MC920 – Trabalho 4

Rafael Sartori M. Santos, 186154

31 de outubro de 2019

1 Introdução

O objetivo do trabalho é a codificação e decodificação de mensagens dentro de imagens e verificar se as novas informações produzem algum artefato visual na imagem. Como exigido pelo enunciado, haverá 2 programas: um para codificação e outro para decodificação.

Faremos esses programas utilizando Python com as bibliotecas *OpenCV*, *NumPy*, *pycrypto* (sua implementação atualizada, *pycryptodome*) e padrão.

2 Metodologia

No programa de codificação, precisamos receber uma imagem e um arquivo a ser codificado. Então abrimos essa imagem, alteramos um único *bit* para cada ponto e camada da imagem, escrevendo a mensagem que desejamos, e salvamos a imagem de saída.

No de decodificação, recebemos a imagem com a mensagem codificada, abrimos e isolamos o *bit* desejado, acumulando num vetor binário que será posteriormente convertido em um arquivo de texto decodificado.

Com esses passos gerais, começamos preparando a imagem, alterando seu formato em um programa externo.

2.1 Preparação da imagem

É um requisito trabalhar de forma *lossless* (sem perdas) para manter todas as informações da imagem intactas, garantindo a integridade da mensagem. Isto é, algoritmos de compressão *lossy* podem degradar a imagem e possivelmente causar a corrupção da mensagem.

Apesar disso, a imagem carregada pelo *OpenCV* não precisa ser de um formato *lossless* pois a biblioteca consegue abrir em um formato e salvar em outro. Há, portanto, apenas uma importância: salvar a imagem sem perdas.

Para isso, o programa sempre exigirá o formato `PNG` para a saída, adicionando a extensão "`.png`" caso não esteja mencionado na entrada.

2.2 Preparação da mensagem

O programa é capaz de encriptar a mensagem de forma segura utilizando uma senha e o algoritmo `AES`, produzindo um vetor de *bytes*. Caso a senha não seja fornecida, a mensagem é interpretada apenas pelos *bytes* que compõem a *string*.

Esses *bytes* serão divididos em uma matriz de mesmo tamanho da imagem, contendo apenas um *bit* por posição correspondendo a imagem. Ou seja, dividiremos cada *byte* em 8 *bits* na matriz que possui dimensão igual à da imagem.

A matriz precisa ser menor ou no máximo de mesmo tamanho à imagem e cada valor precisa ocupar no máximo um *bit*. Caso isso não for verdade, não conseguiremos incluir toda mensagem na imagem sem maiores perdas, pois nos comprometemos pelo enunciado a utilizar apenas um plano de *bits* da imagem. O efeito de qual plano de *bits* devemos escolher para diminuir artefatos visuais será discutido na seção 3.1 com os resultados.

Um problema que haverá com esta metodologia é que, como os *bytes* retornados pelo `AES` não representam um texto em `ASCII`, não podemos confiar que haverá um caractere terminal `\0`. Devemos, portanto, incluir alguma forma de saber onde a mensagem secreta acaba. Fazemos isso através de um inteiro a ser colocado no preâmbulo: o número de *bytes* da mensagem a ser escrita na imagem.

2.3 Colocando a mensagem na imagem

Essa etapa é feita através de operações binárias e vetoriais em toda imagem, o que é bastante eficiente e rápido. Começando com a imagem inicial, precisamos remover o trecho da imagem que utilizaremos para a mensagem.

Fazemos uma máscara M_m que seleciona todas as posições que a mensagem utilizará, pois assim não eliminamos todo o plano de *bits*, apenas as posições que precisamos. Isso dificultará a identificação de uma mensagem dentro da imagem, pois manteremos todo

o plano ocupado com informações, e manterá parte da imagem intacta, sobretudo em pequenas mensagens.

Com a imagem I, plano de *bits* p e a máscara M_m, zeramos apenas as posições que utilizaremos para a mensagem através da eq. (1), produzindo a imagem "com vazios" I_v.

$$I_v = (I \ \& \ (\neg(M_m \ll p))) \tag{1}$$

Juntamos a imagem com espaço para mensagem I_v com a matriz da mensagem M feita na seção 2.2 através da eq. (2), resultando na imagem final I_f, que agora contém a mensagem no plano de *bits* p especificado.

$$I_f = (I_v \ | \ (M \ll p)) \tag{2}$$

2.4 Salvando a imagem final

Utilizando *OpenCV*, salvamos a imagem final I_f em formato `PNG`, que é um formato *lossless*, protegendo a mensagem de degradações.

Assim, acabamos todas as etapas do codificador. Agora veremos a metodologia para o decodificador, que fará praticamente o trabalho oposto: com a imagem final I_f, determinamos a máscara da mensagem M_m e retiramos, utilizando operações binárias novamente, a mensagem M. É importante ressaltar que o plano de *bits* p é dado como argumento do programa e não precisa ser determinado pelo decodificador.

2.5 Isolando a mensagem

Fazemos os mesmos passos para abrir a imagem utilizando *OpenCV*, abrindo a entrada que deve ser dessa vez no formato `PNG`. Os 4 primeiros *bytes* (divididos entre *bits*, como descrito anteriormente) do plano de *bits* selecionado da mensagem deve conter o tamanho da mensagem a seguir.

Com o tamanho da mensagem, conseguimos reconstruir a máscara M_m (multiplicamos esse tamanho por 8 – pois há 8 *bits* dentro de um *byte* – e decrescemos de 1 em 1 a cada vez que adicionamos, da esquerda para direita, de cima para baixo, um *bit* à máscara vazia exceto pelos 4 primeiros *bytes*).

Com a máscara pronta, conseguimos isolar a mensagem M através da eq. (3).

$$M = (I_f \ \& \ (M_m \ll p)) \gg p \tag{3}$$

Agora basta isolar *bit* a *bit* a mensagem, formando os *bytes*.

2.6 Decodificando a mensagem

Com o vetor de *bytes* pronto, se houve a entrada de senha, passamos a mensagem pelo algoritmo de decodificação `AES`. Com ou sem senha, interpretamos o vetor binário final como uma *string*, obtendo um texto.

Após exportação do texto para um arquivo, realizamos todas as etapas e a funcionalidade do programa está completa.

3 Resultados e análise

Agora iremos testar a funcionalidade e analisar o efeito da mensagem na imagem em diversos planos de *bits*.

3.1 Teste de funcionalidade

Para o teste de funcionalidade, codificamos na imagem `imgs/mel.jpg` a mensagem no arquivo de texto `input/mensagem.txt` no plano de *bits* 0, produzindo a imagem `output/mel_coded.png`. Fazemos a decodificação da imagem `output/mel_coded.png` no plano de *bits* 0 produzindo a mensagem `output/mensagem_codificada.txt` e notamos que não há qualquer diferença entre as duas.

Utilizamos o parâmetro `--passphrase`, temos o mesmo resultado. Porém, caso a senha seja incorreta na decodificação, haverá um erro em alguma etapa do algoritmo `AES`, de conversão para *string* ou, com sorte, o texto ficará ilegível.

Podemos tentar perceber as diferenças entre a imagem que possui a mensagem e a original na fig. 1.

(a) Imagem original (b) Imagem codificada

Figura 1: Comparativo entre a imagem original e a com mensagem codificada