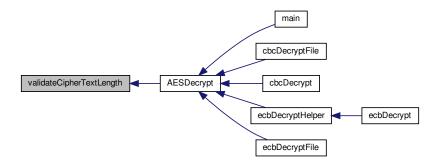
validateCipherTextLength - Function that validates the length of the ciphertext. The validation is done against the AES_BLOCK_SIZE value

Parameters



Definition at line 1015 of file AES.c.

Here is the caller graph for this function:



3.1.2.35 validateNumRounds()

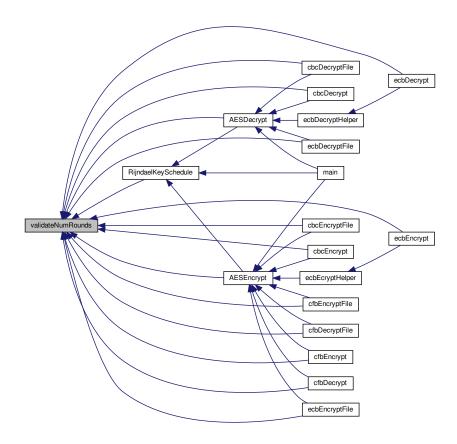
```
void validateNumRounds ( int \ \textit{numRounds}, int \ \textit{keyLength} \ )
```

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

Here is the caller graph for this function:



3.1.2.36 validatePlainTextLength()

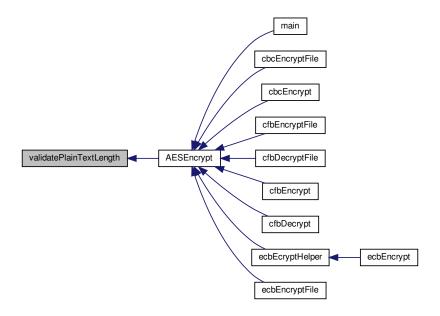
 $validate Plain Text Length \ - \ Function \ that \ validates \ the \ length \ of \ the \ plain text. \ The \ validation \ is \ done \ against \ the \ AES_BLOCK_SIZE \ value$

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

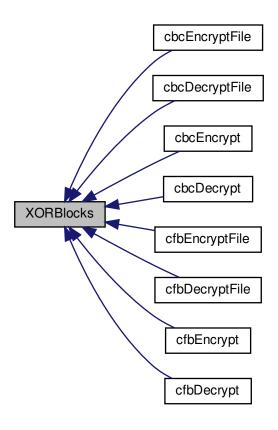
length	and retuns the XOR'd result. User must free the memory returned.	
char	- block1 - First block to be XOR'd.	
char	- block2 - Second block to be XOR'd.	
length	- length of the blocks to be XOR'd.	

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,0xf7,0xe4,0x58,0x50,0xb8,0xb3,0x45,0x06,0xd0,0x2c,0x1e,0x8f,0xca,0x3f,0x0f,0x02,0xc1,0xaf,0xbd,0x03,0x01,0x13,0x8a,0x6b,0x3a,0x91,0x11,0x41,0x41,0x4f,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,0xc6,0x47,0xf1,0x1d,0x20,0x6,0x31,0x11,0x11,0x11,0x14,0x29,0xc5,0x89,0x6f,0xb7,0x62,0x0e,0xaa,0x18,0xbe,0x1b,0xfc,0x56,0x3e,0x4b,0xc6,0x42,0x79,0x20,0x9a,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,0x2c,0x6b,0x3c,0x83,0x53,0x99,0x9c,0xef,0xa0,0xe0,0x3b,0x4d,0xae,0x2a,0xf5,0x0d,0xc8,0xeb,0xbb,0x3c,0x83,0x53,0x99,0x61,0x17,0x2b,0x04,0x7e,0xba,0x77,0xd6,0x26,0xe1,0x6b,0xbb,0x3c,0x83,0x53,0x99,0x61,0x17,0x2b,0x04,0x7e,0xba,0x77,0xd6,0x26,0xe1,0x6b,0x14,0x14,0x63,0x55,0x21,0x0c,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,
                                                                   0xd4,
0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97,
                                                      0x35, 0x6a,
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,
            0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3,
                                                 0xbd,
                                                       0x61,
                                                             0xc2,
                                                                   0x9f,
0x4d, 0x9a,
            0x2f, 0x5e, 0xbc, 0x63, 0xc6,
                                          0x97,
                                                 0x35,
                                                       0x6a,
                                                             0xd4,
                                                                   0xb3,
                              0x83,
                                                 0x74,
                                                             0xcb,
                                                                   0x8d,
0x4a, 0x94,
            0x33, 0x66, 0xcc,
                                    0x1d,
                                          0x3a,
                                                       0xe8,
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,
            0x91, 0x39, 0x72,
0xef, 0xc5,
                              0xe4, 0xd3, 0xbd,
                                                 0x61,
                                                       0xc2,
                                                             0x9f,
                                                                   0x25,
0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8,
                                                             0x8d,
0x04, 0x08,
            0x10, 0x20, 0x40,
                              0x80, 0x1b, 0x36,
                                                 0x6c,
                                                       0xd8,
                                                             0xab,
                                                                   0x4d,
0x2f, 0x5e,
            0xbc, 0x63, 0xc6,
                              0x97,
                                    0x35,
                                          0x6a,
                                                 0xd4,
                                                       0xb3,
                                                             0x7d,
                                                                   0xfa,
                                                                   0x4a,
0xc5, 0x91, 0x39, 0x72, 0xe4,
                              0xd3, 0xbd, 0x61,
                                                0xc2.
                                                       Ox9f.
                                                             0x25.
            0xcc, 0x83, 0x1d,
                              0x3a, 0x74,
                                                0xcb,
                                                       0x8d,
                                                             0x01,
0x33, 0x66,
                                          0xe8,
                                                                   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,
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,
0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31,
                                                                              0x27,
0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a,
                                                0x07,
                                                            0x80,
                                                      0x12,
                                                                  0xb3,
                                                                                    0x2f,
0x09, 0x83,
           0x2c, 0x1a, 0x1b, 0x6e, 0x5a,
                                         0xa0,
                                                0x52, 0x3b,
                                                           0xd6,
                                                                        0x29,
                                                                              0xe3,
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,
0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19,
0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b,
                                         0x5c,
                                                                  0x62,
                                                      0xd3, 0xac,
0xe0, 0x32,
           0x3a, 0x0a, 0x49,
                             0x06, 0x24,
                                                0xc2,
                                                                              0x95,
                                                                                    0xae,
0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9,
                                                0x6c, 0x56, 0xf4, 0xea,
                                                                        0x65,
                                                                              0x7a,
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.

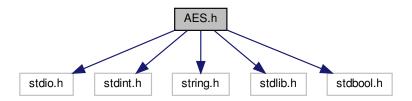
3.2 AES.h File Reference

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.

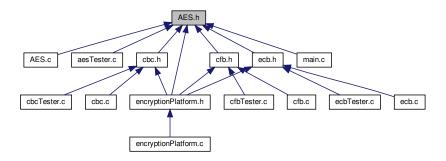
```
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include <stdlib.h>
```

#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"

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

Copyright (c) 2019

3.2.2 Function Documentation

3.2.2.1 AddRoundKey()

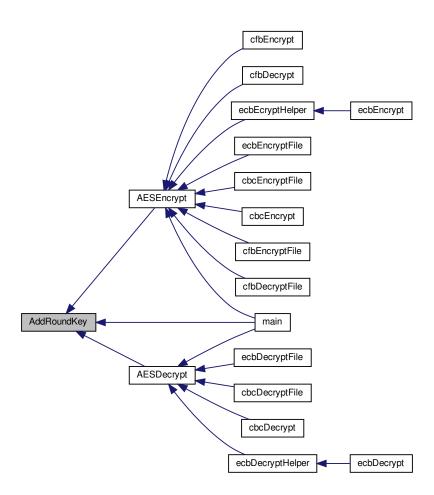
AddRoundKey - Function that performs the Bitwise XOR between state and key as per AES encryption.

Parameters

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

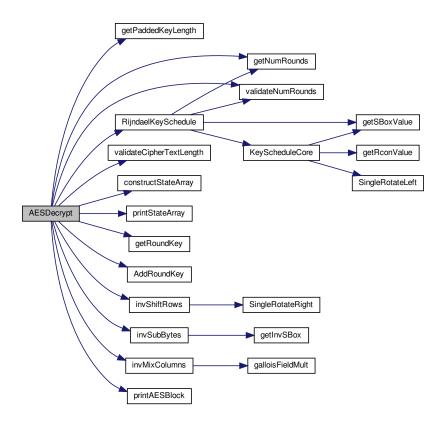
unsigned* char - Plaintext resulting from the decryption of the ciphertext passed in as

Parameters

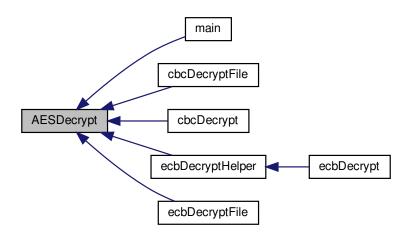
cipherText.

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

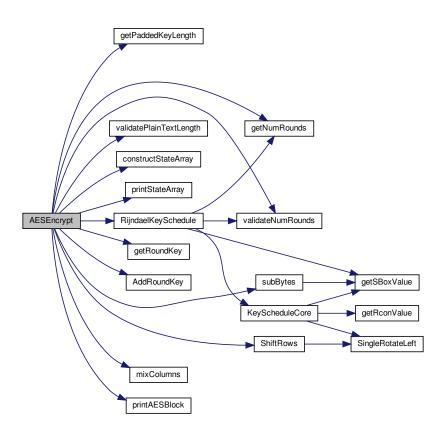
unsigned* char - Ciphertext resulting from the encryption of the plaintext passed in as

D _o			- 4		
Pа	ra	m	eı	e	rs

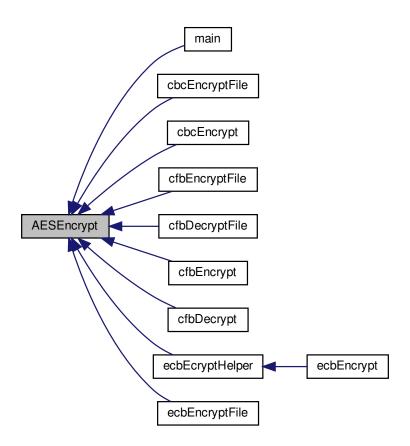
plainText.

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()

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 store	
asciiStringLen - size t containing the length of the ASCII String to be converted.		

Returns

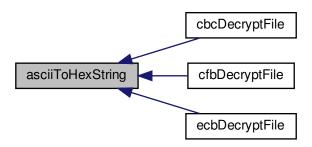
unsigned char* asciiToHexString - pointer to the converted Hex String, pointing to the same memory location as

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] )
```

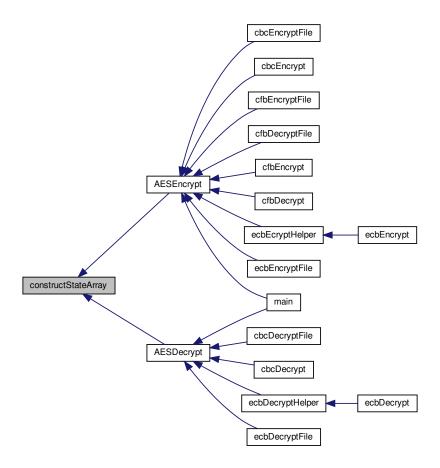
constructStateArray - Function to convert the state array from a flat 1D array to a multidimensional array.

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()

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

Parameters

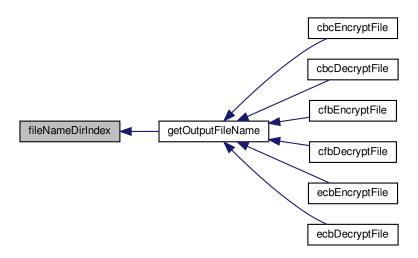
fileName	The path to a file
fileNameLength	The length of the provided file

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()

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

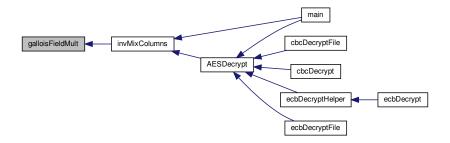
а	- first character to perform Galois field multiplication.
b	- second character to perform Galois field multiplication.

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.

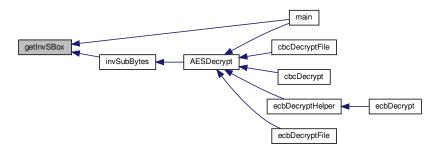
Returns

unsigned char - inverse sBox value for the paramter

Parameters

index.

Here is the caller graph for this function:



3.2.2.9 getNumRounds()

```
\label{eq:conds} \mbox{int getNumRounds (} \\ \mbox{int } keyLength \mbox{ )}
```

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

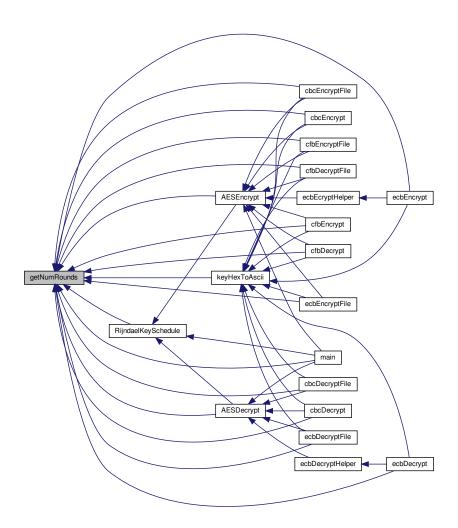
int - the number of rounds based off of the length of the key passed in the parameter

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()

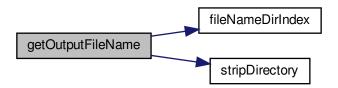
Get the output file name from all the parameters passed in.

Parameters

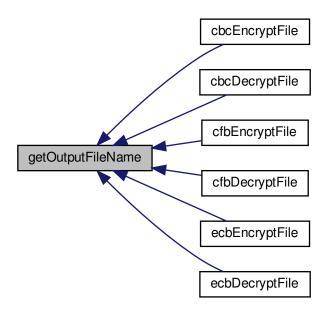
type	0 - Encrypt, 1 - Decrypt
fileName	The name of the input file
outputFileName	The name of the output file
mode	Chipher mode to be used (ECB, CBC, CFB)

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()

```
\label{eq:continuous} \mbox{int getPaddedKeyLength (} \\ \mbox{int } \mbox{currentKeyLength )}
```

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

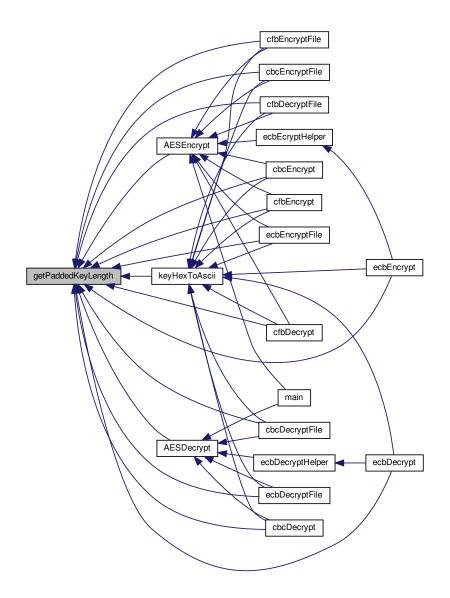
Parameters

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.



3.2.2.12 getRconValue()

```
unsigned char getRconValue ( {\tt unsigned\ char\ \it 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.

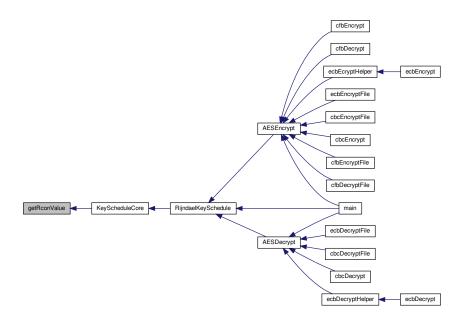
Returns

unsigned char - rCon value for the paramter

Parameters

num.

Definition at line 158 of file AES.c.



3.2.2.13 getRoundKey()

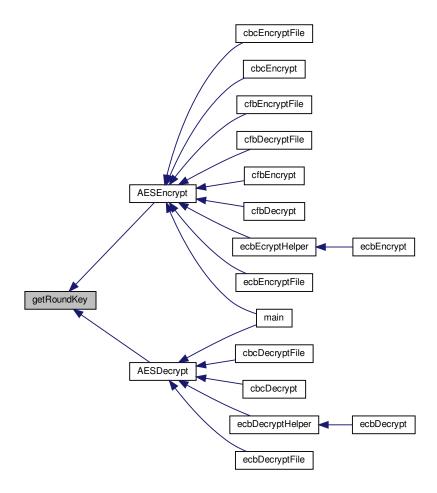
```
void getRoundKey (
          unsigned char * expandedKey,
          unsigned char * roundKey,
          int roundNum )
```

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.



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.

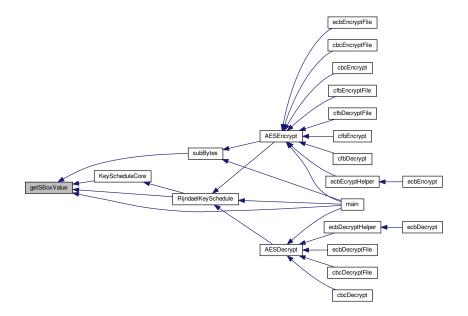
Returns

unsigned char - sBox value for the paramter

Parameters

index.

Definition at line 134 of file AES.c.



3.2.2.15 hexToAscii()

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

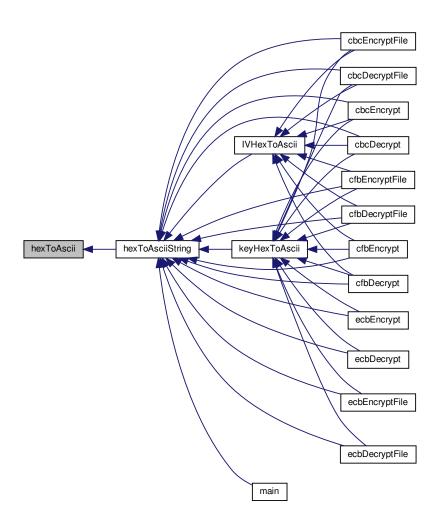
Parameters

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 caller graph for this function:



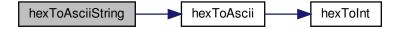
3.2.2.16 hexToAsciiString()

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"

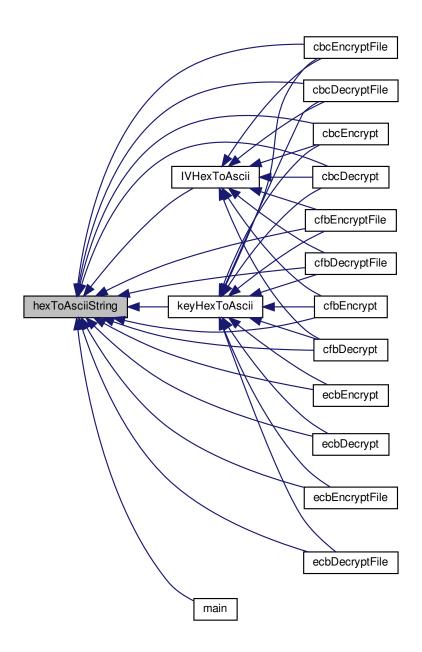
Parameters

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 caller graph for this function:



3.2.2.17 hexToInt()

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

Parameters

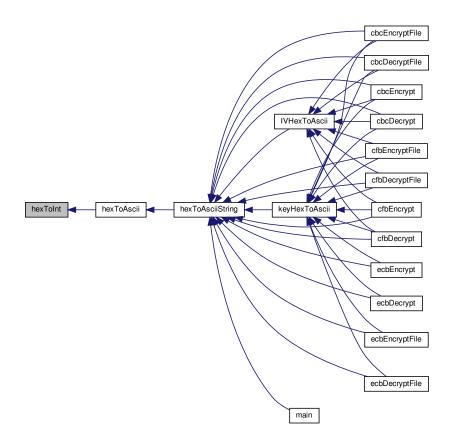
ch - hex value that wil be converted to int.

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

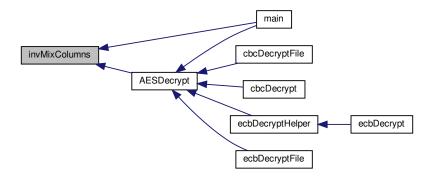
state.	
state	- unsigned char - is the current state of the ciphertext or plaintext during AES encryption or decryption.

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

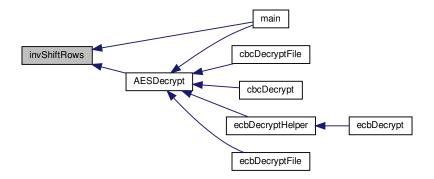
state - unsigned char - is the current state of the ciphertext or plaintext during AES encryption or decryption

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 ( \label{eq:unsigned} unsigned \ char \ state[4][4] \ )
```

invSubBytes - Function that performs the inverse of Function subBytes

Parameters

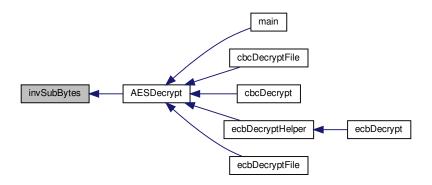
state - unsigned char - is the current state of the ciphertext or plaintext during AES encryption or decryption

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()

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

Parameters

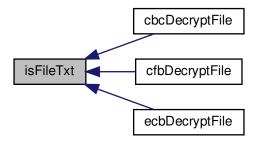
filename	is a text file with extension .txt or not. Returns a 1 if it is and a 0 if it isn't.
fileName	- unsigned char* fileName - path to file to determine if the file is a text file or not.

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

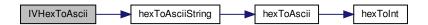
unsigned* - the ASCII representation of the hex IV passed in as parameter

Parameters

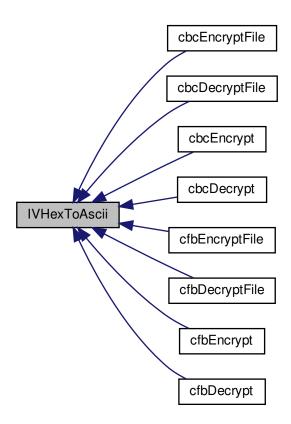


Definition at line 1205 of file AES.c.

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.	

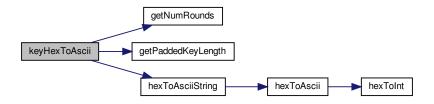
Returns

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

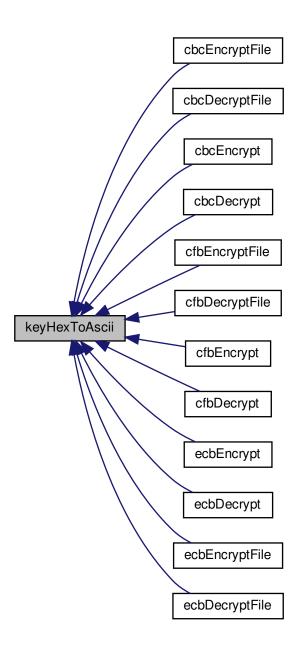
Parameters

hexKey.

Definition at line 1178 of file AES.c.



Here is the caller graph for this function:



3.2.2.24 KeyScheduleCore()

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

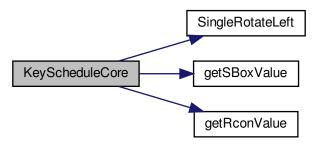
KeyScheduleCore - Function that performs the key schedule core for the Rijndael Key Schedule. Performs a single rotate left of the word passed in as.

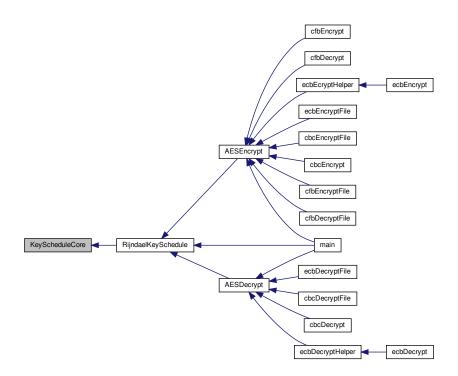
Parameters

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:





3.2.2.25 mixColumns()

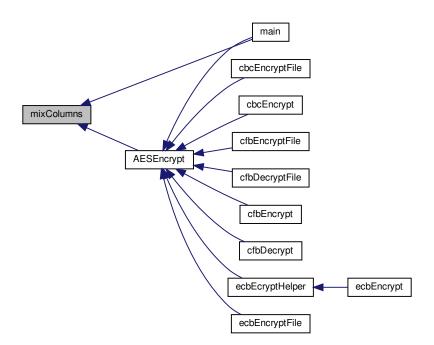
mixColumns - Function that performs the MixColumns step of AES as specified by AES encryption.

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 ( \label{eq:unsigned} \mbox{unsigned char} \ * \ block \ )
```

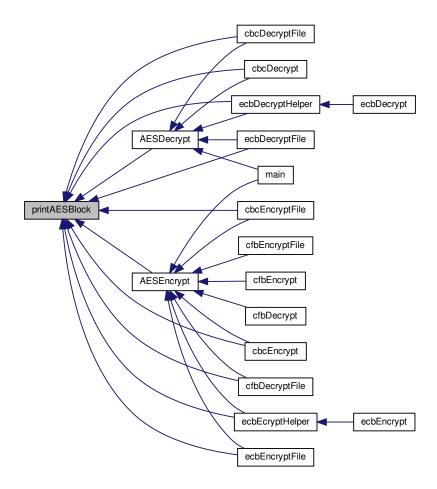
printAESBlock - Function to print a single block in hex format to the terminal.

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()

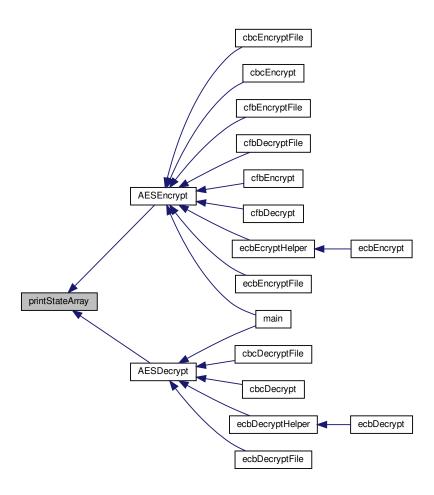
printStateArray - Function to print the state array to the terminal in hex format.

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

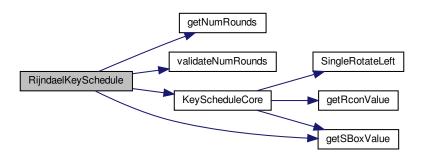
originalKey	and the length of the original key given as parameter. The caller must free the memory allocated and returned.
originalKey	- unsigned char * - An unsigned char pointer to the original key.
keyLength	- int - length of originalKey passed in as a parameter
originalKey	

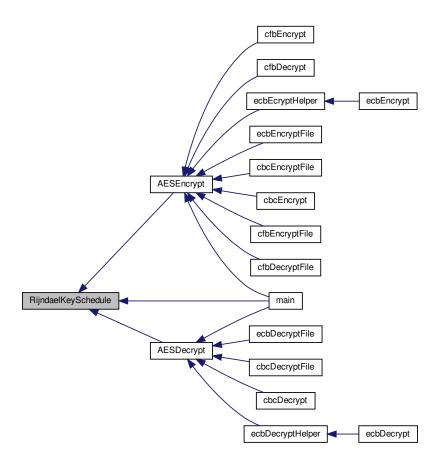
Returns

expandedKey - The key that has been expanded.

Definition at line 577 of file AES.c.

Here is the call graph for this function:





3.2.2.29 ShiftRows()

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

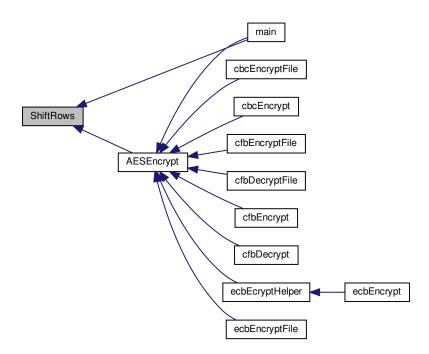
Parameters

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:





3.2.2.30 SingleRotateLeft()

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

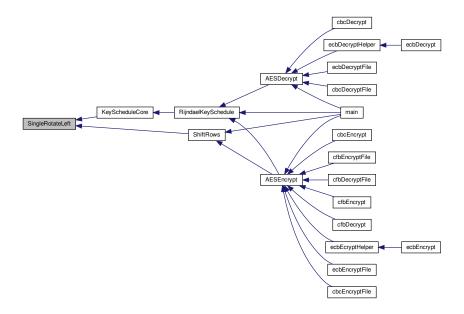
SingleRotateLeft - Function to rotate the array passed in as a paramter.

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 )
```

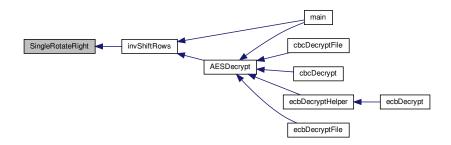
SingleRotateRight - Function to rotate the array passed in as a paramter.

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()

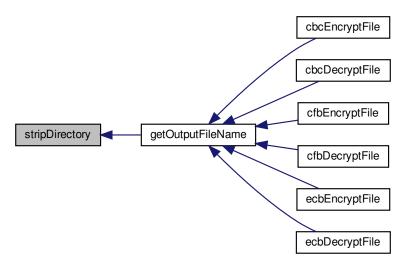
stripDirectory - Function that removes path from the provided path to a file and returns only the file name

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	Generated by Doxygen
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 ( \label{eq:charge_energy} \text{unsigned char } state[4][4] \ )
```

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

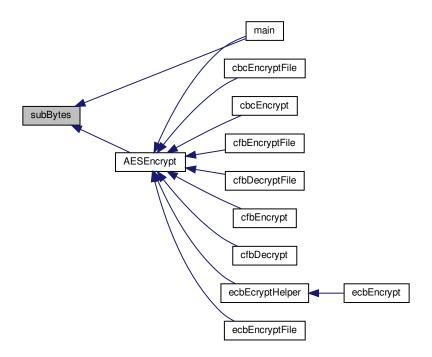
Parameters

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 caller graph for this function:



3.2.2.34 validateCipherTextLength()

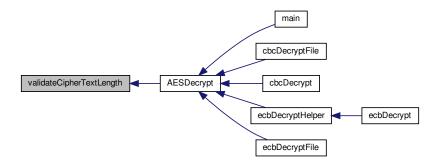
```
\begin{tabular}{ll} \begin{tabular}{ll} void & validate Cipher Text Length & (\\ & int & cipher Text Length & ) \end{tabular}
```

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

Parameters

Definition at line 1015 of file AES.c.

Here is the caller graph for this function:



3.2.2.35 validateNumRounds()

```
void validateNumRounds ( int \ \textit{numRounds}, int \ \textit{keyLength} \ )
```

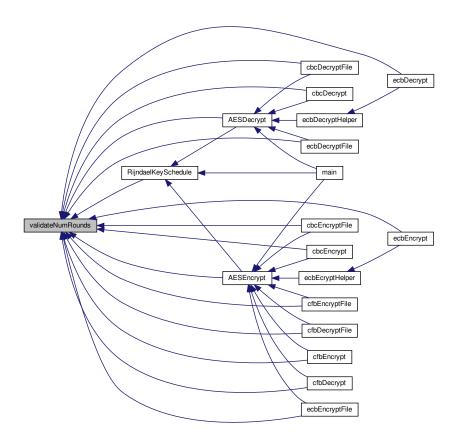
validateNumRounds - Function that validates the number of rounds that have been passed in by the

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()

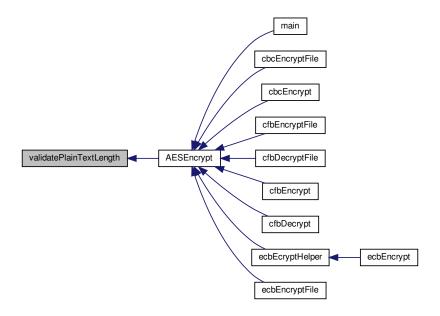
 $validate Plain Text Length \ - \ Function \ that \ validates \ the \ length \ of \ the \ plain text. \ The \ validation \ is \ done \ against \ the \ AES_BLOCK_SIZE \ value$

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

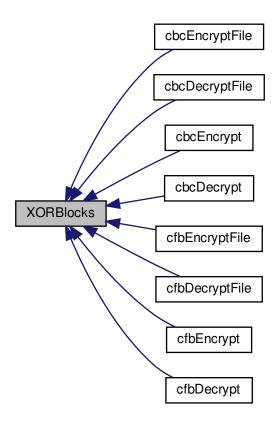
length	and retuns the XOR'd result. User must free the memory returned.
char	- block1 - First block to be XOR'd.
char	- block2 - Second block to be XOR'd.
length	- length of the blocks to be XOR'd.

Returns

unsigned* - Result of the XOR.

Definition at line 1228 of file AES.c.

Here is the caller graph for this function:

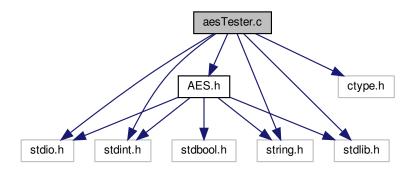


3.3 aesTester.c File Reference

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

Copyright (c) 2019

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};

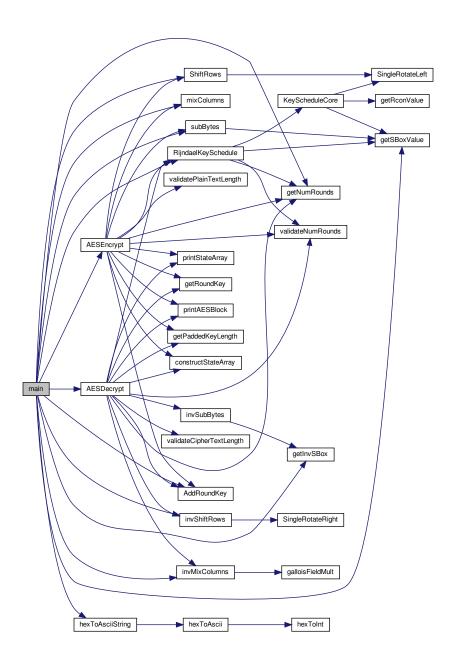
Result Should BE: {0xe9,0xcb,0x3d,0xaf, 0x09,0x31,0x32,0x2e, 0x89,0x07,0x7d,0x2c, 0x72,0x5f,0x94,0xb5};

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.

3.4 cbc.c File Reference 103

Here is the call graph for this function:

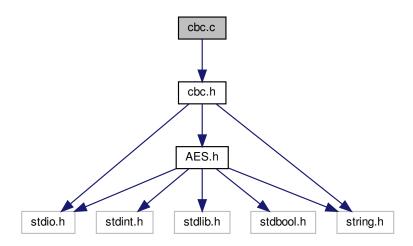


3.4 cbc.c File Reference

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) 3.4 cbc.c File Reference

Version

0.1

Date

2019-03-28

Copyright

Copyright (c) 2019

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 )
```

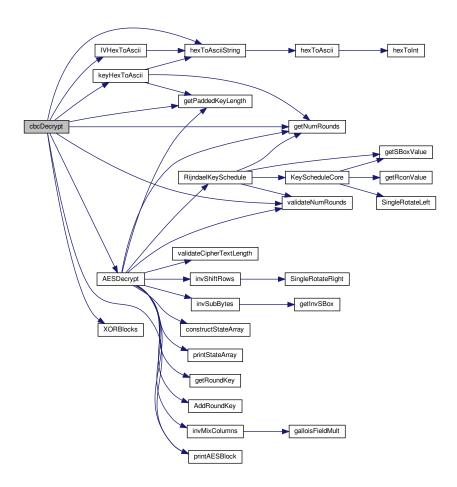
cbcDecrypt - Function to decrypt the user input pointed to by

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)
islvHex	- int - boolean used to signify whether the IV pointed to by
ennitedization vector	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

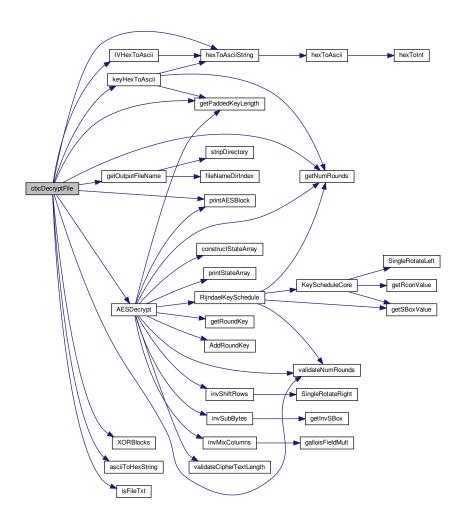
3.4 cbc.c File Reference

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)
islvHex	- 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

3.4 cbc.c File Reference

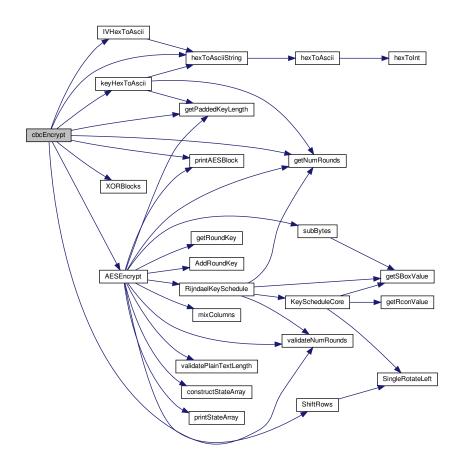
Parameters

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)
islvHex	- 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.

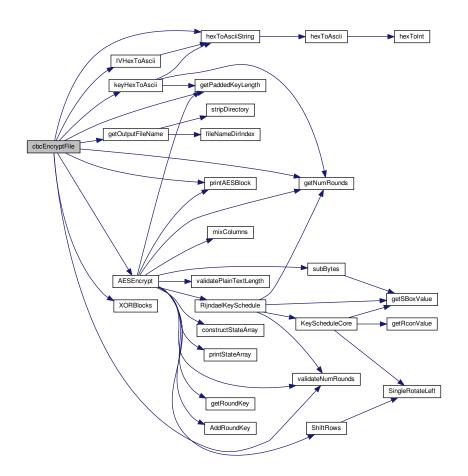
3.4 cbc.c File Reference

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)
islvHex	- 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.

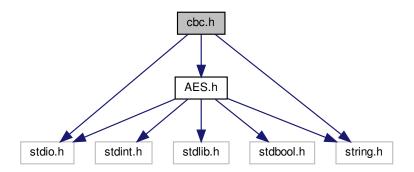
Here is the call graph for this function:



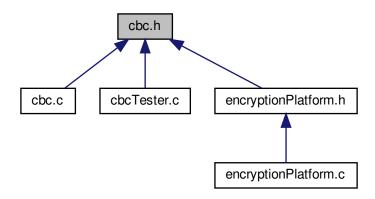
3.5 cbc.h File Reference

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)

3.5 cbc.h File Reference 113

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

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

Copyright (c) 2019

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 )
```

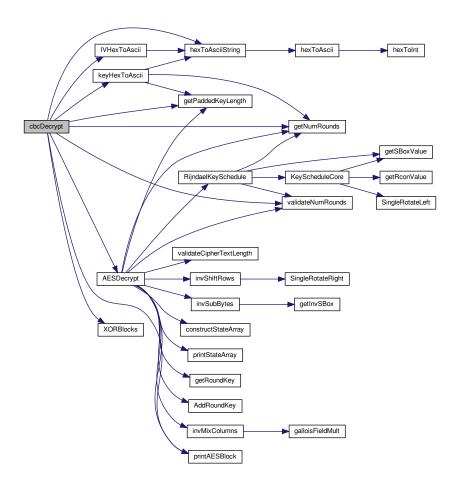
cbcDecrypt - Function to decrypt the user input pointed to by

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)
islvHex	- int - boolean used to signify whether the IV pointed to by
initializationVector	is a hexString or ASCII string. (1 = file is a hexString)

3.5 cbc.h File Reference

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 )
```

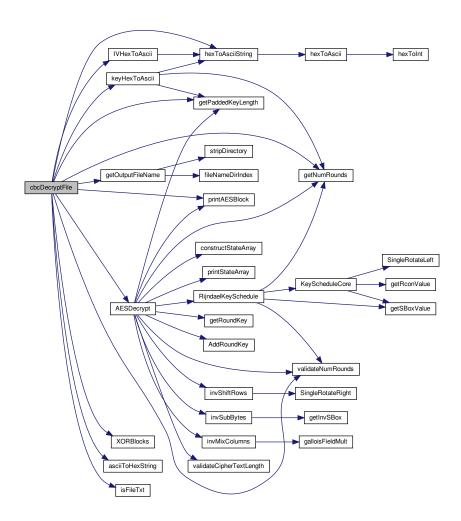
cbcDecryptFile - Function to decrypt the file with name

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)
islvHex	- int - boolean used to signify whether the IV pointed to by
initializationVector	is a hexString or ASCII string. (1 = file is a hexString)

3.5 cbc.h File Reference

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

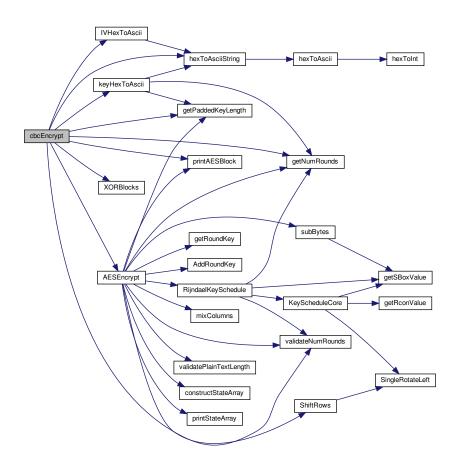
plainText	and print encrypted result in hex to terminal. Performs encryption using the cbc mode prints the result to the terminal for each block in hex. If any input is hex, it will convert it to ascii, perform encryption and print it in hex. Makes use of zero padding.
char	- unsigned char* plainText - 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.

3.5 cbc.h File Reference

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

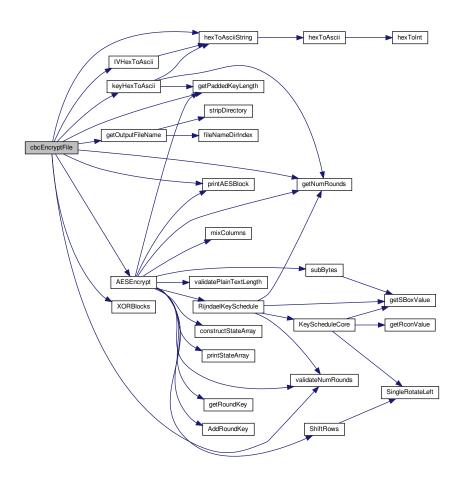
fileName	and write the encrypted version to file with cbcEncrypted appended to the original
	filename. Performs encryption using the cbc mode and writes the result to a file. If
	any input is hex, it will convert it to ascii, perform encryption and write it back as
	ASCII. All terminal output, however,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

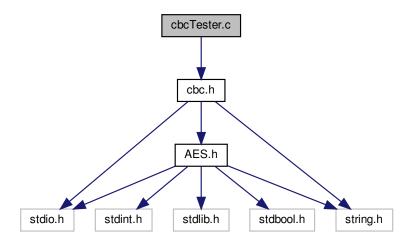


3.6 cbcTester.c File Reference

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

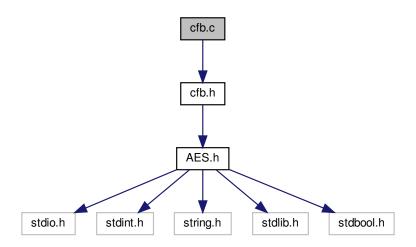
Copyright (c) 2019

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 cfb.c File Reference 123

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

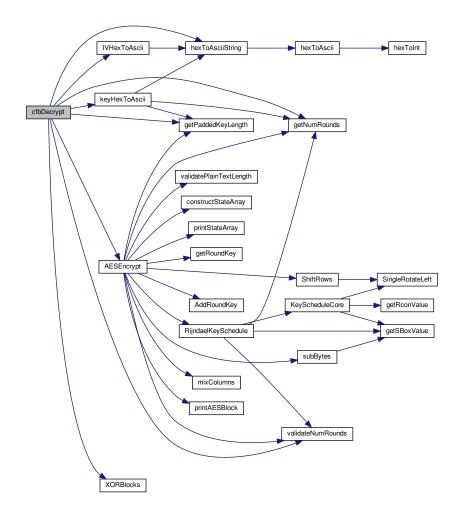
cipherText	and print decrypted result in hex to terminal. Performs decryption using the cfb mode prints the result to the terminal for each block in hex. If any input is hex, it will convert it to ascii, perform decryption and print it in hex. Makes use of zero padding.
char	- unsigned char* cipherText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

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 )
```

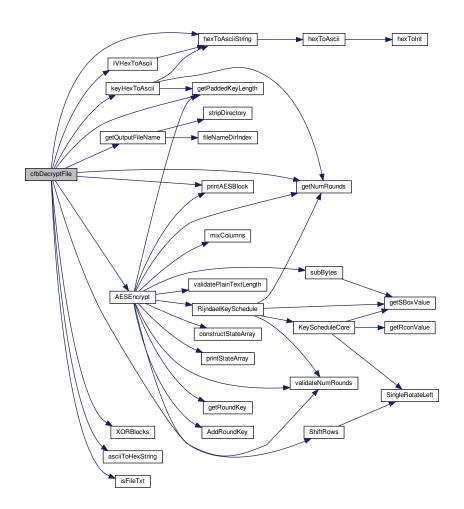
cfbDecryptFile - Function to decrypt the file with name

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

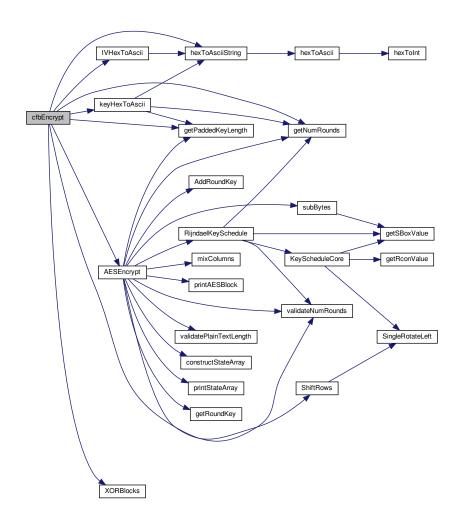
3.7 cfb.c File Reference

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)
islvHex	- 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

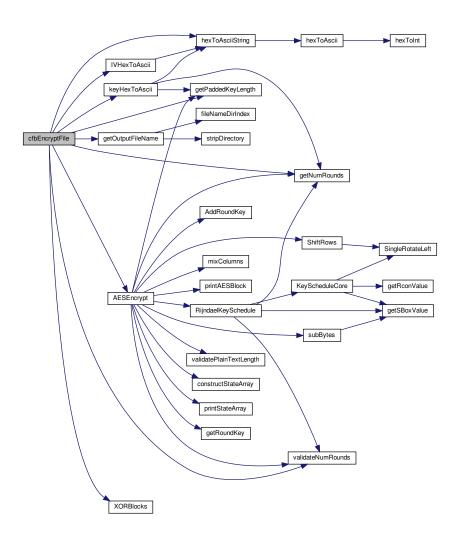
3.7 cfb.c File Reference

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)
islvHex	- 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:



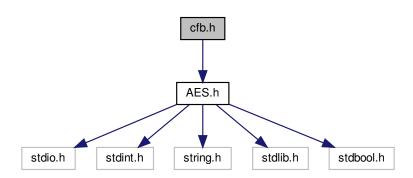
3.8 cfb.h File Reference

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.

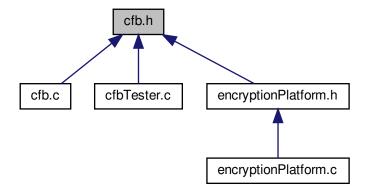
3.8 cfb.h File Reference 131

#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

Copyright (c) 2019

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

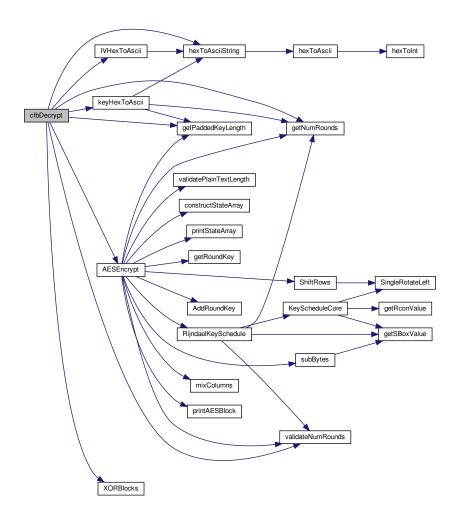
3.8 cfb.h File Reference

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)
islvHex	- 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

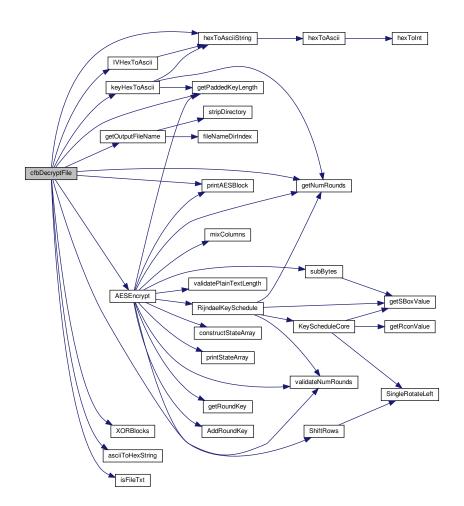
3.8 cfb.h File Reference

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)
islvHex	- 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

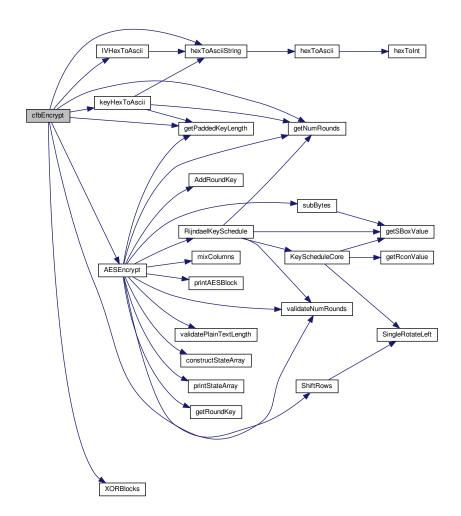
3.8 cfb.h File Reference

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)
islvHex	- 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

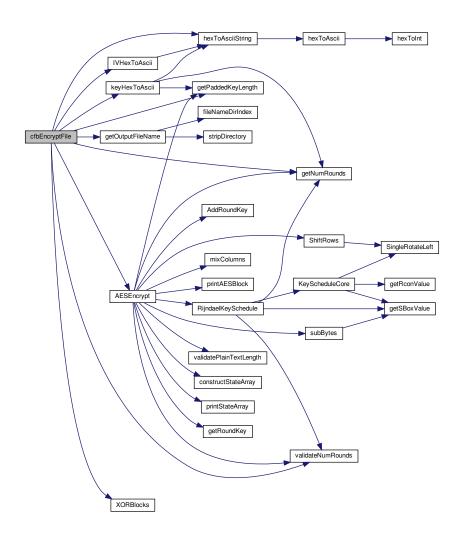
3.8 cfb.h File Reference

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)
islvHex	- 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.

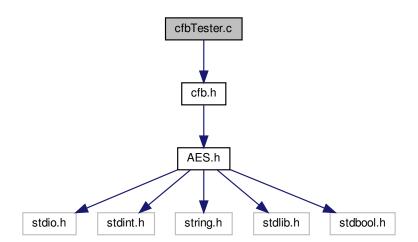
Variable - VERBOSE - specifies if verbose output should be printed or not.

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

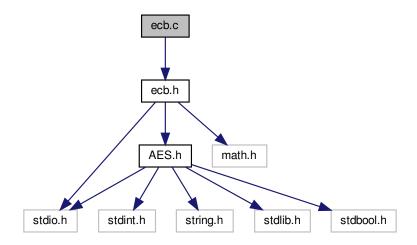
Copyright (c) 2019

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

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

3.10 ecb.c File Reference 143

• 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

Copyright (c) 2019

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 )
```

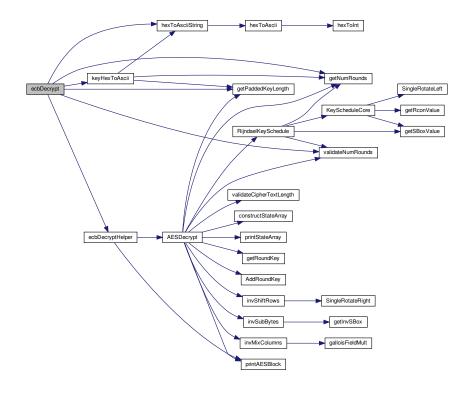
ecbDecrypt - Function to decrypt the user input pointed to by

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,

3.10 ecb.c File Reference

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

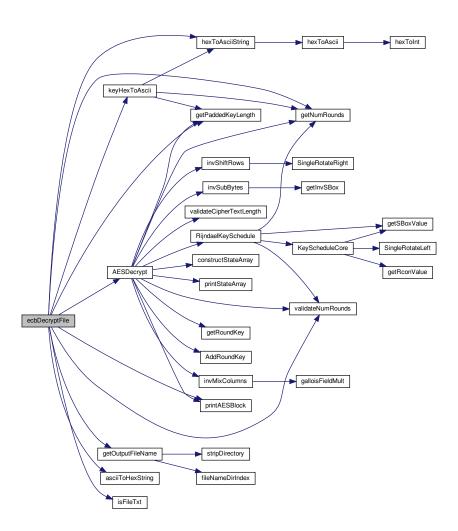
ecbDecryptFile - Function to encrypt the file with name

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.

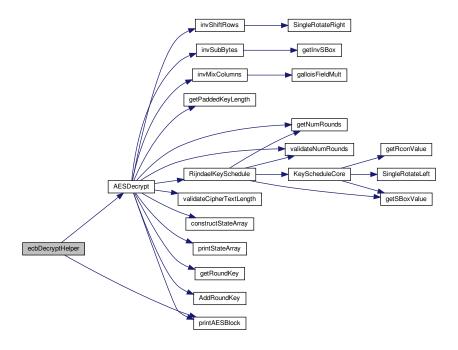
3.10 ecb.c File Reference

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 )
```

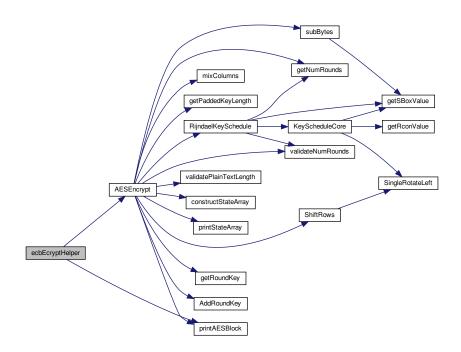
ecbEcryptHelper - Helper function used to encrypt the plaintext pointed to by

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:



3.10 ecb.c File Reference

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 )
```

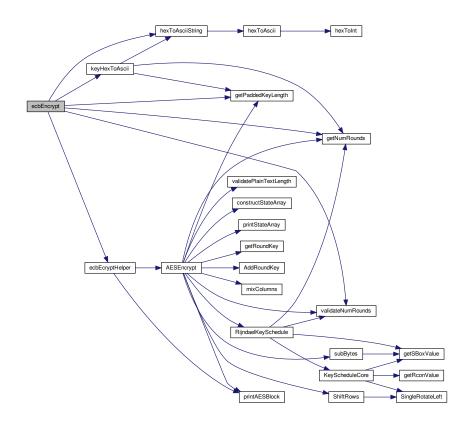
ecbEncrypt - Function to encrypt the user input pointed to by

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

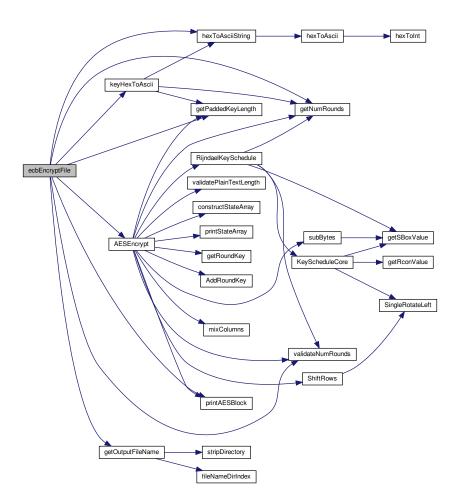
3.11 ecb.h File Reference 151

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:

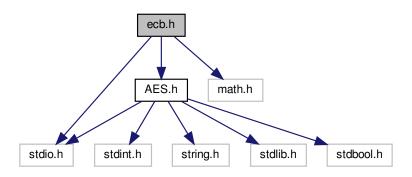


3.11 ecb.h File Reference

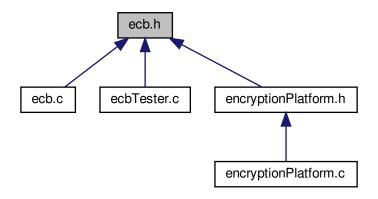
Electronic code book (ECB) - AES header file This file contains the function headers of the functions used for the ECB mode of AES encryption. This system supports both file and user input encryption, as hex or ascii input. If the user inputs data to be encrypted or decrypted, the result will be printed 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)

3.11 ecb.h File Reference 153

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

Copyright (c) 2019

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 )
```

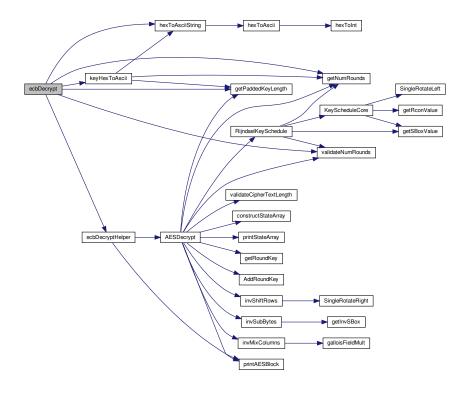
ecbDecrypt - Function to decrypt the user input pointed to by

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,

3.11 ecb.h File Reference

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

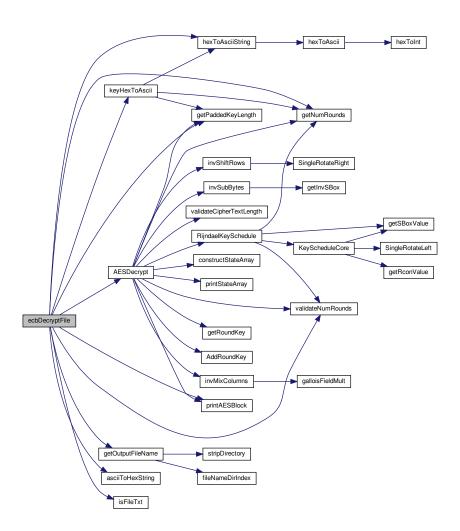
ecbDecryptFile - Function to encrypt the file with name

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.	

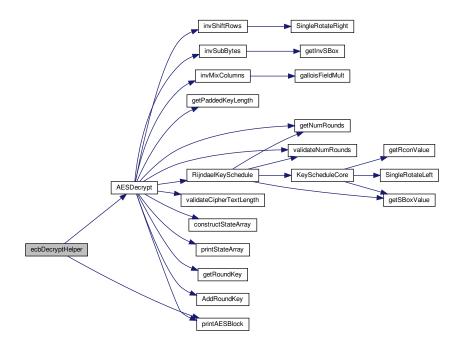
3.11 ecb.h File Reference

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 )
```

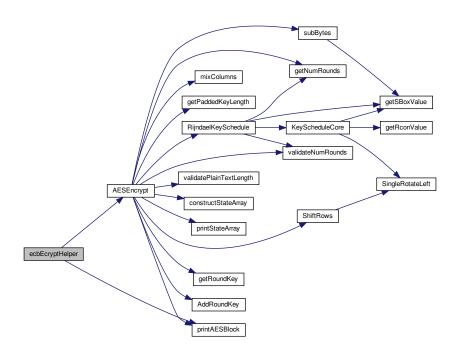
ecbEcryptHelper - Helper function used to encrypt the plaintext pointed to by

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:



3.11 ecb.h File Reference

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 )
```

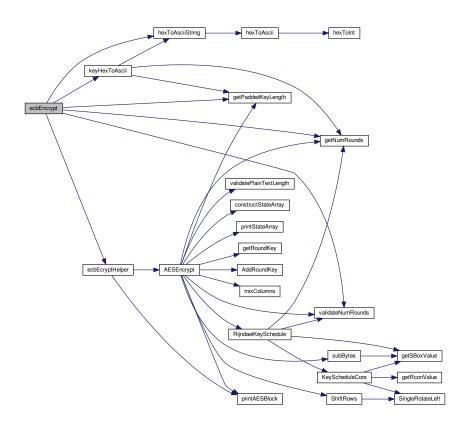
ecbEncrypt - Function to encrypt the user input pointed to by

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

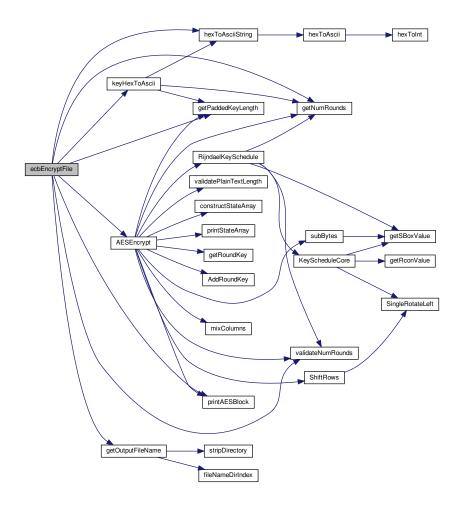
fileName	and write the encrypted version to file with ecbEncrypted appended to the original filename. Performs encryption using the ecb mode and writes the result to a file. If any input is hex, it will convert it to ascii, perform encryption and write it back as ASCII. All terminal output, however, 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.

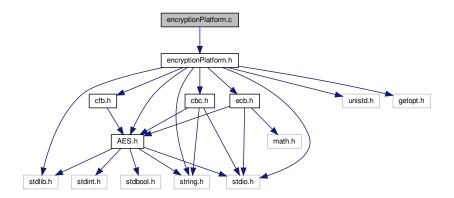
Here is the call graph for this function:



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

Copyright (c) 2019

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

AES.c, 5	fileNameDirIndex, 64
AES_BLOCK_SIZE, 51	galloisFieldMult, 65
AESDecrypt, 9	getInvSBox, 66
AESEncrypt, 11	getNumRounds, 67
AddRoundKey, 8	getOutputFileName, 68
asciiToHexString, 13	getPaddedKeyLength, 69
constructStateArray, 14	getRconValue, 70
fileNameDirIndex, 15	getRoundKey, 71
galloisFieldMult, 16	getSBoxValue, 73
getInvSBox, 17	hexToAscii, 73
getNumRounds, 18	hexToAsciiString, 75
getOutputFileName, 19	hexToInt, 77
getPaddedKeyLength, 20	IVHexToAscii, 82
getRconValue, 21	invMixColumns, 78
getRoundKey, 22	invShiftRows, 79
getSBoxValue, 24	invSubBytes, 80
hexToAscii, 24	isFileTxt, 81
hexToAsciiString, 26	keyHexToAscii, 83
hexToInt, 28	KeyScheduleCore, 85
IVHexToAscii, 33	mixColumns, 88
invMixColumns, 29	printAESBlock, 88
invSBox, 51	printStateArray, 89
invShiftRows, 30	RijndaelKeySchedule, 90
invSubBytes, 31	ShiftRows, 91
isFileTxt, 32	SingleRotateLeft, 92
keyHexToAscii, 34	SingleRotateRight, 93
KeyScheduleCore, 36	stripDirectory, 94
mixColumns, 39	subBytes, 95
printAESBlock, 39	validateCipherTextLength, 96
printStateArray, 40	validateNumRounds, 97
Rcon, 52	validatePlainTextLength, 98
RijndaelKeySchedule, 41	XORBlocks, 99
sbox, 52	AES BLOCK SIZE
ShiftRows, 42	AES.c, 51
SingleRotateLeft, 43	cfb.h, 140
SingleRotateRight, 44	AESDecrypt
stripDirectory, 45	AES.c, 9
subBytes, 46	AES.h, 58
VERBOSE, 53	AES.II, 56 AESEncrypt
	ž.
validateCipherTextLength, 47	AES.c, 11
validateNumRounds, 48	AES.h, 60
validatePlainTextLength, 49	AddRoundKey
XORBlocks, 50	AES.c, 8
AES.h, 53	AES.h, 57
AESDecrypt, 58	aesTester.c, 100
AESEncrypt, 60	main, 102
AddRoundKey, 57	asciiToHexString
asciiToHexString, 62	AES.c, 13
constructStateArray, 63	AES.h, 62

166 INDEX

cbc., 103		
cbcDecryptFile, 106 cbcEncrypt, 108 cbcEncrypt, 108 cbcEncryptFile, 110 cbc.h, 112 cbc.h, 112 cbcDecrypt, 113 cbcDecrypt, 113 cbcDecryptFile, 115 cbcEncrypt, 117 cbcEncryptFile, 119 cbcDecrypt cbc.c, 105 cbc.h, 113 cbcDecrypt cbc.c, 105 cbc.h, 113 cbcDecryptFile cbc.c, 106 cbc.h, 115 cbcDecrypt cbc.c, 106 cbc.h, 117 cbcEncryptFile cbc.c, 106 cbc.h, 117 cbcEncryptFile cbc.c, 108 cbc.h, 117 cbcEncryptFile cbc.c, 108 cbc.h, 117 cbcEncryptFile cbc.c, 110 cbc.h, 119 cbcDester, 121 cbcDester, 121 cbcDester, 121 cbcDester, 122 cbcDecryptFile cbc.c, 110 cbc.h, 119 cbcDester, 121 cbcDester, 121 cbcDester, 122 cbcDecryptFile, 128 cbcDecryptFile, 128 cbcDecryptFile, 128 cbcDecryptFile, 134 cbcDecryptFile, 134 cbcDecryptFile, 134 cbcDecryptFile, 136 vERBOSE, 140 cbcDecryptFile cbc., 125 cbch, 136 cbcDecryptFile cbc., 126 cbch, 137 cbcDecryptFile cbc., 128 cbch, 139 cbcDecryptFile cbc., 128 cbch, 138 cbcDecryptFile cbc., 128 cbch, 139 cbcDecryptFile cbcc, 128 cbch, 139 cbcDecryptFile cbcc, 128 cbch, 139 cbcDecryptFile cbcc, 129 cbcch, 139 cbcDecryptFile cbcch, 139 cbcDecryptFile cbcch, 139 cbccch cbcch, 139 cbccch cbcch, 139 cbccch cbcch cbcch, 139 cbccch cbcch	cbc.c, 103	ecb.h, 151
cbcEncrypt, 108 ecbDecryptHelper, 156 cbch, 112 ecbEncryptHelper, 157 cbcb, 112 ecbEncrypt, 113 cbcDecrypt 113 ecbEncrypt, 159 cbcDecrypt 117 ecb.c, 143 cbcEncryptFile, 119 ecb.n, 153 cbcDecrypt ecb.c, 144 cbch, 113 ecb.n, 153 cbcDecrypt ecb.c, 105 ecb.c, 144 cbch, 113 ecb.c, 144 cbc, 141 ecb., 144 ecb.c, 106 ecb.c, 146 cbc.c, 106 ecb.c, 146 cbc.c, 108 ecb.c, 146 cbc.n, 156 ecberryptHelper cbc.c, 108 ecb.c, 147 cbc.n, 117 ecb.n, 156 cbcEncryptFile ecb.n, 157 cbcEncryptFile ecb.n, 157 cbcEncryptFile ecb.n, 159 cbcTester.c, 121 ecbencryptFile cbc.c, 122 ecb.n, 159 cbberoryptFile, 125 ecb.n, 150 cbberoryptFile, 126 fileNameDirindex cbb.n, 130 AES.b, 64 cbberoryptFile, 134 AES	cbcDecrypt, 105	ecbDecrypt, 153
cbcEncryptFile, 110 ecbEcryptHelper, 157 cbcDerypt, 113 ecbEncrypt, 159 cbcDecrypt, 117 ecb., 143 cbcDecrypt ecb., 153 cbcDecrypt ecb., 153 cbcDecrypt (cbc, 105 ecb., 153 cbcDecryptFile ecb., 144 cbc., 113 ecb., 144 cbc., 106 ecb., 146 cbc., 106 ecb., 146 cbc., 115 ecb., 156 cbcEncrypt ecbEcryptHelper cbc., 106 ecb., 156 cbc., 117 ecb., 156 cbc., 149 ecb., 157 cbc., 110 ecb., 157 cbc., 111 ecb., 157 cbcryptFile ecb., 149 ecb., 159 ecb., 149 ecb., 149 ecb., 159 ecbroryptFile ecb., 150 ecb., 149 ecb., 150 ecb., 150 ecb., 150	cbcDecryptFile, 106	ecbDecryptFile, 154
cbc.h, 112 ecbEncrypt, 159 cbcDecryptFile, 115 ecbEncryptFile, 160 cbcEncryptFile, 117 ecb., 143 cbcEncryptFile, 119 ecb., 143 cbcDecrypt ecb., 153 ecbc, 105 ecb., 154 cbcDecryptFile ecb., 154 cbcb, 113 ecb., 144 cbcb, 115 ecb., 154 cbc, 106 ecb., 156 cbch, 115 ecb., 166 cbc, 146 ecb., 156 cbc, 117 ecb., 156 cbc, 118 ecb., 156 cbc, 119 ecb., 157 cbc, 117 ecb., 157 cbc, 118 ecb., 157 cbc, 119 ecb., 159 cbcTencryptFile ecbEncryptFile cbcTencryptFile ecbEncryptFile cbc, 122 ecb., 150 cbc, 123 ecb., 150 cbcpryptFile, 125 ecbEncryptFile cfbDecrypt, 132 ecb., 160 cfbDecrypt, 132 fileNameDirindex cfb, 130 AES., 64 cfbDecrypt, 132 <	cbcEncrypt, 108	ecbDecryptHelper, 156
cbcDecrypt, 113 ecbDecrypt (16) cbcDecrypt (117) ecbDecrypt (18) cbcDecrypt (117) ecb. (143) cbcDecrypt (118) ecb. (144) cbcDecrypt (113) ecb. (144) cbc. (105) ecb. (144) cbc. (113) ecb. (146) cbc. (106) ecb. (146) cbc. (115) ecb. (146) cbc. (116) ecb. (147) cbc. (117) ecb. (147) cbc. (117) ecb. (147) cbc. (117) ecb. (149) cbc. (119) ecb. (149) cbc. (119) ecb. (149) cbc. (119) ecb. (149) cbc. (119) ecb. (149) cbc. (122) ecbEncrypt (128) cbDecryptFile ecbEncryptFile cbDecryptFile ecbEncryptFile cbEncryptFile ecb. (150)	cbcEncryptFile, 110	ecbEcryptHelper, 157
cbcDecryptFile, 115 ecbCecrypt cbcEncryptFile, 119 ecb., 143 cbcDecrypt ecb., 153 cbcDecryptFile ecb., 154 cbc., 105 ecb., 144 cbc., 106 ecb., 154 cbc.c, 106 ecb., 166 cbc.h, 155 ecb., 166 cbc.h, 117 ecb., 166 cbc.h, 17 ecb., 177 cbc.n, 108 ecb., 177 cbc.n, 110 ecb., 177 cbc.n, 110 ecb., 182 cbc.n, 110 ecb., 149 cbc.n, 122 ecb., 150 cbDecryptFile ecb., 150 cbDecrypt, 123 ecb., 150 cbDecrypt, 126 ecb.h, 160 cbDecryptFile, 125 encryptionPlatform.c, 161 cbDecryptFile, 128 printHelp, 163 cbL, 130 AES.BLOCK_SIZE, 140 cbDecrypt, 132 AES., 64 cbDecrypt, 133 AES., 64 cbDecrypt, 132 AES., 64 cbDecrypt, 133 AES., 64 cbDecrypt, 140 ecb., 128 cbDecr	cbc.h, 112	ecbEncrypt, 159
cbcDecryptFile, 115 ecbCecrypt cbcEncryptFile, 119 ecb., 143 cbcDecrypt ecb., 153 cbcDecryptFile ecb., 154 cbc., 105 ecb., 144 cbc., 106 ecb., 154 cbc.c, 106 ecb., 166 cbc.h, 155 ecb., 166 cbc.h, 117 ecb., 166 cbc.h, 17 ecb., 177 cbc.n, 108 ecb., 177 cbc.n, 110 ecb., 177 cbc.n, 110 ecb., 182 cbc.n, 110 ecb., 149 cbc.n, 122 ecb., 150 cbDecryptFile ecb., 150 cbDecrypt, 123 ecb., 150 cbDecrypt, 126 ecb.h, 160 cbDecryptFile, 125 encryptionPlatform.c, 161 cbDecryptFile, 128 printHelp, 163 cbL, 130 AES.BLOCK_SIZE, 140 cbDecrypt, 132 AES., 64 cbDecrypt, 133 AES., 64 cbDecrypt, 132 AES., 64 cbDecrypt, 133 AES., 64 cbDecrypt, 140 ecb., 128 cbDecr	cbcDecrypt, 113	ecbEncryptFile, 160
cbcEncryptFile, 119 ecb., 143 cbcDecrypt ecbh, 153 cbcDecrypt ecbh, 153 cbc, 105 ecbh, 154 cbch, 113 ecbh, 154 cbch, 113 ecbc, 144 cbc, 106 ecbc, 146 cbc, 115 ecb, 156 cbch, 115 ecb, 156 cbc, 108 ecbc, 147 cbc, 108 ecbc, 147 cbc, 117 ecb, 157 cbcEncrypt ecbc, 147 ecb, 157 cbcEncryptFile ecbcEncrypt cbc, 140 ecbc, 157 cbc, 150 ecbc, 159 ecbc, 150 ecbc, 159 ecbc, 150 ecbc, 150 ecbcryptFile, 125 ecbcryptFile cfbencrypt, 162 febencryptFile, 163 ecberyp		
cbcEncryptFile, 119 ecb.h, 153 cbcDecryptFile ecbDecryptFile cbc., 105 ecb.c, 144 cbc., 113 ecb.c, 144 cbcDecryptFile ecb.ch, 154 cbc., 106 ecb.c, 146 cbc., 115 ecb.c, 146 cbc.h, 115 ecb.c, 146 cbc.h, 117 ecb.c, 147 cbc.c, 108 ecb.c, 147 cbc.c, 110 ecb.c, 149 cbc.c, 110 ecb.n, 157 cbcEcryptFile ecbEncrypt cbc.c, 110 ecb.n, 159 cbcTencryptFile ecbEncrypt cbcTencryptFile ecbEncrypt cbcTencryptFile ecbEncryptFile cbDecryptFile ecb.n, 160 cbDecryptFile ecb.n, 160 cbDecryptFile ecb.n, 161 cbEncryptFile fileNameDirIndex cfb.n, 130 AES_BLOCK_SIZE, 140 cfbDecryptFile, 134 AES.c, 15 cfbDecryptFile, 138 ecbEncryptIndex cfbDecrypt file getloughtindex cfbDecryptFile getloughtindex <td></td> <td></td>		
cbcDecrypt ecbc, 105 ecbc, 144 cbch, 113 ecbch, 154 cbcDecryptFile ecbh, 154 cbc, 106 ecbh, 155 cbc, 106 ecbh, 156 cbcEncrypt ecbch, 156 cbc, 108 ecbc, 147 cbc, 117 ecb, 157 cbch, 117 ecb, 157 cbcEncryptFile ecbErcrypt cbc, 110 ecbc, 149 cbc, 119 ecb, 159 cbcTester, 121 ecbErcryptFile cfbc, 122 ecb, 159 cfberoryptFile, 123 ecbh, 160 cfbEncryptFile, 125 ecryptionPlatform., 161 cfbEncryptFile, 128 primtHelp, 163 cfbEncryptFile, 128 primtHelp, 163 cfbEncryptFile, 134 AES.B, 64 cfbEncryptFile, 134 AES.B, 64 cfbEncryptFile, 138 encryptionPlatform.c, 163 VERBOSE, 140 galloisFieldMult cfb.c, 123 AES.c, 16 cfb.n, 132 AES.c, 16 cfb.c, 125 AES.c, 16 cfb.n, 136 AES.c		ecb.h, 153
cbc.c, 105 ecb.c, 144 cbc.h, 113 ecb.h, 154 cbc.c, 106 ecb.h, 156 cbc.c, 106 ecb.c, 146 cbc.c, 108 ecb.c, 147 cbc.c, 108 ecb.c, 147 cbc.c, 117 ecb.c, 147 cbc.c, 110 ecb.c, 149 cbc.c, 110 ecb.c, 149 cbc.h, 119 ecb.c, 149 cbc.h, 159 ecb.c, 150 cbcTester.c, 121 ecb.c, 150 cbcTester.c, 122 ecb.h, 160 cbcTester.c, 123 ecb.h, 160 cbDecryptFile, 125 encryptionPlatform.c, 161 cbDecryptFile, 126 freeMemory, 163 cbDecryptFile, 128 printHelp, 163 cbDecrypt, 132 AES.c, 15 cbDecryptFile, 134 AES.c, 16 cbDecryptFile, 138 reeMemory cbEncryptFile, 138 encryptionPlatform.c, 163 vERBOSE, 140 reseMemory cbDecryptFile geltnvSBox cbLo, 123 AES.c, 16 cbLo, 125 AES.c, 18 cbL, 136 AES.c, 29		
cbc.h, 113 ecb.h, 154 cbcDecryptFile ecbDecryptHelper cbc.c, 106 ecb.h, 156 cbcEncrypt ecbc.h, 156 cbcEncrypt ecb.c, 147 cbc.c, 108 ecb.c, 147 cbc.h, 117 ecb.h, 157 cbcEncryptFile ecb.c, 149 cbc., 110 ecb.c, 149 cbc., 119 ecb.h, 159 cbcTester.c, 121 ecbEncryptFile cbc, 122 ecb.h, 159 cbDecrypt, 123 ecb.h, 160 cbDecryptFile, 125 encryptionPlatform.c, 161 cbEncryptFile, 128 printHelp, 163 cbEncryptFile, 128 printHelp, 163 cbDecrypt, 132 AES.d, 15 cbDecrypt, 1336 freeMemory cbDecryptFile, 134 AES.h, 64 cbDecrypt galloisFieldMult cbc, 123 AES.h, 65 cbDecryptFile getNomBox cbc, 125 AES.h, 65 cbEncryptFile getNumBounds cbc, 126 AES.h, 66 cbEncryptFile getOutputFileName		
cbcDecryptFile ecbDecryptHelper cbc., 106 ecb., 146 cbc., 115 ecb.h, 156 cbcEncrypt ecbEcryptHelper cbc., 108 ecb., 147 cbc., 117 ecb., 157 cbcEncryptFile ecbEncrypt cbc., 110 ecb., 159 cbcTester., 121 ecbEncryptFile cbc, 122 ecb., 150 cfbDeoryptFile, 123 ecb., 150 cfbDeoryptFile, 125 ecb., 150 cfbEncrypt, 126 ecb., 160 cfbEncrypt, 128 printHelp, 163 cfbLn, 130 freeMemory, 163 printHelp, 163 printHelp, 163 cfbDeryptFile, 134 AES., 15 cfbDeryptFile, 138 AES., 64 cfbDeryptFile, 138 encryptionPlatform.c, 163 vERBOSE, 140 galloisFieldMult cfbDecrypt File getNemory cfb., 123 AES.h, 65 cfbDerryptFile getNumRounds cfb., 132 AES.c, 16 cfb., 136 AES.c, 18 cfb., 136 AES.c, 19		
cbc.c, 106 cbc.h, 115 cbcEncrypt cbc.c, 108 cbc.h, 117 cbcEncrypt cbc.c, 110 cbc.h, 117 cbcEncryptFile cbc.c, 110 cbc.h, 119 cbc.h, 119 cbc.h, 119 cbc.h, 119 cbc.h, 119 cbc.h, 122 cfb.cyptHelper cbc.c, 121 cfb.cyptFile cbc.c, 122 cfb.cyptFile, 125 cfbEncrypt, 123 cfbEncrypt, 126 cfbEncrypt, 126 cfbEncryptFile, 128 cfb.h, 130 AES_BLOCK_SIZE, 140 cfbDecrypt, 132 cfbDecryptFile, 134 cfbEncryptFile, 138 VERBOSE, 140 cfbDecryptFile, 138 VERBOSE, 140 cfbDecryptFile cfb.c, 123 cfb.h, 132 cfbDecryptFile cfb.c, 123 cfb.h, 132 cfbDecryptFile cfb.c, 125 cfb.h, 134 cfbEncrypt cfb.c, 125 cfb.h, 134 cfbEncrypt cfb.c, 125 cfb.h, 134 cfbEncrypt cfb.c, 126 cfb.h, 136 cfbEncrypt cfb.c, 126 cfb.h, 136 cfbEncrypt cfb.c, 126 cfb.h, 134 cfbEncrypt cfb.c, 126 cfb.h, 136 cfbEncryptFile cfb.c, 128 cfb.h, 138 cfbEncrypt cfb.c, 128 cfb.h, 138 cfbEncryptFile cfb.c, 128 cfb.h, 138 cfb.h, 13		
cbc.h, 115 ecb.h, 156 cbcEncrypt ecbEcryptHelper cbc.c, 108 ecbc., 147 cbc.h, 117 ecb., 147 cbc.h, 119 ecb.h, 157 cbc.c, 110 ecb.c, 149 cbc.h, 119 ecb.h, 159 cbcTester.c, 121 ecbEncryptFile cbc, 122 ecb.n, 160 cfbDecrypt, 123 ecb.n, 160 cfbEncrypt, 126 encryptionPlatform.c, 161 cfbEncrypt, 126 freeMemory, 163 cfbEncrypt, 127 printHelp, 163 cfb.n, 130 fileNameDirIndex AES_BLOCK_SIZE, 140 fileNameDirIndex cfbDecryptFile, 134 AES.h, 64 cfbDecryptFile, 136 freeMemory cfbEncrypt, 136 freeMemory cfbEncryptFile, 138 encryptionPlatform.c, 163 VERBOSE, 140 galloisFieldMult cfbDecryptFile, 138 encryptionPlatform.c, 163 vERBOSE, 140 getNumSounds cfbC, 123 AES.h, 65 cfbDecryptFile getNumRounds cfb.c, 125 AES.h, 66 <tr< td=""><td></td><td></td></tr<>		
cbcEncrypt ecbc.c, 147 cbc.h, 117 ecb.c, 147 cbc.h, 117 ecb.h, 157 cbcEncryptFile ecbEncrypt cbc.c, 110 ecb.c, 149 cbc.h, 119 ecb.c, 149 cbc.t, 159 ecb.c, 150 cbcTester.c, 121 ecbEncryptFile cfbccrypt, 122 ecb., 150 cfbDecrypt, 123 ecb.h, 160 cfbEncrypt, 126 freeMemory, 163 cfbEncryptFile, 128 prinHelp, 163 cfb.h, 130 AES_BLOCK_SIZE, 140 cfbDecrypt, 132 AES.a, 64 cfbEncryptFile, 134 AES.a, 64 cfbEncryptFile, 138 encryptionPlatform.c, 163 VERBOSE, 140 galloisFieldMult cfbCc, 123 AES.h, 64 cfbCpcryptFile getInvSBox cfb.c, 123 AES.h, 65 cfb.h, 132 getInvSBox cfbCp.c, 125 AES.h, 65 cfbEncryptFile getNumRounds cfb.c, 126 AES.h, 66 cfbEncryptFile GetDutpttFileName cfb.c, 128 AES		
cbc.c, 108 cbc.h, 117 cbcEncryptFile cbc.c, 110 cbc.h, 119 cbc.h, 119 cbc.h, 159 cbcTester.c, 121 cfb.c, 122 cfbDecrypt, 123 cfbDecrypt, 126 cfbEncrypt, 126 cfbEncrypt, 126 cfbEncrypt, 128 cfbEncrypt, 128 cfbEncrypt, 128 cfbEncrypt, 128 cfbEncryptFile, 128 cfbEncryptFile, 128 cfbEncryptFile, 128 cfbDecrypt, 130 cfbDecrypt, 131 cfbDecryptFile, 134 cfbEncrypt, 136 cfbEncrypt, 136 cfbEncryptFile, 134 cfbEncryptFile cfb.c, 123 cfb.h, 132 cfbDecryptFile cfb.c, 125 cfb.h, 134 cfbEncrypt cfb.c, 125 cfb.h, 134 cfbEncrypt cfb.c, 126 cfb.h, 136 cfbEncrypt cfb.c, 128 cfb.h, 136 cfbEncryptFile cfb.c, 128 cfb.h, 138 cfbTester.c, 141 constructStateArray AES.c, 14 AES.h, 63 getPaddedKeyLength AES.c, 29 ceb.c, 142 ecbDecryptHelper, 144 ecbDecryptHelper, 147 ecbEncrypt, 149 ecbEncrypt, 149 AES.c, 24		•
cbc.h, 117 cbcEncryptFile cbc.c, 110 cbc.h, 119 cbc.h, 119 cbcTester.c, 121 cfb.c, 122 cfbDecrypt, 123 cfbDecryptFile, 125 cfbEncrypt, 126 cfbEncryptFile, 128 cfbLn, 130 AES_BLOCK_SIZE, 140 cfbDecrypt, 132 cfbDecrypt, 132 cfbDecrypt, 132 cfbDecrypt, 136 cfbEncrypt, 136 cfbEncrypt, 136 cfbEncrypt, 136 cfbEncrypt, 136 cfbEncryptFile, 138 VERBOSE, 140 cfbDecrypt cfb.c, 123 cfb.h, 132 cfbDecryptFile cfb.c, 125 cfb.h, 134 cfbEncryptFile cfb.c, 125 cfb.h, 134 cfbEncryptFile cfb.c, 126 cfb.h, 136 cfb.h, 138 cfbEncrypt cfb.c, 126 cfb.h, 136 cfb.h, 138 cfbEncrypt cfb.c, 126 cfb.h, 136 cfbEncrypt cfb.c, 126 cfb.h, 138 cfbEncrypt cfb.c, 128 cfb.h, 138 cfbEncrypt cfb.c, 128 cfb.h, 138 cfbEncrypt cfb.c, 128 cfb.h, 138 cfbEncryptFile cfb.c, 128 cfb.h, 136 cfb.h, 136 cfb.h, 136 c	• •	
cbcEncryptFile ecbc, 149 cbc.b, 119 ecbc, 149 cbc.b, 119 ecb.h, 159 cbcTester.c, 121 ecbEncryptFile cfb.c, 122 ecb.c, 150 cfbDecrypt, 123 ecb.h, 160 cfbEncrypt, 126 ecb.h, 160 cfbEncrypt, 127 freeMemory, 163 cfbEncryptFile, 128 printHelp, 163 cfb.h, 130 AES_BLOCK_SIZE, 140 cfbDecrypt, 132 AES.b, 64 cfbDecrypt, 132 AES.b, 64 cfbDecryptFile, 134 AES.b, 64 cfbEncrypt, 136 freeMemory cfbEncrypt, 136 freeMemory cfbEncrypt, 137 aES.b, 64 cfbEncrypt, 138 percyptionPlatform.c, 163 VERBOSE, 140 galloisFieldMult cfbDecrypt galloisFieldMult AES.c, 16 AES.h, 65 cfbDecrypt getInvSBox cfbDecryptFile getInvSBox cfbLo, 128 AES.h, 66 cfb.h, 136 AES.c, 18 cfbEncryptFile getOutputFileName cfb.c, 128		•
cbc.c, 110 cbc.h, 119 cbc.h, 119 cbc.h, 119 cbc.h, 119 ecb.h, 159 cbcTester.c, 121 cfb.c, 122 cfbDecrypt, 123 cfbDecrypt, 123 cfbDecryptFile, 125 cfbEncrypt, 126 cfbEncryptFile, 128 cfb.h, 130 AES_BLOCK_SIZE, 140 cfbDecrypt, 132 cfbDecryptFile, 134 cfbEncrypt, 136 cfbEncrypt, 136 cfbEncryptFile, 138 vERBOSE, 140 CfbDecrypt cfb.c, 123 cfb.h, 132 cfb.h, 132 cfbDecryptFile cfb.c, 125 cfb.h, 134 cfbEncrypt cfb.c, 125 cfb.h, 136 cfbEncryptFile cfb.c, 125 cfb.h, 136 cfbEncryptFile cfb.c, 126 cfb.h, 136 cfbEncrypt cfb.c, 126 cfb.h, 136 cfbEncryptFile cfb.c, 126 cfb.h, 136 cfbEncryptFile cfb.c, 126 cfb.h, 138 cfbEster.c, 141 constructStateArray AES.c, 14 AES.h, 63 cfbTester.c, 141 constructStateArray AES.c, 14 AES.h, 63 cfbTester.c, 142 ecbDecryptFile, 144 ecbDecryptHelper, 146 ecbEcryptHelper, 146 ecbEcryptHelper, 147 ecbEncryptHelper, 149 AES.c, 24		
cbc.h, 119 ecb.h, 159 cbcTester.c, 121 ecbEncryptFile cfb.c, 122 ecb.c, 150 cfbDecrypt, 123 ecb.h, 160 cfbEncrypt, 126 encryptionPlatform.c, 161 cfbEncrypt, 126 freeMemory, 163 cfbEncryptFile, 128 printHelp, 163 cfb.h, 130 AES_BLOCK_SIZE, 140 cfbDecrypt, 132 AES.c, 15 cfbDecryptFile, 134 AES.h, 64 cfbEncryptFile, 138 encryptionPlatform.c, 163 VERBOSE, 140 galloisFieldMult cfbCerypt galloisFieldMult cfb.c, 123 AES.c, 16 cfb., 132 AES.c, 16 cfb.n, 132 AES.h, 65 cfbDecryptFile getInvSBox cfb.c, 125 AES.h, 65 cfb.n, 134 getNumRounds cfbEncrypt getNumRounds cfbEncryptFile getNumRounds cfbEncryptFile getOutputFileName cfb., 128 AES.c, 19 cfb.n, 138 GetDecryptFile cfbEncryptFile getPaddedKeyLength <t< td=""><td></td><td></td></t<>		
cbcTester.c, 121 ecbEncryptFile cfbDecrypt, 123 ecb.c, 150 cfbDecryptFile, 125 ecb.h, 160 cfbEncrypt, 126 freeMemory, 163 cfbEncryptFile, 128 printHelp, 163 cfb.h, 130 fileNameDirIndex cfb.h, 130 AES_BLOCK_SIZE, 140 cfbDecrypt, 132 AES.c, 15 cfbDecryptFile, 134 AES.h, 64 cfbEncrypt, 136 freeMemory cfbEncryptFile, 138 encryptionPlatform.c, 163 VERBOSE, 140 galloisFieldMult cfbCorypt galloisFieldMult cfb.c, 123 AES.c, 16 cfb., 132 AES.h, 65 cfbDecryptFile getInvSBox cfb.c, 125 AES.h, 65 cfb.h, 134 AES.h, 66 cfbcncrypt getNumRounds cfb.c, 126 AES.h, 67 cfb.n, 136 AES.h, 67 cfbEncryptFile getOutputFileName cfb.c, 128 AES.h, 67 cfb.n, 138 AES.h, 69 cfbTester.c, 141 getPaddedKeyLength constructS		
cfb.c, 122 ecb.c, 150 cfbDecrypt, 123 ecb.h, 160 cfbDecryptFile, 125 encryptionPlatform.c, 161 cfbEncrypt, 126 freeMemory, 163 cfb.h, 130 printHelp, 163 AES_BLOCK_SIZE, 140 fileNameDirIndex cfbDecrypt, 132 AES.c, 15 cfbDecrypt, 136 freeMemory cfbEncryptFile, 138 encryptionPlatform.c, 163 VERBOSE, 140 galloisFieldMult cfbDecrypt galloisFieldMult AES.c, 16 AES.h, 64 cfbDecrypt getInvSBox cfb., 132 AES.h, 65 cfbDecryptFile getInvSBox cfb., 134 AES.c, 17 cfb.h, 134 AES.c, 18 cfbEncrypt getNumRounds cfb.c, 126 AES.h, 66 cfb.n, 136 GetSumRounds cfbEncryptFile getOutputFileName AES.c, 19 AES.h, 67 getDecryptFile AES.c, 19 AES.h, 68 getPaddedKeyLength cfbTester.c, 141 getPaddedKeyLength cost.c, 14		
cfbDecrypt, 123 ecb.h, 160 cfbEncrypt, 126 encryptionPlatform.c, 161 cfbEncrypt, 128 freeMemory, 163 cfb.h, 130 printHelp, 163 AES_BLOCK_SIZE, 140 fileNameDirIndex cfbDecrypt, 132 AES.c, 15 cfbDecryptFile, 134 AES.c, 15 cfbEncrypt, 136 freeMemory cfbEncrypt, 138 encryptionPlatform.c, 163 VERBOSE, 140 galloisFieldMult cfbDecrypt galloisFieldMult cfb.c, 123 AES.c, 16 cfb.h, 132 AES.c, 65 cfbDecryptFile getInvSBox cfb.c, 125 Cfb.n cfb.n, 134 AES.n, 66 cfbEncrypt getNumRounds cfb.c, 126 AES.n, 66 cfb.n, 136 AES.c, 18 cfbEncryptFile getOutputFileName cfb.c, 128 AES.n, 67 cfbTester.c, 141 getPaddedKeyLength constructStateArray AES.c, 20 AES.n, 69 AES.n, 69 getRconValue AES.c, 21 ecb.c, 142		
cfbDecryptFile, 125 encryptionPlatform.c, 161 cfbEncrypt, 126 freeMemory, 163 cfbEncryptFile, 128 printHelp, 163 cfb.h, 130 fileNameDirIndex AES_BLOCK_SIZE, 140 fileNameDirIndex cfbDecrypt, 132 AES.c, 15 cfbDecryptFile, 134 AES.h, 64 cfbEncrypt, 136 freeMemory cfbEncrypt, 138 encryptionPlatform.c, 163 VERBOSE, 140 galloisFieldMult cfbDecrypt galloisFieldMult AES.c, 16 AES.c, 16 AES.h, 65 getInvSBox cfbDecryptFile getInvSBox cfbEncrypt getNumRounds cfb.c, 125 AES.h, 65 cfbEncryptFile getNumRounds cfb.c, 126 AES.h, 66 cfbEncryptFile getOutputFileName cfbEncryptFile getOutputFileName cfbEncryptFile AES.c, 19 cfb.h, 138 AES.n, 68 cfbTester.c, 141 getPaddedKeyLength constructStateArray AES.c, 20 AES.h, 69 AES.c, 21		
cfbEncrypt, 126 cfbEncryptFile, 128 cfb.h, 130 AES_BLOCK_SIZE, 140 cfbDecrypt, 132 cfbDecrypt, 132 cfbDecrypt, 136 cfbEncryptFile, 138 VERBOSE, 140 cfbDecrypt cfb.c, 123 cfb.h, 132 cfb.h, 132 cfbDecryptFile cfb.c, 125 cfb.h, 134 cfbEncrypt cfb.c, 125 cfb.h, 134 cfbEncrypt cfb.c, 125 cfb.h, 136 cfbEncrypt cfb.c, 125 cfb.h, 136 cfbEncrypt cfb.c, 126 cfb.h, 136 cfbEncryptFile cfb.c, 126 cfb.h, 136 cfbEncryptFile cfb.c, 128 cfb.h, 138 cfbTester.c, 141 constructStateArray AES.c, 14 AES.h, 63 cfbTester.c, 144 constructStateArray AES.c, 14 AES.h, 63 cfbTester.c, 142 ecbDecryptFile, 144 ecbDecryptFile, 144 ecbDecryptHelper, 146 ecbEcryptHelper, 147 ecbEncrypt, 149 AES.c, 24	**	
cfbEncryptFile, 128 printHelp, 163 cfb.h, 130 AES_BLOCK_SIZE, 140 fileNameDirIndex cfbDecrypt, 132 AES.c, 15 AES.c, 15 cfbDecrypt, 136 AES.h, 64 AES.h, 64 cfbEncrypt, 136 freeMemory encryptionPlatform.c, 163 VERBOSE, 140 galloisFieldMult AES.c, 16 cfbLe, 123 AES.h, 65 AES.h, 65 cfbDecrypt getInvSBox AES.h, 65 cfbLe, 125 AES.h, 66 AES.c, 17 cfb.h, 134 AES.h, 66 GetNumRounds cfbEncrypt getNumRounds AES.c, 18 cfbEncryptFile getOutputFileName AES.h, 67 cfbEncryptFile getOutputFileName AES.c, 19 AES.h, 68 cfbTester.c, 141 getPaddedKeyLength AES.c, 20 AES.h, 69 cfbTester.c, 141 getPaddedKeyLength AES.c, 21 AES.h, 69 getRonValue AES.c, 21 AES.h, 70 AES.c, 21 AES.h, 70 AES.c, 22 AES.h, 71 AES.c, 22 AES.h, 71 AES.c, 24	**	
cfb.h, 130 AES_BLOCK_SIZE, 140 fileNameDirIndex cfbDecrypt, 132 AES.c, 15 cfbDecryptFile, 134 AES.h, 64 cfbEncrypt, 136 freeMemory cfbEncryptile, 138 encryptionPlatform.c, 163 VERBOSE, 140 galloisFieldMult cfb.c, 123 AES.c, 16 cfb.h, 132 AES.h, 65 cfbDecryptFile getInvSBox cfb.c, 125 AES.h, 65 cfb.n, 134 AES.h, 66 cfb.n, 134 AES.h, 66 cfb.n, 136 AES.h, 66 cfb.n, 136 AES.h, 67 cfbEncryptFile getOutputFileName cfb.c, 128 AES.h, 67 cfb.h, 138 AES.h, 68 cfbTester.c, 141 getPaddedKeyLength constructStateArray AES.h, 69 AES.c, 20 AES.h, 69 qetRoundKey AES.h, 70 getDecrypt, 143 getRoundKey ecbDecryptHelper, 146 AES.h, 71 ecbEcryptHelper, 147 getSBoxValue ecbEncrypt, 149 AES.c, 24		
AES_BLOCK_SIZE, 140 cfbDecrypt, 132 cfbDecryptFile, 134 cfbEncrypt, 136 cfbEncryptFile, 138 VERBOSE, 140 cfbDecrypt cfb.c, 123 cfb.h, 132 cfb.h, 132 cfb.h, 134 cfbEncryptFile cfb.c, 125 cfb.h, 134 cfbEncryptFile cfb.c, 125 cfb.h, 134 cfbEncrypt cfb.c, 126 cfb.h, 136 cfbEncrypt cfb.c, 126 cfb.h, 136 cfbEncrypt cfb.c, 126 cfb.h, 136 cfbEncryptFile cfb.c, 127 cfb.h, 138 cfbEncryptFile cfb.c, 128 cfb.h, 138 cfbEncryptFile cfb.c, 128 cfb.h, 138 cfbEncryptFile cfb.c, 128 cfb.h, 138 cfbEncryptFile cfb.c, 128 cfb.h, 138 cfbTester.c, 141 constructStateArray AES.c, 14 AES.c, 14 AES.h, 63 cfbEncryptFile cebDecrypt, 143 ecbDecryptFile, 144 ecbDecryptHelper, 146 ecbEcryptHelper, 147 ecbEncrypt, 149 AES.c, 24		рпппер, 163
cfbDecrypt, 132 cfbDecryptFile, 134 cfbEncrypt, 136 cfbEncryptFile, 138 VERBOSE, 140 cfbDecrypt cfb.c, 123 cfb.h, 132 cfbDecryptFile cfb.c, 125 cfb.h, 134 cfbEncryptFile cfb.c, 126 cfb.h, 136 cfbEncrypt cfb.c, 127 cfb.h, 138 cfbEncrypt cfb.c, 128 cfb.h, 136 cfbEncrypt cfb.c, 126 cfb.h, 136 cfbEncrypt cfb.c, 126 cfb.h, 136 cfbEncryptFile cfb.c, 128 cfb.h, 138 cfbEster.c, 141 constructStateArray AES.c, 14 AES.h, 63 getPaddedKeyLength AES.c, 20 AES.h, 69 getRconValue AES.c, 21 ecb.c, 142 ecbDecrypt, 143 ecbDecryptFile, 144 ecbDecryptHelper, 146 ecbEcryptHelper, 147 getSBoxValue ecbEncrypt, 149 AES.c, 24		fileNameDirIndex
cfbDecryptFile, 134 cfbEncrypt, 136 cfbEncrypt, 136 cfbEncryptFile, 138 VERBOSE, 140 cfbDecrypt cfb.c, 123 cfb.h, 132 cfbDecryptFile cfb.c, 125 cfb.h, 134 cfbEncrypt cfb.c, 126 cfb.h, 136 cfbEncrypt cfb.c, 126 cfb.h, 136 cfbEncrypt cfb.c, 128 cfb.h, 138 cfbEncryptFile cfb.c, 128 cfb.h, 138 cfbEncryptFile cfb.c, 128 cfb.h, 138 cfbTester.c, 141 constructStateArray AES.c, 14 AES.h, 63 ecbDecrypt, 143 ecbDecryptFile, 144 ecbDecryptHelper, 146 ecbEcryptHelper, 147 ecbEncrypt, 149 AES.c, 24 AES.b, 71 AES.h, 64 freeMemory encryptionPlatform.c, 163 AES.h, 64 freeMemory encryptionPlatform.c, 163 AES.h, 65 galloisFieldMult AES.c, 16 AES.c, 17 AES.h, 65 AES.c, 17 AES.h, 66 GetNumRounds AES.c, 18 AES.h, 67 getNumRounds AES.c, 18 AES.h, 67 getOutputFileName AES.c, 19 AES.h, 68 getPaddedKeyLength AES.c, 20 AES.h, 69 getRconValue AES.c, 21 AES.h, 70 getRoundKey AES.c, 22 AES.h, 71 getSBoxValue AES.c, 24		
cfbEncrypt, 136 cfbEncryptFile, 138 VERBOSE, 140 cfbDecrypt cfb.c, 123 cfb.h, 132 cfbDecryptFile cfb.c, 125 cfb.h, 134 cfbEncrypt cfb.c, 126 cfb.h, 136 cfbEncrypt cfb.c, 125 cfb.h, 134 cfbEncrypt cfb.c, 126 cfb.h, 136 cfbEncrypt cfb.c, 126 cfb.h, 136 cfbEncryptFile cfb.c, 128 cfb.h, 138 cfbTester.c, 141 constructStateArray AES.c, 14 AES.h, 63 ecbDecrypt, 143 ecbDecrypt, 143 ecbDecryptFile, 144 ecbDecryptHelper, 146 ecbEcryptHelper, 147 ecbEncrypt, 149 ecbEncrypt, 149 AES.c, 24	**	
cfbEncryptFile, 138 encryptionPlatform.c, 163 VERBOSE, 140 galloisFieldMult cfb.c, 123 AES.c, 16 cfb.h, 132 AES.h, 65 cfbDecryptFile getInvSBox cfb.c, 125 AES.h, 66 cfb.h, 134 getNumRounds cfbEncrypt getNumRounds cfb.c, 126 AES.c, 18 cfb.h, 136 AES.h, 67 cfbEncryptFile getOutputFileName cfb.c, 128 AES.c, 19 cfb.h, 138 AES.h, 68 cfbTester.c, 141 getPaddedKeyLength constructStateArray AES.c, 20 AES.h, 63 getRconValue AES.h, 69 AES.h, 69 getRconValue AES.c, 21 ecb.c, 142 AES.h, 70 ecbDecrypt, 143 getRoundKey ecbDecryptFile, 144 AES.c, 22 ecbDecryptHelper, 146 AES.h, 71 ecbEcryptHelper, 147 getSBoxValue ecbEncrypt, 149 AES.c, 24	**	
VERBOSE, 140 cfbDecrypt cfb.c, 123 cfb.h, 132 cfbDecryptFile cfb.c, 125 cfb.h, 134 cfbEncrypt cfb.c, 126 cfb.h, 136 cfbEncrypt cfb.c, 126 cfb.h, 136 cfbEncrypt cfb.c, 126 cfb.h, 136 cfbEncryptFile cfb.c, 128 cfb.h, 138 cfbEncryptFile cfb.c, 128 cfb.h, 138 cfbTester.c, 141 constructStateArray AES.c, 14 AES.h, 63 ecb.c, 14 AES.h, 63 ecb.c, 142 ecb.c, 142 ecbDecryptFile, 144 ecbEcryptHelper, 146 ecbEcryptHelper, 147 ecbEcryptHelper, 147 ecbEncrypt, 149 AES.c, 24		
cfbDecrypt galloisFieldMult cfb.c, 123 AES.c, 16 cfb.h, 132 AES.h, 65 cfbDecryptFile getInvSBox cfb.c, 125 AES.h, 66 cfb.h, 134 AES.h, 66 cfbEncrypt getNumRounds cfb.c, 126 AES.c, 18 cfb.h, 136 AES.h, 67 cfbEncryptFile getOutputFileName cfb.c, 128 AES.c, 19 cfb.h, 138 AES.h, 68 cfbTester.c, 141 getPaddedKeyLength constructStateArray AES.c, 20 AES.c, 14 AES.h, 69 getRconValue AES.h, 69 getRconValue AES.h, 70 ecb.c, 142 AES.h, 70 ecbDecrypt, 143 getRoundKey ecbDecryptHelper, 144 AES.c, 22 ecbDecryptHelper, 146 AES.h, 71 ecbEcryptHelper, 147 getSBoxValue ecbEncrypt, 149 AES.c, 24	- ·	encryption lationnic, 100
cfb.c, 123 AES.c, 16 cfb.h, 132 AES.h, 65 cfbDecryptFile getInvSBox cfb.c, 125 AES.h, 66 cfb.h, 134 AES.h, 66 cfbEncrypt getNumRounds cfb.c, 126 AES.c, 18 cfb.h, 136 AES.h, 67 cfbEncryptFile getOutputFileName cfb.c, 128 AES.c, 19 cfb.h, 138 AES.h, 68 cfbTester.c, 141 getPaddedKeyLength constructStateArray AES.c, 20 AES.h, 69 AES.h, 69 getRconValue AES.c, 21 AES.c, 21 AES.h, 70 ecb.c, 142 AES.h, 70 ecbDecryptFile, 144 AES.c, 22 ecbDecryptHelper, 146 AES.h, 71 ecbEcryptHelper, 147 getSBoxValue ecbEncrypt, 149 AES.c, 24		galloisFieldMult
cfb.h, 132 AES.h, 65 cfbDecryptFile getInvSBox cfb.c, 125 AES.h, 66 cfb.h, 134 GetNumRounds cfbEncrypt AES.c, 18 cfb.c, 126 AES.h, 67 cfb.h, 136 GetOutputFileName cfb.c, 128 AES.h, 68 cfb.h, 138 AES.h, 68 cfbTester.c, 141 getPaddedKeyLength constructStateArray AES.c, 20 AES.h, 69 AES.h, 69 getRconValue AES.c, 21 AES.h, 70 GetDecrypt, 143 ecb.c, 142 AES.h, 70 ecbDecryptFile, 144 AES.c, 22 ecbDecryptHelper, 146 AES.h, 71 ecbEcryptHelper, 147 getSBoxValue ecbEncrypt, 149 AES.c, 24		
cfbDecryptFile getInvSBox cfb.c, 125 AES.c, 17 cfb.h, 134 getNumRounds cfb.c, 126 AES.h, 66 cfb.c, 126 AES.h, 67 cfbEncryptFile getOutputFileName cfb.c, 128 AES.c, 19 cfb.h, 138 AES.h, 68 cfbTester.c, 141 getPaddedKeyLength constructStateArray AES.c, 20 AES.h, 69 AES.h, 69 getRconValue AES.c, 21 ecb.c, 142 AES.h, 70 ecbDecrypt, 143 getRoundKey ecbDecryptFile, 144 AES.c, 22 ecbDecryptHelper, 146 AES.h, 71 ecbEcryptHelper, 147 getSBoxValue ecbEncrypt, 149 AES.c, 24		
cfb.c, 125 AES.c, 17 cfb.h, 134 GES.h, 66 cfbEncrypt getNumRounds cfb.c, 126 AES.c, 18 cfb.h, 136 AES.h, 67 cfbEncryptFile getOutputFileName cfb.c, 128 AES.c, 19 cfb.h, 138 AES.h, 68 cfbTester.c, 141 getPaddedKeyLength constructStateArray AES.c, 20 AES.h, 69 AES.h, 69 getRconValue AES.c, 21 AES.h, 70 getRoundKey ecbDecryptFile, 144 AES.c, 22 ecbDecryptHelper, 146 AES.h, 71 ecbEcryptHelper, 147 getSBoxValue ecbEncrypt, 149 AES.c, 24		
cfb.h, 134 cfbEncrypt cfb.c, 126 cfb.h, 136 cfbEncryptFile cfb.c, 128 cfb.h, 138 cfbTester.c, 141 constructStateArray AES.h, 63 ecb.c, 144 AES.h, 63 ecb.c, 142 ecb.c, 142 ecbDecryptFile, 144 ecbEcryptHelper, 146 ecbEcryptHelper, 147 ecbEncrypt, 149 AES.c, 24 AES.c, 24 AES.c, 24 AES.c, 22 AES.c, 24		· ·
CfbEncrypt getNumRounds cfb.c, 126 AES.c, 18 cfb.h, 136 AES.h, 67 cfbEncryptFile getOutputFileName cfb.c, 128 AES.h, 68 cfb.h, 138 AES.h, 68 cfbTester.c, 141 getPaddedKeyLength constructStateArray AES.c, 20 AES.h, 69 AES.h, 69 getRconValue AES.c, 21 ecb.c, 142 AES.h, 70 ecbDecrypt, 143 getRoundKey ecbDecryptFile, 144 AES.c, 22 ecbDecryptHelper, 146 AES.h, 71 ecbEcryptHelper, 147 getSBoxValue ecbEncrypt, 149 AES.c, 24		
cfb.c, 126 AES.c, 18 cfb.h, 136 AES.h, 67 cfbEncryptFile getOutputFileName cfb.c, 128 AES.c, 19 cfb.h, 138 AES.h, 68 cfbTester.c, 141 getPaddedKeyLength constructStateArray AES.c, 20 AES.h, 69 AES.h, 69 getRconValue AES.c, 21 ecb.c, 142 AES.h, 70 ecbDecrypt, 143 getRoundKey ecbDecryptFile, 144 AES.c, 22 ecbDecryptHelper, 146 AES.h, 71 ecbEcryptHelper, 147 getSBoxValue ecbEncrypt, 149 AES.c, 24	•	
ofb.h, 136 AES.h, 67 cfbEncryptFile getOutputFileName ofb.c, 128 AES.c, 19 ofb.h, 138 AES.h, 68 cfbTester.c, 141 getPaddedKeyLength constructStateArray AES.c, 20 AES.h, 69 AES.h, 69 getRconValue AES.c, 21 ecb.c, 142 AES.h, 70 ecbDecrypt, 143 getRoundKey ecbDecryptFile, 144 AES.c, 22 ecbDecryptHelper, 146 AES.h, 71 ecbEcryptHelper, 147 getSBoxValue ecbEncrypt, 149 AES.c, 24		•
cfb.ii, 136 cfbEncryptFile cfb.c, 128 cfb.h, 138 cfbTester.c, 141 constructStateArray AES.c, 14 AES.c, 14 AES.h, 63 GetRconValue AES.c, 21 ecb.c, 142 ecbDecrypt, 143 ecbDecryptFile, 144 ecbDecryptHelper, 146 ecbEcryptHelper, 147 ecbEncrypt, 149 GetCoutputFileName AES.c, 19 AES.h, 68 getPaddedKeyLength AES.c, 20 AES.h, 69 getRconValue AES.c, 21 AES.h, 70 getRoundKey AES.c, 22 AES.h, 71 getSBoxValue ecbEncrypt, 149 AES.c, 24		
cfb.c, 128 cfb.h, 138 cfbTester.c, 141 constructStateArray AES.c, 20 AES.h, 69 AES.h, 69 AES.h, 69 AES.h, 63 GetRconValue AES.c, 21 ecb.c, 142 ecbDecrypt, 143 ecbDecryptFile, 144 ecbDecryptHelper, 146 ecbEcryptHelper, 147 ecbEncrypt, 149 AES.c, 24		· ·
cfb.h, 138 cfbTester.c, 141 constructStateArray AES.c, 14 AES.c, 20 AES.h, 69 AES.h, 69 getRconValue AES.c, 21 ecb.c, 142 ecbDecrypt, 143 ecbDecryptFile, 144 ecbDecryptHelper, 146 ecbEcryptHelper, 147 ecbEncrypt, 149 AES.c, 24		-
cfbTester.c, 141 constructStateArray AES.c, 20 AES.h, 69 AES.h, 63 getRaconValue AES.c, 21 ecb.c, 142 ecbDecrypt, 143 ecbDecryptFile, 144 ecbDecryptHelper, 146 ecbEcryptHelper, 147 ecbEncrypt, 149 getRaconValue AES.h, 70 getRoundKey AES.h, 71 getSBoxValue ecbEncrypt, 149 AES.c, 24		
constructStateArray		
AES.c, 14 AES.h, 63 ecb.c, 142 ecbDecrypt, 143 ecbDecryptFile, 144 ecbDecryptHelper, 146 ecbEcryptHelper, 147 ecbEncrypt, 149 AES.h, 69 getRconValue AES.c, 21 AES.h, 70 getRoundKey AES.c, 22 AES.h, 71 getSBoxValue AES.c, 24		
AES.h, 63 getRconValue AES.c, 21 ecb.c, 142 ecbDecrypt, 143 ecbDecryptFile, 144 ecbDecryptHelper, 146 ecbEcryptHelper, 147 ecbEncrypt, 149 getRconValue AES.c, 21 AES.h, 70 getRoundKey AES.c, 22 AES.h, 71 getSBoxValue AES.c, 24		
AES.c, 21 ecb.c, 142 ecbDecrypt, 143 ecbDecryptFile, 144 ecbDecryptHelper, 146 ecbEcryptHelper, 147 ecbEcrypt, 149 AES.c, 21 AES.h, 70 getRoundKey AES.c, 22 AES.h, 71 getSBoxValue AES.c, 24		
ecb.c, 142 ecbDecrypt, 143 ecbDecryptFile, 144 ecbDecryptHelper, 146 ecbEcryptHelper, 147 ecbEncrypt, 149 AES.h, 71 getSBoxValue AES.c, 24	AES.n, 63	_
ecbDecrypt, 143 getRoundKey ecbDecryptFile, 144 AES.c, 22 ecbDecryptHelper, 146 AES.h, 71 ecbEcryptHelper, 147 getSBoxValue ecbEncrypt, 149 AES.c, 24	ech c 142	
ecbDecryptFile, 144 ecbDecryptHelper, 146 ecbEcryptHelper, 147 ecbEncrypt, 149 AES.c, 22 AES.h, 71 getSBoxValue AES.c, 24		
ecbDecryptHelper, 146 AES.h, 71 ecbEcryptHelper, 147 ecbEncrypt, 149 AES.c, 24		-
ecbEcryptHelper, 147 getSBoxValue ecbEncrypt, 149 AES.c, 24		
ecbEncrypt, 149 AES.c, 24	· · · · · · · · · · · · · · · · · · ·	
• •		_
AES.II, 75		
	eopeniorypunie, 150	ALO.II, /J

INDEX 167

hexToAscii AES.c, 24 AES.h, 73	AES.h, 91 SingleRotateLeft AES.c, 43
hexToAsciiString AES.c, 26	AES.h, 92 SingleRotateRight
AES.h, 75 hexToInt AES.c, 28	AES.c, 44 AES.h, 93 stripDirectory
AES.h, 77 IVHexToAscii	AES.c, 45 AES.h, 94
AES.c, 33 AES.h, 82	subBytes AES.c, 46 AES.h, 95
invMixColumns AES.c, 29 AES.h, 78	VERBOSE
invSBox AES.c, 51	AES.c, 53 cfb.h, 140 validateCipherTextLength
invShiftRows AES.c, 30 AES.h, 79	AES.c, 47 AES.h, 96 validateNumRounds
invSubBytes AES.c, 31	AES.c, 48 AES.h, 97
AES.h, 80 isFileTxt AES.c, 32	validatePlainTextLength AES.c, 49 AES.h, 98
AES.h, 81	XORBlocks
keyHexToAscii AES.c, 34 AES.h, 83	AES.c, 50 AES.h, 99
KeyScheduleCore AES.c, 36 AES.h, 85	
main aesTester.c, 102	
mixColumns AES.c, 39 AES.h, 88	
printAESBlock AES.c, 39	
AES.h, 88 printHelp encryptionPlatform.c, 163	
printStateArray AES.c, 40 AES.h, 89	
Rcon	
AES.c, 52 RijndaelKeySchedule AES.c, 41 AES.h, 90	
sbox AES.c, 52	
ShiftRows AES.c, 42	