**A gold text on a black background

Description automatically generatedCS 430 DL: Assignments**

**Programming Assignment 1**

**CS 356: Computer Security  
Version 1.0**

**Programming Assignment 1**

The purpose of this assignment is to familiarize the students with the concept of block ciphers and stream ciphers. Block ciphers work on blocks of *n* bits at a time. Stream ciphers work on streams *one bit at a time*. Your program will read in an input file and apply the algorithms described below. It will then generate an output file.

**Requirements**

1. This program should be written in C or C++.
2. Your submittal should include a Makefile.
3. 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.
   * 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.
   * The second argument is the input file name.
     + Your program should terminate with an appropriate error message if the file does not exist.
   * The third argument is the output file name.
   * The fourth argument is the keyfile
     + The keyfile contains the symmetric key in ASCII format ("COMPUTER76543210"). The keysize for Block Encryption Mode would be 128 bits (16 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. The keyfile will not contain a terminating "\n"(newline) character.
   * The fifth argument is the 'mode of operation' which can be either 'E' for encryption or 'D' for decryption.
     + Altough the same 'E' or 'D' module can be used for Stream Ciphers, the same does not apply for Block Ciphers as will be explained below in the 'Description of the algorithms' section. In the 'E' mode you would encrypt a plaintext using a key and produce a ciphertext. We can then evaluate your encrypted output against the output produced by our own program with the same key. 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.
4. 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.
5. Your file will write the output file based on the arguments.
   * Your file should be in 128-bit blocks in block mode.
     + This means you must pad blocks that are not 128 bits in length.
     + Each padding byte should be 0X81. 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 0X81 in hex or 129 in decimal. Becasue, the input file is in ASCII and the padding value (0X81) falls outside the ASCII (0-127) range, it allows us distinguish padding bytes from plaintext bytes.

 Your file should be in 8-bit multiples in stream mode.

 Your file will be read in by a program that isn't yours to test.

 If the input file is empty, the output file should be empty.

**Description of the algorithms**

**Block cipher**

1. The **encryption** would be a three step process : pad (if required) -> encrypt (using XOR) -> swap
2. Your block size will be 128 bits. As mentioned earlier, if a block is found to be less than 128 bits, it should be padded with as many (0X81) bytes so as to make 128 bits or 16 bytes.
3. Your algorithm will XOR the 128 bit data block with the 128 bit key in a bitwise manner, i.e. each bit of the key starting from the left most bit will be XORed with each bit of a 128 bit data block, starting from the left hand side.
4. Your algorithm will then swap bytes of the XORed output using the following algorithm.
   * Let 'start' and 'end' be pointers which point to the start and end of the XORed output string.
   * 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.
   * If the value is 0 you do not swap anything and move to the next byte of the ciphertext 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. If the keysize is exhausted, you restart from the first byte of the key. This process is carried on until the 'start' and the 'end' pointers collide. The swap process stops and produces the required encrypted output.
5. The **decryption** process is also of three steps but proceeds in the reverse order as the encryption process : swap -> decrypt (using XOR) -> remove padding (if required). The decrypted ouput should be written to the output file. Because the decryption process occurs in the reverse direction of the encryption process, you cannot use the same module to perform both, altough the underlying logic will be the same.

**Stream cipher**

1. You will XOR your input stream with the following data
   * The key is (in ASCII, again) "COMPUTER76543210".
   * 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. Encryption and decryption will be performed with the same module that you design for the Stream Cipher. In other words, you do not have have to design separate modules for encryption and decryption in the Stream Encryption Mode.

**What to turn in**

You should submit your makefile, your C or C++ programs, and a README to the assignment in Canvas. The README should contain everything the TA should know to execute your program. **Remember: It is your responsibility to test your on the state capital machines - that is where the TA will grade it.**

**Late Policy**

There will be a 10% deduction per day for programs turned in during the late period. No submissions will be accepted after the late period.

**Copyright © 2019: Colorado State University for CS 356.** All rights reserved.