

# Projeto 4 - MC920A - Medidas de imagens

Marcelo Biagi Martins - 183303

## 1 Introdução

A esteganografia é uma técnica relativamente antiga , cujo objetivo é o de embutir uma mensagem, imagem ou vídeo em outra mensagem, imagem ou vídeo. Por ser uma técnica discreta, seus resultados não chamam a atenção e este é um ponto forte em relação à criptografia avançada. No entanto, em contrapartida, é muito mais simples descobrir a mensagem secreta quando se usa esteganografia.

Mesmo se tratando de uma técnica consideravelmente simples, a esteganografia não deixa de ser utilizada. Impressoras a laser da HP e da Xerox, por exemplo, costumam utilizar esses algoritmos para inserir padrões imperceptíveis de amarelo com números de série nas páginas impressas, por exemplo. Além disso, o serviço de inteligência russo já admitiu usar uma variação dessa técnica para mensagens de celular. Ou seja, é um assunto muito interessante e com grande utilidade.

Neste projeto, dois scripts foram desenvolvidos para exercício dessa técnica: o primeiro, encripta uma mensagem dentro de uma imagem nos planos de bits menos significativos das três camadas R, G e B. Já o segundo "descobre" essa mensagem utilizando um algoritmo reverso do primeiro e devolve essa mensagem ao usuário.

## 2 Especificações e funcionamento do programa

O primeiro script python criado foi o de codificação da mensagem na imagem e ele pode ser executado com o comando `python3 codificar.py + flags`. As flags em questão servem para alterar os parâmetros de execução do código e estão dispostas na tabela 1:

Tabela 1: Flags de execução

Flag	Opções	Valor default
--folder	Diretório com as imagens png	imagens/
--image	Qualquer nome de imagem dentro do folder	baboon
--msg	O arquivo txt aonde a mensagem que se deseja encriptar está	-
--bit	A camada de bits em que se deseja encriptar a mensagem (0,1 ou 2)	0

O funcionamento do `codificar.py` é simples. Primeiro, lê-se a imagem dentro da pasta fornecida com a biblioteca `opencv` de Python. Como essa biblioteca lê as imagens de forma invertida, elas são convertidas posteriormente de BGR para RGB. Com isso, o arquivo com a mensagem de entrada é lido e um caracter de parada é adicionado ao final do texto. A mensagem é então transformada em binária e a função `encoder` é chamada caso a mensagem caiba de forma completa dentro da imagem; caso contrário, uma mensagem de erro é gerada. A encoder, por sua vez, vai analisar todos os pixels da imagem na ordem, começando pelas linhas e percorrendo todas as colunas da linha antes de passar para a próxima, e na ordem R, G e B de cada pixel. Cada valor de pixel em cada camada RGB é transformado em um binário de 8 bits, pois os valores de intensidade lidos pela opencv vão de 0 a

255. O valor escolhido pelo usuário com a flag `--bit` será utilizado nesse momento, pois o algoritmo vai alterar os valores apenas da camada escolhida, que pode ser a 0, 1 ou a 2, que possuem os bits menos significativos.

Assim, se o usuário escolher a camada 2, o algoritmo vai percorrer todos os pixels da imagem, alterando apenas a camada 2 e inserindo, para cada camada RGB do pixel, um caracter da mensagem. A condição de parada do algoritmo é quando ele encontrar o caracter de parada '\0', que possui código na tabela ASCII de "00000000", algo impossível de se reproduzir em um teclado convencional e, assim, sabemos que o texto acabou. A imagem gerada é salva da forma `encoded_bit_imageName.png`. Assim, a imagem `encoded_2_baboon.png` utiiza a imagem do babuíno para esconder uma mensagem na sua segunda camada de bits.

Por fim, os planos de bits 0, 1, 2 e 7 da imagem são salvos para a camada R, G e B da imagem com nome `(r,g,b)_plano_hidein_bit_imagename.png`. Assim, se for gerada uma imagem chamada `b_2_hidein_1_baboon.png` pelo script, ela representará o plano de bits 2 da camada B(blue) da imagem `baboon.png`, sendo que a flag "bit" escolhida pelo usuário para esconder sua mensagem foi 1.

O segundo script criado foi o de decodificação da mensagem e ele pode ser executado com o comando `python3 decodificar.py + flags`. As flags em questão servem para alterar os parâmetros de execução do código e estão dispostas na tabela 2:

Tabela 2: Flags de execução

Flag	Opções	Valor default
<code>--folder</code>	Diretório com as imagens png	imagens/
<code>--image</code>	Qualquer nome de imagem dentro do folder	baboon
<code>--msg</code>	O arquivo txt aonde se deseja salvar a mensagem gerada	texto_saida.txt
<code>--bit</code>	A camada de bits em que a mensagem está (0, 1 ou 2)	0

O funcionamento do `decodificar.py` é similar ao anterior. A imagem escolhida é lida e convertida para RGB. Após isso, cada pixel é lido na mesma ordem do `codificar.py`, seu valor é transformado em binário e uma simples comparação é feita: se este valor for "0", um "0" é adicionado na mensagem final; caso contrário, adiciona-se um "1". Após encontrarmos a sequência "00000000", o algoritmo para e a função que gera a mensagem é chamada. Essa função divide o bloco em subsegmentos de 8 caracteres e converte-os para decimal e, com o uso da função `chr()`, esses números são convertidos para seu caracter correspondente na tabela ASCII e a mensagem é gerada, impressa no terminal e salva em um arquivo com nome `bit_imagename_filename.txt`. Assim, um arquivo chamado `2_baboon_texto_saida.txt` é a mensagem gerada através da imagem baboon na camada de bits 2.

### 3 Resultados

Um primeiro resultado que podemos analisar é a respeito do porquê a camada de bits utilizada para encodificar a mensagem ser alguma das três menos significativas. Apesar do script não aceitar encodificação em camadas maiores que a 2, uma mudança momentânea foi feita para podermos adquirir essas imagens e analisar a diferença na figura 1 da encriptação de uma mensagem de 93k de caracteres, que ocupa mais de 75% da área da imagem, nas camadas 0, 1, 2 e 7 e seu impacto na imagem original.

Como pode-se notar, as imagens resultantes da encriptação da mensagem nas primeiras três camadas basicamente não altera o resultado final, pois o olho humano não é capaz de perceber pequenas variações nas cores. A maior diferença, se levarmos em consideração a camada 2, vai acontecer quando R, G e B tiverem 0 nessa camada e todas precisarem mudar para 1, ou vice-versa. Esse caso específico vai fazer cada camada variar em 4 unidades, ou seja, um pixel (250, 120, 2) vai se transformar em (254,

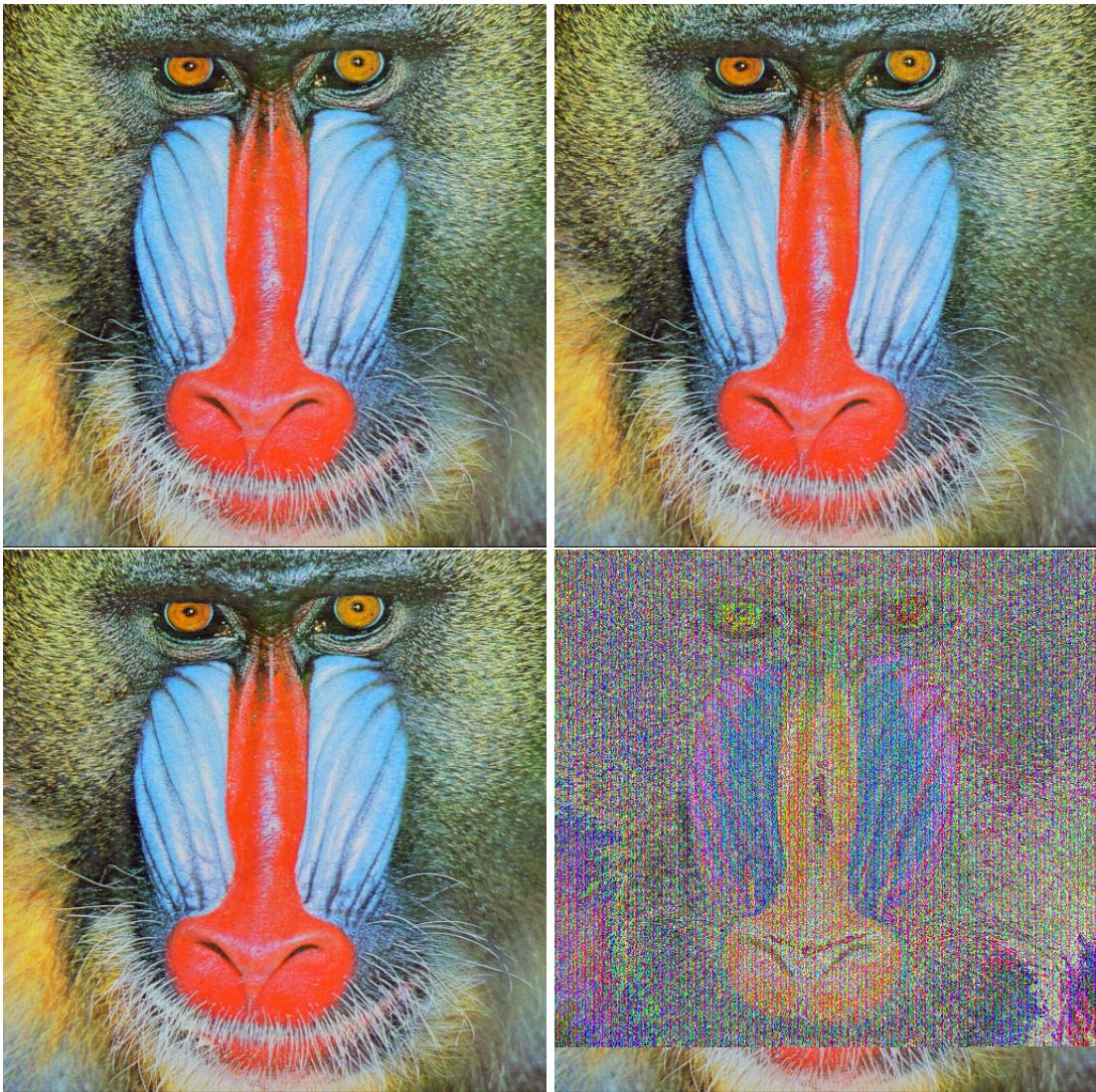


Figura 1: Encriptação de uma mensagem de 93k caracteres nas camadas: 0, 1, 2 e 7, respectivamente.

124, 6), e esse caso, que é raro, é a maior diferença possível para a camada 2. Esse exemplo está na figura 28 e é nitidamente imperceptível pelo olho humano. De forma similar, vemos que a encriptação nas camadas mais altas, como a 7, alteram radicalmente o valor do pixel. A maior variação possível seria no caso em que todos os valores de um pixel na camada 7 são 0 e precisam passar para 1, ou vice-versa. Um exemplo dessa transição seria um pixel de valor (2, 30, 50) que passaria para (130, 158, 178), pois 128 é somado em cada coordenada. Essa variação está na figura 3 e é nitadamente muito mais impactante e, por isso, os resultados na imagem final são muito visíveis e não são úteis para encriptação na prática.

Outro resultado é com relação a diferença entre as camadas que sofrem encriptação e as outras. Para isso, fixando a camada de encriptação como a 2, podemos analisar as camadas 0, 1, 2 e 7 novamente para notar as diferenças entre elas. Assim, temos na figura 4 à 15:

Podemos ver que a camada 2 possui mais linhas verticais, restando uma faixa na parte inferior que deixa bem nítido a localização aproximada do fim da mensagem encriptada e deixando claro que a mensagem está nessa camada.

Para a imagem peppers.png, o resultado é mais visível. A mensagem, neste caso, está no bit 1,

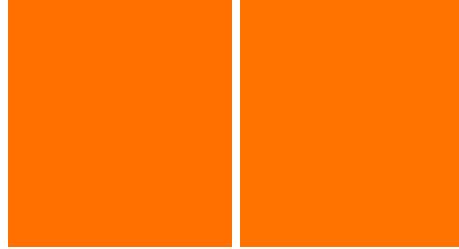


Figura 2: Exemplificação da maior mudança possível na camada 2.

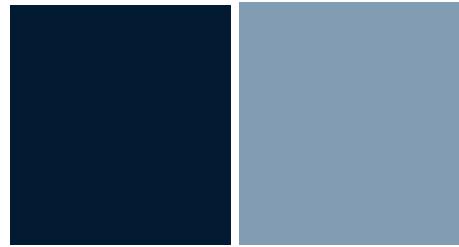


Figura 3: Exemplificação da maior mudança possível na camada 7.

conforme a figura 16 à 27.

## 4 Conclusões

Esse trabalho colocou em prática uma técnica muito utilizada, a esteganografia. Com ela, foi possível encriptar mensagens de diversos tamanhos dentro de uma imagem e recuperá-la em seguida utilizando dois scripts. É notável que a técnica é muito discreta se utilizada nas camadas de bits menos significativas e este fator é uma vantagem em relação à técnicas de criptografia avançadas. No entanto, descobrir a mensagem é tarefa simples e, nesse aspecto, a criptografia se torna mais vantajosa. Além disso, é marcante o potencial desta técnica, que pode ir além da encriptação de mensagens para a encriptação de outra imagem dentro da original, tornando sua utilidade real muito maior do que a estudada neste projeto.

## Referências

- [1] *Esteganografia em python*, Em <https://www.geeksforgeeks.org/image-based-steganography-using-python/>
- [2] *Esconder uma mensagem em uma imagem*, Em <https://dev.to/erikwhiting88/let-s-hide-a-secret-message-in-an-image-with-python-and-opencv-1jf5>
- [3] *Esteganografia e usos*, Em <https://pt.wikipedia.org/wiki/Esteganografia>
- [4] *Escondendo dados em imagens*, Em <https://null-byte.wonderhowto.com/how-to/steganography-hide-secret-data-inside-image-audio-file-seconds-0180936/>

## 5 Anexos

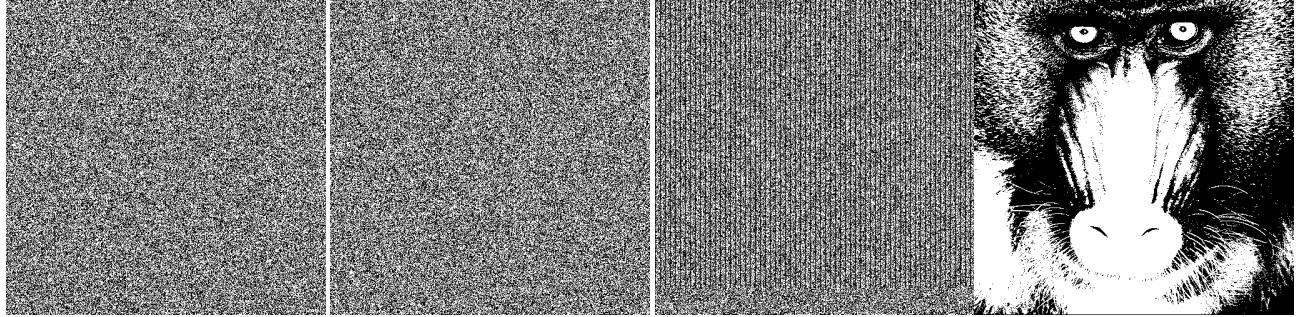


Figura 4: Camada R 0

Figura 5: Camada R 1

Figura 6: Camada R 2

Figura 7: Camada R 7

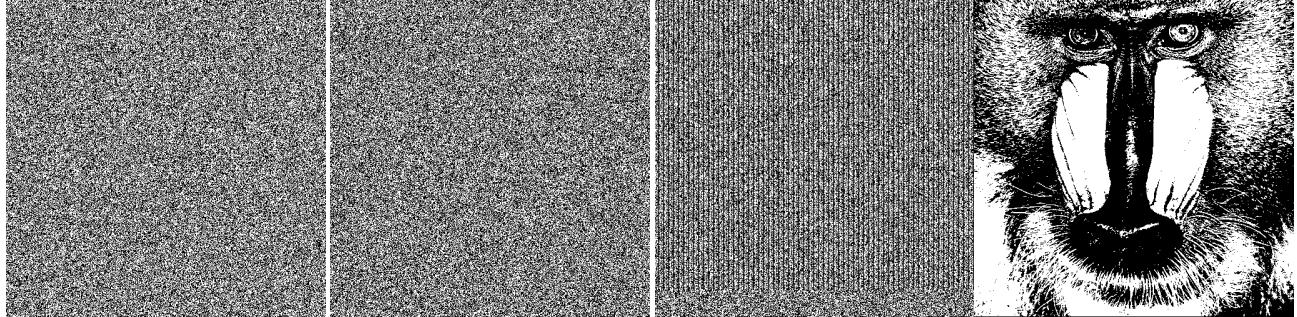


Figura 8: Camada G 0

Figura 9: Camada G 1

Figura 10: Camada G 2

Figura 11: Camada G 7

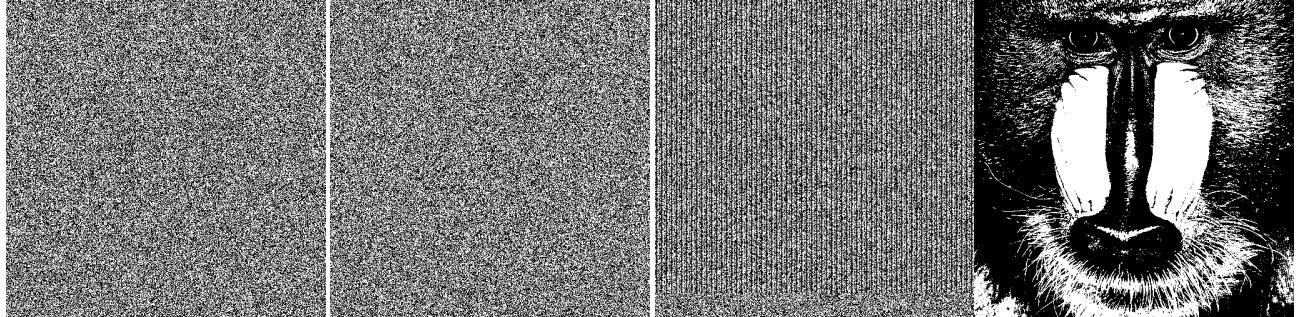


Figura 12: Camada B 0

Figura 13: Camada B 1

Figura 14: Camada B 2

Figura 15: Camada B 7

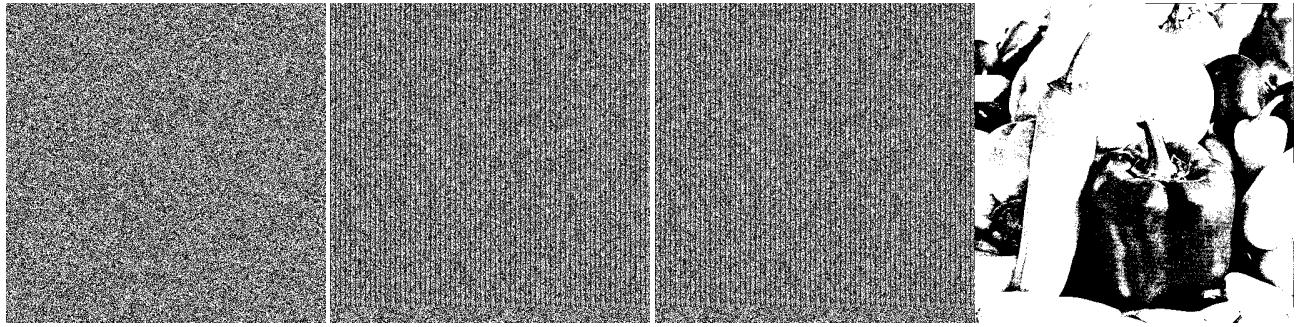


Figura 16: Camada R 0

Figura 17: Camada R 1

Figura 18: Camada R 2

Figura 19: Camada R 7

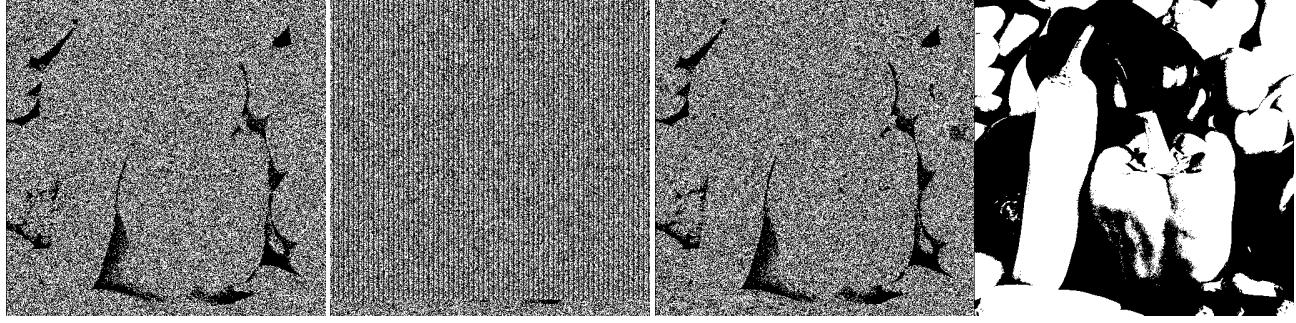


Figura 20: Camada G 0

Figura 21: Camada G 1

Figura 22: Camada G 2

Figura 23: Camada G 7

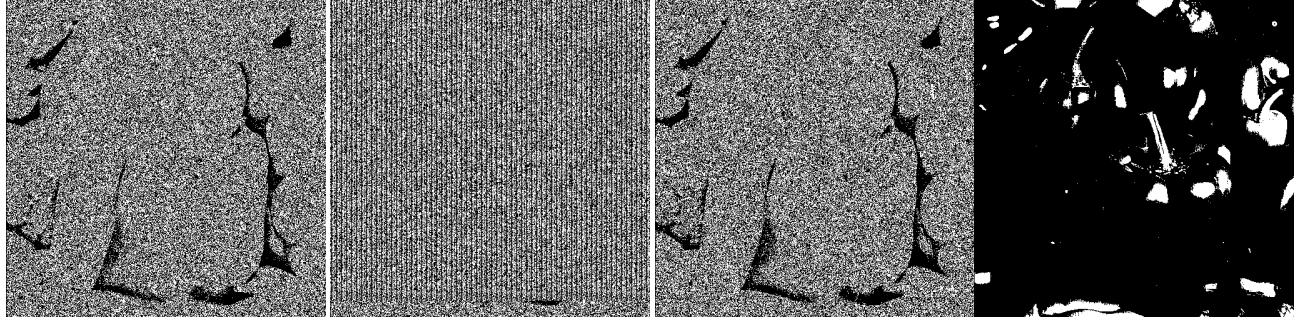


Figura 24: Camada B 0

Figura 25: Camada B 1

Figura 26: Camada B 2

Figura 27: Camada B 7

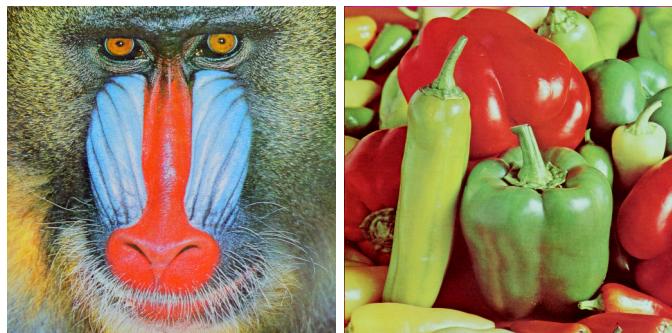


Figura 28: Imagens teste baboon e peppers