Universidade Estadual de Campinas

Instituto de computação

Introdução ao processamento de imagem digital

MC920

Trabalho #4

Nome: Leonardo de Alencar Lopes **RA:** 171928

1. Introdução

O objetivo deste trabalho é implementar um algorítmo de esteganografia, que consiste em ocultar uma mensagem dentro de uma imagem. A técnica utilizada modifica um dos planos de bits da imagem para carregar a informação do binário da mensagem ao invés de informações dos pixels. Dependendo do plano de bits escolhido, pode ser quase impossível identificar que há uma mensagem codificada em uma imagem. Para sua implementação, foi utilizada a linguagem de programação Python 3 e a biblioteca de tratamento de imagens OpenCV.

Junto a este documento está o arquivo comprimido **171928-Trabalho-4.zip**. Nele há todos os scripts que foram implementados e serão citados ao longo deste texto, além das pastas: **img**, que contém as imagens de entrada utilizadas pelo algorítmo; **msg**, que contém as mensagens (textos ou imagens) a serem codificadas; **imgs_codificadas**, que contém as imagens resultantes do processo de codificação; e **msgs_encontradas**, que contém as mensagens encontradas pelo algorítmo de decodificação.

2. Os algorítmos de codificação

Para codificação das mensagens nas imagens, foram implementados os scripts *codificar.py* e *codificar_mp.py*. Ambos recebem como parâmetros uma imagem que carregará a informação desejada (esta imagem deve estar dentro da pasta **img**), uma mensagem a ser codificada e incorporada a imagem do parâmetro anterior (esta mensagem deve estar na pasta **msg** e pode ser tanto um *txt* codificado em *utf-8* quanto uma imagem *png*) e um valor inteiro no intervalo [0,7] que representa o plano de bits em que a mensagem será codificada (caso este parâmetro não seja passado, por padrão a mensagem será codificada no plano de bits 0). Mensagens do tipo *png* são codificadas como monocromáticas para ocupar menos espaço de armazenamento. Caso a msg a ser codificada seja grande demais para a imagem escolhida o programa acusará um erro e finalizará a execução. Um exemplo de execução é *python3 codificar_mp.py watch.png hino.txt 2*.

Inicialmente, a mensagem que desejamos codificar é transformada em uma lista de bytes, em que o primeiro byte informa o tipo de arquivo que está codificado (0 para *txt* e 1 para *png*), os próximos 4 bytes informam as dimensões da mensagem e os bytes subsequentes são a mensagem em si. No caso de arquivos *txt*, os bytes de dimensões informam o tamanho total do texto. Para arquivos *png*, os bytes 2 e 3 são os mais e menos significativos da altura da imagem,

respectivamente, e os 4 e 5 são os mais e menos significativos da largura da imagem, respectivamente.

Table 1: Formato de codificação dos bytes

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	...	Byte n+5
Tipo (1)	$Altura_{MSB}$	$Altura_{LSB}$	$Largura_{MSB}$	$Largura_{LSB}$	Msg_{Byte-1}	Msg_{Byte-2}	...	Msg_{Byte-n}
Tipo (0)	$Tamanho_1$	$Tamanho_2$	$Tamanho_3$	$Tamanho_4$	Msg_{Byte-1}	Msg_{Byte-2}	...	Msg_{Byte-n}

Em seguida, a imagem que carregará a mensagem é separada nos planos de bits que compoem cada uma de suas bandas do RGB. Assim, cada plano de bits é constituído por 3 matrizes de bits. O plano de bits a ser utilizado é aquele informado pelo usuário ou o escolhido por padrão (0), e nele o vetor de bytes da mensagem será codificado. Os bits de cada byte são incorporados nas 3 matrizes do plano de forma sequencial, de maneira que o i-ésimo bit da mensagem é colocado no próximo espaço ainda não alterado da matriz da banda azul se *i%3 == 0*, na banda verde se *i%3 == 1* e na banda vermelha se *i%3 == 2*. Com isso, todas as bandas que formam esse plano de bits são utilizadas para guardar a mensagem desejada. Espaços das matrizes que não contem bits codificados da mensagem não são alterados.

Na sequência, a imagem é remontada com um de seus planos de bits tendo a mensagem codificada. Durante os testes realizados, foi notado que o trecho de código reponsável por realizar essa tarefa é o que necessita de mais processamento. Inicialmente, foi desenvolvido o script *codificar.py* que realiza essa etapa de forma sequencial para arquivos *txt*, ou seja, as 3 bandas são montadas pela mesma thread. Visando acelerar esta etapa da codificação da mensagem, foi implementado o script *codificar_mp.py*, que utiliza a biblioteca multiprocessing para separar o processamento da montagem das 3 bandas entre 3 threads diferentes. A diferença no tempo de processamento total do programa foi notória e pode ser observada na tabela a seguir, o que levou a codificação de arquivos *png* também realizarem essa etapa de forma paralela. Parâmetros utilizados para gerar a tabela: *watch.png hino.txt 0.*

Table 2: Comparação entre processamento paralelo e sequencial

	Processamento sequencial (s)	Processamento paralelo (s)
1	40.66	17.53
2	43.44	18.43
3	43.66	18.72
4	44.68	19.16
5	46.58	18.04
Média	43.80	18.38

Por fim, a imagem remontada com a mensagem codificada será salva na pasta **imgs_codificadas**. Em seu nome será informado o nome do arquivo que contém a mensagem codificada, o plano de bits que ocorreu a codificação e o nome da imagem original. Por exemplo: *hino-0-watch.png*.

3. O algorítmo de decodificação

Para decodificação, foi implementado o script *decodificar.py*, que recebe como parâmetros uma imagem *png* com uma mensagem codificada (deve estar dentro da pasta **imgs_codificadas**) e um valor inteiro no intervalo [0,7] que informa o plano de bits que a mensagem está codificada (caso este parâmetro não seja passado, por padrão o algorítmo buscará a mensagem no plano 0). Caso a mensagem codificada não tenha definido o primeiro byte como um valor válido para o tipo de arquivo, o programa acusará erro e encerrará a execução. Porém, pode acontecer de o primeiro byte ser um valor válido mesmo não tendo uma mensagem codificada naquele plano. Neste caso, o programa não identificará esta falha e executará normalmente, apresentando um resultado inválido.

Inicialmente o processo de decodificação se assemelha muito ao de codificação, pois separa a imagem de entrada em seus planos de bits para facilitar a leitura. Os bytes são lidos no formato descrito anteriormente, alternando entre as 3 matrizes das bandas RGB após cada bit. O primeiro byte identifica o tipo de arquivo que foi codificado na imagem e determina como a mensagem será decodificada (*png* ou *txt*). Os próximos 4 bytes identificam o tamanho da mensagem, no caso de uma imagem, também informa suas dimensões. Em seguida, são lidos **n** bytes e organizados em uma lista.

Por fim, a mensagem é remontada e gravada em disco na pasta **msgs_encontradas**. Para mensagens de texto, é realizada a decodificação do vetor de bytes para o padrão *utf-8*. Para imagens, é realizada a remontagem com base nas dimensões que foram codificadas. O nome do arquivo que será gravado informa que é uma mensagem e o nome da imagem que foi decodificada.

Diferente do processo de codificação, não é necessário remontar a imagem de entrada. Como o trecho de código responsável por este tarefa é o mais custoso em processamento, o programa de decodificação executa de forma expressivamente mais rápida do que o de codificação. Durante os testes, seu tempo médio de execução foi de 0.328 segundos para o comando *python3 decodificar.py watch.png hino.txt 0*.

4. Resultados

A imagem gerada pelo processo de codificação possui um de seus planos de bits corrompido pela mensagem que foi escondida. Isso significa que, dependendo do plano escolhido, pode ocorrer de não se obter uma mensagem realmente criptografada, já que distorções geradas podem ficar muito nítidas. Por isso, é recomendado codificar mensagens em planos de ordens mais baixas, preferencialmente o plano de ordem zero, que afeta os bits menos significativos de cada pixel. As imagens abaixo exemplificam o resultado da codficação. Nelas foram codificadas a Declaração Universal dos Direitos Humanos na imagem *monalisa.png*.

Figure 1: Original monalisa.png

Figure 2: Mensagem codificada no plano 0

Figure 3: Mensagem codificada no plano 1

Figure 4: Mensagem codificada no plano 2

Figure 5: Mensagem codificada no plano 3

Figure 6: Mensagem codificada no plano 4

Figure 7: Mensagem codificada no plano 5

Figure 8: Mensagem codificada no plano 6

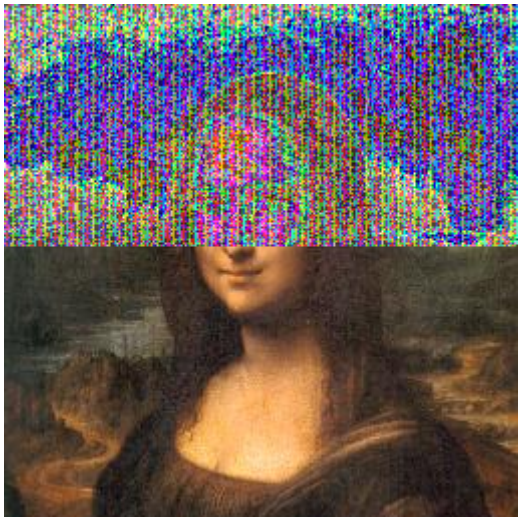

Figure 9: Mensagem codificada no plano 7

A partir do plano de ordem 4 já é possível identificar distorções graves na imagem, invalidando a utilização da esteganografia para esses casos. Utilizando zoom e comparando diretamente com a imagem original, é possível identificar imperfeições para a codificação no plano 3. Os planos 0, 1 e 2 tiveram resultados satisfatórios em esconder uma mensagem na imagem, sendo os mais indicados para utilização.

Como a esteganografia nos menores planos de bits praticamente não gerou distorções, foi realizado um teste que codificou uma mensagem em cada um desses planos em uma mesma imagem. O resultado obtido foi satisfatório, não sendo possível notar distorções na imagem e aumentando a quantidade de informação que a mesma imagem consegue esconder. A imagem obtida deste teste pode ser observada a seguir, em que foi codificada o arquivo *direitos.txt* em seus 3 planos de bits menos sifnificativos.

Figure 10: Codificação de mensagem nos 3 planos de bits menos significativos

5. Conclusões

A esteganografia utilizando a técnica de codificação dos bytes de uma mensagem nos planos de bits que compõem uma imagem se mostrou eficiente para os 3 planos menos significativos, pois não geram distorções visíveis. Além disso, utilizar estes planos simultaneamente aumenta a quantidade de informação que pode ser codificada, permitindo incorporação de outros tipos de arquivos mais complexos, como pequenos audios.

Apesar de conseguir um ganho de desempenho significativo no programa de codificação, para imagens de alta resolução ainda há uma espera para finalizar sua execução. Esse tempo pode ser melhorado utilizando algorítmos mais eficientes para a remontagem da imagem. Por outro lado, o programa de decodificação executa de forma quase instantânea.

Mesmo conseguindo um bom resultado na esteganografia, não é indicada a utilização destes scripts para esconder mensagens importantes. A maneira como a mensagem é codificada é bem simples e sujeita a quebra do seu segredo. Para melhorar a segurança dos scripts, é necessário mudar a forma como os bytes são codificados nos planos de bits, não seguindo padrões tão óbvios quanto este utilizado para este trabalho.