```
/**
 *Automated Brute Force Attack
 *Designed for: S-DES
 *Developer 1: Dr. K.S. Ooi (ksooi@mailexcite.com)
 *Developer 2: Brain Chin Vito (v@chin.tc)
 *University of Sheffield Centre, Taylor's College
 */
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <string.h>
#define UINT unsigned int
#define BYTE unsigned char
#define BYTESIZE CHAR_BIT
#define BLOCKSIZE BYTESIZE
#define KEYSIZE 10
#define SUBKEYSIZE 8
#define SPLITKEYSIZE 5
BYTE ctext = 0;
BYTE E[] = { 0,1,0,0,2,3,3,2 };
BYTE P4[] = { 1,0,3,2 };
BYTE S0[] = { 1,0,2,3,3,1,0,2,2,0,3,1,1,3,2,0 };
BYTE S1[] = { 0,3,1,2,3,2,0,1,1,0,3,2,2,1,3,0 };
/* ======================== general functions ================================ */
/**
 * E.g. of usage:
 * printBin(" 23 is: ", 23, BYTESIZE);
 * printBin(" ",cbin2UINT("10110010",BYTESIZE), BYTESIZE);
 *
 * printBin(" 217 is: ", 217, BYTESIZE);
 * printf("String 11011001 is %u\n", cbin2UINT("11011001",BYTESIZE));
 *
 * printBin(" 729 is: ", 729, 10);
 * printf("String 1011011001 is %u\n", cbin2UINT("1011011001",KEYSIZE));
 *
 */
void printBin(const char *str, unsigned int bInteger, unsigned int nSize) {
        char s[BYTESIZE * sizeof(UINT)];
```

```
	UINT i;
	UINT n = bInteger;
	for (i = 0; i < nSize; i++)
		*(s + i) = '0'; *(s + i) = '\0';
	i = nSize - 1;
	while (n > 0) {
		s[i--] = (n % 2) ? '1' : '0';
		n = n / 2;
	}
	printf("%s%s [%+3u in decimal]\n", str, s, bInteger);
}
/**
 * E.g. of usage:
 * //printf("String 11011001 is %u\n", cbin2UINT("11011001",BYTESIZE));
 * //printf("String 1011011001 is %u\n", cbin2UINT("1011011001",KEYSIZE));
 */
UINT cbin2UINT(char *s, unsigned int nSize) {
	int nLen = strlen(s);
	UINT uResult = 0;
	while (--nLen >= 0)
		if (s[nLen] == '1')
			uResult = 1 << (nSize - nLen - 1) | uResult;
	return uResult;
}
/* ======================== Key Scheduling ================================== */
/**
 *
 * Only valid for max size 10
 * Shift size is two, max
 * printBin(" ",cbin2UINT("10110010",BYTESIZE), BYTESIZE);
 * printBin(" ",leftShift(bin2UINT("10110010",BYTESIZE),2,BYTESIZE), BYTESIZE);
 * printBin(" ",leftShift(bin2UINT("10110010",BYTESIZE),1,BYTESIZE), BYTESIZE);
 *
 */
UINT leftShift(UINT nKey, UINT nShift, UINT nSize) {
	UINT n = nKey >> (nSize - nShift), i, nMask = 0;
	nKey <<= nShift;
```

```
	for (i = 0; i < nSize; i++)
		nMask |= 1 << i;
	return (nKey | n) & nMask;
}
/**
 * Permutation box p10
 * printBin(" ",box_p10(cbin2UINT("1011011001",KEYSIZE)),KEYSIZE);
 */
UINT p10[] = { 9,7,3,8,0,2,6,5,1,4 };
UINT box_p10(UINT key10) {
	UINT uResult = 0, i = 0;

	for (; i < KEYSIZE; i++)
		if (1 << (KEYSIZE - p10[i] - 1) & key10)
			uResult |= 1 << (KEYSIZE - i - 1);

	return uResult;
}
/**
 * Split Key
 * printBin("",splitKey(cbin2UINT("1011011001",KEYSIZE), keyArray),5);
 * where keyArray is an array of 2 '5 bit' keys
 */
void splitKey(UINT p10Key10, UINT uResult[]) {
	/**
	* 31 == 0000011111
	* 992 == 1111100000
	*/
	UINT H_SPLIT5BIT_MASK = 31, L_SPLIT5BIT_MASK = 992;
	uResult[0] = (p10Key10 & L_SPLIT5BIT_MASK) >> SPLITKEYSIZE;
	uResult[1] = p10Key10 & H_SPLIT5BIT_MASK;
}
/**
 * Permutation box p8
 */
UINT p8[] = { 3,1,7,5,0,6,4,2 };
UINT box_p8(UINT key5[]) {
	UINT uResult = 0, uTemp, i = 0;
```

```
	/*
	* 255 = 11111111
	*/
	UINT uMask = 255;
	uTemp = key5[0] << SPLITKEYSIZE | key5[1];
	uTemp &= uMask;
	for (; i < SUBKEYSIZE; i++)
		if (1 << (KEYSIZE - p8[i] - 1) & uTemp)
			uResult |= 1 << (KEYSIZE - i - 3);
	return uResult;
}
/**
 * Key Schedule
 */
void keySchedule(UINT key10, UINT key8[]) {
	UINT key5[2] = { 0,0 }, keyTemp, i;
	keyTemp = box_p10(key10);
	splitKey(keyTemp, key5);
	for (i = 0; i < 2; i++) {
		key5[0] = leftShift(key5[0], i + 1, SPLITKEYSIZE);
		key5[1] = leftShift(key5[1], i + 1, SPLITKEYSIZE);
		key8[i] = box_p8(key5);
	}
}
/* ======================== IP and IP_1 ================================== */
UINT IP[] = { 7,6,4,0,2,5,1,3 };
UINT IP_1[] = { 3,6,4,7,2,5,1,0 };
BYTE per(UINT P[], BYTE input) {
	BYTE bRes = 00;
	int i = 8;
	while (--i >= 0)
		if (01 << (BLOCKSIZE - P[BLOCKSIZE - i - 1] - 1) & input)
			bRes |= (01 << i);
	return bRes;
}
/* ======================== Round ============================== */
void split824(BYTE bInput8, BYTE bLR[]) {
	BYTE L_mask = 240, H_mask = 15;
```

```
	/** left */
	bLR[0] = (bInput8 & L_mask) >> 4;
	/** right */
	bLR[1] = bInput8 & H_mask;
}
/**
 * f-function
 */
BYTE f(BYTE bRight, BYTE key) {
	BYTE bRes = 00, bTemp;
	BYTE sLR4[] = { 0,0 }, r, c;
	int i = SUBKEYSIZE;
	while (--i >= 0)
		if (01 << (4 - E[SUBKEYSIZE - i - 1] - 1) & bRight)
			bRes |= (01 << i);
	bRes ^= key;
	split824(bRes, sLR4);
	c = (sLR4[0] & 6) >> 1;
	r = (sLR4[0] & 8) >> 2 | (sLR4[0] & 01);
	sLR4[0] = S0[4 * r + c] << 2;
	c = (sLR4[1] & 6) >> 1;
	r = (sLR4[1] & 8) >> 2 | (sLR4[1] & 01);
	sLR4[1] = S1[4 * r + c];

	bTemp = sLR4[0] | sLR4[1];
	bRes = 00;
	// permute using P4
	i = 4;
	while (--i >= 0)
		if (01 << (4 - P4[4 - i - 1] - 1) & bTemp)
			bRes |= (01 << i);
	return bRes;
}
/**
 *Takes in a key and encrypt a fixed plaintext using that key to get the corresponding
 *ciphertext. The ciphertext is then compared to the ciphertext ctext(global variable),
 *and if it is the same, this function returns 1, otherwise this function returns a zero.
 */
```

```
int crypt(UINT currentkey, int flag) {

	UINT key8[2] = { 0,0 };
	UINT key10 = currentkey;
	BYTE input8 = (BYTE)cbin2UINT("00100000", BLOCKSIZE), exInput8, i;
	/** left and Right */
	BYTE LR[] = { 00,00 };
	/** display the input */

	keySchedule(key10, key8);
	input8 = per(IP, input8);
	// ===> Start of the round
	exInput8 = input8;
	for (i = 0; i < 2; i++) {
		/** =====> begin round */
		split824(exInput8, LR);
		input8 = (f(LR[1], (BYTE)key8[i]) ^ LR[0]) << 4;
		input8 |= LR[1];
		exInput8 = ((input8 & 240) >> 4) | ((input8 & 15) << 4);
		/** =====> end of round */
	}
	input8 = per(IP_1, input8);
	if (flag == 0) {
		ctext = input8;
		return -1;
	}
	else {
		if (input8 == ctext)
			return 1;
		else return 0;
	}
}
int main(void) {
	UINT i = 0;
	UINT targetKey = 0;
	UINT actualKey = cbin2UINT("0000100000", KEYSIZE);
	char cont, useKey;
	char userKey[] = "0000000000";
```

```
	printf("============Automated Brute Force Attack============");
	putchar('\n');
	printf(" Do you want to use the predetermined key (y/n) ?");
	scanf("%c", &useKey);
	if (useKey != 'y') {
		printf(" Please specify key to use (10 BITS) ?");
		scanf("%s", &userKey);
		actualKey = cbin2UINT(userKey, KEYSIZE);
	}

	crypt(actualKey, 0);
	while (crypt(i, 1) == 0)
	{
		i++;
		targetKey++;
	}
	printBin(" Cracked Key = ", targetKey, KEYSIZE);
	printBin(" Actual Key = ", actualKey, KEYSIZE);
	printf(" Note: Key Suggested May Differ From Actual Key Because Suggested Key Can
Also
		\n");
		printf(" Be Used To Encrypt And Decrypt With Equivalent Results \n");


	return EXIT_SUCCESS;
}
```