CSCI 6331/4331 – Cryptography – Spring 2011
The George Washington University

Homework 2
due 7 February ON BLACKBOARD, 6 pm, **on Blackboard**

Policy on collaboration: All examinations, papers, and other graded work products and assignments are to be completed in conformance with The George Washington University Code of Academic Integrity. You may *not* discuss HWs among yourselves. Each student is expected to work independently on his or her own HW; you may not collaborate with others, you may not copy one another's assignments, even in part. You may only refer to your class notes, the text book, and to class material (such as slides, notes and links provided on the course website) while working on the homework, and not to any other material.

All code must run on your hobbes account. The TA will make no attempt to debug your code, or determine why it does not run. You will be graded on the correctness of your output, and the quality of your code: efficiency and documentation. There will be no exceptions.

Under no circumstances may code be copied from anywhere: classmates, the web, any other source

Any violations will be treated as violations of the Code of Academic Integrity.

Submit all HW in Blackboard by 6 pm on due date. Name your files:
CS284_HW2_LASTNAME_FIRSTNAME.rar or .zip or
CS162_HW2_LASTNAME_FIRSTNAME.rar or .zip

Part A: SPN: Implement an SPN with 4 rows of 4 S-boxes each, each S-box takes as input 8 bits. Use the **same code** for the encryption and the decryption, changing only the variables used by your modules.

Your program takes as input, from a file input_spn.txt, *formatted as you choose though the variables must be in the order below*:
- a key K of length 20 bytes, 4 bytes being used for each of five XORs (there is no key schedule as the key bits used are distinct each time)
- the permutation function σ
- the S-box function f
- m the length of the input
- an input x of length m
- an integer a of value 0 indicating encryption, and value 1 indicating decryption.

Your program produces the following output, in a file named output_spn.txt, *formatted as you choose*

though the variables must be in the order below:
- the input key K of length 20 bytes
- the input permutation function σ in the same format as in your input file
- the input S-box function f in the same format as in your input file
- the value of m
- an encrypted or decrypted output of length m; $E_K(x)$ if $a = 0$, and $D_K(x)$ if $a = 1$
- an integer representing $\bar{a}$.

Thus, if output_spn.txt is fed into the code, it should produce input_spn.txt and vice versa. This will be one of the checks performed on your code.

Part B: Feistel Cipher:

a. Implement a single block of a Feistel cipher assuming an input, from a file named input_feistel.txt *formatted as you choose though the variables must be in the order below*:
- n, the size of each half-block;
- K, a key of length n for the encryption
- X, the input string of $2n$ bits to be encrypted/decrypted;
- an integer a of value 0 indicating encryption, and value 1 indicating decryption.

Your program produces the following output, in a file named output_feistel.txt, such as the one available off the course website:
- n, the size of each half-block;
- K, the encryption key
- Y, the encrypted/decrypted output string;
- an integer representing $\bar{a}$

Assume the function f is as follows: $f(X_r, K) = X_r \times K^2 \ mod \ (2^n)$ (with X and K treated as numbers $mod\ 2^n$).

Use the **same code** for the encryption and the decryption, changing only the variables used by your modules.

b. Assume the following key schedule: $K_1 = key$, $K_2 = \sigma(K)$, where σ is a cyclic shift two bits to the left. Implement an encryption module taking the same inputs as the individual Feistel block above, consisting of 2 rounds. The output has the same format as the output in 2a above, this file may be named output2_feistel.txt.

You are expected to write the Feistel Cipher and SPN code yourself, and not to get it from a library.

Submit code for both parts A and B, as well as valid input files. Your code will be tested by running it on input files you provide. These files will also be modified and the code will also be run on the modified files.