Notebook

October 21, 2024

1 Problem 3: Steganography and codes

1.1 Problem

Sam takes any picture in RGB format, changes the first pixel of it in some way and publishes the modified picture on his web-cite. Betty downloads the picture, analyzes it and takes out the message for her.

What does the Sam do with the picture? He should change it in such a way that nobody can visually fix the changing.

If Sam changes one of bits r6, r7, g6, g7 and b6, let us say that it costs 2 coins, while changing of one bit between r8, g8, b7 and b8 costs 1 coin.

Propose a method of coding a message (through given 16 types) in one pixel such it costs not more than 2 coins (in this case the changing of the picture is still not visual). Propose also the method for Betty how to extract secret messages. It is important that she has no access to the original picture.

1.2 Solution

1.2.1 Selection of Bits Within a Coin Constraint

Let (A) be the set of bits that cost 1 coin, and (B) be the set of bits that cost 2 coins. To spend no more than 2 coins, the following selection methods are available:

- **Selecting no elements**: There is 1 way to do this.
- **Selecting 1 element from set (A)**: There are 4 ways to choose this.
- **Selecting 2 elements from set (A)**: Considering the order, there are $\binom{4}{2}$ ways to make this selection.
- **Selecting 1 element from set (B)**: There are 5 ways to choose this.

Thus, the total number of ways to make selections without exceeding 2 coins is

$$4 + \binom{4}{2} + 5 + 1 = 16.$$

It is just sufficient for the number of messages that need to be sent.

1.2.2 Generate key for each message.

		2	2	1	2	2	1	2	1	1	
message	r6	r7	r8	g6	g7	g8	b6	b7	b8	cost
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	0	1	0	1
3	0	0	0	0	0	0	0	1	1	2
4	0	0	0	0	0	0	1	0	0	2
5	0	0	0	0	0	1	0	0	0	1
6	0	0	0	0	0	1	0	0	1	2
7	0	0	0	0	0	1	0	1	0	2
8	0	0	0	0	1	0	0	0	0	2
9	0	0	0	1	0	0	0	0	0	2
10	0	0	1	0	0	0	0	0	0	1
11	0	0	1	0	0	0	0	0	1	2
12	0	0	1	0	0	0	0	1	0	2
13	0	0	1	0	0	1	0	0	0	2
14	0	1	0	0	0	0	0	0	0	2
15	1	0	0	0	0	0	0	0	0	2

[13]:
```
message_table = [
    [0, 0, 0, 0, 0, 0, 0, 0, 0],  # Message 0
    [0, 0, 0, 0, 0, 0, 0, 0, 1],  # Message 1
    [0, 0, 0, 0, 0, 0, 0, 1, 0],  # Message 2
    [0, 0, 0, 0, 0, 0, 0, 1, 1],  # Message 3
    [0, 0, 0, 0, 0, 0, 1, 0, 0],  # Message 4
    [0, 0, 0, 0, 0, 1, 0, 0, 0],  # Message 5
    [0, 0, 0, 0, 0, 1, 0, 0, 1],  # Message 6
    [0, 0, 0, 0, 0, 1, 0, 1, 0],  # Message 7
    [0, 0, 0, 0, 1, 0, 0, 0, 0],  # Message 8
    [0, 0, 0, 1, 0, 0, 0, 0, 0],  # Message 9
    [0, 0, 1, 0, 0, 0, 0, 0, 0],  # Message 10
    [0, 0, 1, 0, 0, 0, 0, 0, 1],  # Message 11
    [0, 0, 1, 0, 0, 0, 0, 1, 0],  # Message 12
    [0, 0, 1, 0, 0, 1, 0, 0, 0],  # Message 13
    [0, 1, 0, 0, 0, 0, 0, 0, 0],  # Message 14
    [1, 0, 0, 0, 0, 0, 0, 0, 0]   # Message 15
]


def xor_messages(msg1, msg2):
    return [bit1 ^ bit2 for bit1, bit2 in zip(msg1, msg2)]
```

We XOR this key with the 9 bits in the pixel that can be modified. However, the issue is how Betty can recognize it if she cannot know the original image.

1.2.3 Error detecting code

We can use CRC with a polynomial $x^3 + x + 1$ to generate 3 parity bits. Sam must select an image such that r_1, g_1, b_1 are 3 bits that match the parity bits of the 9 modifiable bits.

[14]:
```
def crc_remainder(input_bits, polynomial_bits, n_bits):
    input_bits = input_bits + [0] * (n_bits - 1)

    for i in range(len(input_bits) - (n_bits - 1)):
        if input_bits[i] == 1:
            for j in range(n_bits):
                input_bits[i + j] ^= polynomial_bits[j]

    remainder = input_bits[-(n_bits - 1):]
    return remainder

def generate_crc_message(data, polynomial):
    crc_bits = crc_remainder(data, polynomial, len(polynomial))
    return data + crc_bits

data = [1, 0, 1, 0, 1, 1, 0, 0, 1]
polynomial = [1, 0, 1, 1]

choice = 9
transmitted_message = generate_crc_message(data, polynomial)
received_message = xor_messages(transmitted_message, message_table[choice]) +␣
 ↪transmitted_message[-3:]
print(f"Nine bits Message: {received_message[:9]}")
print(f"R1, G1, B1 bits: {received_message[-3:]}")
```

```
Nine bits Message: [1, 0, 1, 1, 1, 1, 0, 0, 1]
R1, G1, B1 bits: [1, 0, 1]
```

After Betty receives the image, she will extract the first 3 bits and the 9 ciphertext bits, and then use the CRC check function to verify the message.

[15]:
```
def check_crc(received_message, polynomial):
    remainder = crc_remainder(received_message, polynomial, len(polynomial))
    return remainder == [0] * (len(polynomial) - 1)

def detect_error_position(received_message, polynomial):
    for i in range(len(message_table)):
        modified_message = received_message.copy()
        for j in range(len(message_table[i])):
            modified_message[j] ^= message_table[i][j]

        if check_crc(modified_message, polynomial):
            return i
```

```
if check_crc(received_message, polynomial):
    print("Messsage 0")
else:
    print("Message", detect_error_position(received_message, polynomial))

print(choice)
```

```
Message 9
9
```