COMP 2401 -- Assignment #1

Due: Friday, October 11, 2013 at 12:00 (noon)

Goal

You will write a program in C, in the Ubuntu Linux shell environment, to translate an encrypted message into ASCII characters and print out the decrypted message to the screen. The encrypted bytes, set up as array of `unsigned chars`, can be found here: ciphertext.txt

Learning Objectives

- familiarize yourself with the Linux programming environment
- write a small program in C that is modular, correctly designed and documented
- manipulate values at the bit level

Instructions:

1. **Support Functions:**

- **get/set/clear bit**

  You will implement the following bit manipulation functions, just as we saw in class:
  `unsigned char getBit(unsigned char c, int n)` returns the value of the nth bit of character c
  `unsigned char setBit(unsigned char c, int n)` returns the value of character c, with bit n set to 1
  `unsigned char clearBit(unsigned char c, int n)` returns the value of character c, with bit n set to 0

- **rotate**

  You will implement two functions that perform a circular rotation of the bits in a byte. One function is a rotation to the left, and the other is a rotation to the right. The function prototypes are the following:
  `unsigned char rotl(unsigned char c)` returns the value of c with its bits shifted left in a circular fashion
  `unsigned char rotr(unsigned char c)` returns the value of c with its bits shifted right in a circular fashion

- **swap two bits**

  You will implement a function that swaps two bits in a byte. The function prototype is the following:
  `unsigned char swapBits(unsigned char c, int pos1, int pos2)`
  The function returns the value of c with its bits in positions `pos1` and `pos2` switched with each other.

- **encrypt**

  The heart of the decryption algorithm (depicted in Figure 1) is the `encrypt` function, which is a permutation cipher that takes one byte and changes some of its bits depending on the value of a key. The result is the changed byte. You will implement this function, using the following prototype:
  `unsigned char encrypt (unsigned char c, unsigned char key)`
  The overall logic of the `encrypt` function is the following:
  - start with the value of the c as the basis for the resulting byte
  - examine each bit of the key, starting at the most significant bit
  - if the current bit of the key has value 0, swap the following two bits of the resulting byte: the bit in the current bit position, and the bit two positions to the left (circling back to the least significant bits, if necessary)
  - if the current bit of the key has value 1, perform a circular right shift on the resulting byte

2. Decryption algorithm

The decryption algorithm has the following structure:

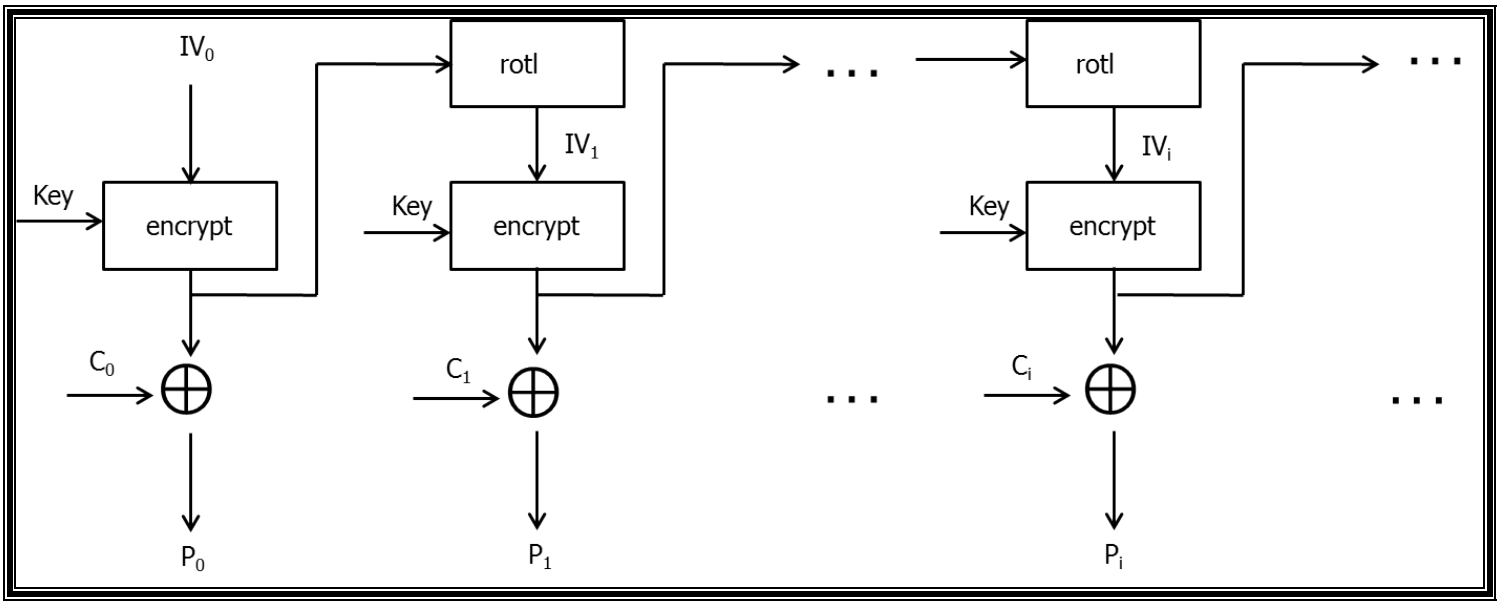

Figure 1 -- Decryption Algorithm

The encrypted message, known as ciphertext, is processed one byte at a time. It is shown in Figure 1 as C_0 through C_n, where n is the length of the ciphertext. The plaintext is the corresponding readable, decrypted byte. It is shown as P_0 through P_n. The **Key** and the initialization vector (**IV**) are initial values that are known beforehand. You can find these values in a header file located here: a1Defs.h

You will implement the decryption algorithm, as depicted in Figure 1. Your program will process the ciphertext one byte at a time, and print out the corresponding plaintext bytes.

The algorithm works as follows. At every iteration `i`:

- encrypt the value of IV_i using the key
- exclusive-or (XOR) the value of C_i with the encrypted IV_i to compute the value of P_i
- perform a circular left shift on the value of the encrypted IV_i to compute the value of IV_{i+1}

Notes: You may use the XOR bitwise operator in C. The `sizeof` function can compute the length of the ciphertext.

Constraints

- you must include the given header file in your program and use the function prototypes exactly as stated
- do **not** use any global variables
- you must use **all** the support functions that you implemented in Part 1, and reuse functions everywhere possible
- your program must be thoroughly commented
- programs that do not compile, do not execute, or violate any constraint are subject to severe deductions
- late assignments are never accepted

Submission

You will submit in *cuLearn*, before the due date and time, **one** tar file that includes all the following:

- all source code
- all data files required for your program to execute
- a readme file, which must include:
  - a preamble (program author(s), purpose, list of source/header/data files)
  - exact compilation command
  - launching and operating instructions

Grading

- **Marking breakdown:**

Component	Marks
get/set/clear bit functions	6
rotate functions	20
swap two bits function	10
encrypt function	35
decryption algorithm	24
printing result	5

- **Assignment grade:**
  - Your grade will be computed based on two criteria:
    - *completeness* of the program functionality and its execution
    - *quality* of the implementation
  - You must familiarize yourself with the grading rules
- **Bonus marks:**
  - Up to 5 extra marks are available for fun and creative additional features