**Projeto**
Esteganografia em Assembly

# 1 Introdução

## 1.1 O que é esteganografia?

A esteganografia é uma área que estuda diversas técnicas para ocultar a **existência** de uma mensagem dentro de outra. Ela difere da criptografia, pois esta última foca-se no estudo de técnicas para ocultar o **significado** de uma mensagem. As duas áreas complementam-se pois um indivíduo pode querer transmitir uma mensagem escondida em outra (i.e., esteganografia) ao mesmo tempo que esta mensagem será enviada cifrada (i.e., criptografia). Neste caso, mesmo que alguém consiga interceptar a mensagem oculta, não conseguirá decifrá-la para ler o seu conteúdo.

## 1.2 Exemplos práticos de esteganografia

Em meios digitais, o conteúdo de uma mensagem pode ser de qualquer tipo de dados que possa ser representado através de dígitos binários. Texto, documentos, imagens, áudios e vídeos são exemplos comuns de tipos de dados enviados em mensagens.

Relativamente à esteganografia em meios digitais, um dos exemplos mais comuns consiste em **esconder uma mensagem de texto numa imagem**, o qual será o tema deste projeto. Outros exemplos, ortogonais a este projeto, incluem esconder textos e imagens em músicas por diversão, marcas d'água em imagens e vídeos para a proteção de direitos de autor e de distribuição, textos subliminares em notícias para contornar meios de comunicação sob censura, etc.

# 2 Contexto

## 2.1 Exemplo de um fluxo de execução

Para contextualizar os objetivos de esconder uma mensagem de texto numa imagem e recuperar esta mensagem, é apresentado o fluxo de execução da Figura 1. Neste exemplo, um *Remetente* escreve uma *Mensagem* num ficheiro de texto e escolhe uma *Imagem* onde a *Mensagem* será escondida. O *Remetente* executa então um programa, chamado *Esconder*, o qual aplica a esteganografia para esconder a *Mensagem* de texto na *Imagem*. O programa receberá apenas três argumentos: o caminho para o ficheiro da *Mensagem* de texto, o caminho para o ficheiro da *Imagem* original e o caminho onde deverá ser escrito o ficheiro da *Imagem** modificada.

Após a *Mensagem* ser ocultada na *Imagem* original, a *Imagem** modificada resultante pode ser enviada através de um meio de comunicação qualquer para um *Recetor*. Caso a *Imagem** modificada seja interceptada por alguém (p. ex., um *Atacante*), este indivíduo apenas verá uma imagem normal, acima de qualquer suspeita. Assumindo que o *Atacante* não conheça esteganografia ou que ele não esteja a verificar se esta técnica foi aplicada nas imagens obtidas, ele então descartará a imagem intercetada.

A seguir, o *Recetor* recebe o ficheiro da *Imagem** modificada. Ele então executa outro programa, chamado *Recuperar*, o qual extrairá a *Mensagem* de texto desta imagem. Este programa recebe, como argumento, apenas o caminho para o ficheiro da *Imagem** modificada e, como resultado, imprime na saída do terminal (*stdout*) a *Mensagem* de texto recuperada.

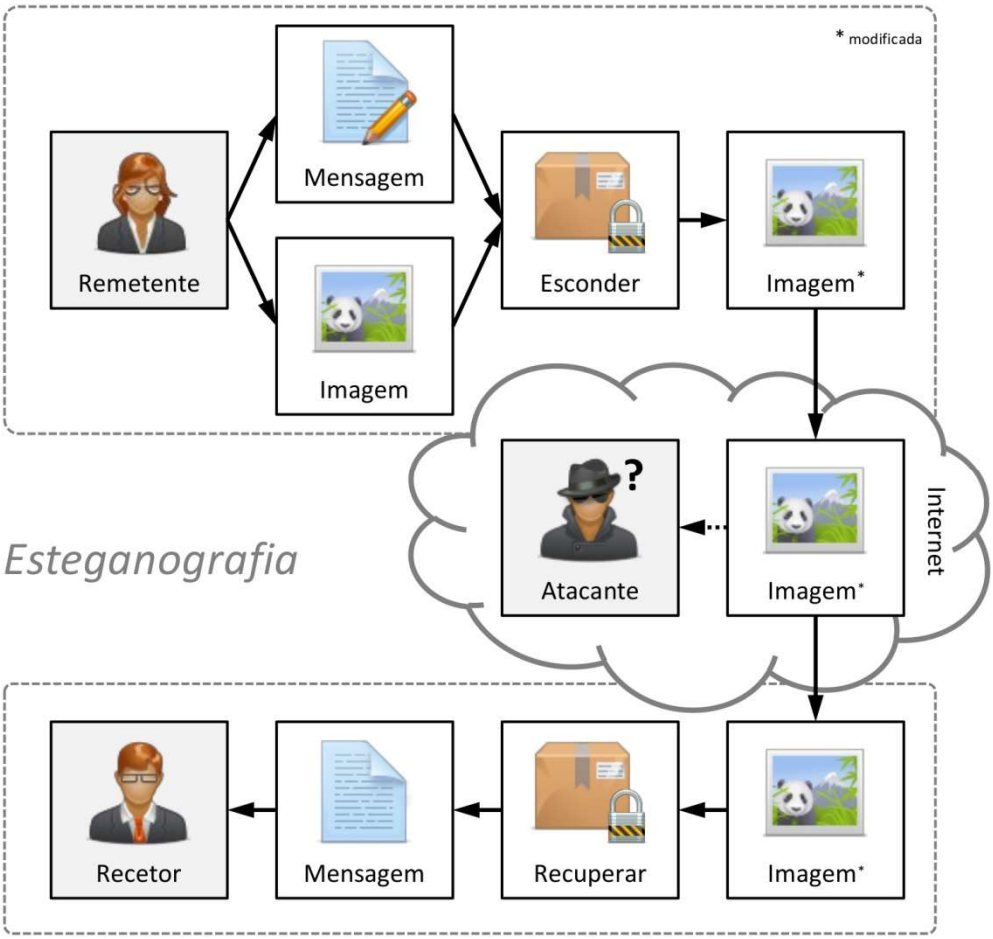


*Figura 1: Exemplo de fluxo de execução do uso da esteganografia.*

Nas próximas duas secções são detalhados os formatos de ficheiros que serão utilizados neste projeto: o formato de texto TXT e o formato de imagem *bitmap* (BMP). A seguir, também será apresentada uma introdução sobre como esconder uma mensagem de texto num ficheiro de imagem.

## 2.2 O formato de texto TXT

O formato de texto TXT armazena os caracteres de uma mensagem. Cada caractere da mensagem é codificado em um byte (8 bits) conforme os valores definidos pela tabela da convenção ASCII. Como os valores em decimal da tabela ASCII vão do 0 ao 127, estes poderiam ser armazenados com apenas 7 bits cada. Porém para facilitar o trabalho, vamos assumir 1 byte por caractere, onde o bit mais significativo é sempre o 0 (zero) e os sete bits menos significativos são os bits normais do valor em ASCII.

Este formato não possui cabeçalho, pelo que o primeiro byte do ficheiro corresponde já ao byte do primeiro caractere. O término do ficheiro é marcado pelo caractere ASCII de valor 0 (`0x00`, *zero*), pois este corresponde ao *null*. Note também que é possível haver caracteres especiais que representam uma nova linha (`0x0A`, *new line*), uma tabulação (`0x09`, *horizontal tab*), etc.

Há diversas formas de se criar um ficheiro de texto TXT, seja através da linha de comandos da consola, de um editor de texto em modo texto na consola (p. ex., o *nano* no Linux), ou um editor de texto em modo gráfico (p. ex., o *gedit* no Linux ou o *Notepad* no Windows).

Por exemplo, para escrever e ler um ficheiro de texto a partir da linha de comandos da consola, pode-se utilizar os seguintes comandos:

```
$ echo "Uma mensagem de texto" > mensagem.txt
$ cat mensagem.txt
Uma mensagem de texto
```

## 2.3 O formato de imagem *bitmap* (BMP)

Os ficheiros *bitmap* (BMP) são ficheiros de imagens onde o valor exato de cada píxel (*px*) da imagem é apresentado explícitamente. Os ficheiros no formato BMP são divididos em duas partes: o cabeçalho e a secção dos píxeis. Estas duas partes são apresentadas na Figura 2.

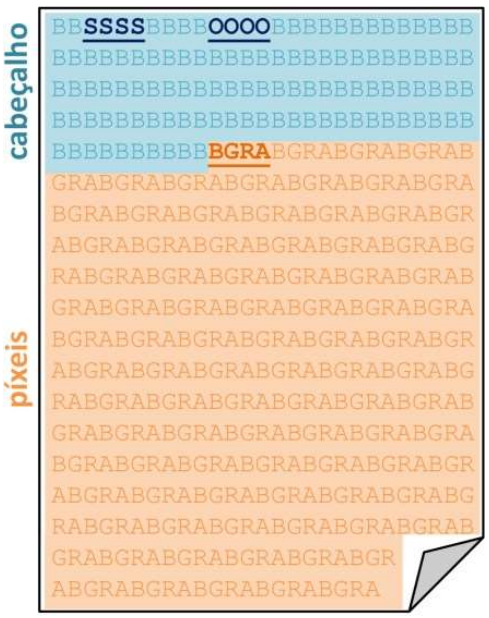

*Figura 2: Estrutura de um ficheiro BMP*

O cabeçalho pode seguir uma de diversas especificações existentes e pode ter um tamanho variável. Porém, para este trabalho, será utilizada a especificação ARGB32 (que será explicada a seguir) e as únicas informações que serão importantes ler do cabeçalho serão o tamanho do ficheiro (*size*), o qual está indicado como **SSSS** na Figura 2, e o deslocamento (*offset*), o qual está indicado como **OOOO** na mesma figura. O *size* é um número de 4 bytes de tamanho que começa após o 2º byte do início do ficheiro e indica o tamanho total do ficheiro BMP em bytes. O *offset* também é um número com 4 bytes de tamanho, porém encontra-se após o 10º byte do início do ficheiro e indica exatamente em qual byte a secção dos píxeis inicia. Tanto o *size* como o *offset* estão no formato *little-endian*.

A secção dos píxeis começa a partir desta posição de deslocamento (*offset*) e é composta por conjuntos também de 4 bytes (i.e., 32 bits) por píxel, representados por **BGRA** na Figura 2. Cada conjunto BGRA contém 1 byte para a intensidade (um número entre 0 e 255) da cor azul (B), 1 byte para o verde (G), 1 byte para o vermelho (R) e 1 byte para a transparência (A, que representa o chamado canal *alpha*).

Esta especificação é chamada ARGB32 e em arquiteturas *little-endian* utilizam esta sequência BGRA para cada píxel. Portanto, um píxel composto pelos bytes `0x0000FFFF` é um píxel vermelho puro sem transparência, um `0x00FF00FF` é um verde puro, um `0xFF0000FF` é um azul puro, um `0x00FFFFFF` é um píxel amarelo e assim por diante. Qualquer píxel do tipo `0x??????00` é um píxel transparente. Neste trabalho, vamos sempre utilizar o valor `0xFF` (i.e., 255) para a transparência (ou seja, o byte A de cada píxel). O fim da secção dos píxeis (e consequentemente do ficheiro BMP) corresponde ao byte na posição indicada pelo número **SSSS** da Figura 2.

## 2.4 Como esconder uma mensagem de texto numa imagem?

Para o caso específico deste projeto, vamos esconder uma mensagem de texto (presente num ficheiro TXT) numa imagem no formato BMP. Primeiramente, obtém-se o conteúdo binário da mensagem de texto, onde cada caractere encontra-se codificado em 1 byte. A seguir, é preciso ler o ficheiro da imagem original onde vai se esconder a mensagem e detetar o início da secção dos píxeis nesta imagem (ver a Secção 2.3).

Após estes passos, para esconder uma mensagem de texto numa imagem, coloca-se cada bit da mensagem de texto no bit menos significativo (LSB, do inglês *Least Significant Bit*) de cada byte das cores dos píxeis da imagem (BGR). Os bytes do canal alpha (A) dos píxeis da imagem não serão utilizados para guardar os bits da mensagem. Isso evita variações de transparência na imagem modificada que poderiam chamar a atenção caso a imagem fosse intercetada.

Como apenas o bit menos significativo de cada byte de cor é alterado, o resultado desta transformação será uma imagem visualmente idêntica à imagem original, porém com uma mensagem escondida nela. Devido ao facto de cada píxel ter 3 bytes de cores (para além de 1 byte para o canal alpha), será possível guardar 3 bits da mensagem por cada píxel da imagem. A título de curiosidade, este modelo implica que uma imagem com `N×N`*px* possa guardar até $(N^2 \times 3)/8$ caracteres duma mensagem.

A ordem com que os bits da mensagem de texto são inseridos nos bytes do píxeis de cor da imagem é muito importante. Esta mesma ordem deve ser seguida tanto para esconder a mensagem na imagem, como para recuperá-la. Mais especificamente, deve-se respeitar a seguinte ordem:

- O **1º** bit do **1º** caracter da mensagem será guardado no LSB do 1º byte (B) do 1º píxel da imagem.
- O **2º** bit do **1º** caracter da mensagem será guardado no LSB do 2º byte (G) do 1º píxel da imagem.
- O **3º** bit do **1º** caracter da mensagem será guardado no LSB do 3º byte (R) do 1º píxel da imagem.
- Nada será guardado no 4º byte (A) do 1º píxel, pois este é o byte do canal alpha.
- O **4º** bit do **1º** caracter da mensagem será guardado no LSB do 1º byte (B) do 2º píxel da imagem.
- ...
- O **8º** bit do **1º** caracter da mensagem será guardado no LSB do 2º byte (G) do 3º píxel.
- O **1º** bit do **2º** caracter da mensagem será guardado no LSB do 3º byte (R) do 3º píxel.
- Novamente, nada será guardado no 4º byte (A) do 3º píxel.
- O **2º** bit do **2º** caracter da mensagem será guardado no LSB do 1º byte (B) do 4º píxel.
- E assim por diante.

### 2.5 Ficheiros de apoio ao projeto

Junto ao enunciado, é disponibilizado um ficheiro ZIP (***bitmaps.zip***) com diversos ficheiros BMP para auxiliar a implementação do projeto. Estes ficheiros estão divididos em dois grupos:

- **Ficheiros básicos:** Estes ficheiros servem para auxiliar a coomprensão dos ficheiros BMP e auxiliar nos estágios iniciais da implementação, pois possuem uma complexidade baixa ao nível de cores. São disponibilizados vários ficheiros com tamanhos `1×1`*px*, `10×10`*px* e `100×100`*px* com os píxeis na cor branca, preta, vermelha, verde ou azul.
- **Ficheiros de teste:** Estes ficheiros servem para auxiliar a implementação dos programas e realizar testes para as fases das entregas. São disponibilizados dois ficheiros, cada um com um tamanho de `150×150`*px*, os quais possuem uma complexidade elevada de cores. O ficheiro ***fcul.bmp*** auxiliará a implementação do programa Esconder, pois ele é um ficheiro original que não possui nenhuma mensagem inserida nele. O ficheiro ***fcul_mod.bmp*** auxiliará a implementação do programa Recuperar pois ele é um ficheiro que já possui uma mensagem oculta nele, a qual foi inserida através do uso de esteganografia.

Para além destes ficheiros, os alunos podem futuramente criar ou utilizar qualquer outra imagem BMP com o código implementado, desde que a especificação utilizada nos ficheiros BMP seja a ARGB32 (conforme mencionado na Secção 2.3). O código pode ainda ser modificado para aceitar outras especificações, mas isto é um trabalho fora do escopo deste projeto.

## 3 Implementação

### 3.1 Fases da implementação

A implementação do projeto está dividida em duas fases, com datas de entrega separadas:

1) **Recuperar:** A primeira fase consiste em implementar apenas o programa ***fcXXXXX_Recuperar.asm***, onde XXXXX é o seu número de aluno. Esta é a parte mais simples e permitirá aos alunos contextualizarem-se melhor com o formato BMP e com algumas funcionalidades que serão úteis também na fase seguinte. Como descrito na Secção 2.5, é vos dado um ficheiro BMP (***fcul_mod.bmp***) com uma mensagem oculta, a qual o vosso programa deverá recuperar.
2) **Esconder:** A segunda fase consiste em implementar o programa ***fcXXXXX_Esconder.asm***, pois este é mais complexo e o seu resultado deverá funcionar corretamente com o programa Recuperar criado na fase anterior. Neste caso, os alunos devem utilizar a outra imagem BMP de teste (***fcul.bmp***), pois esta é considerada uma imagem original, sem nenhuma mensagem oculta nela.

### 3.2 Detalhes de compilação e execução

- O programa ***fcXXXXX_Recuperar.asm*** será compilado e executado com os seguintes comandos:
```
$ nasm -F dwarf -f elf64 fcXXXXX_Recuperar.asm
$ ld -o Recuperar fcXXXXX_Recuperar.o
$ ./Recuperar fcul_mod.bmp
```
- De modo semelhante, o programa ***fcXXXXX_Esconder.asm*** será compilado e executado com os seguintes comandos:
```
$ nasm -F dwarf -f elf64 fcXXXXX_Esconder.asm
$ ld -o Esconder fcXXXXX_Esconder.o
$ ./Esconder mensagem.txt fcul.bmp fcul_mod.bmp
```
- Caso seja criada uma biblioteca de funções auxiliares ***fcXXXXX_Biblioteca.asm***, esta será compilada e ligada aos outros códigos com os seguintes comandos:
```
$ nasm -F dwarf -f elf64 fcXXXXX_Biblioteca.asm
$ ld -o Esconder fcXXXXX_Esconder.o fcXXXXX_Biblioteca.o
$ ld -o Recuperar fcXXXXX_Recuperar.o fcXXXXX_Biblioteca.o
```

### 3.3 Detalhes e sugestões para a implementação

Algumas funcionalidades que serão implementadas podem servir tanto para o Esconder quanto para o Recuperar. Pode-se criar um ficheiro com funções auxiliares que serão utilizadas por cada um destes programas, para evitar ter código repetido no Recuperar e no Esconder. Além disso, muitas das funcionalidades a serem implementadas já foram tema das aulas de ASC ou são abordadas nas referências bibliográficas da disciplina. Seguem alguns exemplos:

- A escrita de mensagens na saída do terminal (*stdout*): Permitirá escrever a mensagem resultante do programa Recuperar. Este tema foi abordado nos guiões [ASC3] e [ASC4a];
- A obtenção dos argumentos passados a um programa: Permitirá receber, através da linha de comandos, o caminho para os ficheiros a serem utilizados pelos programas. Este tema foi abordado no Exercício 3 do guião [ASC4a];
- A leitura de ficheiros: Permitirá ler a mensagem do ficheiro TXT a ser ocultada na imagem, assim como ler as imagens BMP (p.ex., a imagem original no programa Esconder e a imagem modificada no programa Recuperar). Este tema é abordado pela Secção 13.8.2 do livro [Jor20]. Note que na parte final da secção "*Read from file*" do código deste exemplo, o registo *rax* guarda na verdade a quantidade de bytes que foram lidos do ficheiro.
- A escrita de ficheiros: Permitirá escrever a imagem BMP modificada pelo programa Esconder. Este tema é abordado pela Secção 13.8.1 do livro [Jor20];
- Encontrar o tamanho (*size*) e o deslocamento (*offset*) nos ficheiros BMP: Permitirá saber onde começa e onde termina a secção dos píxeis nestes ficheiros. Ver a Secção 2.3.
- Percorrer a memória byte-a-byte: Permitirá obter separadamente cada byte da mensagem ou cada byte dos píxeis da imagem BMP. Este tema foi abordado em diversos exemplo de endereçamento.
- Criar funções e bibliotecas: Permitirá criar ficheiros com bibliotecas de funções que serão utilizadas por outros programas, evitando que o código esteja repetido em múltiplos ficheiros. Este tema foi abordado na Secção 1.2 do guião [ASC4b].

## 4 Entrega do trabalho

### 4.1 Datas das entregas

O trabalho é **individual** e deverá ser entregue nas seguintes datas:

- 1º fase [Recuperar]: Até as 23:59 do dia 03/01/2021.
- 2º fase [Esconder]: Até as 23:59 do dia 17/01/2021.

### 4.2 Formato da entrega

- A entrega será feita através do Moodle, antes do limite do período de entrega, em local indicado para tal.
- Os ficheiros Assembly devem ser gravados com os nomes ***fcXXXXX_Recuperar.asm*** e ***fcXXXXX_Esconder.asm***, onde XXXXX é o seu número de aluno. Pode-se incluir um ficheiro ***fcXXXXX_Biblioteca.asm*** caso queiram separar as funções auxiliares.
- Os alunos devem incluir, no início de cada ficheiro Assembly entregue, uma linha de comentário com o seu número de aluno (p.ex., "; fcXXXXX").
- Os ficheiros Assembly devem ser colocados num ficheiro comprimido ZIP com o número de aluno como o nome do ficheiro (p.ex., ***fcXXXXX.zip***).
- Na 1ª fase, o ficheiro ***fcXXXXX.zip*** deverá conter apenas o ficheiro ***fcXXXXX_Recuperar.asm*** e opcionalmente o ficheiro ***fcXXXXX_Biblioteca.asm***. Na 2ª fase, o ficheiro ***fcXXXXX.zip*** deverá conter também o ficheiro ***fcXXXXX_Esconder.asm***.

### 4.3 Critérios de avaliação

- Funcionamento dos programas Esconder e Recuperar conforme especificado neste enunciado. Serão executados os comandos apresentados na Secção 3.2 com os ficheiros de teste, para além de outros ficheiros BMP e mensagens de texto.
- Implementação das funcionalidades mencionadas na Secção 3.3.
- Conformidade com o formato de entrega descrito na Secção 4.2.
- Organização dos códigos em Assembly. Aspetos que serão valorizados incluem criar uma biblioteca de funções auxiliares, respeitar as convenções de chamada de funções, aplicar boas práticas de uso da memória, evitar referências à memória desnecessárias, etc.
- Documentação dos códigos em Assembly. Aspectos que serão valorizados incluem criar cabeçalhos de documentação nas funções, comentar o funcionamento de trechos de códigos mais complexos e não-triviais, etc.

### 4.4 Plágio

Não é permitido aos alunos partilharem código com soluções, ainda que parciais, a nenhuma parte do projeto com outros alunos (nem através do Fórum da disciplina, nem por qualquer outro meio). Além disso, todos os códigos serão testados por um sistema verificador de plágio. Caso alguma irregularidade seja encontrada, os projetos de todos os alunos envolvidos serão anulados e o caso será reportado aos órgãos responsáveis na Ciências@ULisboa.

## 5 Bibliografia

[ASC3] M. Calha, N. Magaia, V. Cogo. (2020). Guião Arq3 – Lab. Disponível online em https://moodle.ciencias.ulisboa.pt/mod/resource/view.php?id=125539.

[ASC4a] M. Calha, N. Magaia, V. Cogo. (2020). Guião Arq4 - Exercícios. Disponível online em https://moodle.ciencias.ulisboa.pt/mod/resource/view.php?id=126826.

[ASC4b] M. Calha, N. Magaia, V. Cogo. (2020). Guião Arq4 - Lab. Disponível online em https://moodle.ciencias.ulisboa.pt/mod/resource/view.php?id=127701.

[Jor20] Ed Jorgensen. (2020) x86-64 Assembly Language Programming with Ubuntu. Disponível online em http://www.egr.unlv.edu/~ed/x86.html.