Embedding Photos, coded by Photos, inside Photos: A Simple, XOR-based One-Time Pad Cryptography Algorithm for Images

Alkin Kaz

12 May 2020

1 Introduction

Privacy of communication, and hence, the field of cryptology have always been a central concern of humanity. Even in the antiquities, a wide variety of elementary methods like Caesar ciphering were developed and used [1]. When complemented with the increased computation and communication power of the modern era, this field demands further contribution to the research efforts and the arsenal of available applications. In Figure 1, we see a conventional cryptography scheme. Throughout this paper we will call the original message *message*, the fundamental parameter of the encoding and decoding algorithm *key*, the intermediate ciphered message *code*, and the decoded message *decoded* – all conventional usages.

In the Great Wars era, so-called *one-time pads* were used. The distinctive feature of this method from the other methods is the fact that the length of the key is same with the message that is desired to be encoded [1]. Although one-time pad was found to be cumbersome throughout history since the encoding key has a large size, the same argument shows the encoding power of the algorithm as well [1]. Here, we have developed an elementary

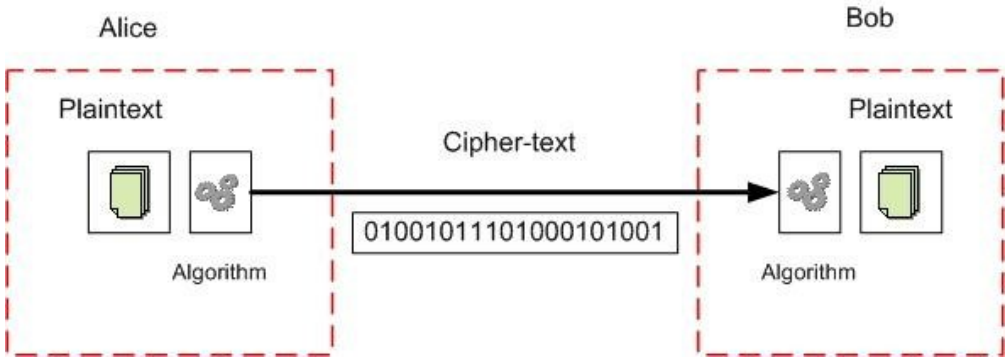

Figure 1: Conventional cryptography scheme, taken from *this* website.

XOR

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Figure 2: Truth table for XOR, taken from [2].

cryptography method to encrypt images with images by using one of them as a pad, and later, by embedding that image to a background image, we aimed to achieve visual secrecy.

2 Method and Algorithm

In the implementation of this idea, we used the bit-wise operation called *exclusive or*, or shortly, *XOR*. The truth table of this operation can be found in Figure 2. This operation has an important property that made it useful in cryptography: Given a message bitstring and key bitstring, if we XOR the message **twice** with the key, we obtain the same result [2].

Now, proceeding to the description of the algorithm, first we will describe the broad scheme. Assume that Alice wants to deliver an image to Bob (called *message*), and they have previously shared another image through a secure communication channel (called *key*). Also, through a secure channel, they have previously decided the size of the image that is to be delivered (namely, a *rowsize* x *colsize* matrix of 8-bit grayscale values), and the block size *blocksize*. Here, Alice aims to embed the result of the XOR or *message* and *key* (namely, B) to an arbitrary (but large enough) background image by the following way: She picks the initial pixel (*rowstart*, *colstart*) in background that she will embed the first pixel of XOR'ed image, and for each of the *blocksize* x *blocksize* blocks in background (starting from the designated initial pixel), she changes their first entry to the respective entry in the B image. Later, she crops out the unused parts of the background, resulting in the image *code*.

Decoding also proceeds in a similar way – Bob knows the key (also related values *rowsize* and *colsize*) and the *blocksize*. He takes the encoded image *code*, gets the first pixel of all *blocksize* x *blocksize* blocks, apply the XOR operation, and obtain the resulting image. The `encoder.m` function and `decoder.m` functions, both written in MATLAB, can be found respectively in Figures 3 and 4.

```
function code = encoder(message, key, background, rowsize, colsize, rowstart, colstart, blocksize)
%ENCODER Embeds message inside background
%   Takes the (possibly RGB) images "message" and "key",
%   resizes them to the desired rowsize & columnsize, bitwise XOR them,
%   and embeds the resulting image to the background image.

gray_message = rgb2gray(message); % message in grayscale
gray_key = rgb2gray(key); % key in grayscale
gray_background = rgb2gray(background); % background in grayscale

% scales the images to the desired size
scaled_message = imresize(gray_message, [rowsize colsize]);
scaled_key = imresize(gray_key, [rowsize colsize]);

% since blocksize and start points are given, we can calculate end points
% for the background embedding
rowend = rowstart + blocksize*rowsize - 1;
colend = colstart + blocksize*colsize - 1;

% scales the background to fit precisely to the image
code = gray_background(rowstart:rowend, colstart:colend);

% encodes the message with key by elementwise bitwise XOR'ing the pixels
B = bitxor(scaled_message, scaled_key);

% embeds the XOR'd image to the background
code(1:blocksize:end, 1:blocksize:end) = B(1:end, 1:end);

end
```

Figure 3: Encoder function

```
function message = decoder(code, key, blocksize, rowsize, colsize)
%DECODER Decodes the code image with given key
%   Detailed explanation goes here

% changes the key to grayscale and scales it
gray_key = rgb2gray(key);
scaled_key = imresize(gray_key, [rowsize colsize]);

% extracts embedded xor'd image
xored = code(1:blocksize:end, 1:blocksize:end);

% xor's again with the key to find the original message
message = bitxor(xored, scaled_key);

end
```

Figure 4: Decoder function

Figure 5: Original image, example 1.

Figure 6: Key image, example 1.

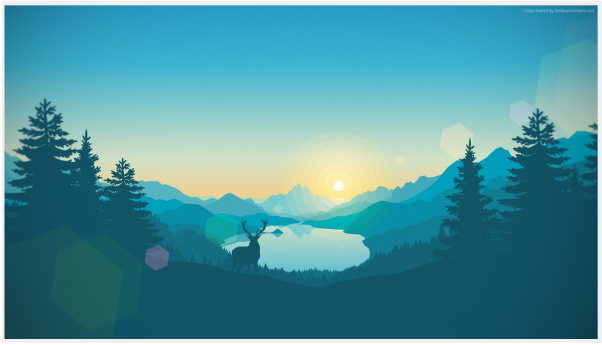

Figure 7: Background image, example 1.

Figure 8: Encoded image, example 1.

3 Results

Here, we show two demonstrations to our code. In all of these examples, we took $rowsize = colsize = 256$, $rowstart = 10$, $colstart = 256$, and $blocksize = 4$. Note that the original message is well hidden inside the encoded images in both examples. For better results, one might increase the blocksize, or select appropriate keys since keys were also found to be an important factor in hiding the original message in a better way.

4 Discussion

The elementary method we devised worked well and we were able to obtain the original images without any loss (on the gray-scale). Although this algorithm might be broken by

Figure 9: Decoded image, example 1

Figure 10: Original image, example 2.

Figure 11: Key image, example 2.

Figure 12: Background image, example 2.

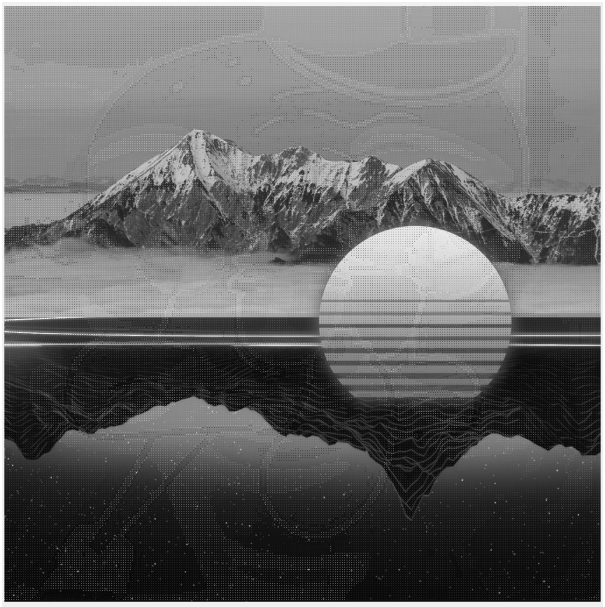

Figure 13: Encoded image, example 2.

Figure 14: Decoded image, example 2

cryptoanalysis experts, it nevertheless demonstrates the power of computing in calculations – we were able to XOR $256 \times 256 \times 8 = 2^{19} = 524288$ pixels, which cannot be easily achieved by pen-and-paper calculations.

5 Acknowledgements

My little brother Arda Kaz, for his artistic taste in which images are to be used in the demonstrations in the "Results" section.

References

[1] Simon Rubinstein-Salzedo. *Cryptography*. Springer Undergraduate Mathematics Series. Springer, 2018. ISBN: 9783319948171.

[2] Robert Sedgewick and Kevin Wayne. *Computer Science: An Interdisciplinary Approach*. 2^{nd} ed. Addison-Wesley, 2017. ISBN: 9780134076423.

I pledge my honor that this paper represents my own work, according to the University regulations.

Alkin Kaz