The Black Box

Premise:

The company you work for has acquired a contract from the Department of Defense to open a Black Box captured from enemy intelligence personnel. Mounted on the Black Box is a LED flashing an encrypted passcode and a light sensor waiting for the decrypted passcode to be flashed back. Fortunately, a defected enemy scientist has revealed the data format and the decryption process.

Objective:

Your task is opening the Black Box by gathering the data provided by the Black Box's flashing LED, processing the data, decrypting the data, and flashing a LED with the correct passcode into the Black Box's light sensor. A correct passcode will automatically open the Black Box.

Provided Information:

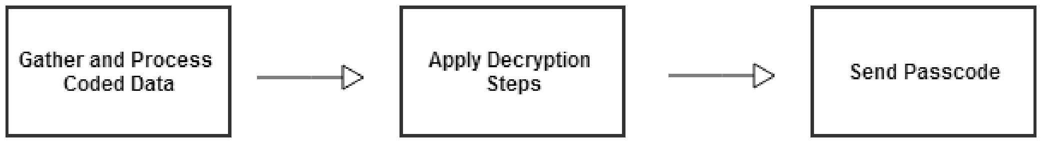
The data provided by the defected enemy scientist indicates that the LED flashes tri-state pseudo serial formatted binary. Data blocks are terminated with the character '#', are formatted in bytes, have a maximum length of ten bytes, and repeat the same encrypted code. The encrypted code and expected passcode changes when the Black Box is power cycled. Correct data blocks viewed as ASCII will spell the last name of a historical scientist.

Once an isolated data block is in decimal format, the passcode can be found by applying the following decryption steps in order: drop the maximum values(s), drop the first even number, reverse the data order, and subtract from all values the rounded standard deviation of the values.

The Black Box expects the passcode to be formatted similar to the flashing LED data.

Three Phases:

The process for opening the Black Box will occur in three phases:



The Sensor System & Black Box:

The sensor system consists of a light brightness sensor, a user controlled output LED, and an output LED replicating the brightness level detected on the brightness sensor. The sensor system is powered by connecting the sensor to the PC via USB.

The Black Box is powered by connecting the Black Box to the PC via USB.

Phase 1: Gather and Process Coded Data

Initialize the Sensor System

Before beginning to gather data from the flashing LED, the sensor system must be initialized by MATLAB.

Add the following code to initialize the sensor system:

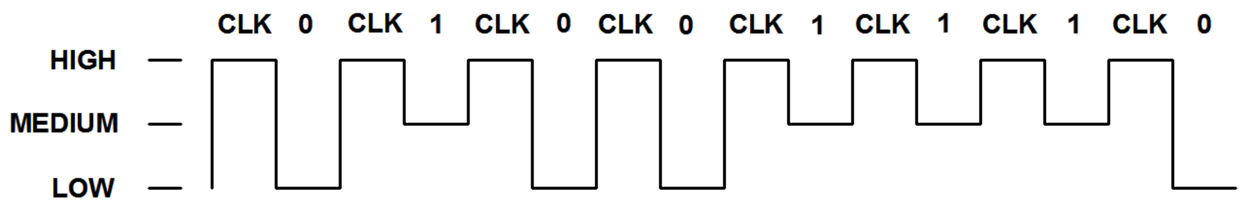
```
% close all serial ports; prevents a port 'in use' error
delete(instrfindall)
% create serial port interface with the indicated COM port number
socket = serial('COM12','BAUD',57600);
% open serial port
fopen(socket);
% read socket once to clear the empty buffer error
fscanf(socket,'%s')
% sensor board needs time to initialize
pause(2);
```

Note: The Black Box's USB cable should be removed from the PC to ensure the correct COM port number is chosen.

Note: Replace the 'X' in 'COMX' with the number of the COM port the sensor system is connected. Find the COM port by opening Window's Device Manager, clicking on "Ports (COM &LPT)", and finding "USB Serial Port (COMX)" where X is the number of the COM port. See appendix for Device Manager example image .

Collect Raw Brightness Data

The following is an example of tri-state pseudo serial formatting using a LED's brightness for data encoding:



Every other pulse is the clock. The bit after a clock pulse is a binary data value of either a zero or one. The above example is 01001110 in binary and the converted decimal value is 78. A decimal value of 78 represents the ASCII character "N".

Use the following code to read the relative brightness of the LED:

```
% read data in the serial buffer (formatted as a string)
% convert the string into a number
data = str2num(fscanf(socket,'%s'));
```

In the case of the Black Box, binary data is coded on the LED as a function of brightness over time. Because the sensor reads relative brightness over a large range, the values of the brightness will fall between 0 and 255 where 0 is the lowest brightness and 255 is the highest brightness.

Note: Considering the provided data states the encrypted code is no more than 10 bytes, in order to insure a whole block of data is captured, at least 840 reads of the light sensor is required. (2 (pulses per bit) * (8 (bits per byte) * 10 (total bytes of data) * 2.5 (data duplication rate) * 2.1 (required data overlap coefficient) = 840)

Hint: Create a loop to store multiple samples of the brightness data.

Normalize Raw Data

Because the sensor reads relative brightness of the flashing LED, the data needs to be normalized into binary and clock equivalents.

Note: After conversion, low brightness = 0, medium brightness = 1, and high brightness = 2 (clock).

Hint: Graph the raw data for a quick visual inspection and extract a range that the three brightness levels fall.

Remove Duplicated Data

Because the flashing LED changes state every 25ms and the light sensor provides brightness samples every 10ms, there will be repeated data. Remove the repeated data.

Hint: Because the clock pulse is every other bit, there will be no sequentially repeating bits.

Remove Clock Data

Considering binary code consists of only ones and twos, the clock data must be removed from the data.

Hint: Avoid making a loop by using MATLAB ‘s built in matrix indexing.

Extract Pertinent Data Block

Because data blocks repeat in the data stream and data collection likely has not begun precisely at the beginning of a data block, find and extract a whole data block from the data.

Note: The end of each data block is terminated with the character ‘#’.

Format Data into Bytes

Considering the data is formatted into bytes, convert the binary data block into a group of bytes.

Hint: MATLAB has a built in binary to byte conversion function.

Convert Bytes into Decimal

Because the decryption process calls for the data to be in decimal format, convert the data bytes into their decimal representation.

Hint: MATLAB has a built in byte to decimal conversion function.

Convert Decimal into ASCII

As a debugging step, the defected enemy scientist indicated correctly read data blocks viewed in ASCII will read as a last name of a historical scientist. Convert the decimal data into ASCII to confirm the data and code for correctness.

Hint: MATLAB has built in decimal to ASCII conversion function.

Phase 2: Apply Decryption Steps

Once the data is formatted in decimal, the following decryption steps can be applied in order:

Step 1: Drop the maximum values(s).

Step 2: Drop the first even number.

Step 3: Reverse the data order.

Step 4: Subtract from all values the rounded standard deviation of the values.

Hint: Several of the decryption steps will not require loops and built in MATLAB functions will suffice.

Phase 3: Send Passcode

The defected enemy scientist indicated the Black Box requires the passcode to be sent in the same tri-state pseudo serial formatted binary.

There are two approaches in sending the passcode by flashing the LED:

1) Create a loop with condition statements that inject clock signals and null terminators within the sending data stream.

2) Create a single data array with built in clock signals and null terminators then create a continuous loop.

The following steps use the second approach.

Convert Decimal into Binary

Convert individual bytes into a binary representation.

Hint: MATLAB has a built in decimal to binary conversion function. However, read carefully the function description and notice the result is a string. Strings are composed of ASCII characters. Since ASCII characters have a decimal equivalent we need to perform another conversion. Look at a decimal to ASCII conversion chart which will indicate a solution to converting ASCII 0's and 1's to decimal 0's and 1's.

Convert the Matrix into a Continuous Array

If the binary data is formatted into a n:8 matrix (where n is the number of bytes), convert the matrix into a single continuous array of 1:n (where n is the number of bits).

Hint: MATLAB has a built in matrix to array conversion.

Add a Terminator

Add a terminator at the end of the continuous binary array.

Note: The terminator will be the character '#'.

Hint: Convert '#' into the binary equivalent.

Add Clock Values

Add a clock value of 2 after every bit.

Send Passcode

The Black Box's light sensor expects code in the same format as the original data where 0's are low brightness, 1's are medium brightness, and the clock (a value of 2) is high brightness.

The following code controls the sensor system's output LED brightness levels:

```
stopasync(socket) % required for continuous data sending
fprintf(socket,'%c','0') % activate LED at low brightness

stopasync(socket) % required for continuous data sending
fprintf(socket,'%c','1') % activate LED at medium brightness

stopasync(socket) % required for continuous data sending
fprintf(socket,'%c','2') % activate LED at high brightness
```

Note: The Black Box's light sensor will only detect correct data when the sending LED's state changes occur at a rate of 25ms or greater.

Hint: In order to make sure the Black Box reads a full data block, continuously send the data block.

Once the correct passcode is properly sent to the Black Box, the Black Box will automatically open and you will be successful at completing the objective.

Appendix

Tips for Success:

Reading a few steps past the current step may give better insight on the purpose of the current step.

Write code in "cells" for quicker execution and efficient step by step debugging process.

For every decoding/decryption step save newly decoded/ decryption data in a unique array. This allows displaying pre manipulated data side by side with manipulated data for verification that the code is working as intended.

Definitions:

Terminator - is a byte containing a system specified value that is placed at the end of data block that indicates the data block's end.

ASCII - American Standard Code for Information Interchange – is a character-encoding scheme that encodes 128 characters, including 0-9, a-z, A-Z, punctuation symbols, and system codes, into a binary and decimal equivalent value.

Suggested Functions:

```
plot() – displays a graph based on the data provided
length() – returns length of an array
zeros() – creates an array or matrix filled with zeros
ones() – creates an array or matrix filled with ones
char() – displays the ASCII equivalent of a decimal value
max() – returns max value of an array or matrix
rem() – returns remainder after division
fliplr() – flips an array or matrix from left to right
round() – rounds a number to the nearest whole value
std() – returns the standard deviation of an array or matrix
bin2dec() – converts binary values to their decimal equivalent
dec2bin() - converts decimal values to their binary equivalent
pause() – halts code execution for the specified amount of time
```

Coding Tips & Tricks:

```
% create a code cell for cell by cell execution and easy debugging
%%

% return the b element of the array or matrix a
a(b)

% creates an array of index values that equal one, then the array stores only
% values that correspond with those index values
b = a(a == 1)

% perform a loop for 10 round trips and increment i by one on each trip
for i = 1:10
   % looped code
```

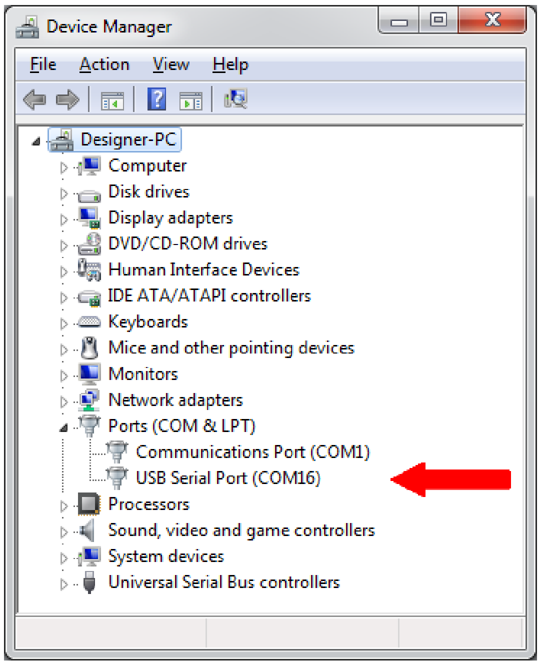
```
end

% exit the loop immediately
break;

% execute when a > b and a < c
if (a > b) && (a < c)
    % code to execute
end

% create the array [0, 1, 0, 1, 0, 1, 0, 1, 0, 1]
a(2:2:10) = ones(1,5)
```

Device Manager:



Designer-PC
Computer
Disk drives
Display adapters
DVD/CD-ROM drives
Human Interface Devices
IDE ATA/ATAPI controllers
Keyboards
Mice and other pointing devices
Monitors
Network adapters
Ports (COM & LPT)
Communications Port (COM1)
USB Serial Port (COM16)
Processors
Sound, video and game controllers
System devices
Universal Serial Bus controllers