cs3843p2Encrypt (60 pts) - last revised 9-14-2019

**©2019 Larry W. Clark,** this must not be copied to any other website.

You will use C to encrypt and decrypt files using byte and bit manipulation. You will also use preprocessor statements to conditionally support debugging.

I have provided a driver, **cs3843p2Driver.c**. to help reduce your coding effort. It does the following

* Handles command switches.
* Reads binary data files. Note that the encrypted file will not be printable; instead, we will use your hexDump.
* Writes binary data files.
* Invokes some low level manipulation routines to help with your debugging.
* Invokes your new encrypt and decrypt functions.
* Prints a lot of output, mostly using your hexDump.

**Command switches (handled in the driver that Larry wrote):**

p2 -l This causes the driver to invoke some low level functions

to help confirm correctness. (That is an el not a one.)

p2 -e inFileNm -k key -o outFileNm

The driver will read the specified inFileNm and encrypt it

using the specified key. The output is written to the

specified outFileNm.

p2 -d encFileNm -k key -o outFileNm

This reads the specified encrypted file, decrypts it and writes

the results to the specified outFileNm.

p2 -b inFileNm -k key -o outFileNm

The driver will read the specified inFileNm and encrypt it

using the specified key. The output is written to the

specified outFileNm. It will also read that file, decrypt it, and

confirm that it is the same as the input.

You can only specify one of -l, -e, -d, -b.

**Input Data**

The specified input file is read assuming there might not be \n on each line. For the encrypted file, the data is binary data and isn't necessarily printable characters.

**You must code the following functions:**

**int hexDump(char \*psbBuffer, int iBufferLength, int iBytesPerLine)**

Your assignment #1 **hexDump** function. **Please change the first printf to NOT print the address.**

**unsigned int bitSwap(unsigned int uiOrig, int k)**

This simply swaps many bits within the four byte unsigned int beginning from the right side, returning the result as the functional value.

|  |  |  |
| --- | --- | --- |
| **Set of bits to swap** | **Positions** | **Positions with k=4** |
| First bits to swap | 0 and k | 0 and 4 |
| Second bits to swap | k+1 and k+1 + k | 5 and 9 |
| Third bits to swap | k+1+ k+1 and k+1+ k+1 + k | 10 and 14 |
| … |  |  |

Assume k is 4 in this example:

|  |  |  |
| --- | --- | --- |
| position (from right side) | 10987654 32109876 54321098 76543210 | Hex |
| bit pattern | 10110101 11011110 00011111 10101010 | B5 DE 1F AA |
| result of swapping every 4 bits | 10010111 11010110 11011011 10101010 | 97 D6 DB AA |

**unsigned int rotateLeft(unsigned int uiOrig, int k)**

This rotates bits to the left which means the bits that were shifted off are then moved to the right side. The rotated value is returned as the function value.

Assume k is 6 in this example:

|  |  |  |
| --- | --- | --- |
| position (from right side) | 10987654 32109876 54321098 76543210 | Hex |
| bit pattern | 10110101 11011110 00011111 10101010 | B5 DE 1F AA |
| result of rotation | 01110111 10000111 11101010 10101101 | 77 87 EA AD |

**void charSwap(char \*psbBuf, int k, int iBuffLgth)**

This swaps every k characters beginning from the left side, modifying the parameter. This is not restricted to four bytes.

Assume k is 4 in this example:

|  |  |
| --- | --- |
| position (from left side) | 01234567 89012345 67890123 4567890 |
| char buffer | abcdefgh ijklmnop qrstuvwz yz |
| result of char swap | ebcdajgh ifolmnkt qrspyvwz uz |

**int encrypt(char \*psbInBuffer, int iInBufferLgth, char \*pszKey, char \*psbOutBuffer**

**, int iOutMaxBufferLgth)**

Using the specified encryption key, this encrypts a buffer and returns it in another buffer.

Parameters:

I char \*psbInBuffer Input buffer to be encrypted. This might not be a zero

terminated string.

I int iInBufferLgth Number of bytes in psbInBuffer

I char \*pszKey A zero terminated string which is the encryption key.

O char \*psbOutBuffer The encrypted result is returned via this parameter

which has a maximum of iOutMaxBufferLgth bytes.

I int iOutMaxBufferLgth The maximum bytes allowed in psbOutBuffer

Notes:

1. You must place the following information in the output buffer (in this order):
   * encryption key
   * 4 byte input buffer length (as an **int** not as printed characters)
   * input buffer
   * padding (see below)
2. How big is the output buffer (before padding)? strlen(pszKey)+4+iInBufferLgth
3. If that total length from #2 is not a multiple of 4, pad it with zero bytes and adjust the total length to make it a multiple of 4. Why are we doing that?
4. You must sum the characters in the pszKey using the driver's sumKey. This sum is used below. We will call it SUM in this discussion.
5. In the output buffer, swap the characters based on k = (SUM % (64 - keyLength) )+ 1. You will call **charSwap** once.
6. In the output buffer, perform a left rotation in every four bytes using k = SUM % 17 + 1. You will call **rotateLeft** many times.
7. In the output buffer, perform a four byte bit swap in every 4 bytes using k = SUM % 7 + 1. You will call **bitSwap** many times.
8. Functionally return **one** of the following:
   * ERR\_ENCRYPT\_KEY\_SIZE
   * ERR\_INPUT\_TOO\_LARGE
   * total length of output buffer including key, buffer length, input buffer, and padding

**int decrypt(char \*psbInBuffer, int iInBufferLgth, char \*pszKey, char \*psbOutBuffer**

**, int iOutMaxBufferLgth)**

Using the specified encryption key, this decrypts a buffer and returns it in another buffer.

Parameters:

I char \*psbInBuffer Input buffer to be decrypted. This might not be a zero

terminated string. It will contain binary data.

I int iInBufferLgth Number of bytes in psbInBuffer

I char \*pszKey A zero terminated string which is the encryption key.

O char \*psbOutBuffer The decrypted result is returned via this parameter

which has a maximum of iOutMaxBufferLgth bytes.

I int iOutMaxBufferLgth The maximum bytes allowed in psbOutBuffer

Notes:

1. You must copy the input buffer to the output buffer. Later we will remove some information from it.
2. You must sum the characters in the pszKey using the driver's sumKey. This sum is used when reversing the encryption.
3. Reverse what we did with encrypt.
4. Check whether pszKey and the embedded key match. If not, return ERR\_KEYS\_DO\_NOT\_MATCH.
5. Use a typecast to copy out the original buffer length.
6. To remove the key and the original buffer length from the output buffer, use **memmove** and shift the data to the left.
7. Functionally return one of the following:
   * ERR\_ENCRYPT\_KEY\_SIZE
   * ERR\_INPUT\_TOO\_LARGE
   * ERR\_KEYS\_DO\_NOT\_MATCH
   * length of output buffer not including key, 4 byte buffer length, and padding

**C Information**

1. You will need to use the bit manipulation from our course notes. This includes
   * How to turn a bit position ON
   * How to turn a bit position OFF
   * How to access a particular bit position
   * How to shift bits
2. Since you will be working with binary data, you will **not** be using: strcpy, strcmp. Instead, you will use **memcpy**, **memcmp**, and probably **memmove**. Recall that memmove is better to use if your source and target overlap in memory.
3. For bit swapping and bit rotation, you will need to go through the entire buffer, grabbing four bytes at a time:
   * Assume you have declared char \*pCh and it will be used to traverse through your binary buffer.
   * To grab four bytes at pCH and convert to an unsigned int:

unsigned int uiValue = \*((unsigned int \*)pCh);

1. For a left rotational shift, you will have to do two shifts:

* Copy the four byte unsigned int so that you can get the portion that falls off.
* To shift left: uiValue << kShiftBits (that didn't assign it)
* To get the first kShiftBits so that we can place them on the right side, we realize that four bytes is 32 bits. If we shifted 4 bits to the left, we must shift the original value 28 bits to the right (32-4).

**Notes**

1. Your code must be written according to my programming standards.
2. Your code must include debug output which is controlled using preprocessor statements. You need to be able to turn off your debug output using a #DEFINE. The debug output should match my sample below.
3. I have provided input files, an include file, and a driver.
4. **Any use of code from another web site will result in a 0 on this assignment, may result in an F for this course, and may cause you to be expelled from UTSA.**
5. Turn in a zip file named *abc123.*zip containing (do not add a folder around these)
   * cs3843p2.c (which also contains your hexDump function, please keep the preprocessor debug output turned ON)
   * p2OneOut.txt – encrypt stdout for p2One.txt **without the debugging** output use **-e p2One.txt -k onekey -o p2One.enc > p2OneOut.txt**
   * Assuming you **turned ON the #define for debug output,** you can run all these using the **runDebug.bash** file that I provided.
   * p2LowOut.txt – output for the **-l** switch
   * p2OneOutDebug.txt – encrypt stdout for p2One.txt **with the debugging** output use **-e p2One.txt -k onekey -o p2One.enc > p2OneOutDebug.txt**
   * p2OneOutDecDebug.txt – decrypt stdout for p2One.enc **with the debugging** output, use **-d p2One.enc -k onekey -o p2One.dec > p2OneOutDecDebug.txt**
   * p2OneOutBad.txt – decrypt stdout for p2One.enc **with the debugging** output, use **-d p2One.enc -k xyz -o p2One.bad > p2OneOutBad.txt**
   * p2TwoOutDebug.txt – encrpyt output for p2Two.txt **with the debugging** text use **-e p2Two.txt -k twokey -o p2Two.enc > p2TwoOutDebug.txt**
   * p2TwoOutDecDebug.txt – decrypt stdout for p2Two.enc **with the debugging** output, use **-d p2Two.enc -k twokey -o p2Two.dec > p2TwoOutDecDebug.txt**

**Some Sample Output for the cases shown**

**$ ./p2 –l (that is an el not a one)**

PROGRAM 2 Output:

\*\*\* Low Level Test

>>> Testing character swap of every 4th character (0<->4, 5<->9, 10<->14, ...)

origStr="abcdefghijklmnopqrstuvwzyz Computer Science is the best major in the world. We have fun! "

resStr="ebcdajghifolmnktqrspyvwzum CozrutepeSci ice ne thstbes omaj inrwhe t.rldo We aveh un!f" CORRECT

>>> Testing six bit left rotate of bits (not entire buffer)

Orig 4 bytes=B5DE1FAA

Res 4 bytes=7787EAAD CORRECT

>>> Testing bit swap of every 4 bits in four bytes (not entire buffer)

Orig 4 bytes=B5DE1FAA

Res 4 bytes=97D6DBAA CORRECT

**$ ./p2 -e p2One.txt -k onekey -o p2One.enc (without debug)**

PROGRAM 2 Output:

\*\*\* Encryption Only

>>> Hexdump of Original Data, 77 bytes

hexDump of 77 bytes

0000 I f y o u w o u l d l i k e m o r e i n f o r m a t i o n a b o u t

49662079 6F752077 6F756C64 206C696B 65206D6F 72652069 6E666F72 6D617469 6F6E2061 626F7574

0028 e n c r y p t i o n , t a k e C S 4 3 6 3 C r y p t o g r a p h y

20656E63 72797074 696F6E2C 2074616B 65204353 34333633 20437279 70746F67 72617068 79

>>> Hexdump of Encrypted Data, 88 bytes

hexDump of 88 bytes

0000 . g . . . . . # & . . a ' . . . . . . . ' ' & . . . . & . . . . & . e g . g . .

E567A7E6 80A6AD23 26008061 2784EDE7 2EC4EEA7 27272604 84E5A526 A6A72E84 26846567 E467ACA7

0028 ' . . g g . g . . . . . m . d . $ , . . . . . . . . . & . . ( . ` . L . n . - ,

27AEE567 67EC67E4 A62E2E84 6DE764AC 242CAEA5 2E040704 84E4A526 CCE028CE 60CE4CE7 6EAC2D2C

0050 . . f . . , . -

E4E766AC 002CA52D

**$ ./p2 -e p2One.txt -k onekey -o p2One.enc (with debug)**

PROGRAM 2 Output:

\*\*\* Encryption Only

>>> Hexdump of Original Data, 77 bytes

hexDump of 77 bytes

0000 I f y o u w o u l d l i k e m o r e i n f o r m a t i o n a b o u t

49662079 6F752077 6F756C64 206C696B 65206D6F 72652069 6E666F72 6D617469 6F6E2061 626F7574

0028 e n c r y p t i o n , t a k e C S 4 3 6 3 C r y p t o g r a p h y

20656E63 72797074 696F6E2C 2074616B 65204353 34333633 20437279 70746F67 72617068 79

padded with 1 bytes for a total of 88 bytes

hexDump of 88 bytes

0000 o n e k e y M . . . I f y o u w o u l d l i k e m o r e i n f o r m a

6F6E656B 65794D00 00004966 20796F75 20776F75 6C64206C 696B6520 6D6F7265 20696E66 6F726D61

0028 t i o n a b o u t e n c r y p t i o n , t a k e C S 4 3 6 3 C r y p t

74696F6E 2061626F 75742065 6E637279 7074696F 6E2C2074 616B6520 43533433 36332043 72797074

0050 o g r a p h y .

6F677261 70687900

swapping every 14 bytes, size = 88 bytes

hexDump of 88 bytes

0000 o n e k e y M . . . I f y o o w o u l d l i k e m u e i n f o r m a

6F6E656B 65794D00 00004966 20796F6F 20776F75 6C64206C 696B6520 6D752065 20696E66 6F726D61

0028 t i o n r o b o u t e n c r y p t i a , t a k e C S 4 3 6 3 n C r y p t

74696F6E 726F626F 75742065 6E637279 70746961 202C2074 616B6520 43533433 36336E43 72797074

0050 o g r a p h y .

6F677261 70687900

left rotational shift by 6 bits, size = 88 bytes

hexDump of 88 bytes

0000 . . [ . @ Y ^ . . . @ . . H . . . . . [ . . . . H . Z . Y [ . H . H . . . . \ [

DA9B5BD9 40595E13 19004092 1B48DEDB 1DC8DD5B 1B1B1908 48DA5A19 595B1D48 19489A9B D89B5C5B

0028 . ] . . . . . . Y . . H . . . \ . . ] Z . . . . H . Z . . . . . . . . . . \ . .

1B5DDA9B 9BDC9BD8 591D1D48 9EDB985C 181C5D5A 1D080B08 48D85A19 CCD014CD 90CD8CDB 9D5C1E1C

0050 . . . \ . . Z .

D8DB995C 001C5A1E

swap every 1 bits, size = 88 bytes

hexDump of 88 bytes

0000 . g . . . . . # & . . a ' . . . . . . . ' ' & . . . . & . . . . & . e g . g . .

E567A7E6 80A6AD23 26008061 2784EDE7 2EC4EEA7 27272604 84E5A526 A6A72E84 26846567 E467ACA7

0028 ' . . g g . g . . . . . m . d . $ , . . . . . . . . . & . . ( . ` . L . n . - ,

27AEE567 67EC67E4 A62E2E84 6DE764AC 242CAEA5 2E040704 84E4A526 CCE028CE 60CE4CE7 6EAC2D2C

0050 . . f . . , . -

E4E766AC 002CA52D

>>> Hexdump of Encrypted Data, 88 bytes

hexDump of 88 bytes

0000 . g . . . . . # & . . a ' . . . . . . . ' ' & . . . . & . . . . & . e g . g . .

E567A7E6 80A6AD23 26008061 2784EDE7 2EC4EEA7 27272604 84E5A526 A6A72E84 26846567 E467ACA7

0028 ' . . g g . g . . . . . m . d . $ , . . . . . . . . . & . . ( . ` . L . n . - ,

27AEE567 67EC67E4 A62E2E84 6DE764AC 242CAEA5 2E040704 84E4A526 CCE028CE 60CE4CE7 6EAC2D2C

0050 . . f . . , . -

E4E766AC 002CA52D

**$ ./p2 -d p2One.enc -k onekey -o p2One.dec (with debug)**

PROGRAM 2 Output:

\*\*\* Decryption Only

>>> Hexdump of Encrypted Data, 88 bytes

hexDump of 88 bytes

0000 . g . . . . . # & . . a ' . . . . . . . ' ' & . . . . & . . . . & . e g . g . .

E567A7E6 80A6AD23 26008061 2784EDE7 2EC4EEA7 27272604 84E5A526 A6A72E84 26846567 E467ACA7

0028 ' . . g g . g . . . . . m . d . $ , . . . . . . . . . & . . ( . ` . L . n . - ,

27AEE567 67EC67E4 A62E2E84 6DE764AC 242CAEA5 2E040704 84E4A526 CCE028CE 60CE4CE7 6EAC2D2C

0050 . . f . . , . -

E4E766AC 002CA52D

swap every 1 bits, size=88 bytes

hexDump of 88 bytes

0000 . . [ . @ Y ^ . . . @ . . H . . . . . [ . . . . H . Z . Y [ . H . H . . . . \ [

DA9B5BD9 40595E13 19004092 1B48DEDB 1DC8DD5B 1B1B1908 48DA5A19 595B1D48 19489A9B D89B5C5B

0028 . ] . . . . . . Y . . H . . . \ . . ] Z . . . . H . Z . . . . . . . . . . \ . .

1B5DDA9B 9BDC9BD8 591D1D48 9EDB985C 181C5D5A 1D080B08 48D85A19 CCD014CD 90CD8CDB 9D5C1E1C

0050 . . . \ . . Z .

D8DB995C 001C5A1E

right rotational shift by 6 bits, size=88 bytes

hexDump of 88 bytes

0000 o n e k e y M . . . I f y o o w o u l d l i k e m u e i n f o r m a

6F6E656B 65794D00 00004966 20796F6F 20776F75 6C64206C 696B6520 6D752065 20696E66 6F726D61

0028 t i o n r o b o u t e n c r y p t i a , t a k e C S 4 3 6 3 n C r y p t

74696F6E 726F626F 75742065 6E637279 70746961 202C2074 616B6520 43533433 36336E43 72797074

0050 o g r a p h y .

6F677261 70687900

swapping every 14 bytes, size=88 bytes

hexDump of 88 bytes

0000 o n e k e y M . . . I f y o u w o u l d l i k e m o r e i n f o r m a

6F6E656B 65794D00 00004966 20796F75 20776F75 6C64206C 696B6520 6D6F7265 20696E66 6F726D61

0028 t i o n a b o u t e n c r y p t i o n , t a k e C S 4 3 6 3 C r y p t

74696F6E 2061626F 75742065 6E637279 7074696F 6E2C2074 616B6520 43533433 36332043 72797074

0050 o g r a p h y .

6F677261 70687900

Keys match!!!

>>> Hexdump of Decrypted Data, 77 bytes

hexDump of 77 bytes

0000 I f y o u w o u l d l i k e m o r e i n f o r m a t i o n a b o u t

49662079 6F752077 6F756C64 206C696B 65206D6F 72652069 6E666F72 6D617469 6F6E2061 626F7574

0028 e n c r y p t i o n , t a k e C S 4 3 6 3 C r y p t o g r a p h y

20656E63 72797074 696F6E2C 2074616B 65204353 34333633 20437279 70746F67 72617068 79

**$ ./p2 -d p2One.enc -k xyz -o p2One.dec (with debugging)**

PROGRAM 2 Output:

\*\*\* Decryption Only

>>> Hexdump of Encrypted Data, 88 bytes

hexDump of 88 bytes

0000 . g . . . . . # & . . a ' . . . . . . . ' ' & . . . . & . . . . & . e g . g . .

E567A7E6 80A6AD23 26008061 2784EDE7 2EC4EEA7 27272604 84E5A526 A6A72E84 26846567 E467ACA7

0028 ' . . g g . g . . . . . m . d . $ , . . . . . . . . . & . . ( . ` . L . n . - ,

27AEE567 67EC67E4 A62E2E84 6DE764AC 242CAEA5 2E040704 84E4A526 CCE028CE 60CE4CE7 6EAC2D2C

0050 . . f . . , . -

E4E766AC 002CA52D

swap every 7 bits, size=88 bytes

hexDump of 88 bytes

0000 . . . g . ' . . & . . . . . . . . E o . . . & . . . . & ' . . . & . . . e . - .

E5E6A767 0127ADA2 260001E0 A605EDE7 2E456FA7 A6A62604 05E5A526 27A72E05 2605E4E6 65E62DA7

0028 . / . . . m . e ' . . . . . d - $ , / . . . . . . e . & M a ( O ` O L . n - . ,

A62FE5E6 E66DE665 272E2E05 ECE7642D 242C2FA5 2E048604 0565A526 4D61284F 604F4CE7 6E2DAC2C

0050 e . f - . , . .

65E7662D 002CA5AC

right rotational shift by 7 bits, size=88 bytes

hexDump of 88 bytes

0000 . O . . N Z E . . . . M . . . M . . N ] M M . L . K M . N ] . N . . . M . [ N .

CD4FCFCA 4E5A4503 0002C04D 0BDACF4D 8ADE4E5D 4D4D084C CA4B4D0A 4E5D0A4E 0AC8CD4D CC5B4ECB

0028 \_ . . M . . . . \ \ . N . . Z . X ^ J I . . . \ . J M . . P . . . . . . Z X Y .

5FCACD4D DBCCCBCC 5C5C0A4E CFC95AD8 585E4A49 080C095C CA4A4D0A C2509E9A 9E98CEC1 5A5859DC

0050 . . Z . X J Y .

CECD5ACA 584A5901

swapping every 59 bytes, size=88 bytes

hexDump of 88 bytes

0000 I O . . N Z E . . . . M . . . M . . N ] M M . L . K M . N ] . N . . . M . [ N .

494FCFCA 4E5A4503 0002C04D 0BDACF4D 8ADE4E5D 4D4D084C CA4B4D0A 4E5D0A4E 0AC8CD4D CC5B4ECB

0028 \_ . . M . . . . \ \ . N . . Z . X ^ J . . . . \ . J M . . P . . . . . . Z X Y .

5FCACD4D DBCCCBCC 5C5C0A4E CFC95AD8 585E4ACD 080C095C CA4A4D0A C2509E9A 9E98CEC1 5A5859DC

0050 . . Z . X J Y .

CECD5ACA 584A5901

keys do not match

ERROR: decrypt returned: decrypted key doesn't match encrypt key