# Manually Computing Encryption

In this module, you looked at the various methods used to create ciphertext. In this exercise, you will manually create ciphertext using the given building blocks described in each part.

# Instructions

Apply the inputs below to each of the methods listed, in order. For Part 2, use the right shift operation *only*. Solutions will be posted on Bright Space three days after the assignment deadline.

Use the following inputs:

1. 0000 0000
2. 0110 1010
3. 231
4. A916
5. 1101 1011
6. 1111 1111

## Part 1: S-box

Substitution boxes are used to manipulate the order of each input bit to deter cryptanalysis.

Consider the following S-box. The output represents where from the input stream the output bit will appear:

|  |
| --- |
| S-box |
| 2 6 3 1 4 8 5 7 |

In this example, input **ABCD EFGH** produces the output **BFCA DHEG**.

Inputs: Output:

1. 0000 0000 ---→0000 0000
2. 0110 1010 --→1010 0011
3. 231= 1110 0111 --→ 1111 0101 =245
4. A916 = 1010 1001 → 0011 0110 = 3616
5. 1101 1011 → 1001 1111
6. 1111 1111 --->

## Part 2: Left/Right Shift

Shifting bits is another way to obscure the original message. In this process, the value of each position remains the same, while the entire sequence is shifted in a circle. Note that the number of bits to shift is usually represented by the number of chevrons “>” or a number following the L or R character (e.g., L-2 or << would shift 2 bits to the left).

|  |  |  |
| --- | --- | --- |
| Left Shift |  | Right Shift |
| < |  | >>> |

In this example, input **ABCD EFGH** produces the output **BCDE FGHA** for the left rotation and **FGHA BCDE** for the right rotation.

OutPut

1. 0000 0000
2. 0110 1010
3. 231 = 1110 0111
4. A916 = 1010 1001
5. 1101 1011
6. 1111 1111

## Part 3: Expansion

Sometimes it can be useful to create a new number of bits from a given input text. This is one way to pad the data to a convenient size (e.g., for block encryption). In the following example, the output is determined in the same way as in Part 1 but in this case, positions can be repeated.

|  |
| --- |
| Expansion |
| 4 6 8 5 4 2 3 1 7 7 |

In this example, input **ABCD EFGH** produces the output **DF HEDB CAGG**.

1. 00 0000 0000
2. 00 0101 1011
3. 01 1001 1111 = 415
4. 00 1100 1100 = CC16
5. 10 1111 0111
6. 11 1111 1111

## Part 4: Compression

Sometimes it is effective to remove some of the input bits. In this example, not every input bit is mapped to an output bit. Substitution may be applied here as well.

|  |
| --- |
| Compression |
| 3 7 5 8 1 4 |

In this example, input **ABCD EFGH** produces the output **CGE HAD**.

output

1. 00 0000
2. 11 1000
3. 11 0110 = 54
4. 10 1110 = 2E16
5. 01 1111
6. 11 1111

## Part 5: XOR

XOR or Exclusive OR, is a logical function that has an input of two bits and returns true (1) if, and only if, exactly one of the inputs are set to true (1). Usually this step is used to combine a keystream into our input function. This function is usually denoted with the **⊕** symbol. Use the input stream of **0110 1101** and computer the bitwise XOR with the given inputs.

1 **⊕** 1 => 0 0 **⊕** 1 => 1 1 **⊕** 0 => 1 0 **⊕** 0 => 0

In this example, input **ABCD EFGH** produces the output **0BC0 EF0H**.

output

1. 0110 1101
2. 0000 0111
3. 1000 1010 = 138
4. 1100 0100 = C416
5. 1011 0110
6. 1001 0010

## Part 6: Inversion

Inversion is a process that is normally used to restore a cipher text into its original order. If you were to take the S-box from Part 1 and move the output into this S-box observe what the output becomes.

|  |  |  |
| --- | --- | --- |
| S-box |  | Inversion (S-Box)-1 |
| 2 6 3 1 4 8 5 7 |  | 4 2 3 5 7 2 8 6 |

You can see that the using input **ABCD EFGH** for S-box, you get an output of **BFCA DHEG.** Using the ouput from the S-box as the input to the inversion, you would receive the output **ABCD EFGH**.

output

1. 0000 0000
2. 1010 0011
3. 1111 0101 = 245
4. 0011 0110 = 3616
5. 1001 1111
6. 1111 1111

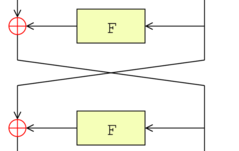
output after Inversion: -

1. 0000 0000
2. 0110 1010
3. 1110 0111 = 231
4. 1010 1001 = A916
5. 1101 1011
6. 1111 1111

## Part 7: Swap

Swapping of portions of a bit sequence is another way to help protect from cryptanalysis. Usually the process is symmetric, as in the sequence is split into two and each half is swapped for the other side.

Figure 1 shows the swap function as two crossing lines. Occasionally, this is represented as a square box with a cross through it.



**Figure 1: Section of DES Algorithm**

Adapted from: Matt Crypto, <https://commons.wikimedia.org/wiki/File:DES-main-network.png> (Public Domain)

In this example, input **ABCD EFGH** produces the output **EFGH ABCD**.

output

1. 0000 0000
2. 1010 0110
3. 0111 1110 = 126
4. 1001 1010 = 9A16
5. 1011 1101
6. 1111 1111