CPS 472/572: Programming Assignment #1

100 pts, two weeks

No submission will be accepted after the deadline

Receive an ***F*** for this course if any academic dishonesty occurs

Receive 5 bonus points if submit it without errors at least one day before deadline

1. Purpose

This homework builds an understanding of classic block ciphers and cryptanalytic attacks.

2. Description

2.1. TEA

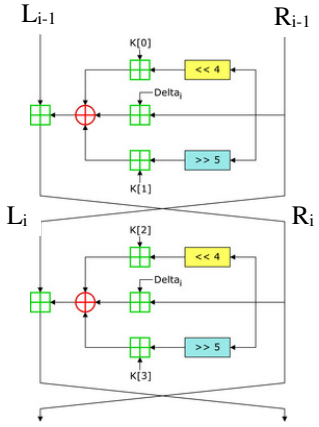
The Tiny Encryption Algorithm (TEA) block cipher operates on 64-bit blocks of plaintext using a 128-bit key. The plaintext is divided into two 32-bit blocks (L_0, R_0), and the key is divided into four 32-bit blocks (K_0, K_1, K_2, K_3). Encryption involves repeated application of two Feistel rounds (making up one cycle of TEA), defined as follows for round i and i+1:

$L_i = R_{i-1}$

$R_i = L_{i-1} \boxplus \mathbf{F}(R_{i-1}, K_0, K_1, \delta_i)$;

$L_{i+1} = R_i$

$R_{i+1} = L_i \boxplus \mathbf{F}(R_i, K_2, K_3, \delta_{i+1})$;



where $\boxplus$ denotes modulus +, the logical left shift of x by y bits is denoted by x << y, the logical right shift of x by y bits is denoted by x >> y, δ_i is a sequence of predetermined constants, and **F** is defined as

$\mathbf{F}(X, K_j, K_k, \delta_i) = ((X << 4) \boxplus K_j) \oplus ((X >> 5) \boxplus K_k) \oplus (X + \delta_i)$.

The encryption function is given next, written in C, for encoding with key k[0] … k[3]. Data is in v[0] and v[1] and there are 32 cycles (i.e., 64 rounds).

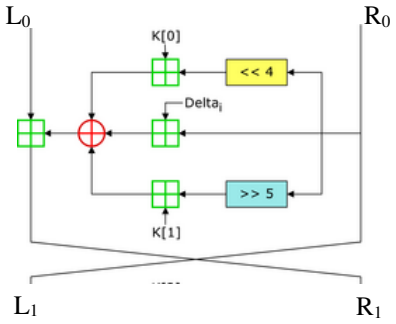
```
void code(unsigned long* v, unsigned long* k) {
   unsigned long y=v[0],z=v[1], sum=0,   /* set up */
    delta=0x9e3779b9,   /* a key schedule constant */
    n=32 ;
   while (n-->0) {            /* basic cycle start */
     sum += delta ;
       y += ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
       z += ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;
                 }    /* end cycle */
   v[0]=y ; v[1]=z ; }
```

2.2. Attack Method

In this project, we will develop a **C++ program** that performs an attack on 1 round of TEA. In 1 round of TEA the **key size** is effectively reduced to **64 bits**, because only half of the 128 bit key is used in 1 round of TEA. Using known plaintexts and their resulting ciphertexts, we will perform a **brute force** attack on the 64-bit key by repeatedly **guessing 32 bits of the key**, which will eventually lead us to deduce the other 32 bits.

We only need to search through 2^{32} possible keys, since given a subkey K_0, we can calculate a value for the other subkey K_1 by using known plaintext/ciphertex pairs (i.e., **derive an equation for K_1 in terms of K_0 and a pair of plaintext/ciphertext**).

In this project, first implement the TEA encryption method and then generate 100 random plaintexts and their corresponding ciphertexts using **1 round** of TEA encryption (see Figure below). That is, each plaintext is $<L_0, R_0>$ and its corresponding ciphertext is simply $<L_1, R_1>$. Output the plaintext/ciphertext pairs to a file.



The following steps/pseudocode outline the general procedure of the attack program.

1. Read in the file with the plaintext/ciphertext pairs into lists. These values are then converted from strings in the file to 32 bit unsigned integers.
2. Guess a value for subkey K_0, starting at 0.
3. Calculate the value for K_1 using our guess K_0 and the **first** plaintext/ciphertext pair.
4. Calculate the value for K_1 using our guess K_0 and the **second** plaintext/ciphertext pair.
5. Do those two values of K_1 match?

    6. If not, this is the incorrect guess for K_0. Increment our guess and go back to Step 3.
    6. If yes, we need to verify that this guess for subkey K_0 is correct
        7. Repeat steps 3 and 4 ten times with different plaintext/ciphertext pairs
        8. If the values of K_1 did not match every time, increment our guess for K_0 and go back to Step 3.

        9. If the values of K_1 matched every time, this is the correct guess for K_0 and we've found the key!! Print the key and the run time of your project.

2.3. Submission

- Print **two** sample runs of the program.
- Zip your entire project, sample runs, and a README file.
- Submit the zip to **isidore.udayton.edu**.