Project 1
CS356

CS356

Colorado State University

September 6, 2015

Contents

Program Requirements

1. This program should be written in C or C++.
2. Your submission should include a Makefile.

Program Arguments I

Your program will take 5 arguments in the order defined below. Your program should ensure there are 5 arguments, and that each argument is error-checked appropriately.

1. The first argument is either a 'B' or a 'S'.
   - B means you will use your block cipher function.
   - S means you will use your stream cipher function.
   - Your program should terminate if other than a 'B' or an 'S' is entered with an appropriate error message.
2. The second argument is the input file name.
   - Your program should terminate with an appropriate error message if the file does not exist.
3. The third argument is the output file name.
4. The fourth argument is the keyfile name.
   - The keyfile contains the symmetric key in ASCII format.
   - The key size for Block Encryption Mode would be 64 bits (8 bytes) and can be of any length for Stream Encryption Mode.
   - Your program will read in the contents of the keyfile and use it to encrypt the plaintext provided in the input file.

Program Arguments II

- The keyfile will not contain a terminating newline character.

Example

File 'keyFile' contains "ABCDEFGH" for block cipher and "EDRFT" for stream cipher.

5. The fifth argument is the 'mode of operation' which can be either 'E' for encryption or 'D' for decryption.
   - In the 'E' mode you would encrypt a plaintext using a key and produce a ciphertext.
   - Similarly, in the 'D' mode your program is expected to decrypt an already encrypted text, with the same symmetric key, and produce expected plaintext results.

Program IO

1. Your program will read in the input file.
   - End of line characters are data just like any other character.
   - You may assume the file is in multiples of 8 bits.

Example

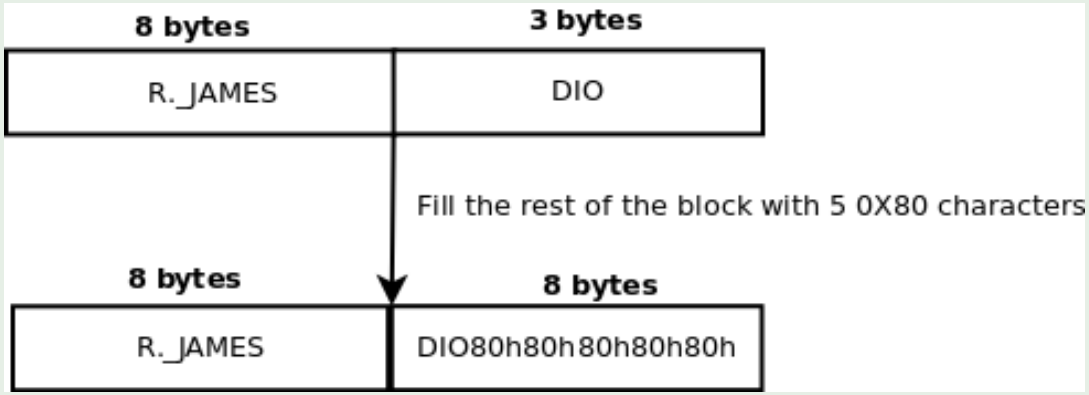
File “input” contains ”FRANK ZAPPA\n” $->$ 12 bytes or 12*8 bits.

2. Your file will write the output file based on the arguments.
   - No need to preserve white-spaces from the input in the output.
   - No newlines at the end of the output file.
   - If the input file is empty the output file should also be empty.

Padding

1. You must pad input blocks that are not 64 bits or 8 bytes in length.
2. Each padding byte should be 0X80.
3. Because the input file will be a multiple of 8 bits or 1 byte, padding (if required) can also be done in multiples of 8 bits or 1 byte. For the purpose of this project you will be using the byte value 0X80 in hex or 128 in decimal. Because, the input file is in ASCII and the padding value (128 or 0X80) falls outside the ASCII (0-127 or 0X7F) range, it allows us distinguish padding bytes from plaintext bytes. Stream mode does not require any padding.

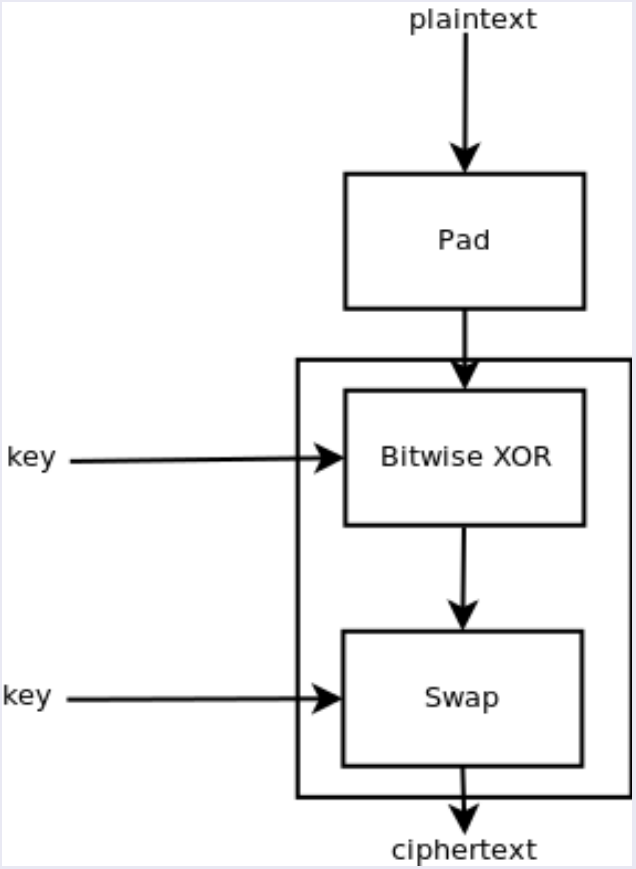
Example



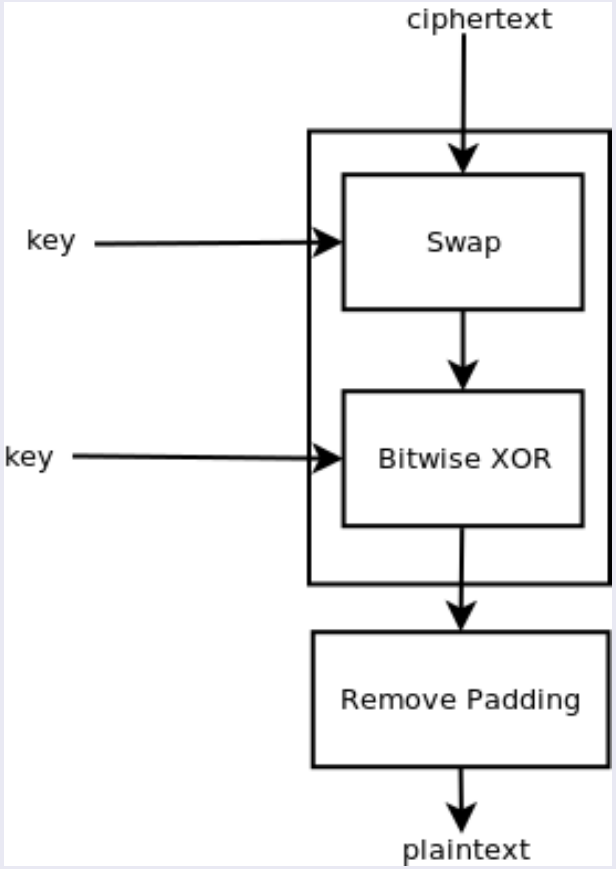
Description of the algorithms

Block Cipher

Encrypt



Decrypt



XOR

A two byte example

```
01000001 01000010 => 65 66 =>   A B  (plaintext)
00110001 00110010 => 49 50 =>   1 2  (key)
-----------------------------------------
01110000 01110000 => 112 112 => p p (ciphertext)
```

Notice the power of XOR. The XOR operation produces 'p' in both cases. So given the ciphertext 'p' there would be a 50-50 chance of the plaintext being either 'A' or 'B'. This causes more trouble for the evil person who is trying to break our block cipher algorithm.
You have to figure out a clever way to implement *bitwise XOR* in C/C++ using other boolean operators.

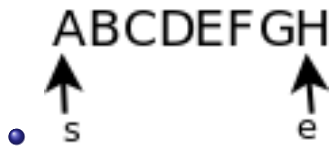
Description of the algorithms

Block Cipher

Swap

Algorithm

1. Let 'start' and 'end' be pointers which point to the starting and ending byte of the XORed output string (*Note:* 'end' points to the end of the string and not the block.Eg. 'start' begins at 0 and 'end' begins at 7 if there are 8 bytes in the XORed output string)
2. For each byte of the key, starting from the left most byte or 0th byte, you calculate the following : (ASCII value of the byte or character)mod2. This would give you either 0 or 1.
3. If the value is 0 you do not swap anything and move to the next byte of the plaintext by incrementing the 'start' pointer. Otherwise, you swap the byte pointed by the 'start' pointer with that pointed by the 'end' pointer. Then increment the 'start' pointer so that it points to the next higher byte and decrement the 'end' pointer so that it points to the next lower byte.
4. If the keysize is exhausted, you restart from the first byte of the key.
5. This process is carried on until the 'start' and the 'end' pointers collide or crossover.



key = **V**ANHALEN
keybyte = V (86)
86 mod 2 = 0
swap = NO

Description of the algorithms

Block Cipher

Swap

Algorithm

1. Let 'start' and 'end' be pointers which point to the starting and ending byte of the XORed output string (*Note:* 'end' points to the end of the string and not the block.Eg. 'start' begins at 0 and 'end' begins at 7 if there are 8 bytes in the XORed output string)
2. For each byte of the key, starting from the left most byte or 0th byte, you calculate the following : (ASCII value of the byte or character)mod2. This would give you either 0 or 1.
3. If the value is 0 you do not swap anything and move to the next byte of the plaintext by incrementing the 'start' pointer. Otherwise, you swap the byte pointed by the 'start' pointer with that pointed by the 'end' pointer. Then increment the 'start' pointer so that it points to the next higher byte and decrement the 'end' pointer so that it points to the next lower byte.
4. If the keysize is exhausted, you restart from the first byte of the key.
5. This process is carried on until the 'start' and the 'end' pointers collide or crossover.

key = V**A**NHALEN
keybyte = A (65)
65 mod 2 = 1
swap = YES [B and H]

Description of the algorithms

Block Cipher

Swap

Algorithm

1. Let 'start' and 'end' be pointers which point to the starting and ending byte of the XORed output string (*Note:* 'end' points to the end of the string and not the block.Eg. 'start' begins at 0 and 'end' begins at 7 if there are 8 bytes in the XORed output string)
2. For each byte of the key, starting from the left most byte or 0th byte, you calculate the following : (ASCII value of the byte or character)mod2. This would give you either 0 or 1.
3. If the value is 0 you do not swap anything and move to the next byte of the plaintext by incrementing the 'start' pointer. Otherwise, you swap the byte pointed by the 'start' pointer with that pointed by the 'end' pointer. Then increment the 'start' pointer so that it points to the next higher byte and decrement the 'end' pointer so that it points to the next lower byte.
4. If the keysize is exhausted, you restart from the first byte of the key.
5. This process is carried on until the 'start' and the 'end' pointers collide or crossover.

key = VA**N**HALEN
keybyte = N (78)
78 mod 2 = 0
swap = NO

Description of the algorithms

Block Cipher

Swap

Algorithm

1. Let 'start' and 'end' be pointers which point to the starting and ending byte of the XORed output string (*Note:* 'end' points to the end of the string and not the block.Eg. 'start' begins at 0 and 'end' begins at 7 if there are 8 bytes in the XORed output string)
2. For each byte of the key, starting from the left most byte or 0th byte, you calculate the following : (ASCII value of the byte or character)mod2. This would give you either 0 or 1.
3. If the value is 0 you do not swap anything and move to the next byte of the plaintext by incrementing the 'start' pointer. Otherwise, you swap the byte pointed by the 'start' pointer with that pointed by the 'end' pointer. Then increment the 'start' pointer so that it points to the next higher byte and decrement the 'end' pointer so that it points to the next lower byte.
4. If the keysize is exhausted, you restart from the first byte of the key.
5. This process is carried on until the 'start' and the 'end' pointers collide or crossover.

key = VAN**H**ALEN
keybyte = H (72)
72 mod 2 = 0
swap = NO

Description of the algorithms

Block Cipher

Swap

Algorithm

1. Let 'start' and 'end' be pointers which point to the starting and ending byte of the XORed output string (*Note:* 'end' points to the end of the string and not the block.Eg. 'start' begins at 0 and 'end' begins at 7 if there are 8 bytes in the XORed output string)
2. For each byte of the key, starting from the left most byte or 0th byte, you calculate the following : (ASCII value of the byte or character)mod2. This would give you either 0 or 1.
3. If the value is 0 you do not swap anything and move to the next byte of the plaintext by incrementing the 'start' pointer. Otherwise, you swap the byte pointed by the 'start' pointer with that pointed by the 'end' pointer. Then increment the 'start' pointer so that it points to the next higher byte and decrement the 'end' pointer so that it points to the next lower byte.
4. If the keysize is exhausted, you restart from the first byte of the key.
5. This process is carried on until the 'start' and the 'end' pointers collide or crossover.

key = VANH**A**LEN
keybyte = A (65)
65 mod 2 = 1
swap = YES [E and G]

Description of the algorithms

Block Cipher

Swap

Algorithm

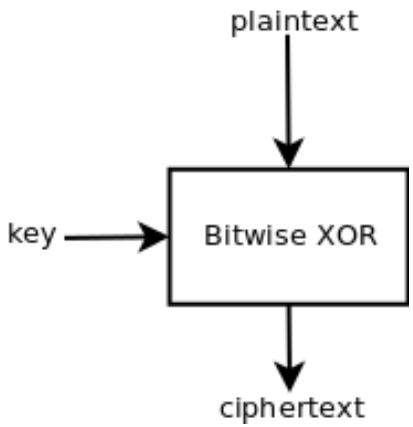
1. Let 'start' and 'end' be pointers which point to the starting and ending byte of the XORed output string (*Note:* 'end' points to the end of the string and not the block.Eg. 'start' begins at 0 and 'end' begins at 7 if there are 8 bytes in the XORed output string)
2. For each byte of the key, starting from the left most byte or 0th byte, you calculate the following : (ASCII value of the byte or character)mod2. This would give you either 0 or 1.
3. If the value is 0 you do not swap anything and move to the next byte of the plaintext by incrementing the 'start' pointer. Otherwise, you swap the byte pointed by the 'start' pointer with that pointed by the 'end' pointer. Then increment the 'start' pointer so that it points to the next higher byte and decrement the 'end' pointer so that it points to the next lower byte.
4. If the keysize is exhausted, you restart from the first byte of the key.
5. This process is carried on until the 'start' and the 'end' pointers collide or crossover.

- s and e point to the same character. So end swapping process.

Description of the algorithms

Stream Cipher

1. You will XOR your input stream with the following data
   - The key is (in Hex) "0123456789ABCDEF"
   - Bit 0 from the input file will be XORed with Bit 0 from the key
   - Bit 1 from the input file will be XORed with Bit 1 from the key
   - ...
   - Bit n-1 from the input file will be XORed with Bit n-1 from the key
2. If you reach the end of the key, start over from the beginning.
3. Note on the number of the bits. Since this is a stream, bit 0 is the first bit to arrive, bit 1 is the second bit and so forth. That means the leftmost bit of the first byte is bit 0. Same with the key.
4. No padding is required.



Grading and Deliverables

You should submit your makefile, your C or C++ programs, and a README to the assignment in Canvas. Remember: It is your responsibility to test your on the state capital machines - that is where the TA will grade it.
You will be graded on 4 criteria.

1. Clarity of README, ability to compile and run.
2. Encrypt a plaintext and decrypt it.
3. Encrypt and compare the ciphertext with that produced by us using the same plaintext and key.
4. Decrypt a random ciphertext using symmetric key which was used to encrypt it.