APPENDIX

//----------------------------------------------------

//

// Code: DES Encryption Header File

// Authors: Tyler Travis & Justin Cox

// Date: 3/23/16

//

//----------------------------------------------------

#ifndef DES\_HH

#define DES\_HH

#include <stdio.h>

#include <stdlib.h>

#include <stdint.h>

#include <string.h>

extern const uint8\_t IP**[**64**];**

extern const uint8\_t FP**[**64**];**

extern const uint8\_t E**[**48**];**

extern const uint8\_t P**[**32**];**

extern const uint8\_t PC\_1**[**56**];**

extern const uint8\_t PC\_2**[**48**];**

extern const uint8\_t ISV**[**16**];**

extern const uint8\_t S\_box1**[**4**][**16**];**

extern const uint8\_t S\_box2**[**4**][**16**];**

extern const uint8\_t S\_box3**[**4**][**16**];**

extern const uint8\_t S\_box4**[**4**][**16**];**

extern const uint8\_t S\_box5**[**4**][**16**];**

extern const uint8\_t S\_box6**[**4**][**16**];**

extern const uint8\_t S\_box7**[**4**][**16**];**

extern const uint8\_t S\_box8**[**4**][**16**];**

void encrypt**(**uint8\_t **\***plain\_text**,** uint16\_t plain\_text\_size**,** uint8\_t **\***cipher\_text**,** uint8\_t key**[**8**]);**

void decrypt**(**uint8\_t **\***plain\_text**,** uint8\_t **\***cipher\_text**,** uint8\_t key**[**8**]);**

void generate\_subkeys**(**uint8\_t key**[**8**],** uint8\_t subkey**[][**6**]);**

void desRound**(**uint8\_t leftHalve**[],** uint8\_t rightHalve**[],** uint8\_t subkey**[**6**]);**

void fFunction**(**uint8\_t rightHalve**[],** uint8\_t subKey**[**6**]);**

void copy\_bit**(**uint8\_t source**[],** uint8\_t dest**[],** uint16\_t source\_bit**,** uint16\_t dest\_bit**);**

void circular\_shift\_array**(**uint8\_t array**[**4**],** uint8\_t shift**);**

void combine\_CD**(**uint8\_t C**[**4**],** uint8\_t D**[**4**],** uint8\_t dest**[**7**]);**

void getPlainText**(**uint8\_t plainText**[],** uint32\_t genNum**);**

#endif

//----------------------------------------------------

//

// Code: DES Encryption C File

// Authors: Tyler Travis & Justin Cox

// Date: 3/23/16

//

//----------------------------------------------------

#include "cpu/o3/des.hh"

const uint8\_t IP**[**64**]** **=** **{**57**,** 49**,** 41**,** 33**,** 25**,** 17**,** 9**,** 1**,**

59**,** 51**,** 43**,** 35**,** 27**,** 19**,** 11**,** 3**,**

61**,** 53**,** 45**,** 37**,** 29**,** 21**,** 13**,** 5**,**

63**,** 55**,** 47**,** 39**,** 31**,** 23**,** 15**,** 7**,**

56**,** 48**,** 40**,** 32**,** 24**,** 16**,** 8**,** 0**,**

58**,** 50**,** 42**,** 34**,** 26**,** 18**,** 10**,** 2**,**

60**,** 52**,** 44**,** 36**,** 28**,** 20**,** 12**,** 4**,**

62**,** 54**,** 46**,** 38**,** 30**,** 22**,** 14**,** 6**};**

const uint8\_t FP**[**64**]** **=** **{**39**,** 7**,** 47**,** 15**,** 55**,** 23**,** 63**,** 31**,**

38**,** 6**,** 46**,** 14**,** 54**,** 22**,** 62**,** 30**,**

37**,** 5**,** 45**,** 13**,** 53**,** 21 **,**61**,** 29**,**

36**,** 4**,** 44**,** 12**,** 52**,** 20**,** 60**,** 28**,**

35**,** 3**,** 43**,** 11**,** 51**,** 19**,** 59**,** 27**,**

34**,** 2**,** 42**,** 10**,** 50**,** 18**,** 58**,** 26**,**

33**,** 1**,** 41**,** 9**,** 49**,** 17**,** 57**,** 25**,**

32**,** 0**,** 40**,** 8**,** 48**,** 16**,** 56**,** 24**};**

const uint8\_t E**[**48**]** **=** **{**31**,** 0**,** 1**,** 2**,** 3**,** 4**,**

3**,** 4**,** 5**,** 6**,** 7**,** 8**,**

7**,** 8**,** 9**,** 10**,** 11**,** 12**,**

11**,** 12**,** 13**,** 14**,** 15**,** 16**,**

15**,** 16**,** 17**,** 18**,** 19**,** 20**,**

19**,** 20**,** 21**,** 22**,** 23**,** 24**,**

23**,** 24**,** 25**,** 26**,** 27**,** 28**,**

27**,** 28**,** 29**,** 30**,** 31**,** 0**};**

const uint8\_t P**[**32**]** **=** **{**15**,** 6**,** 19**,** 20**,** 28**,** 11**,** 27**,** 16**,**

0**,** 14**,** 22**,** 25**,** 4**,** 17**,** 30**,** 9**,**

1**,** 7**,** 23**,** 13**,** 31**,** 26**,** 2**,** 8**,**

18**,** 12**,** 29**,** 5**,** 21**,** 10**,** 3**,** 24**};**

// Left

const uint8\_t PC\_1**[**56**]** **=** **{**56**,** 48**,** 40**,** 32**,** 24**,** 16**,** 8**,**

0**,** 57**,** 49**,** 41**,** 33**,** 25**,** 17**,**

9**,** 1**,** 58**,** 50**,** 42**,** 34**,** 26**,**

18**,** 10**,** 2**,** 59**,** 51**,** 43**,** 35**,**

// Right

62**,** 54**,** 46**,** 38**,** 30**,** 22**,** 14**,**

6**,** 61**,** 53**,** 45**,** 37**,** 29**,** 21**,**

13**,** 5**,** 60**,** 52**,** 44**,** 36**,** 28**,**

20**,** 12**,** 4**,** 27**,** 19**,** 11**,** 3**};**

const uint8\_t PC\_2**[**48**]** **=** **{**13**,** 16**,** 10**,** 23**,** 0**,** 4**,** 2**,** 27**,**

14**,** 5**,** 20**,** 9**,** 22**,** 18**,** 11**,** 3**,**

25**,** 7**,** 15**,** 6**,** 26**,** 19**,** 12**,** 1**,**

40**,** 51**,** 30**,** 36**,** 46**,** 54**,** 29**,** 39**,**

50**,** 44**,** 32**,** 47**,** 43**,** 48**,** 38**,** 55**,**

33**,** 52**,** 45**,** 41**,** 49**,** 35**,** 28**,** 31**};**

const uint8\_t ISV**[**16**]** **=** **{**1**,**1**,**2**,**2**,**2**,**2**,**2**,**2**,**1**,**2**,**2**,**2**,**2**,**2**,**2**,**1**};**

const uint8\_t S\_box1**[**4**][**16**]** **=** **{**

**{**14**,**4**,**13**,**1**,**2**,**15**,**11**,**8**,**3**,**10**,**6**,**12**,**5**,**9**,**0**,**7**},**

**{**0**,**15**,**7**,**4**,**14**,**2**,**13**,**1**,**10**,**6**,**12**,**11**,**9**,**5**,**3**,**8**},**

**{**4**,**1**,**14**,**8**,**13**,**6**,**2**,**11**,**15**,**12**,**9**,**7**,**3**,**10**,**5**,**0**},**

**{**15**,**12**,**8**,**2**,**4**,**9**,**1**,**7**,**5**,**11**,**3**,**14**,**10**,**0**,**6**,**13**}**

**};**

const uint8\_t S\_box2**[**4**][**16**]** **=** **{**

**{**15**,**1**,**8**,**14**,**6**,**11**,**3**,**4**,**9**,**7**,**2**,**13**,**12**,**0**,**5**,**10**},**

**{**3**,**13**,**4**,**7**,**15**,**2**,**8**,**14**,**12**,**0**,**1**,**10**,**6**,**9**,**11**,**5**},**

**{**0**,**14**,**7**,**11**,**10**,**4**,**13**,**1**,**5**,**8**,**12**,**6**,**9**,**3**,**2**,**15**},**

**{**13**,**8**,**10**,**1**,**3**,**15**,**4**,**2**,**11**,**6**,**7**,**12**,**0**,**5**,**14**,**9**}**

**};**

const uint8\_t S\_box3**[**4**][**16**]** **=** **{**

**{**10**,**0**,**9**,**14**,**6**,**3**,**15**,**5**,**1**,**13**,**12**,**7**,**11**,**4**,**2**,**8**},**

**{**13**,**7**,**0**,**9**,**3**,**4**,**6**,**10**,**2**,**8**,**5**,**14**,**12**,**11**,**15**,**1**},**

**{**13**,**6**,**4**,**9**,**8**,**15**,**3**,**0**,**11**,**1**,**2**,**12**,**5**,**10**,**14**,**7**},**

**{**1**,**10**,**13**,**0**,**6**,**9**,**8**,**7**,**4**,**15**,**14**,**3**,**11**,**5**,**2**,**12**}**

**};**

const uint8\_t S\_box4**[**4**][**16**]** **=** **{**

**{**7**,**13**,**14**,**3**,**0**,**6**,**9**,**10**,**1**,**2**,**8**,**5**,**11**,**12**,**4**,**15**},**

**{**13**,**8**,**11**,**5**,**6**,**15**,**0**,**3**,**4**,**7**,**2**,**12**,**1**,**10**,**14**,**9**},**

**{**10**,**6**,**9**,**0**,**12**,**11**,**7**,**13**,**15**,**1**,**3**,**14**,**5**,**2**,**8**,**4**},**

**{**3**,**15**,**0**,**6**,**10**,**1**,**13**,**8**,**9**,**4**,**5**,**11**,**12**,**7**,**2**,**14**}**

**};**

const uint8\_t S\_box5**[**4**][**16**]** **=** **{**

**{**2**,**12**,**4**,**1**,**7**,**10**,**11**,**6**,**8**,**5**,**3**,**15**,**13**,**0**,**14**,**9**},**

**{**14**,**11**,**2**,**12**,**4**,**7**,**13**,**1**,**5**,**0**,**15**,**10**,**3**,**9**,**8**,**6**},**

**{**4**,**2**,**1**,**11**,**10**,**13**,**7**,**8**,**15**,**9**,**12**,**5**,**6**,**3**,**0**,**14**},**

**{**11**,**8**,**12**,**7**,**1**,**14**,**2**,**13**,**6**,**15**,**0**,**9**,**10**,**4**,**5**,**3**}**

**};**

const uint8\_t S\_box6**[**4**][**16**]** **=** **{**

**{**12**,**1**,**10**,**15**,**9**,**2**,**6**,**8**,**0**,**13**,**3**,**4**,**14**,**7**,**5**,**11**},**

**{**10**,**15**,**4**,**2**,**7**,**12**,**9**,**5**,**6**,**1**,**13**,**14**,**0**,**11**,**3**,**8**},**

**{**9**,**14**,**15**,**5**,**2**,**8**,**12**,**3**,**7**,**0**,**4**,**10**,**1**,**13**,**11**,**6**},**

**{**4**,**3**,**2**,**12**,**9**,**5**,**15**,**10**,**11**,**14**,**1**,**7**,**6**,**0**,**8**,**13**}**

**};**

const uint8\_t S\_box7**[**4**][**16**]** **=** **{**

**{**4**,**11**,**2**,**14**,**15**,**0**,**8**,**13**,**3**,**12**,**9**,**7**,**5**,**10**,**6**,**1**},**

**{**13**,**0**,**11**,**7**,**4**,**9**,**1**,**10**,**14**,**3**,**5**,**12**,**2**,**15**,**8**,**6**},**

**{**1**,**4**,**11**,**13**,**12**,**3**,**7**,**14**,**10**,**15**,**6**,**8**,**0**,**5**,**9**,**2**},**

**{**6**,**11**,**13**,**8**,**1**,**4**,**10**,**7**,**9**,**5**,**0**,**15**,**14**,**2**,**3**,**12**}**

**};**

const uint8\_t S\_box8**[**4**][**16**]** **=** **{**

**{**13**,**2**,**8**,**4**,**6**,**15**,**11**,**1**,**10**,**9**,**3**,**14**,**5**,**0**,**12**,**7**},**

**{**1**,**15**,**13**,**8**,**10**,**3**,**7**,**4**,**12**,**5**,**6**,**11**,**0**,**14**,**9**,**2**},**

**{**7**,**11**,**4**,**1**,**9**,**12**,**14**,**2**,**0**,**6**,**10**,**13**,**15**,**3**,**5**,**8**},**

**{**2**,**1**,**14**,**7**,**4**,**10**,**8**,**13**,**15**,**12**,**9**,**0**,**3**,**5**,**6**,**11**}**

**};**

void getPlainText**(**uint8\_t plainText**[],** uint32\_t genNum**){**

plainText**[**4**]** **=** **(**genNum **&** 0xFF000000**)** **>>** 24**;**

plainText**[**5**]** **=** **(**genNum **&** 0x00FF0000**)** **>>** 16**;**

plainText**[**6**]** **=** **(**genNum **&** 0x0000FF00**)** **>>** 8**;**

plainText**[**7**]** **=** **(**genNum **&** 0x000000FF**);**

plainText**[**0**]** **=** 0**;**

plainText**[**1**]** **=** 0**;**

plainText**[**2**]** **=** 0**;**

plainText**[**3**]** **=** 0**;**

**}**

void encrypt**(**uint8\_t **\***plain\_text**,** uint16\_t plain\_text\_size**,** uint8\_t **\***cipher\_text**,** uint8\_t key**[**8**])**

**{**

// 2d array to hold all the generated subkeys for DES

uint8\_t subkey**[**16**][**6**];**

uint8\_t plain\_textIP**[**8**]** **=** **{**0**};**

uint8\_t cipher\_textPreFP**[**8**]** **=** **{**0**};**

uint8\_t leftHalve**[**4**]** **=** **{**0**};**

uint8\_t rightHalve**[**4**]** **=** **{**0**};**

uint8\_t i**;**

// Create the subkeys from the key

generate\_subkeys**(**key**,** subkey**);**

// Send Input through IP (Initial Permutation)

**for(**i **=** 0**;** i **<** 64**;** **++**i**)**

**{**

copy\_bit**(**plain\_text**,** plain\_textIP**,** IP**[**i**],** i**);**

**}**

// Split 64 bits into two 32 bit chunks (L & R)

leftHalve**[**0**]** **=** plain\_textIP**[**0**];**

leftHalve**[**1**]** **=** plain\_textIP**[**1**];**

leftHalve**[**2**]** **=** plain\_textIP**[**2**];**

leftHalve**[**3**]** **=** plain\_textIP**[**3**];**

rightHalve**[**0**]** **=** plain\_textIP**[**4**];**

rightHalve**[**1**]** **=** plain\_textIP**[**5**];**

rightHalve**[**2**]** **=** plain\_textIP**[**6**];**

rightHalve**[**3**]** **=** plain\_textIP**[**7**];**

// Round 1 through 16

**for(**i **=** 0**;** i **<** 16**;** i**++)**

**{**

desRound**(**leftHalve**,** rightHalve**,** subkey**[**i**]);**

**}**

// Recombine final Left and Right Halves

cipher\_textPreFP**[**0**]** **=** rightHalve**[**0**];**

cipher\_textPreFP**[**1**]** **=** rightHalve**[**1**];**

cipher\_textPreFP**[**2**]** **=** rightHalve**[**2**];**

cipher\_textPreFP**[**3**]** **=** rightHalve**[**3**];**

cipher\_textPreFP**[**4**]** **=** leftHalve**[**0**];**

cipher\_textPreFP**[**5**]** **=** leftHalve**[**1**];**

cipher\_textPreFP**[**6**]** **=** leftHalve**[**2**];**

cipher\_textPreFP**[**7**]** **=** leftHalve**[**3**];**

// Send 64 bit recombination into FP (Final Permutation)

**for(**i **=** 0**;** i **<** 64**;** i**++){**

copy\_bit**(**cipher\_textPreFP**,** cipher\_text**,** FP**[**i**],** i**);**

**}**

// printf("Done with encryption ");

//End of Function

**}**

void decrypt**(**uint8\_t **\***plain\_text**,** uint8\_t **\***cipher\_text**,** uint8\_t key**[**8**])**

**{**

// 2D array to hold subkeys

uint8\_t subkey**[**16**][**6**];**

uint8\_t cipher\_textIP**[**8**]** **=** **{**0**};**

uint8\_t plain\_textPreFP**[**8**]** **=** **{**0**};**

uint8\_t leftHalve**[**8**]** **=** **{**0**};**

uint8\_t rightHalve**[**8**]** **=** **{**0**};**

uint16\_t i**;**

// Generate the subkeys

generate\_subkeys**(**key**,** subkey**);**

**for(**i **=** 0**;** i **<** 64**;** i**++){**

copy\_bit**(**cipher\_text**,** cipher\_textIP**,** IP**[**i**],** i**);**

**}**

// Split 64 bits into two 32 bit chunks (L & R)

leftHalve**[**0**]** **=** cipher\_textIP**[**0**];**

leftHalve**[**1**]** **=** cipher\_textIP**[**1**];**

leftHalve**[**2**]** **=** cipher\_textIP**[**2**];**

leftHalve**[**3**]** **=** cipher\_textIP**[**3**];**

rightHalve**[**0**]** **=** cipher\_textIP**[**4**];**

rightHalve**[**1**]** **=** cipher\_textIP**[**5**];**

rightHalve**[**2**]** **=** cipher\_textIP**[**6**];**

rightHalve**[**3**]** **=** cipher\_textIP**[**7**];**

// Round 1 through 16

**for(**i **=** 0**;** i **<** 16**;** i**++)**

**{**

desRound**(**leftHalve**,** rightHalve**,** subkey**[**15 **-** i**]);**

**}**

// Recombine final Left and Right Halves

plain\_textPreFP**[**0**]** **=** rightHalve**[**0**];**

plain\_textPreFP**[**1**]** **=** rightHalve**[**1**];**

plain\_textPreFP**[**2**]** **=** rightHalve**[**2**];**

plain\_textPreFP**[**3**]** **=** rightHalve**[**3**];**

plain\_textPreFP**[**4**]** **=** leftHalve**[**0**];**

plain\_textPreFP**[**5**]** **=** leftHalve**[**1**];**

plain\_textPreFP**[**6**]** **=** leftHalve**[**2**];**

plain\_textPreFP**[**7**]** **=** leftHalve**[**3**];**

// Send 64 bit recombination into FP (Final Permutation)

**for(**i **=** 0**;** i **<** 64**;** i**++){**

copy\_bit**(**plain\_textPreFP**,** plain\_text**,** FP**[**i**],** i**);**

**}**

// printf("Decryption Done ");

//End of Function

**}**

void generate\_subkeys**(**uint8\_t key**[**8**],** uint8\_t subkey**[][**6**])**

**{**

// K+ permuted key array, Use the PS\_1 permutation array to move the bits around

uint8\_t permuted\_key**[**7**]** **=** **{**0**};**

uint32\_t i**,** j**;**

uint8\_t C**[**17**][**4**];**

uint8\_t D**[**17**][**4**];**

uint8\_t temp\_array**[**7**];**

//uint8\_t temp;

//printf("%x%x %x%x %x%x %x%x\n", key[0], key[1], key[2], key[3], key[4], key[5], key[6], key[7]);

**for(**i **=** 0**;** i **<** 56**;** **++**i**)**

**{**

copy\_bit**(**key**,** permuted\_key**,** PC\_1**[**i**],** i**);**

**}**

//printf("%x%x %x%x %x%x %x\n", permuted\_key[0], permuted\_key[1], permuted\_key[2],

// permuted\_key[3], permuted\_key[4], permuted\_key[5], permuted\_key[6]);

// Initial setup for splitting the key

// For the C array:

// The mask: 0x0FFFFFFF should perserve all data

// That is the reason for the shifting

// For the D array:

// Everything already aligns well

C**[**0**][**0**]** **=** permuted\_key**[**0**]** **>>** 4 **&** 0x0F**;**

C**[**0**][**1**]** **=** **(**permuted\_key**[**1**]** **>>** 4 **&** 0x0F**)** **|** permuted\_key**[**0**]** **<<** 4**;**

C**[**0**][**2**]** **=** **(**permuted\_key**[**2**]** **>>** 4 **&** 0x0F**)** **|** permuted\_key**[**1**]** **<<** 4**;**

C**[**0**][**3**]** **=** **(**permuted\_key**[**3**]** **>>** 4 **&** 0x0F**)** **|** permuted\_key**[**2**]** **<<** 4**;**

D**[**0**][**0**]** **=** permuted\_key**[**3**]** **&** 0x0F**;**

D**[**0**][**1**]** **=** permuted\_key**[**4**];**

D**[**0**][**2**]** **=** permuted\_key**[**5**];**

D**[**0**][**3**]** **=** permuted\_key**[**6**];**

// Generate the next 15 C-D pairs by circular shifting

// using the ISV array

**for(**i **=** 1**;** i **<** 17**;** **++**i**)**

**{**

// Copy the previous C and D to the current C and D

// Then perform the shifting on these

memcpy**(**C**[**i**],** C**[**i**-**1**],** **sizeof(**C**[**i**]));**

memcpy**(**D**[**i**],** D**[**i**-**1**],** **sizeof(**D**[**i**]));**

circular\_shift\_array**(**C**[**i**],** ISV**[**i**-**1**]);**

circular\_shift\_array**(**D**[**i**],** ISV**[**i**-**1**]);**

// Combine the C and D arrays to the temp\_array

combine\_CD**(**C**[**i**],** D**[**i**],** temp\_array**);**

// Use PC\_2 to get the correct subkey

// Need to start the subkey at index 0

// Hence the i-1 subscript

**for(**j **=** 0**;** j **<** 48**;** **++**j**)**

**{**

copy\_bit**(**temp\_array**,** subkey**[**i**-**1**],** PC\_2**[**j**],** j**);**

**}**

**}**

**}**

void desRound**(**uint8\_t leftHalve**[],** uint8\_t rightHalve**[],** uint8\_t subkey**[**6**]){**

uint8\_t rightTemp**[**4**];**

//Keep copy of R\_i

rightTemp**[**0**]** **=** rightHalve**[**0**];**

rightTemp**[**1**]** **=** rightHalve**[**1**];**

rightTemp**[**2**]** **=** rightHalve**[**2**];**

rightTemp**[**3**]** **=** rightHalve**[**3**];**

//Send R\_i and subKey into fFuntion()

fFunction**(**rightHalve**,** subkey**);**

//XOR Output of fFunction() with L\_i

rightHalve**[**0**]** **=** rightHalve**[**0**]** **^** leftHalve**[**0**];**

rightHalve**[**1**]** **=** rightHalve**[**1**]** **^** leftHalve**[**1**];**

rightHalve**[**2**]** **=** rightHalve**[**2**]** **^** leftHalve**[**2**];**

rightHalve**[**3**]** **=** rightHalve**[**3**]** **^** leftHalve**[**3**];**

//Make L\_i+1 = R\_i for next round

leftHalve**[**0**]** **=** rightTemp**[**0**];**

leftHalve**[**1**]** **=** rightTemp**[**1**];**

leftHalve**[**2**]** **=** rightTemp**[**2**];**

leftHalve**[**3**]** **=** rightTemp**[**3**];**

//End of Function

**}**

void fFunction**(**uint8\_t rightHalve**[],** uint8\_t subKey**[**6**]){**

uint8\_t rightHalveE**[**6**]** **=** **{**0**};**

uint8\_t rowBits **=** 0**;**

uint8\_t colBits **=** 0**;**

uint8\_t post\_Sbox\_R**[**4**]** **=** **{**0**};**

uint32\_t i**;**

//Send rightHalve to E permutation

**for(**i **=** 0**;** i **<** 48**;** i**++){**

copy\_bit**(**rightHalve**,** rightHalveE**,** E**[**i**],** i**);**

**}**

//XOR subKey with output of E

**for(**i **=** 0**;** i **<** 6**;** i**++){**

rightHalveE**[**i**]** **=** rightHalveE**[**i**]** **^** subKey**[**i**];**

**}**

//printf("E: %02x%02x %02x%02x %02x%02x\n", rightHalveE[0], rightHalveE[1], rightHalveE[2],

// rightHalveE[3], rightHalveE[4], rightHalveE[5]);

//Send output of XOR into Switch Boxes

//------------------------------------------------------------------------

// 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111

// ^ ^ ^ ^ ^ ^ ^ ^

// 0 6 4 2 0 6 4 2

//------------------------------------------------------------------------

//\*\*\*\*\*\*\*\*\*\*\*\*\*

// Sbox1

//\*\*\*\*\*\*\*\*\*\*\*\*\*

rowBits **=** **(**rightHalveE**[**0**]** **&** 0x80**)** **>>** 6**;**

rowBits **|=** **(**rightHalveE**[**0**]** **&** 0x04**)** **>>** 2**;**

colBits **=** **(**rightHalveE**[**0**]** **&** 0x78**)** **>>** 3**;**

post\_Sbox\_R**[**0**]** **=** **(**S\_box1**[**rowBits**][**colBits**]** **<<** 4**);**

//\*\*\*\*\*\*\*\*\*\*\*\*\*

// Sbox2

//\*\*\*\*\*\*\*\*\*\*\*\*\*

rowBits **=** **(**rightHalveE**[**0**]** **&** 0x02**);**

rowBits **|=** **(**rightHalveE**[**1**]** **&** 0x10**)** **>>** 4**;**

colBits **=** **(**rightHalveE**[**0**]** **&** 0x01**)** **<<** 3**;**

colBits **|=** **(**rightHalveE**[**1**]** **&** 0xE0**)** **>>** 5**;**

post\_Sbox\_R**[**0**]** **|=** S\_box2**[**rowBits**][**colBits**];**

//printf("Sbox1&2: %02x \n", post\_Sbox\_R[0]);

//\*\*\*\*\*\*\*\*\*\*\*\*\*

// Sbox3

//\*\*\*\*\*\*\*\*\*\*\*\*\*

rowBits **=** **(**rightHalveE**[**1**]** **&** 0x08**)** **>>** 2**;**

rowBits **|=** **(**rightHalveE**[**2**]** **&** 0x40**)** **>>** 6**;**

colBits **=** **(**rightHalveE**[**1**]** **&** 0x07**)** **<<** 1**;**

colBits **|=** **(**rightHalveE**[**2**]** **&** 0x80**)** **>>** 7**;**

post\_Sbox\_R**[**1**]** **=** **(**S\_box3**[**rowBits**][**colBits**]** **<<** 4**);**

//\*\*\*\*\*\*\*\*\*\*\*\*\*

// Sbox4

//\*\*\*\*\*\*\*\*\*\*\*\*\*

rowBits **=** **(**rightHalveE**[**2**]** **&** 0x20**)** **>>** 4**;**

rowBits **|=** **(**rightHalveE**[**2**]** **&** 0x01**);**

colBits **=** **(**rightHalveE**[**2**]** **&** 0x1E**)** **>>** 1**;**

post\_Sbox\_R**[**1**]** **|=** S\_box4**[**rowBits**][**colBits**];**

//printf("Sbox3&4: %02x \n", post\_Sbox\_R[1]);

//\*\*\*\*\*\*\*\*\*\*\*\*\*

// Sbox5

//\*\*\*\*\*\*\*\*\*\*\*\*\*

rowBits **=** **(**rightHalveE**[**3**]** **&** 0x80**)** **>>** 6**;**

rowBits **|=** **(**rightHalveE**[**3**]** **&** 0x04**)** **>>** 2**;**

colBits **=** **(**rightHalveE**[**3**]** **&** 0x78**)** **>>** 3**;**

post\_Sbox\_R**[**2**]** **=** **(**S\_box5**[**rowBits**][**colBits**]** **<<** 4**);**

//\*\*\*\*\*\*\*\*\*\*\*\*\*

// Sbox6

//\*\*\*\*\*\*\*\*\*\*\*\*\*

rowBits **=** **(**rightHalveE**[**3**]** **&** 0x02**);**

rowBits **|=** **(**rightHalveE**[**4**]** **&** 0x10**)** **>>** 4**;**

colBits **=** **(**rightHalveE**[**3**]** **&** 0x01**)** **<<** 3**;**

colBits **|=** **(**rightHalveE**[**4**]** **&** 0xE0**)** **>>** 5**;**

post\_Sbox\_R**[**2**]** **|=** S\_box6**[**rowBits**][**colBits**];**

//printf("Sbox5&6: %02x \n", post\_Sbox\_R[2]);

//\*\*\*\*\*\*\*\*\*\*\*\*\*

// Sbox7

//\*\*\*\*\*\*\*\*\*\*\*\*\*

rowBits **=** **(**rightHalveE**[**4**]** **&** 0x08**)** **>>** 2**;**

rowBits **|=** **(**rightHalveE**[**5**]** **&** 0x40**)** **>>** 6**;**

colBits **=** **(**rightHalveE**[**4**]** **&** 0x07**)** **<<** 1**;**

colBits **|=** **(**rightHalveE**[**5**]** **&** 0x80**)** **>>** 7**;**

post\_Sbox\_R**[**3**]** **=** **(**S\_box7**[**rowBits**][**colBits**]** **<<** 4**);**

//\*\*\*\*\*\*\*\*\*\*\*\*\*

// Sbox8

//\*\*\*\*\*\*\*\*\*\*\*\*\*

rowBits **=** **(**rightHalveE**[**5**]** **&** 0x20**)** **>>** 4**;**

rowBits **|=** **(**rightHalveE**[**5**]** **&** 0x01**);**

colBits **=** **(**rightHalveE**[**5**]** **&** 0x1E**)** **>>** 1**;**

post\_Sbox\_R**[**3**]** **|=** S\_box8**[**rowBits**][**colBits**];**

//printf("Sbox7&8: %02x \n", post\_Sbox\_R[3]);

//Send output of Switch Boxes into P permutation

**for(**i **=** 0**;** i **<** 32**;** i**++){**

copy\_bit**(**post\_Sbox\_R**,** rightHalve**,** P**[**i**],** i**);**

**}**

//printf("P: %02x%02x %02x%02x\n", rightHalve[0], rightHalve[1], rightHalve[2],

// rightHalve[3]);

//End of Function

**}**

void copy\_bit**(**uint8\_t source**[],** uint8\_t dest**[],** uint16\_t source\_bit**,** uint16\_t dest\_bit**)**

**{**

// Find which byte to look in

uint8\_t source\_byte **=** source\_bit**/**8**;**

// Get the bit offset relative to the byte

uint8\_t source\_bit\_offset **=** 7**-**source\_bit**%**8**;**

// Get the data from at the byte

uint8\_t source\_byte\_data **=** source**[**source\_byte**];**

// Get what the bit is

uint8\_t source\_bit\_data **=** **((**0x1 **<<** source\_bit\_offset**)** **&** source\_byte\_data**)** **>>** source\_bit\_offset**;**

// Find which byte to store the value in

uint8\_t dest\_byte **=** dest\_bit**/**8**;**

// Get the bit taht we need to store the value in

uint8\_t dest\_bit\_offset **=** 7**-**dest\_bit**%**8**;**

// Get the byte data

uint8\_t dest\_byte\_data **=** dest**[**dest\_byte**];**

// mask for the bit destination

uint8\_t dest\_mask**;**

// the bit data for the destination

uint8\_t dest\_bit\_data **=** source\_bit\_data **<<** dest\_bit\_offset**;**

// If the the bit needs to be a 1,

// then the final byte data just needs to be ORed

// for it to work correctly

**if(**source\_bit\_data **==** 0x1**)**

**{**

dest\_mask **=** 0x0**;**

dest\_byte\_data **=** dest\_byte\_data **|** **(**dest\_mask **|** dest\_bit\_data**);**

**}**

// If the bit needs to be a 0, then we need to and it with a mask

// of 0x11..0..11 where the 0 is the position where the zero needs

// to be.

**else**

**{**

dest\_mask **=** **~(**0x1 **<<** dest\_bit\_offset**);**

dest\_byte\_data **=** dest\_byte\_data **&** dest\_mask**;**

**}**

// Store the data back into the array

dest**[**dest\_byte**]** **=** dest\_byte\_data**;**

**}**

void circular\_shift\_array**(**uint8\_t array**[**4**],** uint8\_t shift**)**

**{**

// The temp array holds the values that need to be

// moved between indices

uint8\_t temp**[**4**];**

// For DES, there are only two options for this

// shifting step: shift left by 1 or shift left

// by 2.

**if(**shift **==** 1**)**

**{**

// 0bxxxx xxxx.xxxx xxxx.xxxx xxxx.xxxx xxxx

// ^ | | | <- temp[0]

// ^ | | <- temp[1]

// ^ | <- temp[2]

// ^ <- temp[3]

temp**[**0**]** **=** **(**array**[**0**]** **&** 0x08**)** **>>** 3**;**

temp**[**1**]** **=** **(**array**[**1**]** **&** 0x80**)** **>>** 7**;**

temp**[**2**]** **=** **(**array**[**2**]** **&** 0x80**)** **>>** 7**;**

temp**[**3**]** **=** **(**array**[**3**]** **&** 0x80**)** **>>** 7**;**

// This performs the shifting, and carries over the

// bits that would have been outside of the operation

// for array[0], ANDing with 0x0F perserves the structure

// of the array.

array**[**0**]** **=** **(**array**[**0**]** **<<** shift **&** 0x0F**)** **|** temp**[**1**];**

array**[**1**]** **=** **(**array**[**1**]** **<<** shift**)** **|** temp**[**2**];**

array**[**2**]** **=** **(**array**[**2**]** **<<** shift**)** **|** temp**[**3**];**

array**[**3**]** **=** **(**array**[**3**]** **<<** shift**)** **|** temp**[**0**];**

**}**

**else** **if(**shift **==** 2**)**

**{**

// 0bxxxx xxxx.xxxx xxxx.xxxx xxxx.xxxx xxxx

// ^^ || || || <- temp[0]

// ^^ || || <- temp[1]

// ^^ || <- temp[2]

// ^^ <- temp[3]

temp**[**0**]** **=** **(**array**[**0**]** **&** 0x0C**)** **>>** 2**;**

temp**[**1**]** **=** **(**array**[**1**]** **&** 0xC0**)** **>>** 6**;**

temp**[**2**]** **=** **(**array**[**2**]** **&** 0xC0**)** **>>** 6**;**

temp**[**3**]** **=** **(**array**[**3**]** **&** 0xC0**)** **>>** 6**;**

array**[**0**]** **=** **(**array**[**0**]** **<<** shift **&** 0x0F**)** **|** temp**[**1**];**

array**[**1**]** **=** **(**array**[**1**]** **<<** shift**)** **|** temp**[**2**];**

array**[**2**]** **=** **(**array**[**2**]** **<<** shift**)** **|** temp**[**3**];**

array**[**3**]** **=** **(**array**[**3**]** **<<** shift**)** **|** temp**[**0**];**

**}**

**}**

void combine\_CD**(**uint8\_t C**[**4**],** uint8\_t D**[**4**],** uint8\_t dest**[**7**])**

**{**

// Combine the C and D array into one array.

// The tricky part here is that the first element

// of the C and D array are padded with 4 bits of 0's.

// The shifting, ORing, and ANDing take care of this

// for the C array. The D array is already aligned correctly

// once we get to the second element

dest**[**0**]** **=** **(**C**[**0**]** **<<** 4**)** **|** **((**C**[**1**]** **>>** 4**)** **&** 0x0F**);**

dest**[**1**]** **=** **(**C**[**1**]** **<<** 4**)** **|** **((**C**[**2**]** **>>** 4**)** **&** 0x0F**);**

dest**[**2**]** **=** **(**C**[**2**]** **<<** 4**)** **|** **((**C**[**3**]** **>>** 4**)** **&** 0x0F**);**

dest**[**3**]** **=** **(**C**[**3**]** **<<** 4**)** **|** **(**D**[**0**]** **&** 0x0F**);**

dest**[**4**]** **=** D**[**1**];**

dest**[**5**]** **=** D**[**2**];**

dest**[**6**]** **=** D**[**3**];**

**}**