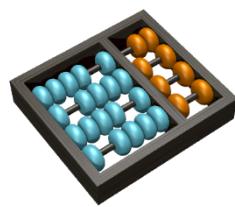




**Universidade Estadual de Campinas**  
**Instituto de Computação**



Disciplina do 2º Semestre de 2020

IC - UNICAMP

Curso: Bacharelado em Ciência da Computação

## **MC920 - Introdução ao Processamento de Imagens Digitais**

### **Trabalho 4 - Esteganografia em Imagens Coloridas**

**Alunos:** Gabriel Volpato Giliotti    **RA:** 197569

**Professor:** Hélio Pedrini

Campinas – SP  
2020

# **Trabalho 4 - Introdução ao Processamento de Imagem Digital**

**Nome:** Gabriel Volpato Giliotti - **RA:** 197569

## **1 - Introdução**

O objetivo deste trabalho é o estudo e realização do processo de Esteganografia, uma técnica onde é possível ocultar mensagens ou até mesmo outros tipos de arquivos realizando um processo de modificação dos bits menos significativos de uma imagem dada como entrada, junto da mensagem/arquivo que se deseja ocultar. Vale ressaltar que nesse projeto estamos fazendo uso apenas de mensagens em formato .txt e imagens coloridas no formato .png (Portable Network Graphics), mas é possível aplicar tal método em qualquer arquivo/imagem que seja possível sua binarização. Além disso, procuramos realizar uma análise dos planos de bits entre as imagens originais e de saída para avaliar o processo.

Atualmente, o processamento de imagens é um artifício muito utilizado em mídias sociais, aparelhos eletrônicos e softwares, para aplicação de filtros e outros recursos em imagens. Assim, a esteganografia oferece um modo de se ocultar uma mensagem ou arquivo dentro de imagens que podem ser enviadas para outras pessoas, a fim de impedir que terceiros tenham acesso facilitado. Vale lembrar que esse é um processo diferente da criptografia, isto é, na criptografia sabe-se sobre existência da mensagem/arquivo e a busca é pela chave para descriptografar, já na esteganografia a idéia é que a mensagem arquivo passe despercebido até que chegue a seu destino, sem a necessidade de criptografia ou chave.

Junto desse relatório será entregue o arquivo .zip ou .tgz, que possui todos os arquivos citados nesse relatório e que foram utilizados para processar as imagens dadas como entrada.

## **2 - Programas**

Os scripts para processamento das imagens foram implementados em Python 3.7.6 junto das bibliotecas Numpy 1.19.2 e OpenCV 4.4.0 que são voltadas para o processamento de imagens. Outras bibliotecas como Argparse (utilizada para leitura de parâmetros em python) também foram utilizadas, mas não oferecem recursos específicos para processamento de imagens.

### **2.1 - Como Executar**

Inicialmente, antes de executar o script, devemos criar uma pasta chamada “Outputs” que corresponde ao local de saída das imagens da execução dos arquivos .py e .txt referentes ao trabalho 4. Para se obter as imagens de saída, imagens de plano de bits das imagens de

entrada e saída e arquivo texto com a mensagem esteganografada, você deve criar a pasta no mesmo diretório onde executará os arquivos .py.

Para o trabalho 4 foram criados três arquivos com extensão tipo .py responsáveis respectivamente por ocultar a mensagem de texto dentro de uma imagem, por decodificar a mensagem presente na imagem para um novo arquivo .txt e para apresentar uma imagem do plano de bits de uma imagem especificada (seja ela esteganografada ou não). A seguir estão os formatos de construção para chamada dos arquivos via linha de comando (Windows: CMD/ Linux: Bash):

```
C:\> python codificar.py "pathImagemOriginal" "pathArquivoTxt" "valorPlanoBits"  
"nomeImagenSaida"
```

```
C:\> python decodificar.py "pathImagenTxtOculto" "valorPlanoBits" "nomeTxtSaida"
```

```
C:\> python bitplane.py "pathImagenEscolhida" "valorPlanoBits"
```

```
C:\> python bitplane2.py "pathImagenEscolhida" "valorPlanoBits"
```

Vale ressaltar que só realizamos a codificação e decodificação dos planos de bits menos significativos, sendo eles os planos 0, 1 e 2. Qualquer valor de plano diferente desses irá disparar uma mensagem de erro. Para o arquivo de exibição de plano de bits (bitplane.py), o valor do plano que se deseja apresentar pode ser qualquer um no intervalo [0,7].

## 2.2 - Entradas

Como entrada para os arquivos .py do projeto são apresentadas as descrições:

- pathImagenOriginal: caminho até a imagem onde se deseja ocultar uma mensagem de texto.
- pathArquivoTxt: caminho até o arquivo de texto onde se encontra a mensagem que desejamos ocultar.
- valorPlanoBits: valor do plano de bits da imagem onde vamos ocultar a mensagem. Os planos possíveis para esse projeto são 0, 1 e 2.
- nomeImagenSaida: nome da imagem de saída, que contém a mensagem oculta em seus bits. Especificar .png é opcional
- pathImagenTxtOculto: caminho até a imagem que contém a mensagem que precisa ser decodificada. É o arquivo com nomeImagenSaida.png.
- nomeTxtSaida: nome do arquivo de texto onde será escrita a mensagem decodificada. Especificar .txt é opcional.
- pathImagenEscolhida: caminho até uma imagem (com mensagem codificada ou não) que se deseja exibir o plano de bits.

## **2.3 - Um pouco sobre o Algoritmo**

O algoritmo de codificação realiza a leitura da imagem original e do arquivo de texto, armazenando seus conteúdos em variáveis. Em seguida, transforma a mensagem de texto em valores binários onde cada letra corresponde a 8 bits (ou 1 byte). Por fim, ele itera sobre o cada pixel e seus valores de banda, substituindo o valor do bit de uma banda de cor no plano de bit especificado pelo valor de um bit que faz parte da representação da mensagem.

Por outro lado, o algoritmo de decodificação faz o processo inverso, realizando a leitura da imagem que contém a mensagem oculta e armazenando-a em uma variável. Assim, dado o plano de bit especificado onde a mensagem se encontra, são percorridos os pixel e seus valores de banda no plano especificado, onde armazenamos o valores dos bits em uma string que, em seguida, é convertida para caracteres, reconstruindo a mensagem. Por fim, o algoritmo escreve a mensagem em um arquivo .txt onde o nome é especificado por parâmetro.

Além disso, para auxiliar na comparação das imagens originais e estenografadas, temos um terceiro algoritmo que, através de operações de bitwise disponíveis em python, filtra e gera uma imagem com o plano de bit especificado de uma imagem passada por parâmetro.

## **2.4 - Saídas**

As saídas (imagens codificadas, arquivo de texto após decodificação e imagens de planos de bits) são obtidas na pasta “Outputs” criada previamente à execução do arquivo. Todas as imagens de saída serão coloridas, no formato PNG (Portable Network Graphics) e com a mesma resolução de pixels da imagem de entrada.

## **3 - Parâmetros Utilizados**

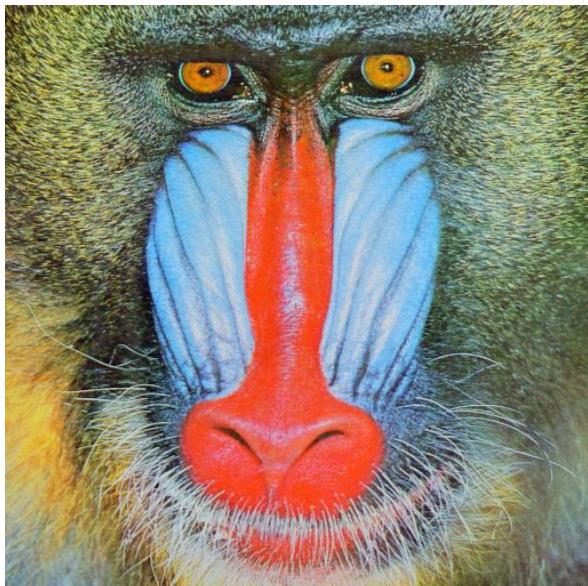
Os parâmetros utilizados para esse projeto são o path das imagens a serem codificadas e decodificadas, o arquivo texto com mensagem a ser codificada/decodificada, valores de planos de bits e nomes de arquivos de saída que podemos especificar. Todos esses parâmetros estão citados previamente neste relatório.

## **4 - Soluções**

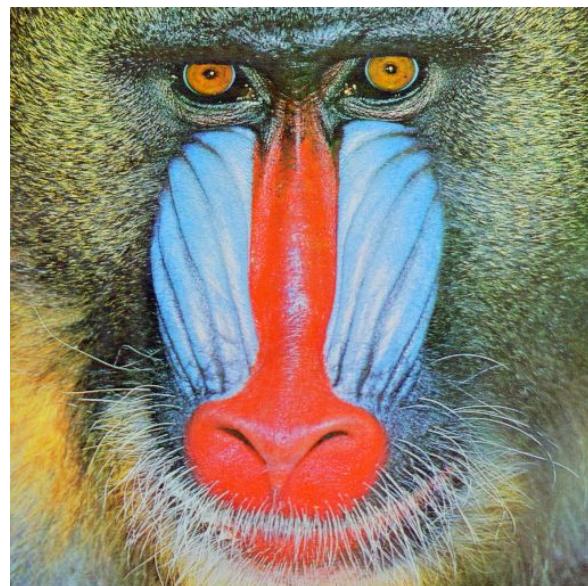
A seguir estão as comparações e observações da aplicação de cada algoritmo com suas correspondentes imagens de saída ou arquivo texto de saída. A escolha das imagens foi aleatória, na tentativa de melhor se apresentar os resultados obtidos. Além disso, tentamos apresentar as aplicações de forma mais abrangente possível.

#### 4.1 - Comparação entre imagens Originais e Codificadas

Inicialmente podemos observar que as imagens originais e codificadas são visualmente idênticas, ou seja, não é possível identificar a olho nú que uma imagem armazena uma mensagem oculta, qualquer que seja o plano de bits escolhido para codificar a mensagem. Observe as imagens a seguir:



baboon.png Original



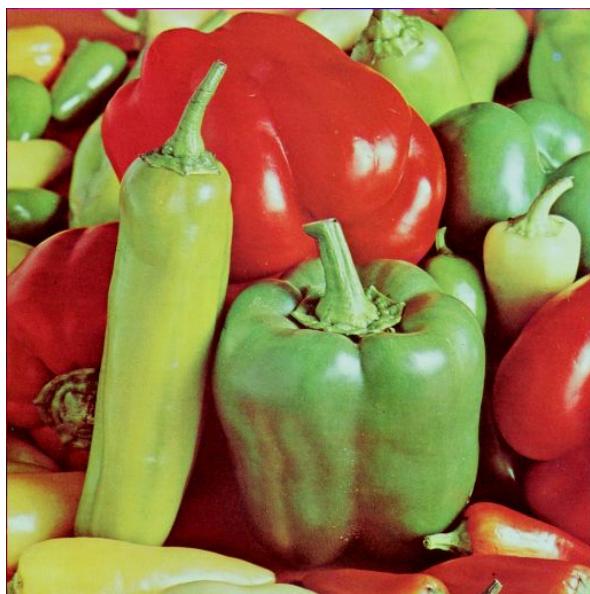
baboon.png Codificado



monalisa.png Original



monalisa.png Codificada

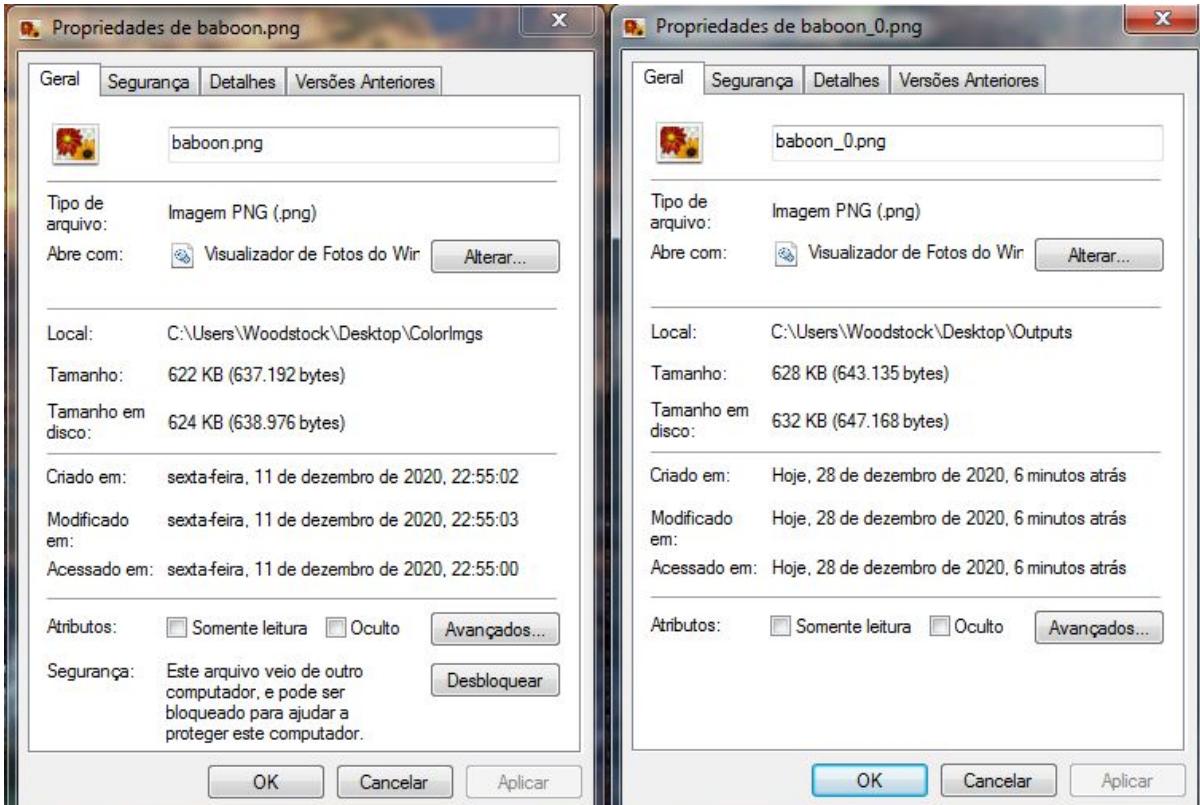


peppers.png Original



peppers.png Codificada

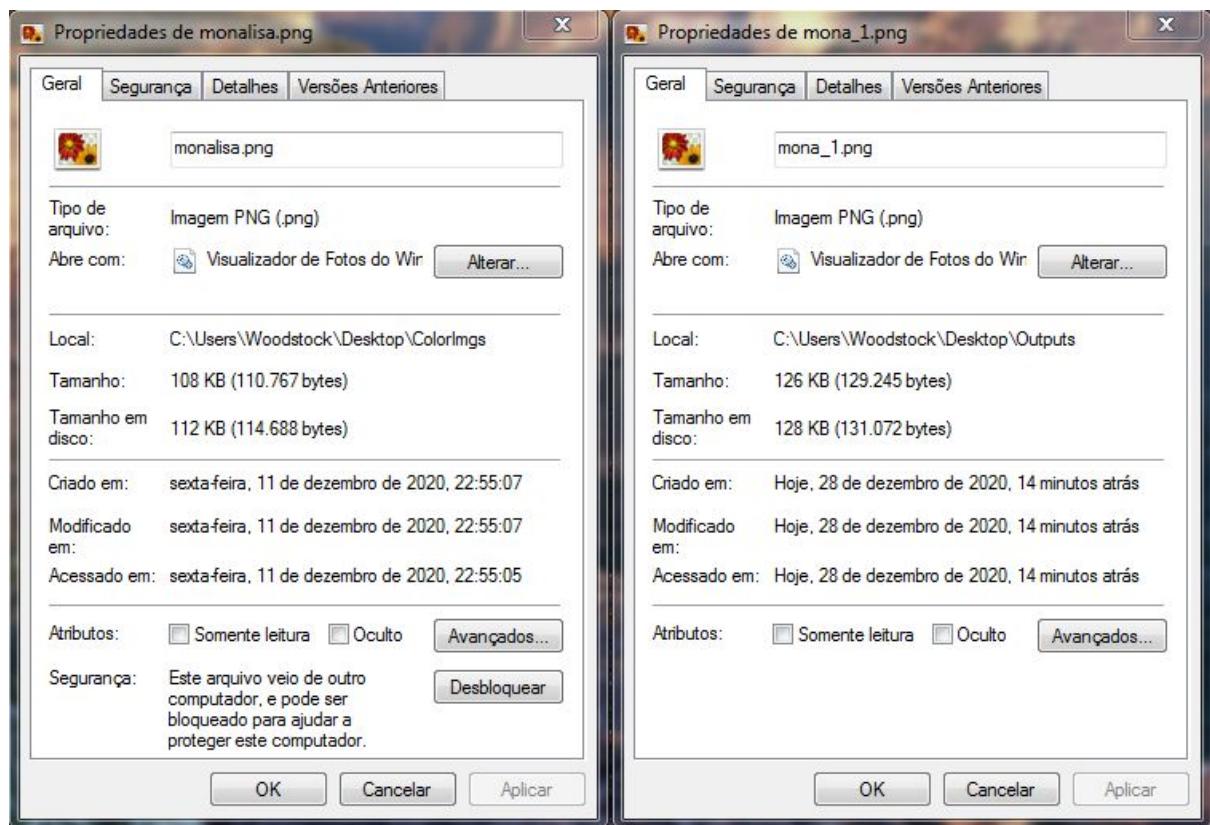
Entretanto, é possível identificar que as imagens armazenadas em disco possuem tamanhos diferentes e, portanto, podem indicar que foram modificadas e podem armazenar alguma mensagem ou arquivo oculto. Observe os quadros comparativos com os tamanhos das imagens originais e codificadas:



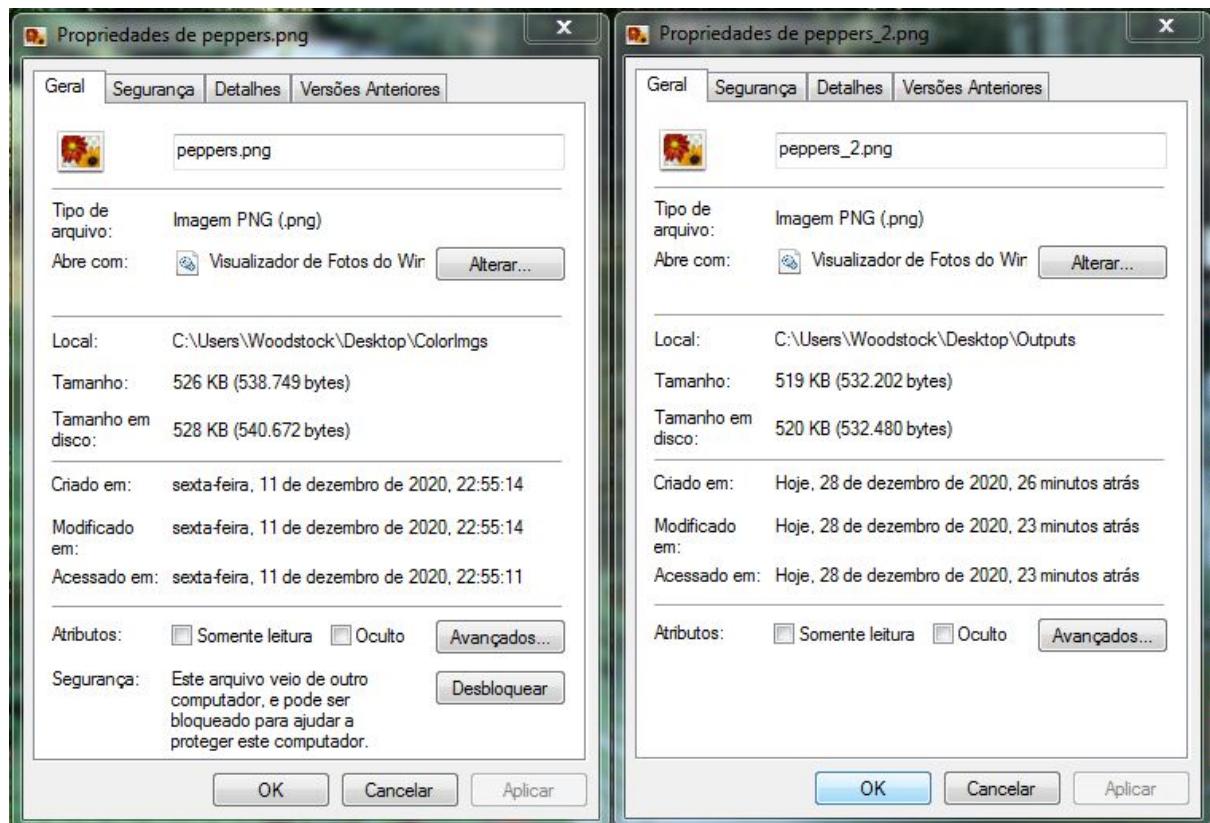
The image shows two side-by-side Windows file property dialogs. The left dialog is for 'baboon.png' and the right is for 'baboon\_0.png'. Both files are listed as 'Imagem PNG (.png)'.

Propriedade	baboon.png	baboon_0.png
Local:	C:\Users\Woodstock\Desktop\ColorImgs	C:\Users\Woodstock\Desktop\Outputs
Tamanho:	622 KB (637.192 bytes)	628 KB (643.135 bytes)
Tamanho em disco:	624 KB (638.976 bytes)	632 KB (647.168 bytes)
Criado em:	sexta-feira, 11 de dezembro de 2020, 22:55:02	Hoje, 28 de dezembro de 2020, 6 minutos atrás
Modificado em:	sexta-feira, 11 de dezembro de 2020, 22:55:03	Hoje, 28 de dezembro de 2020, 6 minutos atrás
Acessado em:	sexta-feira, 11 de dezembro de 2020, 22:55:00	Hoje, 28 de dezembro de 2020, 6 minutos atrás
Atributos:	<input type="checkbox"/> Somente leitura <input type="checkbox"/> Oculto <input type="button" value="Avançados..."/>	<input type="checkbox"/> Somente leitura <input type="checkbox"/> Oculto <input type="button" value="Avançados..."/>
Segurança:	Este arquivo veio de outro computador, e pode ser bloqueado para ajudar a proteger este computador. <input type="button" value="Desbloquear"/>	

Comparativo entre baboon Original e Modificado (localmente chamado de baboon\_0.png)



Comparativo entre monalisa Original e Modificado (localmente chamado de mona\_1.png)



Comparativo entre peppers Original e Modificado (localmente chamado de peppers\_2.png)

Obs: No último caso, ao invés do aumento de tamanho da imagem, ocorreu uma redução do tamanho em disco. Não encontramos causa aparente para tal comportamento, mas de qualquer forma, isso indica que as imagens apresentam diferenças apesar de serem visualmente iguais.

## 4.2 - Plano de bits com Mensagem/Texto curto

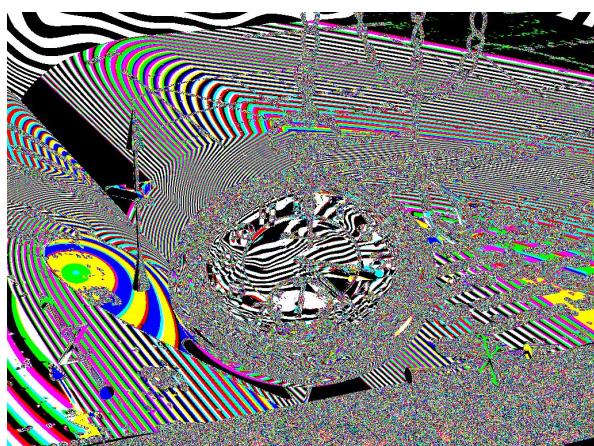
No caso da mensagem a ser codificada para dentro da imagem ser pequena, podemos verificar que para qualquer um dos planos de bits menos significativos, alterações passam quase que despercebidas. Observe um comparativo entre mensagens curtas nos diferentes planos de bits de uma mesma imagem, isto é, aplicamos uma mensagem no plano de bits 0 de uma imagem original, em seguida, na imagem de saída aplicamos novamente a codificação de uma segunda mensagem no plano de bits 1 e em seguida, fizemos o mesmo para o plano de bits 2:



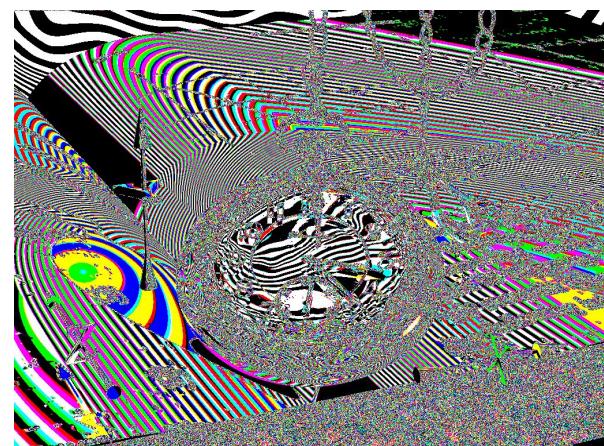
watch.png Original



watch.png com mensagens curtas nos planos de bits 0,1 e 2



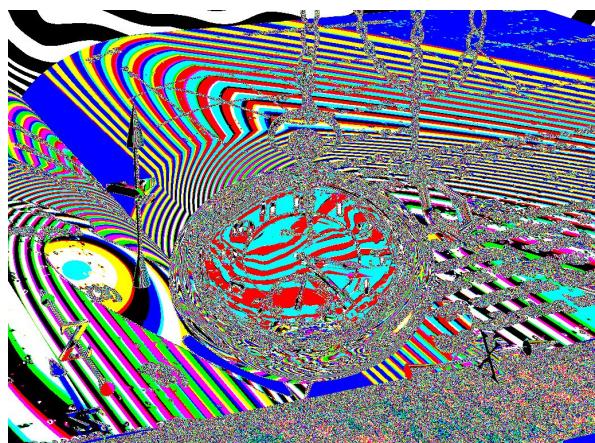
plano de bits 0 watch.png Original



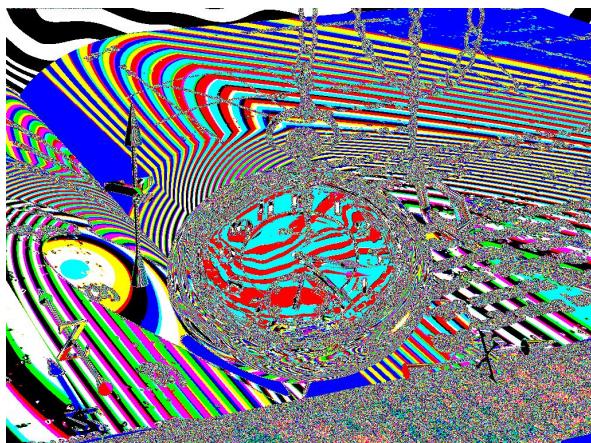
plano de bits 0 watch.png codificada

Observe a diferença sutil de coloração dos pixels bem no topo das imagens, no que seria uma primeira linha dos pixels bem fina. Essa pequena diferença indica que existe uma mensagem

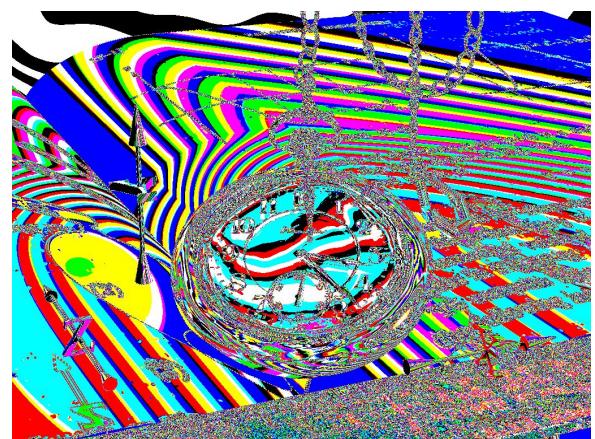
codificada em uma das imagens e pode ser imperceptível e passar despercebida. Assim, o mesmo padrão se repete para os outros planos.



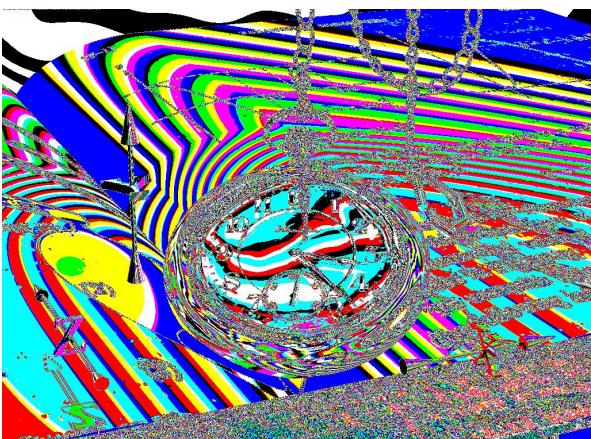
plano de bits 1 watch.png Original



plano de bits 1 watch.png codificada



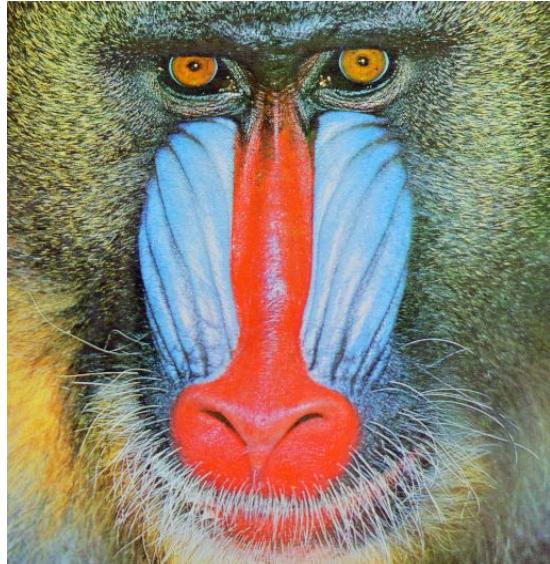
plano de bits 2 watch.png Original



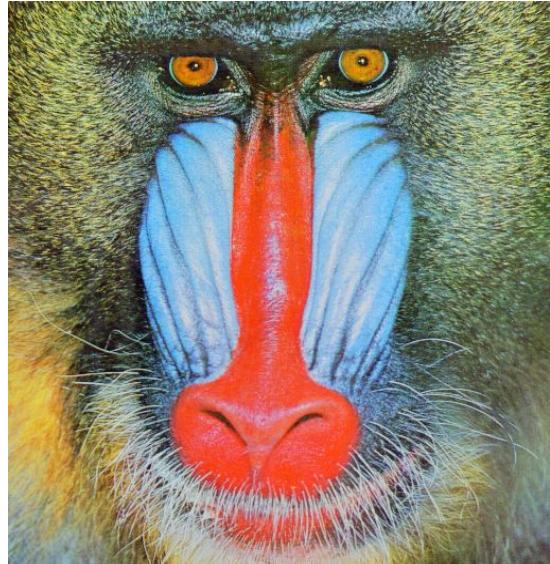
plano de bits 2 watch.png codificada

#### 4.3 - Plano de bits com Mensagem/Texto longo

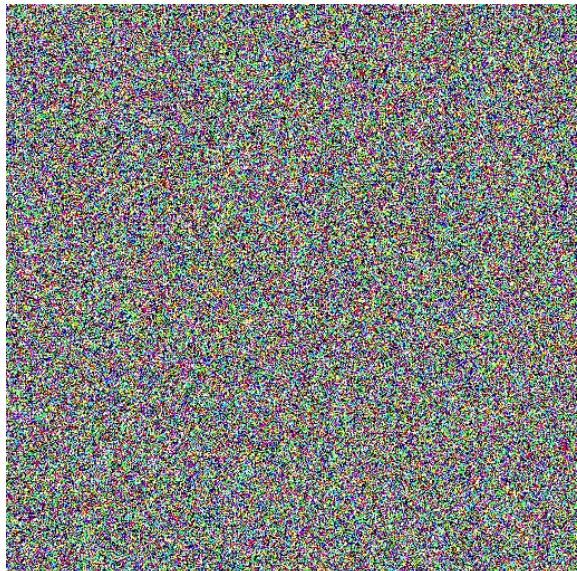
No caso da mensagem a ser codificada para dentro da imagem ser grande, é possível identificar sua presença pela distorção do plano de bits, ou seja, fazendo um comparativo entre plano de bits da imagem original e plano de bits da imagem codificada, haverá alguma diferença bastante visível. Observe um comparativo entre mensagens longas nos diferentes planos de bits de uma mesma imagem, isto é, aplicamos uma mensagem no plano de bits 0 de uma imagem original, em seguida, na imagem de saída aplicamos novamente a codificação de uma segunda mensagem no plano de bits 1 e em seguida, fizemos o mesmo para o plano de bits 2 (como feito para mensagens curtas):



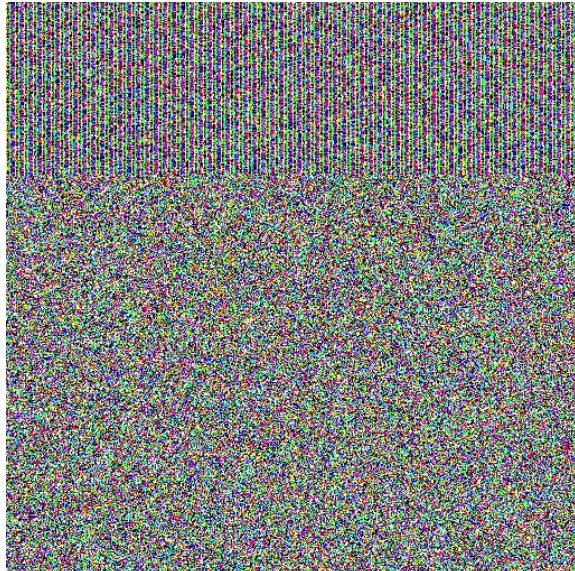
baboon.png Original



baboon.png com mensagens longas nos planos de bits 0,1 e 2

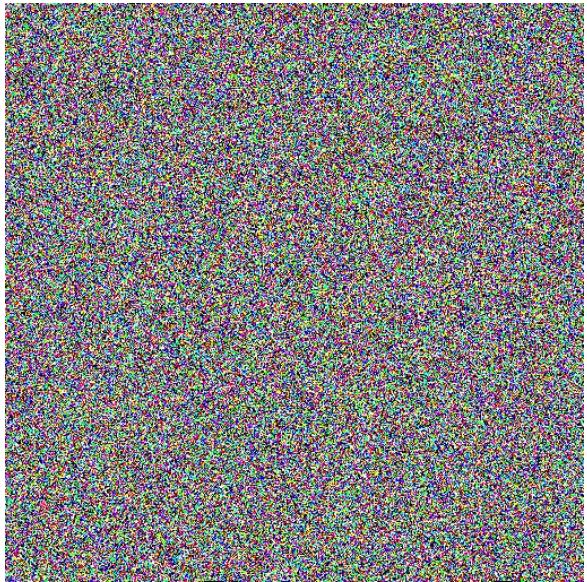


plano de bits 0 baboon.png Original

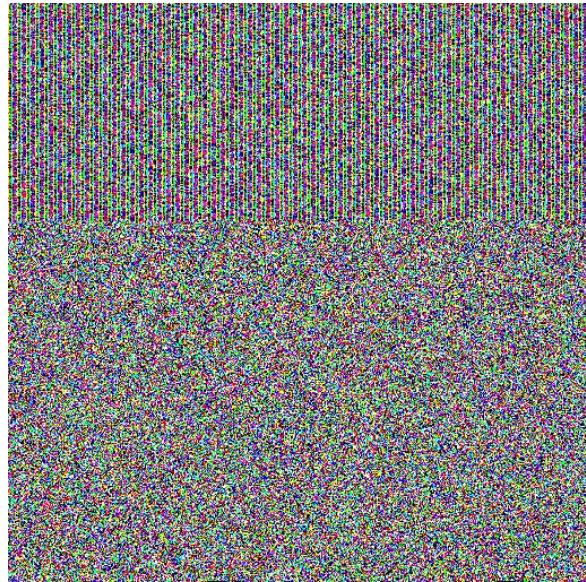


plano de bits 0 baboon.png codificada

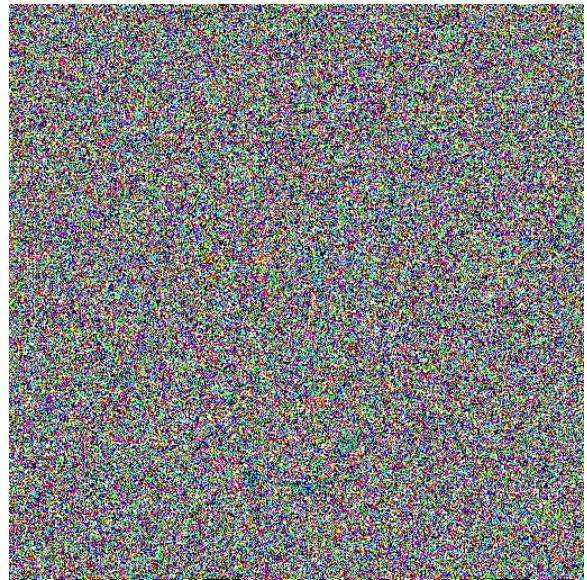
Observe que na imagem da direita temos na região superior do plano de bits o que parece ser uma distorção, que indica que os os valores dos bits estão modificados. Logo, no caso de mensagens mais longas, tal comportamento é mais visível e difícil de se esconder em qualquer um dos planos. Veja como o mesmo comportamento se repete em outros planos:



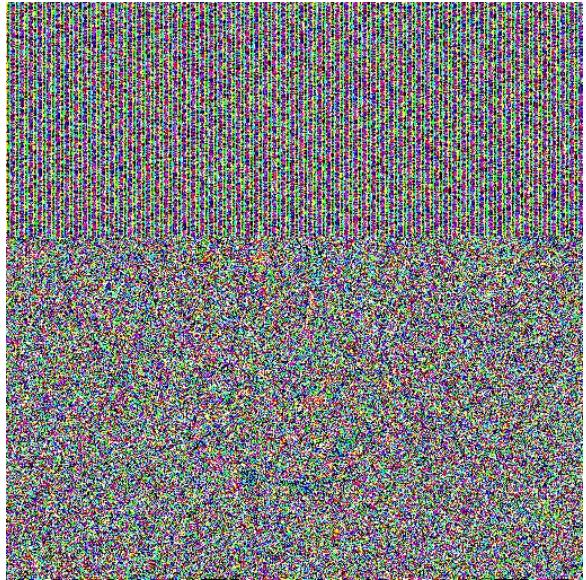
plano de bits 1 baboon.png Original



plano de bits 1 baboon.png codificada



plano de bits 2 baboon.png Original



plano de bits 2 baboon.png codificada

Além disso, podemos gerar os planos de bits com as bandas de cores azul, verde e vermelha separadamente, executando o arquivo `bitplane2.py`. Entretanto, a aparência das imagens de saída se assemelham muito com o exemplo anterior a este parágrafo, sendo imagens com uma única banda de cor. Outro ponto a se ressaltar é que, apesar de não apresentarmos os textos (curtos e longos) no relatório, é possível extraí-los das imagens codificadas utilizando o arquivo `decodificar.py` submetido junto a esse relatório. Tentar extrair uma mensagem de uma imagem não codificada pode acabar gerando erros inesperados, pois o decodificador busca por um caractere de marcação que indica a existência (início e fim) da mensagem dentro da imagem. Os arquivos utilizados para testes (imagens `.png` e textos `.txt`) foram submetidos juntos ao relatório, e estão na pasta “`ColorImgs`”, e podem ser alterados se desejado. Os comandos para execução foram previamente descritos neste relatório.

## **5 - Conclusão**

Por fim podemos concluir que o processo de esteganografia pode ser aplicado para diferentes imagens, mas não é à prova de balas pois de acordo com a mensagem/arquivo que se codifica dentro de uma imagem, tal processo pode ser identificado observando os planos de bits da imagem codificada. Outro ponto interessante é que tal processo nos trás maior aptidão para se trabalhar com planos de bits específicos em imagens, que podem ser aplicados também para se reproduzir marcas d'água em imagens restritas. Enfim, com o Projeto 4 obtivemos contato com a técnica de esteganografia de imagens coloridas, passando pelo processo de codificação, decodificação, exibição de diferentes planos de bits de uma imagem e identificação de mensagens ocultas.