Lab 07 - Block Ciphers and Block Cipher Modes

Intro

In lecture we discussed the differences between stream ciphers and block ciphers. We spent some time looking at older stream ciphers such as A5/1 which was used in GSM (gen 2) and RC4 (which was used in WEP and SSL/TLS), as well as the newer ChaCha20-Poly1305 whose use has been widely included in hardware, software, and protocols. We also are looking at block ciphers such as DES and AES and the different modes block ciphers can run in.

The most well-known and influential Feistel cipher was the Data Encryption Standard (DES). Also, the Soviet Union/Russia had a similar cipher named GOST. While DES has been deprecated, there are still numerous Feistel ciphers available for use today. The better known are Blowfish, Twofish (one of the finalists to become AES), CAST (also a contender to be AES), and Camellia.

There is a Tiny Encryption (TEA) algorithm that is commonly used to simply demonstrate a Feistel algorithm. A Baby DES implementation also exists. Although both of those have more specifications than we need to implement in lab. We just want to get an understanding of a Feistel network. We will implement a smaller, simple 4-bit Baby Feistel cipher and only run it for three rounds. This is also a commonly implemented Feistel network in labs and classrooms as a teaching device. You will encrypt and decrypt with your Baby Feistel cipher.

Additionally, you will decrypt a message encoded with AES and encrypt a message for the TAs. Finally, we will look at the different block cipher modes comparing the speeds and what information can be leaked when using them.

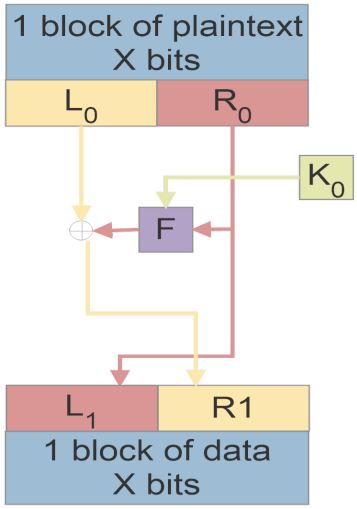
Part 1

You will implement a 4-bit block cipher with a 4-bit key and it output 4-bits of data. It will run for 3 rounds. You will not be implementing a block mode so processing a full message won't be possible. You will be working with the first 4-bits of a single character since you are working at the binary level. Remember, even if you implemented a block cipher mode and could process an entire message, this is a very simple Feistel cipher which, of course, is not secure.

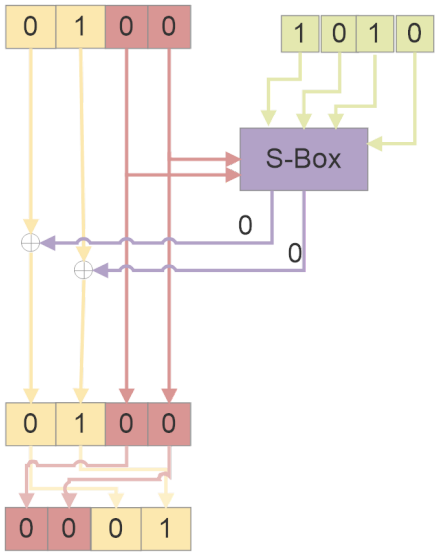
1. Remember how a Feistel network functions. The image used in lecture is provided below for reference.
    a. The function starts by splitting the block of X bits (in this case 4 bits) into two parts, a left half (L_0) and a right half (R_0)
    b. After it is split into halves, the round function (F) is applied to one half using a round key. In DES and in our 4-bit implementation this will be the right half (R_0).
    c. The output of the round function is XOR'd with the other half. In our

implementation this will be the left half (L_0).

d. The halves are then transposed so that output of step c above becomes R_1 and the old R_0 moves down to become L_1.



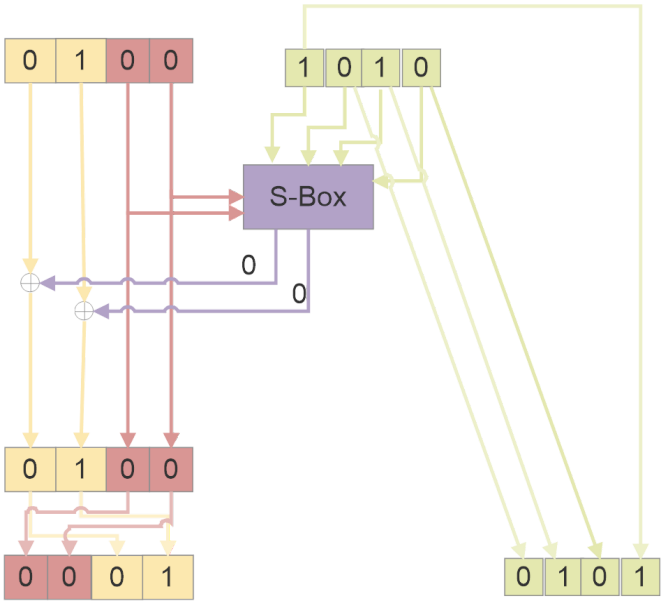
2. **As an example, we will use 0100 as the plaintext and the key of 1010.** The R_0 bits are input to the S-Box and the L_0 bits are XOR'd with the output from the S-Box. The description of how the S-Box works is in the next step. The final step is to swap the halves, making the bits in the L_0 new right half and the bits in the R_0 position the new left half. These bits become L_1 and R_1 in the next round.



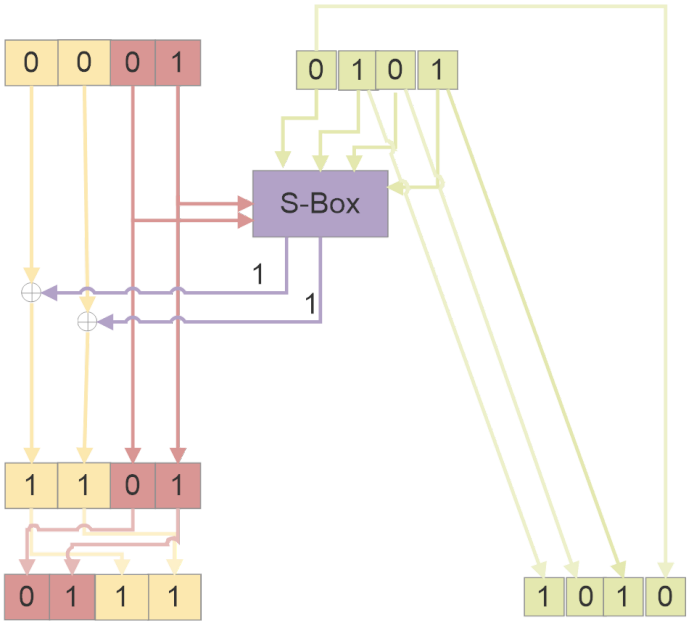
3. The S-Box we are using is a simple lookup table. It is comparable to what happens in DES. In our version, you use the key value as the index to the rows and the two bits from R_0 as the index to the columns. For this example, the row value is 1010 and the column value is 00. The intersection of those two provides you with 00 as the output.

	Last 2 bits of block			
Key	*10*	*01*	*00*	*11*
0000	10	10	00	11
0001	00	11	10	00
0010	00	10	01	00
0011	01	00	11	11
0100	11	00	10	01
0101	10	11	01	10
0110	11	01	11	10
0111	10	01	00	01
1000	00	01	11	10
1001	01	00	01	11
1010	11	11	00	00
1011	11	10	01	01
1100	01	11	10	10
1101	00	01	11	00
1110	10	00	00	01
1111	01	10	10	11

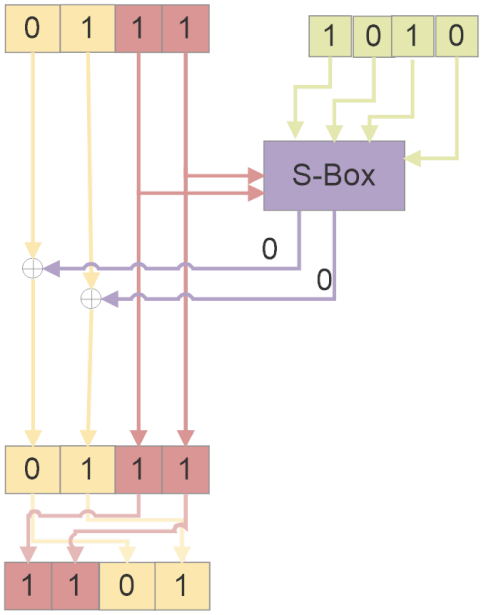
4. Before you enter the next round, you need to do a key rotation. In DES this is called the round key. Again, we will do a simple process of moving the left most bit in the key to the right most position while shifting the remaining 3 bits one position to the left.



5. You now start the second round with 0001 as your block of input and 0101 as your round key. The image illustrates the process.



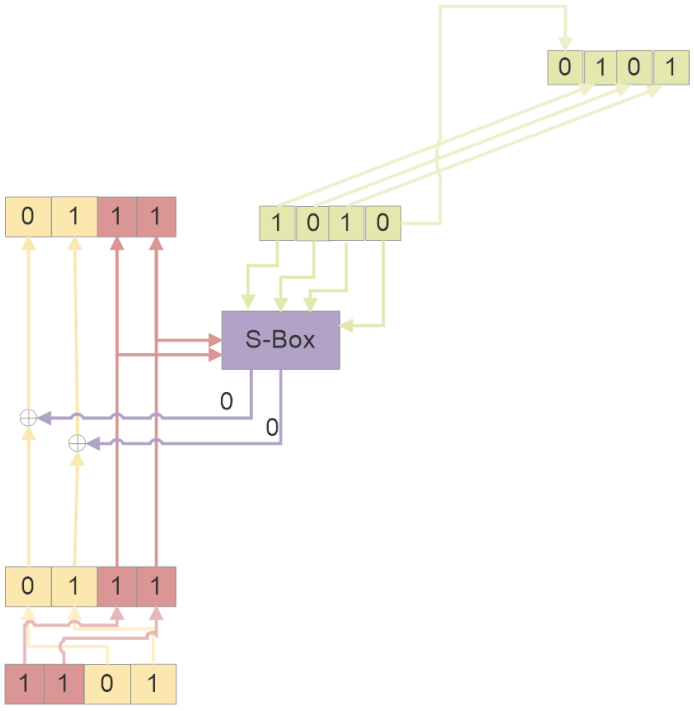
6. The third round is shown in the image below. Notice we don't need to generate a next round key.

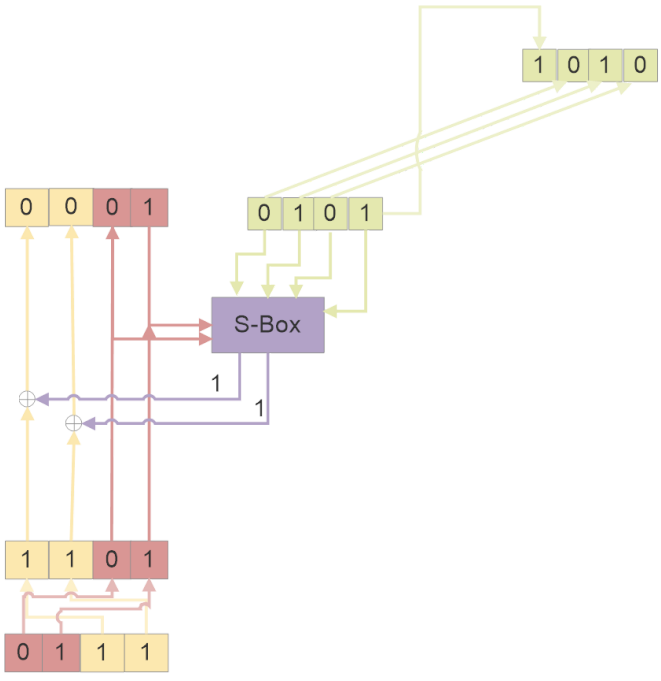


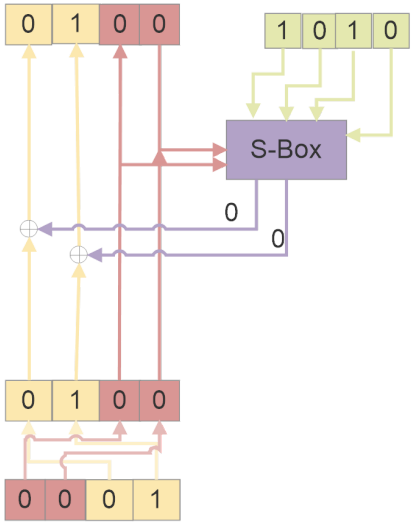
7. The 4-bits of ciphertext for this simple demonstration is 1101 which is the output from the final round. You may want to print your output for each round similar to what is below just for your own troubleshooting.

Round 1: [0, 0, 0, 1]
Round 2: [0, 1, 1, 1]
Round 3: [1, 1, 0, 1]

8. To decrypt simply run the process in reverse starting with the ciphertext and the last key used. Because the right 2 bits do not change, you can still do a lookup in the S-Box table using these bits and the key. At the end of the round, the round key gets modified by moving the right most bit to the left most position and other three bits shift right. You are getting the key ready for the second round of decryption. Of course, you must run three rounds in decrypt mode to get the plaintext message of 0100 back out.







9. **The 4-bits of input you will use is 1010 and the key is 0111.** You need to write a program in python that will **encrypt** and **decrypt** the 4-bit message. Do not hard code this. You will upload your commented python code to Canvas. You will include your 4-bits of cipher text in your lab report. Also, include a screenshot of your code successfully encrypting and decrypting the 4-bit message.

Part 2

Begin by downloading files from repo.331.com via the command shown below.

```
cpre331@desktop:~/Downloads/lab07$ scp -r repo@repo.331.com:/home/repo/lab07/* .
repo@repo.331.com's password:
cipher_f22_1.enc                              100% 1865     4.6MB/s   00:00
plaintext_f22_2.txt                           100%  457     1.6MB/s   00:00
favoriteOS_f22.bmp                            100% 2204KB  25.3MB/s   00:00
```

1. You will use openssl to read a file I have encrypted. You will have to record the author and plaintext in your lab report. The file you download from repo.331.com was encrypted with openssl using aes-256 and the password "fish." The plaintext was encoded with base64 before it was encrypted. I also had to use the password-based key derivation function 2. This allows the key and the IV to be derived from the password with the default number of iterations (many, many iterations). I could have asked you type a series of numbers using the -k and -iv flags, but I took pity on you and just provided the password.
    a. openssl enc -d -aes256 -base64 -pbkdf2 -in cipher_f22_1.enc -out plaintext_f22_1.txt
2. **Record the title, the author, and the plaintext in your lab report**.
3. Now, you need to encrypt the file plaintext_f22_2.txt which is also found on

repo.331.com. You need to first encode it with base64 and then encrypt it using aes-256 and the password "winter". Again, use the -pbkdf2 flag. **Name the outfile with your netid in it (cipher_netid_f22_2.enc), please.**

4. **Upload the encrypted file to Canvas and the TA will decrypt it**.

Part 3

You will conduct a speed test of AES using two different block modes. You will use the speed option in openssl to conduct this evaluation. The two different block modes are aes-256-gcm and aes-256-cbc

1. Run the command *openssl speed -evp aes-256-gcm aes-256-cbc*.
2. Looking at the output from using 8192 byte size blocks, how many iterations can each cipher generate in 3 seconds?
3. Looking at the output from using 8192 byte size blocks, how many megabytes per second can each cipher encrypt?
4. Which one is faster? Provide a **technical explanation** as to why one would be faster than the other. Saying it is faster because it encrypts more in the same amount of time will earn you 0 points.

Part 4

This part will show you visually the differences between ECB and CBC modes of block ciphers. **This is not steganography.** I am only trying to demonstrate why ECB (Electronic Code Book) mode is a very bad idea. This should also make you think about the paper codebook without an additive.

Because I teach at a university, I have to publicly have no opinion about operating systems. However, I decided to send a picture of the logo of my preferred OS to a friend just for fun. I encrypted the bmp logo file using AES with a 128 bit key. Open the image encrypted_favoriteOS_f22.bmp and see if you can figure out what my favorite OS is. It is also in the download from repo.331.com.

a. What's the name of my favorite OS?
b. What mistake did I make?
c. What problem with this particular type of cipher block mode does this picture demonstrate?
d. How could I get rid of this problem?

Now, I want you to take a **bmp of your favorite OS's mascot** and encrypt it the same way.

1. Put your **bmp** into your home directory. (**Hint**: You may need to use GIMP or some other image tool to convert your image from jpg or png to bmp. You also may want to use smaller images. Some of the very large images can cause issues).
2. You will use openssl to encrypt. Run the following command in terminal:

openssl enc –aes-128-ecb –in your.bmp –out encrypted_your.bmp

3. This process encrypted all the headers, so it can't be opened by any graphics program. So, now we have to put the headers back onto the encrypted file, so an image viewer can read it. Run the following command in terminal:

    dd if=your.bmp of=encrypted_your.bmp bs=1 count=54 conv=notrunc

4. Now copy the encrypted image to your desktop and try to open it. You can play with other encryption algorithms and cipher block modes, but please **submit the file you encrypted using these steps**.

Lab 04 Template

1) Part 1:
    a. **Upload python code for babyFeistel cipher**
    (20 points total; 10 points encrypt, 10 points decrypt)

    b. **4-bit ciphertext**
    (10 points)

    c. **Screen shot of encrypting and decrypting**
    (10 points total; 5 points encrypt, 5 points decrypt)

2) Part 2:
    a. **Decrypt cipher and record the title, author, plaintext**
    (10 points total; 4 points title, 3 points author, 3 points plaintext)

    b. **Encrypt the plaintext provided and upload to Canvas**
    (10 points)

3) Part 3:
    a. **How many iterations for each cipher in 3 seconds using 8192 byte size blocks?**
    (5 points)

    b. **How many megabytes can each cipher encrypt per second using 8192 byte size blocks?**
    (5 points)

    c. **Which one is faster? Why?**
    (5 points)

4) Part 4:
    a. **Submit the name of my favorite OS**
    (10 points)

    b. **Answer questions about my favorite OS**
    (10 total points; 1 point name, 3 points for each of the other questions)

    c. **Submit an encrypted bmp of your favorite OS's mascot**
    (5 points)