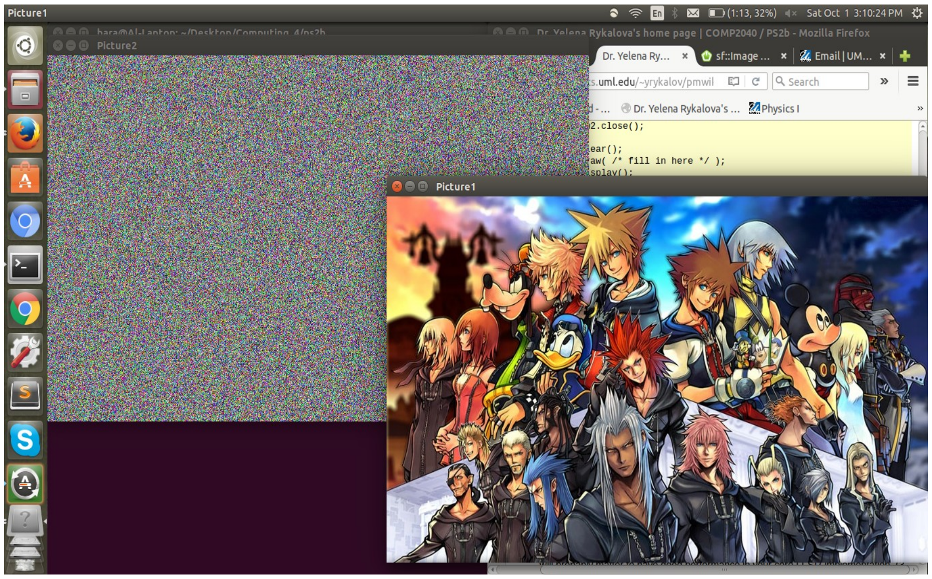
<u>PS2b</u>:

Ps2b assignment was make a main client that uses LFSR to encrypt and decrypt pictures. To accomplish this task, we needed to pass a image. Each pixel would be passed go through LFSR.

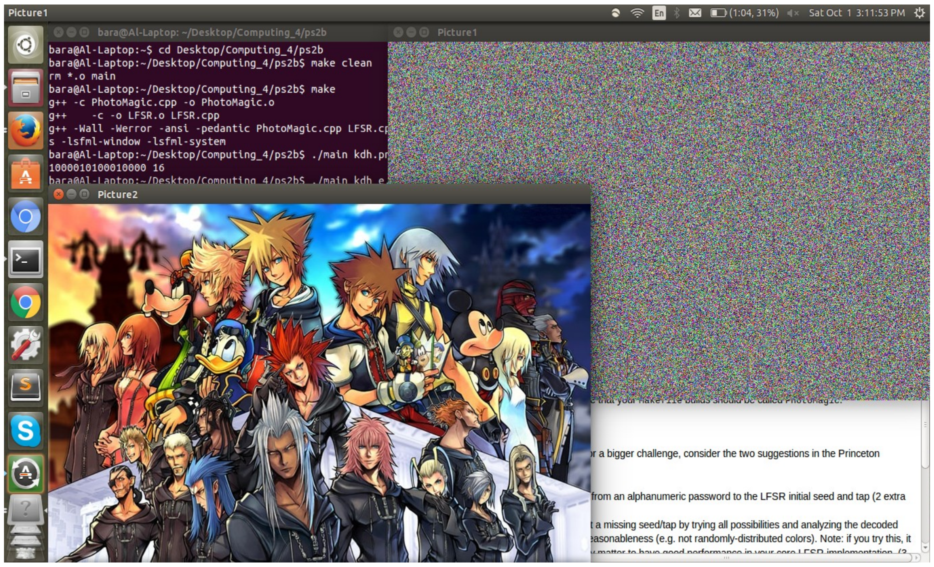
To accomplish this task, I created a main and created a object of class LFSR. The accepted input of the image name would then be stored. I then used a function that would call the generate function and XOR the green, blue, and red. This would be done using the SFML Color type. Then create the output file and insert the new encrypted picture.

Learning about the amount of pixels in different types pictures was helpful to know in the future. XOR was nice to know what type of algorithm it is. Loading images and creating new image files was another skill I learned from this project.


Picture1
Picture2
Dr. Yelena Rykalova's home page | COMP2040 / PS2b - Mozilla Firefox
Search
Physics I

```
2.close();

ear();
aw( /* fill in here */ );
splay();
```



```
bara@Al-Laptop:~$ cd Desktop/Computing_4/ps2b
bara@Al-Laptop:~/Desktop/Computing_4/ps2b$ make clean
rm *.o main
bara@Al-Laptop:~/Desktop/Computing_4/ps2b$ make
g++ -c PhotoMagic.cpp -o PhotoMagic.o
g++    -c -o LFSR.o LFSR.cpp
g++ -Wall -Werror -ansi -pedantic PhotoMagic.cpp LFSR.cp
s -lsfml-window -lsfml-system
bara@Al-Laptop:~/Desktop/Computing_4/ps2b$ ./main kdh.pr
1000010100010000 16
bara@Al-Laptop:~/Desktop/Computing_4/ps2b$ ./main kdh.e
```

r a bigger challenge, consider the two suggestions in the Princeton

from an alphanumeric password to the LFSR initial seed and tap (2 extra

t a missing seed/tap by trying all possibilities and analyzing the decoded
easonableness (e.g. not randomly-distributed colors). Note: if you try this, it

```
cc = g++
#all

all : PhotoMagic

PhotoMagic : PhotoMagic.o LFSR.o
	$(cc) -Wall -Werror -ansi -pedantic PhotoMagic.cpp LFSR.cpp -o PhotoMagic -lsfml-graphics -lsfml-window -lsfml-system

PhotoMagic.o : LFSR.hpp
	$(cc) -c PhotoMagic.cpp -o PhotoMagic.o

LFSR : LFSR.cpp LFSR.hpp
	$(cc) -c LFSR.cpp -o LFSR.o

clean:
	rm *.o main

```

```
/*
Name: Albara Mehene
Date: 10/1/2016
Computing IV

*/

#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>

#include "LFSR.hpp"

sf::Image transform(sf::Image picture, LFSR lfsr);

int main(int argc, char* argv[])
{
        if(argc < 5){

                        std::cout << "input-file.png, output-file.png, seed, tap" << std::endl;
                        return -1;
        }

        std::string input = argv[1];
        std::string output = argv[2];
        std::string i_seed = argv[3];
        int i_tap = atoi(argv[4]);

        sf::Image image;
        if (!image.loadFromFile(input))
                return -1;

        sf::Image image_e = image;
        sf::Image temp_e;

        //Pass the seed and tap function
        LFSR l(i_seed, i_tap);
        temp_e = transform(image_e, l);


        sf::Vector2u size = image.getSize();
        sf::Vector2u size2 = temp_e.getSize();
        sf::RenderWindow window(sf::VideoMode(size.x, size.y), "Picture1");
        sf::RenderWindow window1(sf::VideoMode(size2.x, size2.y), "Picture2");




        sf::Texture texture;
        texture.loadFromImage(image);

        sf::Texture texture_e;
        texture_e.loadFromImage(temp_e);

        sf::Sprite sprite;
        sprite.setTexture(texture);

        sf::Sprite sprite_e;
        sprite_e.setTexture(texture_e);

        while (window.isOpen() && window1.isOpen())
        {
                sf::Event event;
                while (window.pollEvent(event))
                {
                        if (event.type == sf::Event::Closed)
                                window.close();
                }
                while (window1.pollEvent(event))
                {
                        if (event.type == sf::Event::Closed)
                                window1.close();
                }


                window.clear(sf::Color::White);
                window1.clear(sf::Color::White);
                window.draw(sprite);
                window1.draw(sprite_e);
                window.display();
                window1.display();
        }

        // fredm: saving a PNG segfaults for me, though it does properly
        //   write the file
        if (!temp_e.saveToFile(output))
                return -1;

        return 0;
}

sf::Image transform(sf::Image picture, LFSR lfsr){
        // p is a pixel
        sf::Color p;
        int temp;
        sf::Vector2u size = picture.getSize();

        // create photographic negative image of upper-left 200 px square
        for (unsigned int x= 0; x < size.x; x++) {
                for (unsigned int y = 0; y < size.y; y++) {
                        p = picture.getPixel(x, y);

                        temp = lfsr.generate(8);
                        p.r = p.r ^ temp;

                        temp = lfsr.generate(8);
                        p.g = p.g ^ temp;

                        temp = lfsr.generate(8);
                        p.b = p.b ^ temp;

                        picture.setPixel(x, y, p);
                }
        }
        return picture;
}




```

```


#include <iostream>
#include <string>
#include <cmath>


class LFSR {

public:
  LFSR(std::string seed_, int tap_);
  ~LFSR();
  int step();
  int generate(int k);

  friend std::ostream& operator<< (std::ostream &out, LFSR &lfsr);

private:
  std::string seed;
  int tap;
};

```

```
/*
Name: ALbara Mehene
Date: 9/25/2016
Computing IV

*/


#include <iostream>
#include <string>
#include <cmath>
#include "LFSR.hpp"

//constructor
LFSR::LFSR(std::string seed_, int tap_){

        seed = seed_;
        tap = tap_;

}


int LFSR::step(){
        int bit;
        int size;

        size = seed.length();//stored the amount of elements

        bit =  seed.at(0) ^ seed [size - tap - 1]; //Took the total elements and subtracted by the tap and by 1


        seed.erase(0, 1); // erased the front element

        if(bit == 1){//condition if its 1, it would return the chracter 1
                seed.push_back('1');
        }
        else{//returns 0 if its anything else
                seed.push_back('0');
        }
        //returns bit to test the test.cpp
        return bit;
}

int LFSR::generate(int k){
        int temp = 0;

        //Condition to test the generate function in test.cpp
        for(int i = k - 1; i >= 0; i--){
                if(step() == 1){
                        temp += pow(2,i);
                }
        }
        return temp;

}
//prints out the string if I were to use the a main. It was not required in this assigment
std::ostream& operator<< (std::ostream &out, LFSR &lfsr){
        out << lfsr.seed;
        return out;
```

```cpp
}



LFSR::~LFSR(){

}




```