Assignment 1

❖ In this assignment you are asked to implement 5 ciphers.

❖ You have to use C to implement them from scratch.

❖ You should create 2 files:
  ➢ cs457_crypto.h, containing function declarations
  ➢ cs457_crypto.c, containing the implementation of the functions
1) One-time pad

❖ It’s a theoretically unbreakable cipher

❖ It uses a pre shared-key that is at least the size of the plaintext

❖ The algorithm XORs each byte of the plaintext with the corresponding key byte
1) One-time pad

Plaintext:      HelloWorld

Key:           randombyte

Output:        (H⊕r)(e⊕a)(l⊕n)(o⊕d)(W⊕m)(o⊕b)(r⊕y)(l⊕t)(d⊕e) =

Hex:           3A 04 02 08 00 3A 0D 0B 18 01
1) One-time pad

- In order to generate a random key, use /dev/urandom (use a Linux base system)

- Decryption can be done by XORing the ciphertext with the key

- You have to store the key used for encryption, in order to successfully decrypt the ciphertext
1) One-time pad

❖ You should implement the functions:

➢ one_time_pad_encr
➢ one_time_pad_decr

which take as arguments the plaintext/ciphertext and the key and return the result of the operation.

❖ The plaintext can contain only digits (0-9) and english letters (A-Za-z)
1) One-time pad

Assume that a programmer used the same “one time pad” to encrypt two English words. The result of the encryption is “e9 3a e9 c5 fc 73 55 d5” and “f4 3a fe c7 e1 68 4a df”. Can you find which these two words are?
2) Rail fence cipher

- It’s a form of transposition cipher, also called a zigzag cipher.
- Plaintext is written on alternate “rails” diagonally and we obtain the ciphertext by combining the individual rows.
2) Rail fence cipher

Plaintext: Hello World

Number of rails: 3

Ciphertext: Horel ollWd

Note that the space is written and read normally as the other characters.
2) Rail fence cipher

❖ You should implement the functions:

➢ `rail_fence_encr`
➢ `rail_fence_decr`

which take as arguments the plaintext/ciphertext and the number of rails and return the result of the operation.
3) Beaufort cipher

- The Beaufort cipher is a substitution cipher that uses a tableau (shown below) and a keyword to encipher the plaintext.

- Assume that we will be encrypting and decrypting only the capital letters of the English alphabet.

- The key size must match the size of the plaintext. For example:
  - Plaintext: “ATTACK AT DAWN” (length = 12 after removing spaces)
  - Keyword: “LEMON” (length = 5)
  - Keyword should be transformed to: “LEMONLEMONLE” (length = 12)
3) Beaufort cipher: The tableau

- The tableau used for the encryption and decryption algorithms is shown in the figure at the right.
- The way you wish to implement this (either statically or dynamically) is up to you.
3) Beaufort cipher : Encryption

❖ Having the tableau and the correct keyword for our plain text:
  ■ Plaintext : “ATTACKATDAWN” (length = 12)
  ■ Keyword : “LEMONLEMONLE” (length = 12)
❖ The steps to encrypt are the following (with the letter T for example):
  ➢ Find the plaintext letter in the topmost horizontal row
    ■ The letter T position in the table is table[0][19]
  ➢ Travel down the column, until you find the current key letter
    ■ Traveling down column 0 we find E at table[11][19]
  ➢ The leftmost letter in the current row is the new ciphertext letter
    ■ The leftmost letter is at position table[11][0] which is L
❖ The above example should be encrypted as :
  ➢ Ciphertext : “LLTOLBETLNPR”
3) Beaufort cipher: Encryption (example)

1) Find the plaintext letter (T) in the topmost horizontal row
3) Beaufort cipher: Encryption (example)

2) Travel down the column, until you find the current key letter
3) Beaufort cipher: Encryption (example)

3) The leftmost letter in the current row is the new ciphertext letter
3) Beaufort cipher : Decryption

- Decryption works the same way as encryption only now instead of the plaintext you use the ciphertext which the decryption function returned.
3) Beaufort cipher

- You are asked to implement the functions:
  ◆ **beaufort_encr**
  ◆ **beaufort_decr**

Which receive as arguments the plaintext/ciphertext (respectively) and the keyword for the algorithm.

Both functions should return the result of each operation.
4) Affine cipher

- Affine cipher is a substitution cipher, where each letter of the alphabet is mapped to its numeric equivalent, encrypted using a mathematical function and converted back to a letter.

- Assume that we correspond each letter with its position in the alphabet starting from 0 (A-0, B-1, C-2 etc.) including only the capital letters.

- Each letter is enciphered with the function \((ax + b) \mod m\), where:
  - \(m\) is the size of the alphabet.
  - \(a, b\) are the keys of the cypher.
  - (!) \(a\) must be coprime to \(m\) (the only positive integer divisor of both is 1)
4) Affine cipher : Encrypt

- The encryption function for a single letter L is:
  - Enc(x) = (a * x + b) % m
  - Enc(x) returns the index of the encrypted letter in the alphabet
  - where x is the mapped position of the letter L in the alphabet

- Assume:
  - Plaintext = “AFFINE CIPHER”
  - a = 5
  - b = 8
  - m = 26

- The encryption works as follows
4) Affine cipher: Encrypt

<table>
<thead>
<tr>
<th>Plaintext</th>
<th>A</th>
<th>F</th>
<th>F</th>
<th>I</th>
<th>N</th>
<th>E</th>
<th>C</th>
<th>I</th>
<th>P</th>
<th>H</th>
<th>E</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>8</td>
<td>13</td>
<td>4</td>
<td>2</td>
<td>8</td>
<td>15</td>
<td>7</td>
<td>4</td>
<td>17</td>
</tr>
<tr>
<td>(5*x+8) mod 26</td>
<td>8</td>
<td>7</td>
<td>7</td>
<td>22</td>
<td>21</td>
<td>2</td>
<td>18</td>
<td>22</td>
<td>5</td>
<td>17</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>Ciphertext</td>
<td>I</td>
<td>H</td>
<td>H</td>
<td>W</td>
<td>V</td>
<td>C</td>
<td>S</td>
<td>W</td>
<td>F</td>
<td>R</td>
<td>C</td>
<td>P</td>
</tr>
</tbody>
</table>
4) Affine cipher: Decrypt

- The decryption function for a single letter L is:
  - \[ \text{Dec}(x) = a^{-1} * ((x - b) \mod m) \]
  - \( \text{Dec}(x) \) returns the index of the decrypted letter in the alphabet
  - where \( x \) is the mapped position of the letter L in the alphabet

- (1) \( a^{-1} \) does not translate into \( 1/a \):
  - where \( a^{-1} \) is the **modular multiplicative inverse of a mod m**, meaning it satisfies the equation:
    - \[ a a^{-1} \mod m = 1 \]
4) Affine cipher

❖ You are asked to implement:
   ➢ affine_encr
   ➢ affine_decr

Which take as arguments the plaintext/ciphertext (respectively) and return the result of the operation.

❖ Assume that:
   ➢ a = 11
   ➢ b = 19
   ➢ m = 26 (only the capital letters of the english alphabet)
5) Feistel cipher

- It’s a structure used to develop many block ciphers such as DES.

- **Block ciphers** encrypt a fixed size of block of the plaintext at a time (instead of a byte a time).

- Feistel cipher consists of rounds and a separate key is used for each one.
5) Feistel cipher

- For encryption, first you need to divide the plaintext into blocks of fixed size (padding is used if plaintext doesn't fill the last block).

- Each block is encrypted in rounds. For each round:
  - Divide the block into two halves, \( L_i \) and \( R_i \)
  - Compute \( L_{i+1} = R_i \) and \( R_{i+1} = L_i \text{ XOR } f(R_i, K_i) \)

After \( n \) rounds the final ciphertext is \((R_{n+1}, L_{n+1})\)
5) Feistel cipher

- Since encryption is performed in blocks, the size of ciphertext will be a multiple of block size.
- Decryption is performed in a similar way, with the difference of using the keys in the reverse order.
5) Feistel cipher

❖ You should implement the functions:
  ➢ feistel_encr
  ➢ feistel_decr

which take as arguments the plaintext/ciphertext, the keys and the size of the message we want to encrypt/decrypt and return the result of the operation.

❖ Assume that \( n(\text{rounds}) = 4, \) block size = 128, size of \( R_i = 64, \) size of \( L_i = 64, \) key size = 64 and \( f(R_i, K_i) = (R_i \times K_i) \mod (2^{64}). \)

❖ You should again use /dev/urandom to generate the keys.