Chapter 2: Set two: Encryption Part I.
Deadline: Friday, September 29, 23:59 (11:59 PM)

Please consult the 'Practical assignments instructions and rules' document before proceeding with the exercises.

Exercise 1. [1.5 points]
Purpose of this exercise: construct a tool to work with the Vernam cipher.

Write a program that, given a key and input data, applies the key as a one time pad (Vernam cipher) to the input data to encrypt or decrypt it. The input of the program is given as follows: first a key consisting of *n* bytes, followed by the binary value *0xFF*, followed by the input data consisting of *n* bytes as well.

The output should be the encrypted/decrypted data, also in binary.

Note: as this program requires you to read a binary file, functions designed for reading text files will not work.

Exercise 2. [2 points]
Purpose: learn to implement a stream cipher.

Implement the RC4 stream cipher. The input for this program will be given in a similar format as the input for the Vernam cipher program earlier: first a key, followed by the binary value *0xFF*, followed by the input data.

The output should be the encrypted/decrypted data, also in binary.

Please be aware that there is an attack possible against RC4. Read the RC4 section in the book *Information Security: Principles and Practices* (starting on page 55) on how to prevent that attack. This prevention is needed to pass the test cases on Themis.

Note: as this program requires you to read a binary file, functions designed for reading text files will not work.

Exercise 3. [2.5 points] Purpose: learn to work with a simple Feistel cipher.

Define and implement a simple Feistel cipher in a little program. The input of this program is given in binary and formatted as follows: first the binary value of *d (0x64)* or *e (0x65)* specifying whether to decrypt or encrypt, followed by the binary value *0xFF*, followed by a key, then another *0xFF*, and finally the input data. The key size is a multiple of 4 bytes, while the input data is a multiple of 8 bytes.

The key bytes are used in the key-schedule, described below.

Implement the feistel function as a function processing blocks of 8 bytes. Each block is split into a left half (LH) and right half (RH).

As usual, RH becomes the LH of the next round, and LH xor f(key) becomes the RH of the next round.

Although the ECB block cipher should be avoided, it is used in this exercise to encrypt subsequent blocks of plain text. This exercise concentrates on the Feistel method, and once that's available it can be used in combination with other block cipher methods as well.

Key Schedule

To fully complete this method an additional manipulation would be used in which the RH and key are manipulated (see also 3DES). Eventually RH2 becomes

```
        LH1 ^ F(RH1, key)
```

where the key schedule is a separate operation, and the RH may also be manipulated, combining them to `F(RH, key)`.

This part (the key schedule handling) is not further elaborated in this exercise: Instead, the function $f(key)$ simply returns the key again.

Exercise 4. [1.5 points]
Purpose: learn to validate public and private keys of a superincreasing knapsack.

Write a program that, given as input *m*, *n*, a private and a public key, checks if this private key is a valid key and whether the public key corresponds to it. The output of the program should be *-1* if the private key is invalid, *0* if the public key is invalid and *1* if both the public and private key are valid. Keep in mind that the public key can never be valid if the private key is not.

The input is structured as follows (all in text, not in binary):

```
m n
[private key]
public key
```

Exercise 5. [2.5 points]
Purpose: Implement encryption and decryption using a superincreasing knapsack.

Write a program that, given as input the appropriate keys and the input data, encrypts or decrypts the input data and shows the encrypted/decrypted text. The input is formatted at follows (in text, not in binary):

For encryption:

```
e
[public key]
<integer value 0>
<integer value 1>
<integer value 2>
etc.
```

For decryption:

```
d
<m> <n>
[superincreasing knapsack]
<integer value 0>
<integer value 1>
<integer value 2>
etc.
```