Homework 2: A Programming Project on a Symmetric Encryption Algorithm - TEA
Due Date*: 12:00pm (noon) 09/02/2019, Cutoff Date*: 12:00pm (noon) 09/04/2019
***Late penalty will apply for past-due late submission, **Submission will NOT be accepted after the cutoff deadline**
Submission: 1) TWO .java files on Blackboard, and 2) (see Task II) two java programs and two .txt files on cs3750a

An option of peer programming: You may choose to work on this assignment individually or in a team of two students. If you choose to work in a ***team of two students***, you **must** 1) **add** both team members' first and last names as the **comments** on Blackboard when submitting the .java files (both team members are required to submit the same .java files on Blackboard), and 2) put a ***team.txt*** file including both team members' names under your HW2/ on cs3750a (both team members are required to complete Task II under both home directories on cs3750a). For grading, I will randomly pick the submission in one of the two *home directories* of the team members on cs3750a, and then give both team members the same grade. If the team information is *missing* on Blackboard or cs3750a, two or more submissions with similar source codes with just variable name/comment changes will be considered to be a violation of the Integrity described in the course policies and both or all will be graded as 0.

Grading: Your programs will be graded via testing and points are associated with how much task it can complete. A program that cannot be compiled or crashes while running will receive up to 5% of the total points. A submission of .java files that are similar to any online Java program with only variable name changes will receive 0% of the total points.

*Other programming languages such as Python may be used but must be pre-approved by the instructor.

Task I (90%): Write two Java* programs, one for encryption and the other for decryption, to implement TEA

(modified from Textbook): Perhaps the simplest "serious" symmetric block encryption algorithm is the Tiny Encryption Algorithm (TEA). TEA 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)**. As shown in the diagram, encryption involves repeated application of a pair of rounds, defined as follows for rounds i and $i + 1$ (i starts with 1):

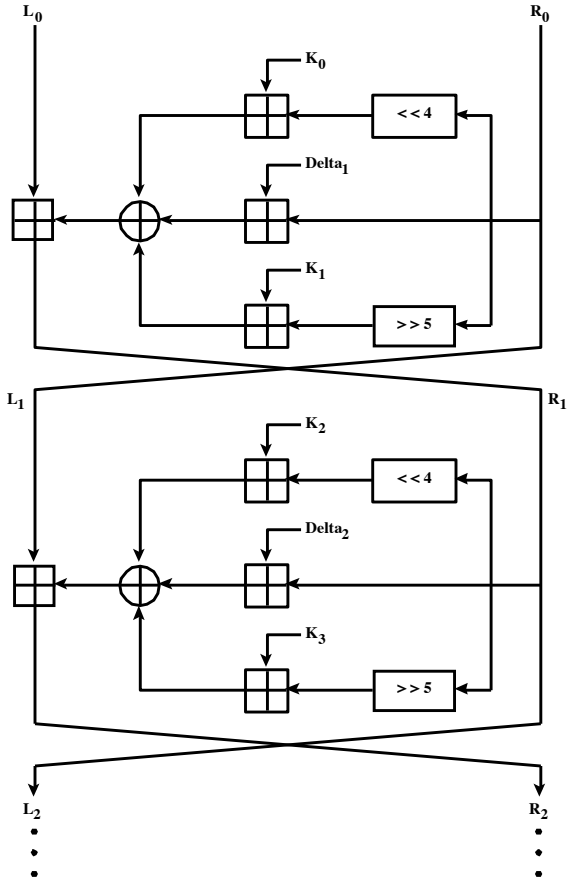
$L_i = R_{i-1}$ $\quad R_i = L_{i-1} \boxplus F(R_{i-1}, K_0, K_1, \delta_i)$

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

where F is defined as

$F(X, K_m, K_n, \delta_y) = ((X << 4) \boxplus K_m) \oplus ((X >> 5) \boxplus K_n) \oplus (X \boxplus \delta_y)$

and where 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 (Delta$_i$) is a sequence of predetermined constants; and $\boxplus$ denotes addition-mod-2^{32}.



a. If only one ***pair*** of rounds, i.e., rounds 1 and 2, is used, then ***the ciphertext is the 64- bit block (L_2, R_2).*** You may express the ***encryption*** algorithm by representing L_2 as a function of L_0, R_0, K_0, K_1, and δ_1, and representing R_2 as a function of L_2, R_0, K_2, K_3, and δ_2.

b. The ***decryption*** algorithm is given as below. You may verify it by reverting the calculation in the ***block diagram***.

$R_0 = R_2 \boxminus [[(L_2 << 4) \boxplus K_2] \oplus [L_2 \boxplus \delta_2] \oplus [(L_2 >> 5) \boxplus K_3]]$

$L_0 = L_2 \boxminus [[(R_0 << 4) \boxplus K_0] \oplus [R_0 \boxplus \delta_1] \oplus [(R_0 >> 5) \boxplus K_1]]$

where $\boxminus$ denotes subtraction-mod-2^{32}.

Program TEA_Encryption: Write a Java program to implement TEA ***Encryption*** including only one *pair* of rounds: rounds 1 and 2

- Data and inputs:
  - declare a constant ***int*** DeltaOne with a hex value of 0x11111111
  - declare a constant ***int*** DeltaTwo with a hex value of 0x22222222
  - declare an ***int*** array with 4 elements: K[0], K[1], K[2], and K[3], get their values via user inputs in Hex strings
    For each element K[i], where i = 0, 1, 2, or 3, display the following prompt message, read the user input as a String, and then convert the user input String into an integer by treating it as a Hex value, and assign this integer to K[i]. E.g., if the user input is "F3579BD1" for K[0], then K[0] = 0xF3579BD1.
    `Please input K[i] in Hex String (without "0x"):`
  - declare two ***int*** arrays L[ ] and R[ ], each having a size of **3**.

For each of **L[0]** and **R[0]**, display a prompt message like the following one for L[0], read the user input as a String, and then convert the user input String into an integer by treating it as a Hex value, and assign this integer to L[0] or R[0]. E.g., if the user input is "2468ACE0" for L[0], then L[0] = 0x2468ACE0.

```
Please input L[0] in Hex String (without "0x"):
```

Initialize **L[1]**, **L[2]**, **R[1]**, and **R[2]** as 0x00000000's.

- The program and outputs:
  - Implement the **encryption** algorithm to calculate L[1] and R[1] first, and then L[2] and R[2]
  - Display the hex values of L[*i*] and R[*i*], where *i* = 0, 1, and 2, in Hex strings. E.g., assuming L[0] = 0x2468ACE0,

```
L[0] = 2468ACE0      R[0] = ……
L[1] = ……            R[1] = ……
L[2] = ……            R[2] = ……
```

Program TEA_Decryption: Write a Java program to implement TEA ***Decryption*** including only one *pair* of rounds: rounds 1 and 2

- Data and inputs:
  - declare a constant ***int*** DeltaOne with a hex value of 0x11111111
  - declare a constant ***int*** DeltaTwo with a hex value of 0x22222222
  - declare an ***int*** array with 4 elements: K[0], K[1], K[2], and K[3], get their values via user inputs in Hex strings
    For each element K[*i*], where *i* = 0, 1, 2, or 3, display the following prompt message, read the user input as a String, and then convert the user input String into an integer by treating it as a Hex value, and assign this integer to K[*i*]. E.g., if the user input is "F3579BD1" for K[0], then K[0] = 0xF3579BD1.

    ```
    Please input K[i] in Hex String (without "0x"):
    ```

  - declare two ***int*** arrays L[ ] and R[ ], each having a size of **3**.
    For each of **L[2]** and **R[2]**, display a prompt message like the following one for L[2], read the user input as a String, and then convert the user input String into an integer by treating it as a Hex value, and assign this integer to L[2] or R[2]. E.g., if the user input is "2468ACE0" for L[2], then L[2] = 0x2468ACE0.

    ```
    Please input L[2] in Hex String (without "0x"):
    ```

    Initialize **L[0]**, **L[1]**, **R[0]**, and **R[1]** as 0x00000000's.
- The program and outputs:
  - Implement the **decryption** algorithm to calculate L[1] and R[1] first, and then L[0] and R[0]
  - Display the hex values of L[*i*] and R[*i*], where *i* = 2, 1, and 0, in Hex strings. E.g., assuming L[2] = 0x2468ACE0,

```
L[2] = 2468ACE0      R[2] = ……
L[1] = ……            R[1] = ……
L[0] = ……            R[0] = ……
```

Task II (10%): **Test your program on the virtual server.**

Warning: to complete this part, especially when you work at home, you must first (1) **connect to the MSUDenver VPN** using your student VPN account (please read "**how to set up VPN for … for students**" at https://msudenver.edu/vpn/); then (2) **connect to the virtual server cs3750a.msudenver.edu** using ***sftp*** and ***ssh*** command on MAC/Linux or ***PUTTY*** and ***PSFTP*** on Windows. For details, you may refer to **Lab 1, Part I.**

1. MAKE a directory "**HW02**" under your home directory on **cs3750a.msdenver.edu**.
2. UPLOAD, COMPILE, and TEST your both programs under "**HW02**" on **cs3750a.msdenver.edu**.
3. SAVE two ***files named "tst_Encryption.txt" and "tst_Decryption.txt"*** under "**HW02**" on **cs3750a.msudenver.edu**, which captures the outputs of your programs as a proof that you have tested both programs on cs3750a. You can use the following commands to redirect the standard output (stdout) to a **file** on UNIX, Linux, or Mac, and view the contents of the file

```
javac prog_name.java     //compile .java file into byte code into .class file
java prog_name_args | tee tst_Encryption.txt  //copy stdout to the given .txt file
cat file-name      //display the file's contents.
```

4. If you work in a team of two students, you must put a ***team.txt*** file including both team members' names under your HW2/ on cs3750a (both team members are required to complete Task II under their own home directories on cs3750a). For grading, I will randomly pick the submission in one of the two *home directories* of the team members on cs3750a, and then give both team members the same grade.