Steganog Lab Write-Up

David Ericson

How my algorithm works:

(Prime Iterator)

My prime iterator works fairly simply. The next set of non-primes are marked out every time next is called. So the first time all of the multiples of two are crossed out. Next time all of the multiples of three. This is done because in the case that you don’t use all the primes, which happens all the time in steganography, you won’t have to use the extra massive overhead to load in all of the unused prime numbers.

(Reading)

Reading is done fairly simply. I create a prime iterator of the width\*height of the image file. Then I check if I have a next prime and that I haven’t hit an escape character yet (I’ll get into that later). I grab the pixel at the prime location, after figuring out the correct width and height values. I take the color value then use bitwise operations on each of the red, green, and blue values to get the total ascii value of the pixel’s hidden character ( after I add 32). If the character does not belong to a list of escape characters then it is added to the message string. The escape characters are simply a list of garbage characters that few people would use on an image. These characters allow you to escape the while loop early and not spend time calculating unnecessary prime numbers.

(Writing)

Writing is a bit trickier. I create a prime iterator of the message.length\*20. This will create enough primes to fill the required string up to several million characters. I loop through for every character that is in the message. I grab the ascii value-32 of the current character then I figure out which pixel to embed this character into. After I have the pixel I grab the red, green, and blue values and obtain their respective values except for the last two bits in each number (0-3). I then calculate the value that needs to be added to the red, green, and blue pixels with the embedded info. For red I take the ascii value and divide it by 16, since the last two values in a six bit character are 16 and 32 I have to divide the ascii by 16 to obtain the number (0-3) of red, after that I subtract the ascii value by 16\*the number I just got. For green I do the same thing except with 4. After that I am left with just the blue value. I then make a new pixel with the red, green, and blue values obtained after chopping off the bits at the end and adding on the new values for each color. After the message is finished I save and return the picture.

Help:

The idea of calculating the prime only when next was called was Jon Pearl’s idea. The escape character’s string was Drew’s idea.

Mathematical analysis:

All of my calculations are constant other than the number of times through the while loop and the time it takes to call PrimeIterator.next(). The performance of the Sieve of Eratosthenes is O(n log log n). Where n is the max value you send into your PrimeIterator. This means that assuming worst case for writing my algorithm performs at O(n log log n ) where n = width\*height.

Empirical analysis:

Reading:

|  |  |  |  |
| --- | --- | --- | --- |
| Picture Width | Picture Height | Message Length | Time |
| 256 | 256 | 77 | 3 ms |
| 4272 | 2848 | 77 | 24 ms |
| 4272 | 2848 | 3748 | 79 ms |

(For the writing time I commented out the save function call on the new picture, because it was taking about 50ms.)

Writing:

|  |  |  |  |
| --- | --- | --- | --- |
| Picture Width | Picture Height | Message Length | Time |
| 256 | 256 | 77 | 2 ms |
| 4272 | 2848 | 77 | 2 ms |
| 4272 | 2848 | 3748 | 25 ms |