Nicky Nocerino

ECE102 Project Report

5/11/16

**Proposal:**

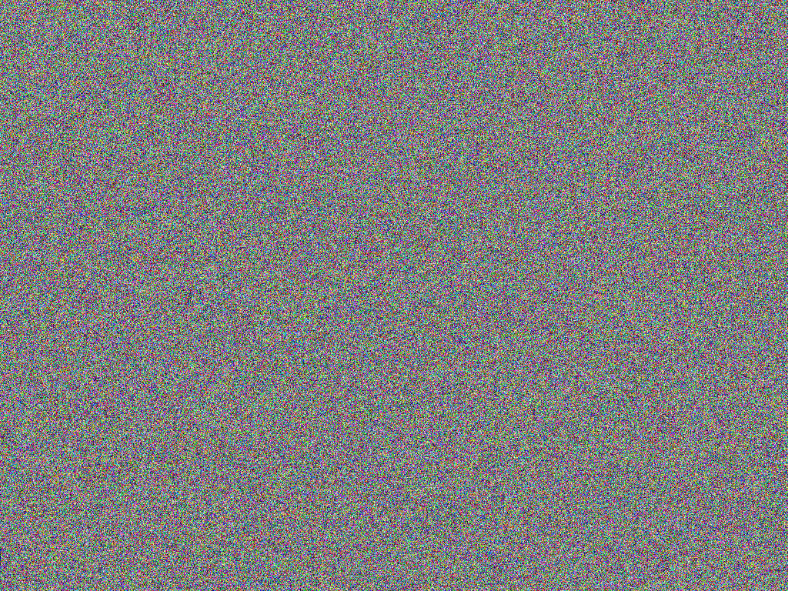
While I initially proposed to do an encryption based project, after completing C challenge 9, and with the similarity between my idea and C challenge 12 I would like to do something different. In order to do this I would like to propose an idea for encrypting files into bit map images. This project would involve 2 programs, one that would be able to take in a message and a key and then use it to generate a bit map image, the other would take in a bit map image and a key and would return the original message. the key in this instance does no serve to alter the data like in the other types of encryption, but instead hide it in what otherwise appears to be a random bitmap. Since a bit map allows you to store 3 integers between 0-255 in each pixel, it would be extremely difficult to find the hidden data amongst the junk data without the key to tell you were to look.

**Goals:**

* Create a method of encryption using bitmaps.
  + Must be hard to notice
  + Must be difficult to break
  + Must be fairly simple to implement
* Create 2 programs using this method of encryption
  + Encrypt
  + Decrypt
  + Both must take in the same key

**Project Summary:**

Project consists of 2 programs, bitcrypt.c and bitdecrypt.c. Through the combination of these 2 programs simple text files can be passed back and forth in a manner so that it is disguised as an image in the bitmap format for all intermediary steps. Bitcrypt.c hides data within a bitmap based on the 3 part key provided, and bitdecrypt.c uses the same 3 part key to decode the bitmap, drawing the initial information out.



**Usage:**

* To Encrypt
  + ./bitcrypt.exe example.txt key1 key2 key3
    - Argument 1: name of text file you wish to encrypt
    - Argument 2: key1 any int 1-999
    - Argument 3: key2 any int 1-999
    - Argument 4: key3 any int 1-999
  + Will create a file called bitcrypted.bmp
* To Decrypt
  + ./bitdecrypt.exe bitcrypted.bmp key1 key2 key3
    - Argument 1: name of bitmap to decrypt
    - Argument 2: key1
    - Argument 3: key2
    - Argument 4: key3
  + Will output a file debitcrypted.txt with the same content as the original file passed in

**Problems:**

* Only works for text files
* Relatively easy to break, 999^3 possible keys

**Sources:**

<https://en.wikipedia.org/wiki/BMP_file_format>

<https://www.daubnet.com/en/file-format-bmp>

<http://www.fileformat.info/format/bmp/egff.htm>

**Code:**

**Bitcrypt.c**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <time.h>

#include <unistd.h>

#define WIDTH 1024

#define HEIGHT 768

void drawbmp (char \* filename);

int red\_pixel[WIDTH][HEIGHT];

int green\_pixel[WIDTH][HEIGHT];

int blue\_pixel[WIDTH][HEIGHT];

int main (int argc, char \* argv[])

{

int x, y; // Source coordinates of particle being tracked

char filename[200];

int mcount;

int scount;

FILE\* input;

int rkey;

int gkey;

int bkey;

int mlength;

mlength=atoi(argv[2]);

char message[mlength+1];

char temp[500];

srand(time(NULL));

for (x = 0; x < WIDTH; x++)

{

for (y = 0; y < HEIGHT; y++)

{

red\_pixel[x][y] = rand()%255;

green\_pixel[x][y] =rand()%255;

blue\_pixel[x][y] = rand()%255;

}

}

// green\_pixel[0][1]=10;

input=fopen(argv[1],"r");

while(fgets(temp,mlength+1,input))

{

sprintf(message,"%s%s",message,temp);

}

mcount=0;

scount=1;

rkey=atoi(argv[3]);

gkey=atoi(argv[4]);

bkey=atoi(argv[5]);

for (x = 0; x < WIDTH&& mcount<mlength; x++)

{

for (y = 0; y < HEIGHT&& mcount<mlength; y++)

{

if(scount%rkey==0)

{

red\_pixel[x][y] = message[mcount];

mcount++;

}

if(scount%gkey==0)

{

green\_pixel[x][y] =message[mcount];

mcount++;

}

if(scount%bkey==0)

{

blue\_pixel[x][y] = message[mcount];

mcount++;

}

scount++;

}

}

sprintf(filename, "bitcrypted.bmp");

drawbmp(filename);

printf("Done.\n");

return 0;

}

void drawbmp (char \* filename)

{

unsigned int headers[13];

FILE \* outfile;

int extrabytes;

int paddedsize;

int x; int y; int n;

int red, green, blue;

extrabytes = 4 - ((WIDTH \* 3) % 4); // How many bytes of padding to add to each

// horizontal line - the size of which must

// be a multiple of 4 bytes.

if (extrabytes == 4)

extrabytes = 0;

paddedsize = ((WIDTH \* 3) + extrabytes) \* HEIGHT;

// Headers...

// Note that the "BM" identifier in bytes 0 and 1 is NOT included in these "headers".

headers[0] = paddedsize + 54; // bfSize (whole file size)

headers[1] = 0; // bfReserved (both)

headers[2] = 54; // bfOffbits

headers[3] = 40; // biSize

headers[4] = WIDTH; // biWidth

headers[5] = HEIGHT; // biHeight

// Would have biPlanes and biBitCount in position 6, but they're shorts.

// It's easier to write them out separately (see below) than pretend

// they're a single int, especially with endian issues...

headers[7] = 0; // biCompression

headers[8] = paddedsize; // biSizeImage

headers[9] = 0; // biXPelsPerMeter

headers[10] = 0; // biYPelsPerMeter

headers[11] = 0; // biClrUsed

headers[12] = 0; // biClrImportant

outfile = fopen(filename, "wb");

//

// Headers begin...

// When printing ints and shorts, we write out 1 character at a time to avoid endian issues.

//

fprintf(outfile, "BM");

for (n = 0; n <= 5; n++)

{

fprintf(outfile, "%c", headers[n] & 0x000000FF);

fprintf(outfile, "%c", (headers[n] & 0x0000FF00) >> 8);

fprintf(outfile, "%c", (headers[n] & 0x00FF0000) >> 16);

fprintf(outfile, "%c", (headers[n] & (unsigned int) 0xFF000000) >> 24);

}

// These next 4 characters are for the biPlanes and biBitCount fields.

fprintf(outfile, "%c", 1);

fprintf(outfile, "%c", 0);

fprintf(outfile, "%c", 24);

fprintf(outfile, "%c", 0);

for (n = 7; n <= 12; n++)

{

fprintf(outfile, "%c", headers[n] & 0x000000FF);

fprintf(outfile, "%c", (headers[n] & 0x0000FF00) >> 8);

fprintf(outfile, "%c", (headers[n] & 0x00FF0000) >> 16);

fprintf(outfile, "%c", (headers[n] & (unsigned int) 0xFF000000) >> 24);

}

//

// Headers done, now write the data...

//

for (y = 0; y < HEIGHT; y++) // BMP image format is written from bottom to top...

{

for (x = 0; x < WIDTH; x++) // left to right

{

red = red\_pixel[x][y];

green = green\_pixel[x][y];

blue = blue\_pixel[x][y];

if (red > 255)

red = 255;

if (red < 0)

red = 0;

if (green > 255)

green = 255;

if (green < 0)

green = 0;

if (blue > 255)

blue = 255;

if (blue < 0)

blue = 0;

// Also, it's written in (b,g,r) format...

fprintf(outfile, "%c", blue);

fprintf(outfile, "%c", green);

fprintf(outfile, "%c", red);

}

if (extrabytes) // See above - BMP lines must be of lengths divisible by 4.

{

for (n = 1; n <= extrabytes; n++)

{

fprintf(outfile, "%c", 0);

}

}

}

fclose(outfile);

return;

}

**Bitdecrypt.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#define WIDTH 1024

#define HEIGHT 768

int rdata[WIDTH][HEIGHT];

int gdata[WIDTH][HEIGHT];

int bdata[WIDTH][HEIGHT];

int main( int argc, char\* argv[])

{

FILE\* input;

FILE\* output;

int i;

int j;

int length=atoi(argv[2]);

// int junk;

int count = 0;

long scount=1;

char message[length];

input = fopen(argv[1], "r");

int rkey=atoi(argv[3]);

int gkey=atoi(argv[4]);

int bkey=atoi(argv[5]);

fseek(input,54,SEEK\_SET);

printf("header stripped\n");

for(i=0;i<HEIGHT;i++)

{

for(j=0;j<WIDTH;j++)

{

bdata[j][i] = getc(input);

gdata[j][i] = getc(input);

rdata[j][i] = getc(input);

}

}

// printf("%d",gdata[0][1]);

fclose(input);

printf("data got, file closed\n");

while(count<length)

{

for(i=0;i<WIDTH&&count<length;i++)

{

for(j=0;j<HEIGHT&&count<length;j++)

{

if(scount%rkey==0)

{

message[count]=rdata[i][j];

count++;

}

if(scount%gkey==0)

{

message[count]=gdata[i][j];

count++;

}

if(scount%bkey==0)

{

message[count]=bdata[i][j];

count++;

}

scount++;

}

}

}

output=fopen("debitcrypted.txt","w");

fputs(message,output);

fclose(output);

return 0;

}