

Trabalho 4

MO443/MC920 - Introdução ao Processamento de Imagem Digital

Lucas de Oliveira Silva - 220715

1 Introdução

Neste trabalho o objetivo foi aplicar a técnica de esteganografia para "esconder" arquivos dentro de imagens digitais. Dado uma imagem, um arquivo e um plano de bits o programa codifica o arquivo na imagem alterando apenas bits do plano escolhido. A solução foi feita com Python 3.8.5 em um único arquivo .py. Foram utilizadas as seguintes bibliotecas: numpy (1.19.1) e scikit-image (0.16.2).

2 Como executar

A implementação está no *script* trabalho4.py, as imagens e arquivos de teste na raiz e as saídas obtidas na pasta output. O programa precisa necessariamente de uma imagem de entrada (-i img) em um formato png e um arquivo de saída (-o arq), que indica a imagem resultante da codificação (em png) ou o arquivo da decodificação. O resto dos parâmetros são opcionais e alguns possuem valores padrões. Os parâmetros opcionais são os seguintes:

- -f O nome do arquivo de entrada que será codificado na imagem.
- -p Um inteiro que indica o plano de bits a ser usado. O padrão é zero.
- -e Se o programa deve codificar (0) ou decodificar (1). O padrão é 0, ou seja, codificação.

Ex: python3 trabalho4.py -i baboon-encoded.png -o decoded.txt -e 1 -p 1

Para mais detalhes é possível executar o *script* com o parâmetro -h.

A imagem é lida usando a função *io.imread* do skimage, e logo depois os valores são convertidos para uint8. Durante a codificação o arquivo é lido como uma lista de bytes, o programa só funciona se esse arquivo tiver até $2^{32} - 1$ bytes (unsigned int de 4 bytes). Isso pode ser facilmente aumentado mudando uma constante no código, mas já que esse limite é bem alto resolvi deixar fixo.

Se o arquivo de entrada não couber na imagem destino uma exceção é lançada com uma mensagem de erro. Antes de salvar a imagem em arquivo é feito uma conversão para uint8 novamente.

3 Avaliação dos resultados

No geral a codificação é totalmente imperceptível a olho nu quando usamos bits pouco significativos, mas a medida que o plano vai avançando percebemos que claramente há modificações (figura 2). A imagem baboon, que possui mais textura, suporta alterações maiores do que a imagem watch, pois nessa já percebemos alterações (manchas) usando o plano três.

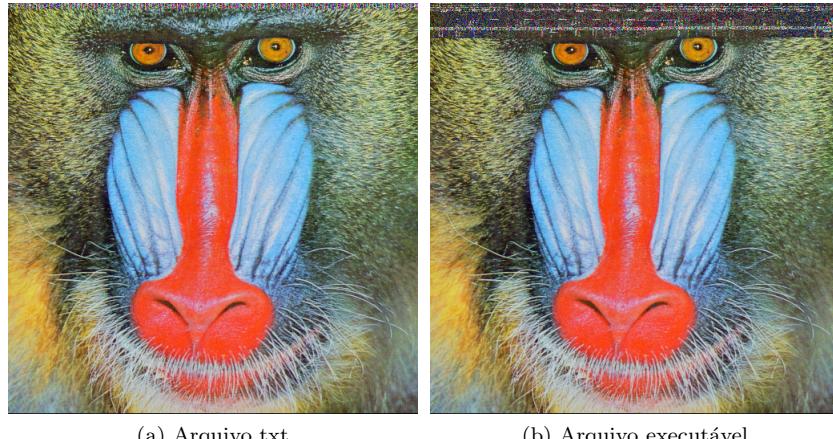
Com relação ao tipo de conteúdo a ser codificado, o texto simples produziu um ruído mais uniforme, ao passo que nos arquivos binários percebemos artefatos como linhas (figura 1). Isso provavelmente vem do fato de que caracteres de texto são normalmente restritos a poucos valores da tabela ASCII.

Após o processo de esteganografia apenas um plano de bits é alterado e nas figuras 4-5 é mostrado os planos 0, 1, 2, 5, 6 e 7 após uma mensagem ser codificada em cada plano. Ao codificar arquivos de texto o efeito ainda é imperceptível, sendo apenas claro no plano 7. Já o arquivo executável fica bem destacado com grandes faixas pretas na imagem, e é possível a olho nu perceber a codificação mesmo nos planos menos significativos. Dado essa variação no efeito fica muito difícil detectar codificações esteganográficas em imagens, mas algum cálculo como o desvio padrão, numa janela local, poderia ajudar.

4 Conclusão

O programa é limitado a operar em apenas um plano e portanto há um limite de bytes no arquivo a ser codificado, mas fora isso o programa não apresenta nenhuma restrição com relação a entrada já que o tamanho é salvo num pequeno cabeçalho. O programa gera apenas saídas no formato png, só que isso pode ser facilmente alterado. É necessário apenas usar formatos sem perda, para que o arquivo não seja ocasionalmente corrompido.

Os resultados foram bastante promissores, pois mesmo salvando grandes arquivos, a alteração é imperceptível a olho nu. E mesmo isolando os planos de bits alterados ainda é difícil, para alguns tipos de arquivos, perceber a codificação.



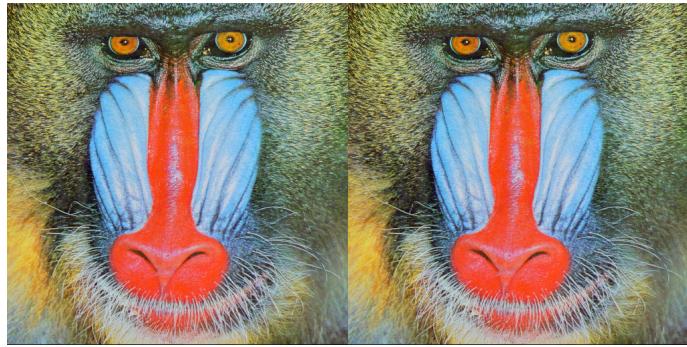
(a) Arquivo txt

(b) Arquivo executável



(c) Arquivo png

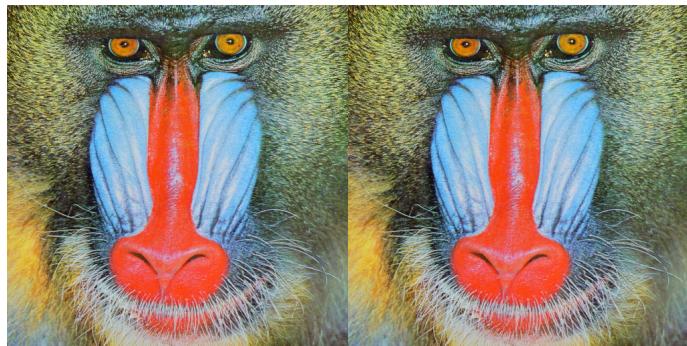
Figura 1: Resultado de diferentes tipos de arquivos codificados no plano 7



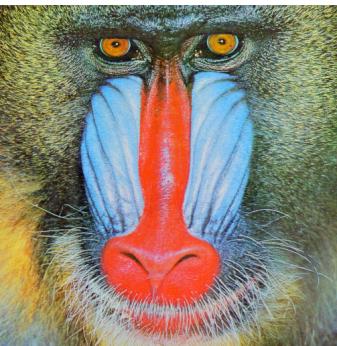
(a) Plano 0



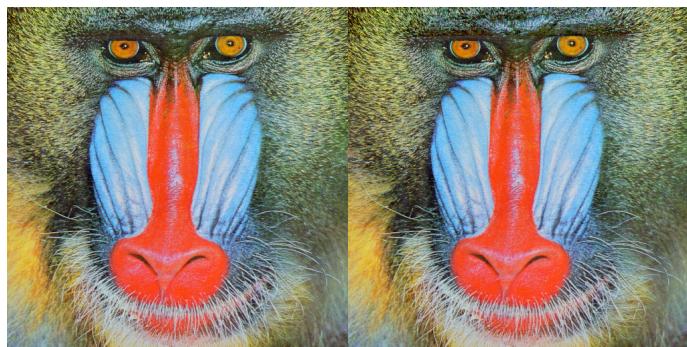
(b) Plano 1



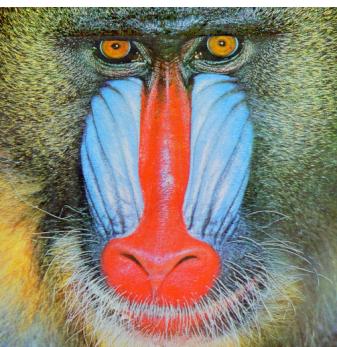
(c) Plano 2



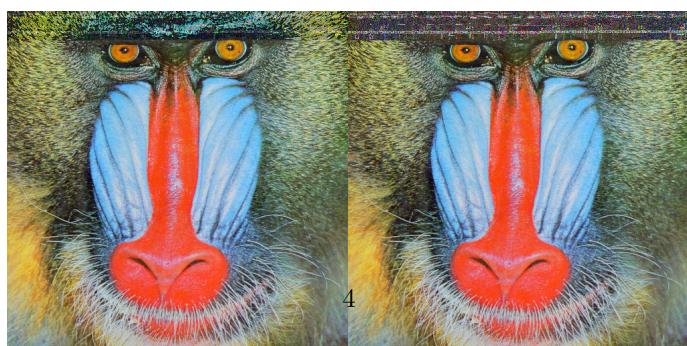
(d) Plano 3



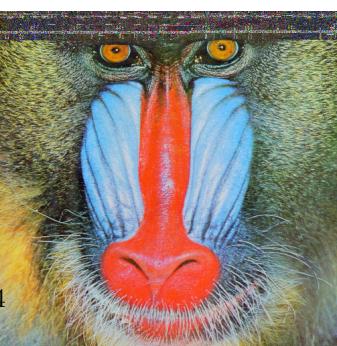
(e) Plano 4



(f) Plano 5



(g) Plano 6



(h) Plano 7

Figura 2: Alteração do efeito visual ao variar o plano de bits na codificação de um executável

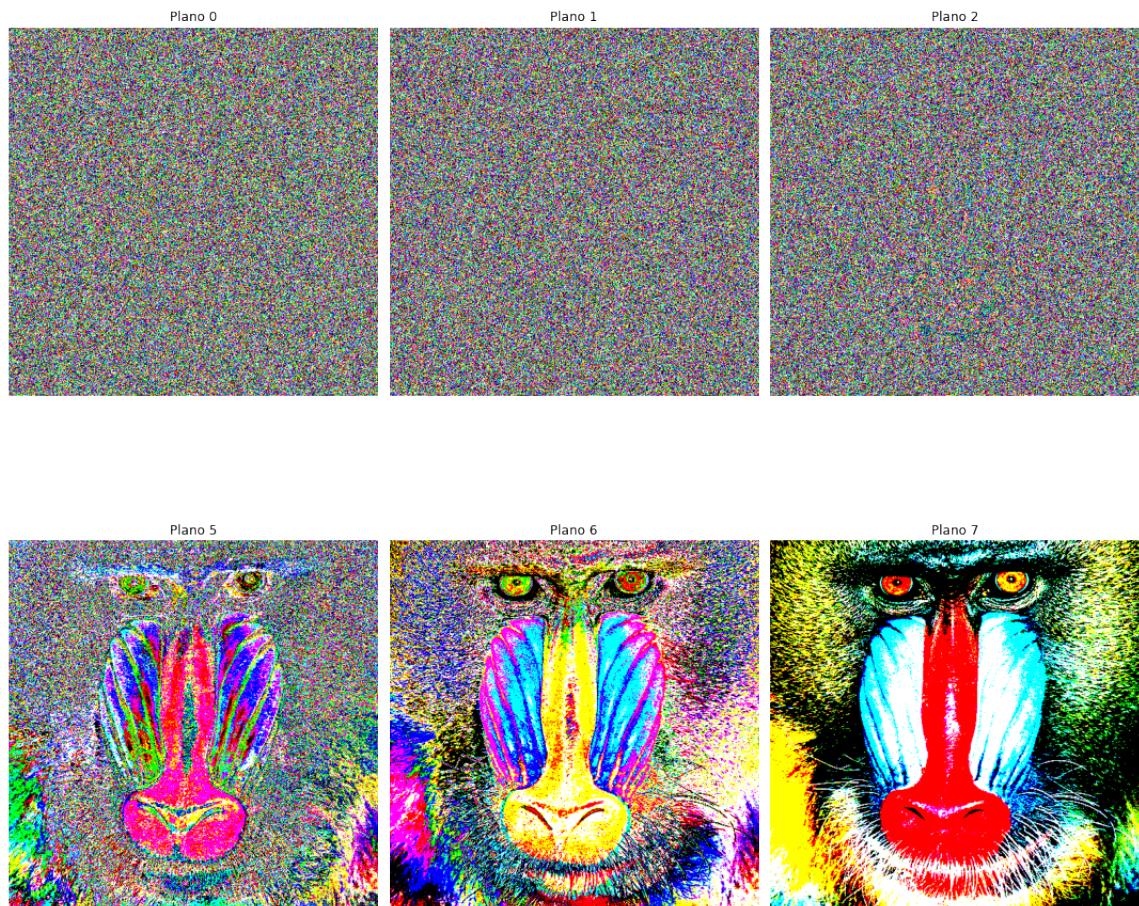


Figura 3: Planos de bits da imagem original baboon

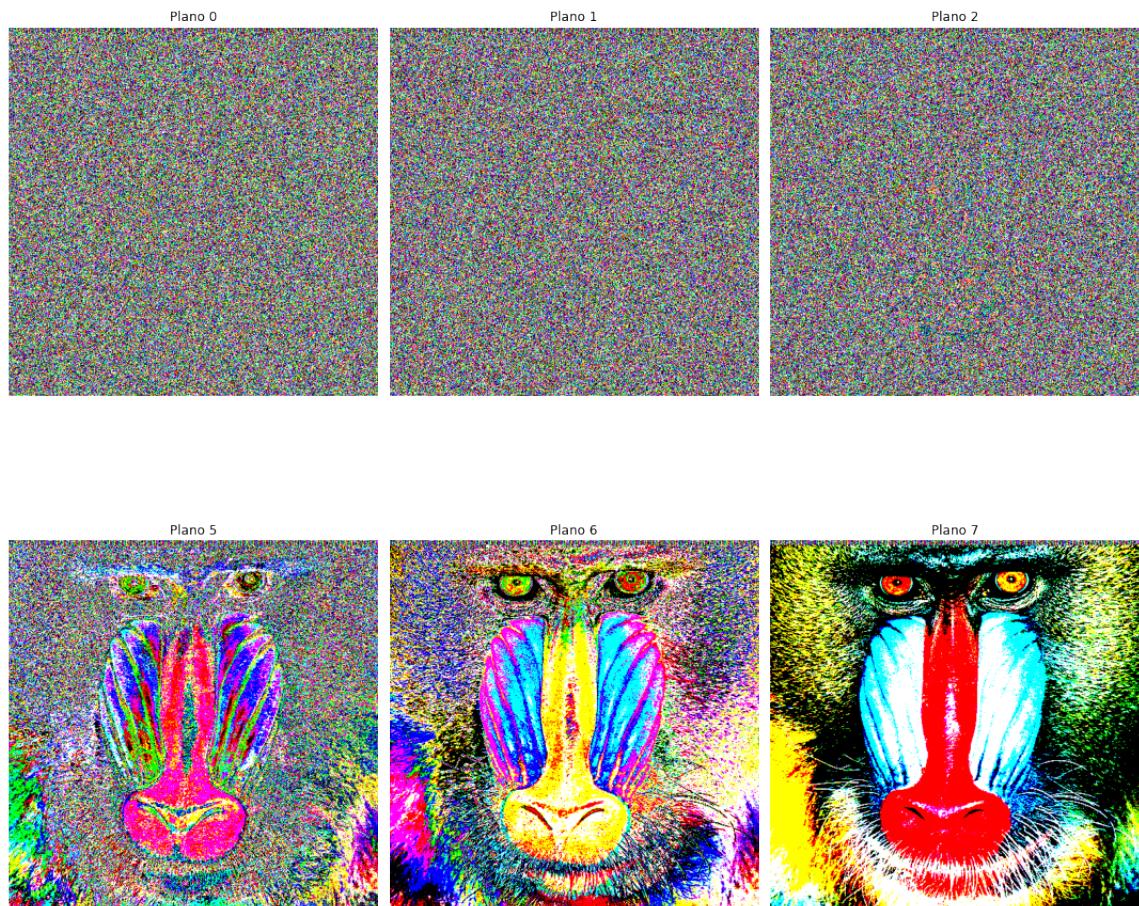


Figura 4: Planos de bits da imagem baboon com texto codificado

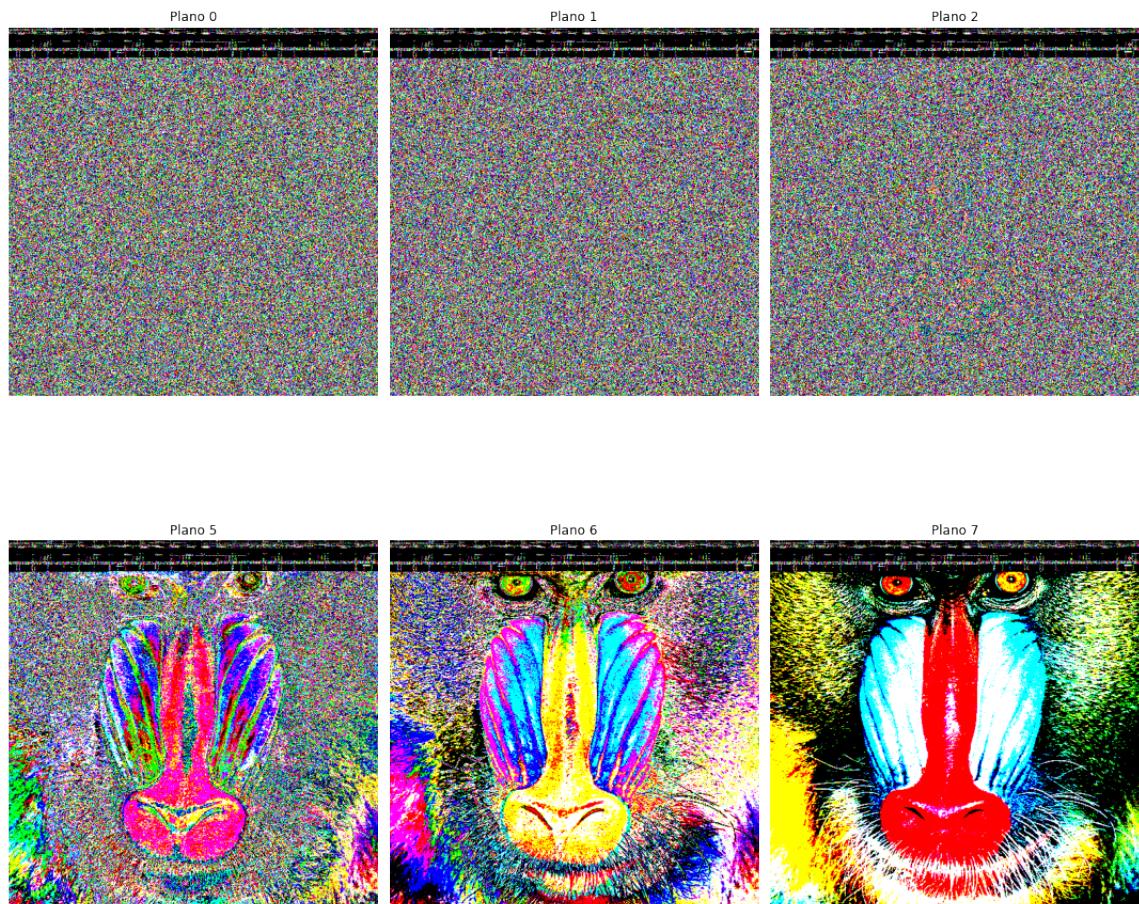


Figura 5: Planos de bits da imagem baboon com executável codificado