CS 427/527 - Computer Security
Spring 2019
Program 1
Due Friday, February 15

200 points

Write a program for the block-encryption algorithm called **WSU-CRYPT** (based on Twofish by Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall and SKIPJACK by the NSA combined). WSU-CRYPT is described below. Implement **WSU-CRYPT** using a 64 bit block size and a 64 bit key. Graduate students are required to implement a 64 bit block size with an 80 bit key.

The majority of the code MUST be your own work. Code that is not yours (for example from a book) must be clearly documented. You may use any standard programming language that the grader will not have difficulty compiling and testing on the WSU lab machines. Give this some thought since you are manipulating blocks of bits. Make sure your language supports the operations easily. Please test your program on the lab machines or contact the grader to determine if your platform is compatible with his. The grader will deduct points if he can not easily compile and run your code. If you have questions about your choice of language, then also contact the grader.

Input and Output
As input your program should take a standard ASCII text file (to standardize grading, call the file **plaintext.txt**) and a randomly chosen secret key represented in HEX (put this in a file called **key.txt**). As output your program should return a HEX text file (call the file **cyphertext.txt**) which is the encryption of the input file under **WSU-CRYPT**. This is of course reversed for decryption.

Overall Structure of WSU-CRYPT
WSU-CRYPT is a block encryption algorithm where the block size is 64 bits. The input file is encrypted 64 bits at a time. Each 64 bit input block is divided into 4 words, w_0, w_1, w_2, w_3 (16 bits each).

To Encrypt:
- Input whitening step:
    - The input is the 64 bit block divided into 4 words w_0, w_1, w_2, w_3.
    - XOR each word with 16 bits of the key $K_0 K_1 K_2 K_3$.
    - The output is
        - $R_0 = w_0 \oplus K_0$, $R_1 = w_1 \oplus K_1$,
        - $R_2 = w_2 \oplus K_2$, $R_3 = w_3 \oplus K_3$.
- Set the round number to **round=0**. After each of the 16 rounds increment **round** by one.
- Compute **F(R_0, R_1, round)**. This function returns two values F_0 and F_1.

- o Compute $R_1 \oplus F_0$ and rotate this value right by 1 bit. This value is R_1' for the next round.
- o Rotate R_2 left by 1 bit to get R_2' and compute $R_2' \oplus F_1$. This value is R_3' for the next round.
- o The value R_0 becomes R_2 for the next round. The value R_1 becomes R_3 for the next round.
- o Repeat for 16 rounds (but not the whitening step this only happens at the start of the first round).

- After the last of the 16 rounds:
  - o Undo the last swap by setting
    y_0 to R_2 from the end of the 16th round;
    y_1 to R_3 from the end of the 16th round;
    y_2 to R_0 from the end of the 16th round and
    y_3 to R_1 from the end of the 16th round.

- Output whitening step:
  - o XOR each word with 16 bits of the key $K_0 K_1 K_2 K_3$.
    The input is y_0, y_1, y_2, y_3. The output is
    $C_0 = y_0 \oplus K_0$, $C_1 = y_1 \oplus K_1$,
    $C_2 = y_2 \oplus K_2$, $C_3 = y_3 \oplus K_3$.
    The value $C_0 C_1 C_2 C_3$ is the ciphertext.

To Decrypt:

The algorithm to decrypt is the same as the algorithm to encrypt except that:
  - o The keys are generated and used in reverse order (see the sections below on keys).
  - o Instead of: computing $R_1 \oplus F_0$ and then rotating this value right by 1 bit you must rotate R_2 left by 1 bit to get R_2' and compute $R_2' \oplus F_0$.
  - o Instead of: rotating R_2 left by 1 bit to get R_2' and then computing $R_2' \oplus F_1$ you must compute $R_1 \oplus F_1$ and then rotate this value right by 1 bit. These rotates are a small change that makes this a modified Feistel cipher.

Subroutines

The program can be broken into several logical subroutines. Function **F( )**, the **G-permutation** and the **Key Schedule** are explained in terms of their inputs and outputs.

Functions F(R0, R1, round)

  - o To Encrypt:
    - o Input: R_0, R_1 and the **round** number (or the round number may be a global variable).
    - o Note: In every call to **F()** a total of 12 subkeys are used, 4 in each call to **G()** and 4 directly in **F()**. You may want to compute all 12 subkeys at the start of **F()** since you will need them generated in reverse order for decryption. Then pass keys to **G()**.
  - Compute **T0 = G(R0, round)**
  - Compute **T1 = G(R1, round)**

Compute **F$_0$= (T$_0$+ 2·T$_1$+ concatenate(K(4·round) K(4·round+1))) mod 2**

Compute **F$_1$= (2·T$_0$+ T$_1$+ concatenate(K(4·round+2) K(4·round+3))) mod 2**

Note: **concatenation(x, y)** is just the concatenation of two bytes. **K(x)** is a call to the key scheduler to get a key based on value x.

Output: **F$_0$, F$_1$**.

- To Decrypt: The **F()** function is not run in reverse for decryption. Only the subkeys are used in reverse order. It may be easier to understand if, at the entry to the **F()** function, you generate all the subkeys for that call and then use them. This implies passing keys to the **G()** function as arguments.
- For encryption the order of construction and use of subkeys is:

  GK$_0$ = K(4·round)   first call to **G()**

  GK$_1$= K(4·round+1)

  GK$_2$= K(4·round+2)

  GK$_3$= K(4·round+3)

  GK$_4$= K(4·round)   second call to **G()**

  GK$_5$= K(4·round+1)

  GK$_6$= K(4·round+2)

  GK$_7$= K(4·round+3)

  K$_9$ = K(4·round)   for **F$_0$** and **F$_1$**

  K$_{10}$= K(4·round+1)

  K$_{11}$= K(4·round+2)

  K$_{12}$= K(4·round+3)

- For decryption the subkeys are used as the reverse list:

  K$_{12}$= K(4·round+3)   used first

  K$_{11}$= K(4·round+2)   used second

  K$_{10}$= K(4·round+1)   used third, etc.

  K$_9$ = K(4·round)

  GK$_7$= K(4·round+3)

  GK$_6$= K(4·round+2) etc., until **GK$_0$ = K(4·round)**

  For decryption, the algorithms for **F()** and **G()** are not changed only the order of the subkeys.

G-permutation G(w, round)

The input is 16 bits, **w** and the **round** number (and maybe the subkeys). A fixed table called, the **F-table,** is used to perform a substitution (see last page).

- To Encrypt and Decrypt: Algorithm **G**

  Input**: w, round:**

  Let **g$_0$** be the left (high) 8 bits of **w**.

  Let **g$_1$** be the right (low) 8 bits of **w**.

  g$_2$ = Ftable(g$_1$ ⊕ K(4·round)) ⊕ g$_0$

  g$_3$ = Ftable(g$_2$ ⊕ K(4·round+1)) ⊕ g$_1$

  g$_4$ = Ftable(g$_3$ ⊕ K(4·round+2)) ⊕ g$_2$

  g$_5$ = Ftable(g$_4$ ⊕ K(4·round+3)) ⊕ g$_3$

  Output: **g$_4$ g$_5$** (the concatenation).

The Key Schedule K(x)

- o Encryption:

  Input: A number **x** and it is assumed that **K()** has access to the current, stored, 64 bit key **K**. The key is 64 bits and so 8 bytes long. Label the bytes 0 through 7, so $K = k_7, k_6, k_5, k_4, k_3, k_2, k_1, k_0$. Left circular rotate **K** by 1 bit and store this rotated value as the new key **K'**.

  Output: The **x mod 8** byte of the key **K'**. So if **x = 18** then since **18 mod 8 = 2** output the third byte (k_2) of **K**. This allows us to output the first byte when, for example **x = 16** and **16 mod 8 = 0**, so the output byte is k_0

- o Decryption:

  Input: A number **x** and it is assumed that **K()** has access to the current, stored, 64 bit key **K**. The key is 64 bits and so 8 bytes long. Label the bytes 0 through 7 as for encryption. Now **z** is the **x mod 8** byte of the key **K**. Right rotate **K** by 1 bit and store this value as the new key **K'**. Unlike encryption, the subkey is gotten before the rotation of the key.

  Output: z.

More on Keys

For the keys: The **F()** function is called with **R** only
- o The **F()** function calls the **G()** function twice
    - o Each call to **G()** call **K()** 4 times   8 keys
    - o The **F()** function then directly call **K()** 4 times 4 keys
    - o So 1 call to the **F()** functions uses 12 subkeys keys

There are 16 calls to the **F()** function so 16·12 = 192 subkeys are generated from the initial secret key **K**. Each time **K()** is called (and a subkey is generated) the key **K** is left rotated by 1 bit. Since the initial secret key **K** is 64 bits long and 64·3 = 192 then after 16 rounds the final key (after all the rotations) should be same as the initial secret key. This is important for decryption. Also note that the input and output whitening steps do not shift the key bits. It is probably easiest to generate ALL keys at the beginning of the encryption or decryption.

When you are testing your program, if you use a smaller number of rounds (say 2), then you must save the final value of the key (since it will not be rotated enough to be the same as the original secret key) and then use that to decrypt.

The generation of subkeys for decryption is different then that for encryption.
- o Key Generation for Encryption is:
    - § Left rotate
    - § Pick out the key bits
- o Key Generation for Decryption is:
    - § Pick out the key bits
    - § Right rotate

F-table

The SKIPJACK F-table is in hexadecimal. The high order 4 bits of input index the row and the low order 4 bits index the column. Rows and columns are indexed starting at 0. For example **F(7a) = d6** since the 7 row and the 10 column (HEX for 10 is **a**) contains **d6**. The F-table values:

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

Graduate Students

Graduate students are required implement 64-bit block and 80-bit key sizes. This will then mean that you cipher will need to do 20 rounds so that the key is completely rotated 3 times.

Graduate students are also required to make their programs interoperate. Test your programs to see if they will interoperate by sending each other encrypted messages. It is also advisable for you, as a group, to use/build test vectors for the different subroutines so that you can jointly check your progress.

Submitting your solution

Turn in electronically to the grader: All of your source code, documentation, etc. archived in single compressed file. When you send in your code include the course number / project name in the subject line, and the grader will reply to you with confirmation. Also include a text file named **README** that includes

1. Your name and an email address you can be contacted at.
2. A brief description of what you are submitting.
3. A precise description of how to build and use your program on the WSU system. The grader will deduct points if he needs to communicate with you in order to test your code.
4. A list of files that should be in the archive, and a one line description of each file.