HO CHI MINH CITY NATIONAL UNIVERSITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE & ENGINEERING

Cryptography and Network Security

Assignment

STEGANOGRAPHY

Instructor:	Prof. Nguyễn Đức Thái	
Students:	Ôn Quân An	1852221
	Nguyễn Gia Huy	1852405
	Trần Nguyên Huân	1852394

HO CHI MINH CITY, APRIL 2021

Summary

1 Introduction

In recent years, technology plays a critical roles in our life in connecting people, solve difficult tasks, transport data from places to places, and even storing sensitive information, etc. For the past years, there are many crimes related to stealing private information. That is why many people are using encryption to send their secret message to the receiver. One of the most popular method to secretly send message is using Steganography. It encodes message into another form like music (mp3) or image (png, jpg, etc.) which makes the message imperceptible. In this project, we will apply steganography to hide our message in an image.

2 Steganography algorithm

We found a lot of method to encoded the message in an image, after discussion, we argreed to use the **Least Significant Bits (LSB)** method because it is simple and imperceptible. In an image file, it is displayed as an combination of pixels, where each pixel is in form of RGB (Red Green Blue), we chose to change the last bit of the Blue color. Since it changes the value of that pixel by 1, it is unable to notice any different by humans eyes.

However, just LSB alone can be solved easily by attackers and they will get the message that we try to hide. That is why we will use LSB with addition of a **Pseudorandom Number Generator (PRNG)** to further protect our message, so even if attackers can manage to get the changed bits out of the picture, it will still be too hard for them to get the right order of bits.

For PRNG, we use an algorithm that will not generate same number under some conditions. The function we are using is:

$$f(x) = x^2 \,\%\, prime \tag{1}$$

Where:

- $f(x)$ is the index of the pixel we want to encode the hidden bit in.
- x is the index of a bit of our message, ranging from 0 to message length in bits.
- $prime$ is a prime number that satisfies $prime \geq 2 * (message\_len)$

Example of Random Number Generator: Given $message\_len$ is 100 bits, p is prime number $p \geq 2 * message\_len$, so $p = 211$

x	0	1	2	3	4	5	6	7	8
prime	211								
Index	0	1	4	9	16	25	36	49	64

However, with the proposed hash function, the result is still somewhat obvious and easy to guess. Suppose the indexes of the first bits are 0, 1, 2, 3, 4, etc., the function output will be 0, 1, 2, 9, 16, etc. respectively. It is quite clear that we are using some kind of x^2 function. So to add up another layer of protection, we come up with this solution, instead of putting x alone, we will put $x + \alpha$, where $\alpha = \lceil \sqrt{prime} \rceil$

So the new hash function is:

$$f(x) = (x + \alpha)^2 \,\%\, new\_prime \tag{2}$$

Where

- x runs from 0 to message length in bits.
- $\alpha = \lceil \sqrt{prime} \rceil$.
- new_prime is a prime number that satisfies $new\_prime \geq 2 * (message\_len + \alpha)$

Example of Advanced Random Number Generator: Given $message\_len$ is 100 bits, by calculation we find that $\alpha = 15$, $new\_prime = 233$

x	0	1	2	3	4	5	6	7	8
alpha	15								
prime	233								
Index	225	23	56	91	128	167	208	18	63

Using $x+\alpha$ will significantly make our random number harder to guess since the new function output is more "random"comparing to previous one. In order to figure out the index pattern, the attackers will need to know new_prime, $alpha$, how we choose those 2 keys and the algorithm we are using with those 2 keys. But in addition, we will need to encode new_prime, $alpha$ at the end of the image for the decoding side to retrieve our message.

3 Encode

The purpose of using steganography as we have already known is to hide a secret message inside a normal picture. Hence, encoding a picture is an extremely important task that we have to do.

To encode a picture, we must do the following steps:

- Launch image from path
- Determine new_prime and α numbers
- Encode new_prime and α numbers at the end of the image
- Encode the message to the image

Algorithm 1 Check if a number is prime

Require: num, lst
  if lst is not empty **then**
    return True
  end if
  for $i = 0; i < length\,of\,lst; step = 1$ **do**
    if $num \,\%\, lst[i] = 0$ **then**
      return False
    end if
  end for
  for $i = length\,of\,lst, i < \lfloor\sqrt{num}\rfloor, step = 2$ **do**
    if $num \,\%\, i = 0$ **then**
      return False
    end if
  end for
  return True

Algorithm 2 Get α and *prime* number based on length of message

Require: $mess\_length$
  $prime\_list \leftarrow [2]$
  Note: $prime\_list$ has element 2 because it is the only even prime number.
  $prime \leftarrow 0$
  $\alpha \leftarrow 0$
  for $i = 3; i < 3 * mess\_length; step = 2$ **do**
    if i is a prime number **then**
      $prime\_list$ appends i
    end if
  end for
  for num in $prime\_list$ **do**
    if $2 * mess\_length \leq num$ **then**
      $prime = num$
      break
    end if
  end for
  $\alpha \leftarrow \lceil \sqrt{prime} \rceil$
  $pime \leftarrow (mess\_length + \alpha) * 2$
  for num in $prime\_list$ **do**
    if $prime < num$ **then**
      $prime = num$
      break
    end if
  end for

Algorithm 3 Encode *message* to the image

Require: $new\_prime \vee \alpha \vee pixel\_list$
  $message\_loc \leftarrow getRandomQuadraticResidues(new\_prime, \alpha)$
  Comment: The $getRandomQuadraticRedidues()$ function generate the locations of message after being encoded.
  if $length\,of\,message > length\,of\,image$ **then**
    ERROR: Need larger image
  else
    $index \leftarrow 0$
    for $pixel$ in $message\_loc$ **do**
      if $index < length\,of\,message$ **then**
        $last\,bit\,of\,pixel\_list[pixel] \leftarrow message[index]$
        $index \leftarrow index + 1$
      end if
    end for
  end if
  Save the encoded image

4 Decode

In order to decode an encoded picture, we have to follow these steps:

- Launch image from path
- Extract the *pixel_list* from the image
- Extract *prime* and α numbers from *pixel_list*
- Extract *message* from *prime*, α numbers and *pixel_list*

Algorithm 4 Extract *prime* and α numbers from *pixel_list*

Require: *pixel_list*
  prime $\leftarrow 0$
  $\alpha \leftarrow 0$
  hidden_key_bits $\leftarrow''$
  hidden_key $\leftarrow''$
  bit_counter $\leftarrow 0$
  for *pixel* in *reversed*(*pixel_list*) **do**
    hidden_key_bits $\leftarrow$ *hidden_key_bits* + *last bit of binary*(*pixel*)
    bit_counter $\leftarrow$ *bit_counter* $+ 1$
    if *bit_counter* $= 8$ **then**
      letter $\leftarrow$ *binary_to_string*(*last eight elements of hidden_key_bits*)
      hidden_key $\leftarrow$ *hidden_key* + *letter*
      if *letter* $=' \#'$ **then**
        break
      end if
    end if
  end for
  if *length of hidden_key* $= 2$ **then**
    prime $\leftarrow$ *hidden_key*[0]
    $\alpha \leftarrow$ *hidden_key*[1]
  end if
  return *prime*, α

The *prime* and α numbers will be our main variables to decode the message. We will continue with the pseudo-code for extracting message from *prime* and α numbers by using the algorithm as follow:

Algorithm 5 Extract *message* from *prime*, α numbers and *pixel_list*

Require: $prime \vee \alpha \vee pixel\_list$
  $hidden\_bits \leftarrow$ ""
  $message \leftarrow$ ""
  $message\_loc = getRandomQuadraticRedidues(prime, \alpha)$
  Comment: The $getRandomQuadraticRedidues()$ function generate the locations of message after being encoded.
  for *pixel* in *message_loc* **do**
    $hidden\_bits \leftarrow hidden\_bits + last\,bit\,of\,binary(pixel)$
  end for
  for $i = 0; i < length\,of\,hidden\_bits; step = 8$ **do**
    $hidden\_bits[i] \leftarrow hidden\_bits[i \rightarrow i + 7]$
  end for
  for $i = 0; i < length\,of\,hidden\_bits; step = 1$ **do**
    if $last\,four\,characters\,of\,message =$ "$END.$" **then**
      break
    end if
    $message \leftarrow message + hidden\_bits[i]$
  end for
  return *message*

5 Complexity

5.1 Encode

The **Encoding** process consists of 3 parts:
Call:

- N: length of *key* string
- M: length of the message

5.1.1 Calculating prime number and alpha value

This process will call the $getAlphaAndPrime()$ function. The function will be ...

5.1.2 Encoding prime and alpha at the end of the image

First, we initial a *key* string with the format: ***prime_number,alpha_value#*** (the symbol *#* is to mark the end of the key sequence for decoding convenience). Then we turn this string *key* into a sequence of *8-bit binary key*. The operation to do this will cost **O(N)**.

Next, we will iterative encode each bit of the *8-bit binary key* into each pixel, starting from the last pixel of the image. This procedure will cost **O(N*8) = O(N)** (because one character is 8-bit long).

5.1.3 Encoding message to the image

We will call $getRandomQuadraticResidues()$ (**O(M*8) = O(M)**) to get a list of random quadratic residues. The last job is to encode each bit of the message into the image with the

corresponding index of pixels. This will cost **O(M*8) = O(M)**.

Overall, the Encoding function will have the complexity: **O(N) + O(N) + O(M) + O(M) = O(N) + O(M)**.

5.2 Decode

The **Decoding** process consists of 2 parts:
Call:

- N: length of *key* string
- M: length of the message
- I: size of the image

5.2.1 Extracting key and alpha value from the end of the image

We will start from the end of the image, taking every last bit of the pixels iterative. For every 8 bits we have extracted, we will convert it into the corresponding character. If the character is #, we will break the loop and try to pull out the *prime* and *alpha* value.

The best case is that we are able to find the hidden *key*, so we only need to iterate $N \times 8$ times: **O(N)**.

The worst case is when there is no *key* hidden in the message, we will have to iterate through every pixels in the image, so the complexity is **O(I)**.

5.2.2 Extract message from the image

We will call *getRandomQuadraticResidues*() (**O(M*8) = O(M)**) to get a list of random quadratic residues. Then we will extract the last bit of the pixels corresponding to the quadratic residues we have found, the complexity will be **O(M*8) = O(M)**.

Overall, the Decoding function will the complexity: **O(N)+O(M)** in the best case, **O(I)** in the worst case.

6 Result and Drawbacks

6.1 Function

The first requirement of Steganography is that the encoded file can still be opened normally like the original one. During encoding, we invest a lot of time to research how to read the image of different type (jpeg, png) and how to change necessary bits to hide the message for each type. Finally, we successfully generate a encoded picture that can be opened and displayed normally.

6.2 Imperceptible

With LSB method, we assure that the encoded image is hard to distinguish from the original one since we only change 1 bit of Green colour in a pixel.

Hình 1: Original cat image

Hình 2: Enconded cat image

6.3 Security

To deal with security problem, we initially put the changed bits in ascending order, which is too easy to guess. Then we come up with Random Number Generator, that function can put the encoded bits into different, random index. The problem which that function is that it is still easy to guess as we explain in section 2. Therefore we create Advance Random Number Generator to make it even harder to get the message with out knowing the hash function and necessary key numbers.

6.4 File size

One of our difficulties is trying to ensure that the encoded image will still have the same size as the original one. We tried adding or subtracting the differences in size that our message might affect the image. Concretely, if the last bit of the Blue color is 1 but we want to change it into 0 according to our message bit, meaning that the size of the image is subtracted by 1 bit, we tried to add this 1 bit difference into the Green or Red color, similarly with case of the changing the original bit from 0 to 1. But the output image could not retain its old size. We had some researched online and concluded that the size of the new image is depended on the way the library we are using compress the image.

We tried with 2 Python libraries for image processing: Pillow and OpenCV. OpenCV gave better result for image compressing since it produced the closest size to the original image (when the input and output image format are both PNG), for an image with 2.77MB in size, the difference in size when encoding is just about 10-50 bytes, depending on our message size.

6.5 Lossless compression

Another drawbacks for LSB is that compression needs to be lossless, because lossy compression can destroy the last bit of every pixel. Meaning that the output image can only be in the format of PNG.