# Esteganografia

L'esteganografia és la ciència d'amagar dades a la vista. Concretament en el món digital, l'esteganografia (el nom de la qual és la unió de les paraules grecs "ocult" i "grafia") pretén ocultar dades dins de fitxers d'imatges, àudios, vídeos, etc sense que aquestes dades es vegin (o s'escoltin) a primera vista. L'avantatge que té l'esteganografia sobre la criptografia és que les dades ocultes no criden l'atenció: quan algú veu dades criptogràfiques, en canvi, reconeix immediatament que aquestes dades estan xifrades. Així doncs, l'objectiu és passar desapercebut

> NOTA: La tècnica del "watermarking" empra mecanismes similars a l'esteganografia però el seu objectiu en aquest cas és diferent: és identificar unívocament l'objecte portador (per exemple, per tenir el control de "copyrights" i autoria).
> NOTA: Òbviament, es poden combinar les dues tècniques: xifrar alguna dada mitjançant criptografia i amagar aquesta dada xifrada mitjançant esteganografia. D'aquesta manera no només caldrà detectar la presència de la dada sinó també caldrà desxifrar-la

## Tècniques esteganogràfiques: LSB

En imatges representades com a mapes de bits (PNG, GIF, BMP...), és habitual que cada píxel sigui representat mitjançant tres bytes: el byte "R", que ens indica la quantitat de vermell, el byte "G", que ens indica la quantitat de verd i el byte "B", que ens indica la quantitat de blau. Atès que la modificació d'aquests bytes en unes poques unitats no és perceptible per al sistema visual humà, això pot ser aprofitat per amagar informació.

> NOTA: També existeixen les imatges que afegeixen un quart byte, el "A", per indicar la quantitat de transparència que tindrà cada píxel en particular (0=totalment transparent, 255=totalment opac)
> NOTA: En les imatges blanc i negre, en canvi, cada píxel se sol representar només amb un sol byte, on el valor 0 representa el negre, el valor 255 el blanc i els valors intermitjos la diferent gradació del gris

És a dir, ja que cada byte està format per 8 bits, una manera senzilla d'ocultar informació sense alterar gaire el valor del píxel consistiria a substituir el valor del bit menys significatiu (LSB o Least Significant Bit...o el que és el mateix, el bit de "més a la dreta"), pel valor d'un bit del missatge que volem amagar. Per exemple: suposem que els primers quatre píxels d'una imatge RGB tenen els valors següents...:

| | **R** | **G** | **B** |
|---|---|---|---|
| **Píxel 1:** | 160 | 60 | 53 |
| **Píxel 2:** | 128 | 111 | 43 |
| **Píxel 3:** | 84 | 125 | 125 |
| **Píxel 4:** | 23 | 204 | 97 |

Si representem els seus valors en binari, ens quedaria...

| | **R** | **G** | **B** |
|---|---|---|---|
| **Píxel 1:** | 10100000 | 00111100 | 00110101 |
| **Píxel 2:** | 10000000 | 01101111 | 00101011 |
| **Píxel 3:** | 01010100 | 01111101 | 01111101 |
| **Píxel 4:** | 00010111 | 11001100 | 01100001 |

Si ara volem amagar, per exemple, la lletra ‘C’ , sabem que aquest caràcter a la taula ASCII ve representat pel número 67, que en binari és **01000011**. Per ocultar aquesta informació, només haurem de substituir el valor del LSB de cada byte de la imatge pel bit corresponent del valor del missatge. És a dir...

| | **R** | **G** | **B** |
|---|---|---|---|
| **Píxel 1:** | 1010000**0** | 0011110**1** | 0011010**0** |
| **Píxel 2:** | 1000000**0** | 0110111**0** | 0010101**0** |
| **Píxel 3:** | 0101010**1** | 0111110**1** | 01111101 |
| **Píxel 4:** | 00010111 | 11001100 | 01100001 |

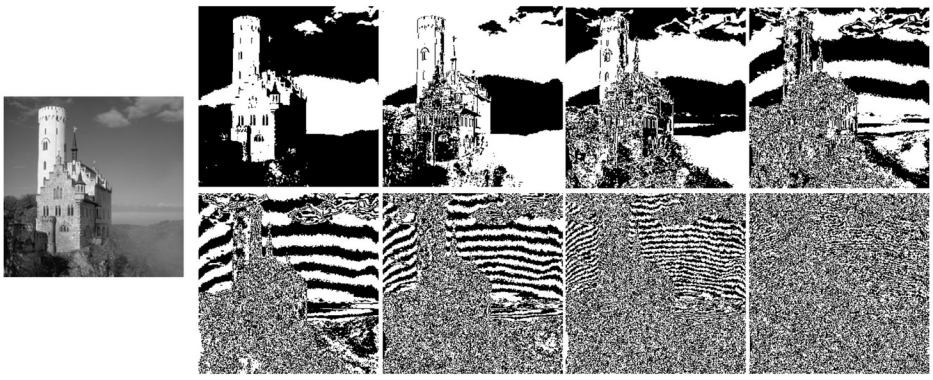
Si tornem a representar aquests nous valors en decimal, per comparar amb els valors de la imatge original, ens quedarà això...

| | R | G | B |
|---|---|---|---|
| **Píxel 1:** | 160 | **61** | **52** |
| **Píxel 2:** | 128 | **110** | **42** |
| **Píxel 3:** | **85** | 125 | 125 |
| **Píxel 4:** | 23 | 204 | 97 |

... i si veiéssim la imatge resultant amb un visor de fotografies qualsevol, no veuríem cap canvi perceptible perquè els valors alterats dels bytes són tan propers a l'original que el color resultant serà pràcticament idèntic visualment parlant.

Cal dir, però, que aquest tipus d'inserció és fàcilment detectable amb els anomenats "atacs estructurals" (com ara el "Sample Pairs Analysis" o el "Weighted Stego", entre altres) ja que altera la distribució estadística dels valors dels bytes de la imatge d'una forma estadísticament determinista, el que fa que es pugui detectar a nivell binari un patró reconegut en les imatges estaganogràfiques d'aquest tipus.

NOTA: En concret, cal fixar-se que els píxels modificats que inicialment tenien un valor imparell (és a dir, acabats en 1) sempre passen a 0 (és a dir, es modifiquen amb -1) i tots els parells (és a dir, acabats en 0) sempre passen a 1 (és a dir, es modifiquen amb +1)
NOTA: El fet de només voler modificar els LSBs d'una imatge i no la resta de bits de cada byte és perquè llavors l'ocultació es farà prou evident. A continuació es mostra l'efecte d'anar alterant el darrer bit (LSB), els dos darrers bits, els tres darrers bits, etc d'una determinada imatge en blanc i negre original (mostrada a l'esquerra)



La tècnica anterior s'anomena "LSB replacement", però existeix una altra tècnica similar anomenada "LSB matching" que no té els problemes mencionats al paràgraf anterior i que, per tant, és immune als atacs estructurals (i més resistent a la detecció en general). En lloc de substituir els LSBs que no coincideixen amb els bits del missatge que volem ocultar, el "LSB matching" consisteix en sumar o restar 1 a aquests valors. Per exemple, si tornem a la mateixa imatge de l'exemple anterior...

| | R | G | B |
|---|---|---|---|
| **Píxel 1:** | 10100000 | 00111100 | 00110101 |
| **Píxel 2:** | 10000000 | 01101111 | 00101011 |
| **Píxel 3:** | 01010100 | 01111101 | 01111101 |
| **Píxel 4:** | 00010111 | 11001100 | 01100001 |

...i volem amagar la lletra ‘C’ (en binari **01000011**), podem fer-ho sumant o restant un 1 aleatòriament (només en els bytes on sigui necessari, és clar). És a dir:

| | R | G | B |
|---|---|---|---|
| **Píxel 1:** | 1010000**0** | 0011110**1** (+1) | 0011010**1**0 (+1) |
| **Píxel 2:** | 1000000**0** | 0110111**0** (-1) | 0010101**0** |
| **Píxel 3:** | 0101010**1** (+1) | 0111110**1** | 01111101 |
| **Píxel 4:** | 00010111 | 11001100 | 01100001 |

Com es pot veure, la diferència fonamental entre el "LSB replacement" i el "LSB matching" és que el segon genera carregament -implicant, llavors, més bits modificats- mentre que el primer no (això no vol dir que la imatge sigui visualment més diferent, ja que continuem modificant el valor global del color del píxel només en una unitat).

NOTA: Existeixen molts altres mètodes, més sofisticats, d'incrustació d'informació esteganogràfica dins d'imatges. Podem mencionar la tècnica F5 (que empra algoritmes de tipus "Wet Paper Codes" per ocultar informació en l'histograma de coeficients DCT de la imatge; només funciona, no obstant, en imatges JPG, que són comprimides) i d'altres que es troben resumides en aquest article: https://arpitbhayani.me/blogs/image-steganography

**EXERCICIS:**

**1.-**Instal.la (en una màquina virtual qualsevol) el paquet "steghide". Després de llegir l'ajuda bàsica de la comanda homònima mostrada a continuació, utilitza-la per:

***steghide embed [-p "contrasenya"] -ef secret.txt -cf coberta.jpg [-sf final.jpg]*** : Incrusta el contingut del fitxer "secret.txt" (de tipus text) dins de l'estructura interna del fitxer binari "coberta.jpg" (el qual pot ser de tipus JPG, BMP i WAV). Si s'indica el paràmetre *-sf*, però, el fitxer "coberta.jpg" no serà modificat sinó que, en canvi, es generarà un altre fitxer ("final.jpg") amb l'assimilació de "secret.txt" en "coberta.jpg" resultant. En fer la incrustació es preguntarà interactivament una contrasenya, a no ser que s'indiqui amb el paràmetre *-p*; aquesta contrasenya es demanarà en el moment de voler extreure el missatge secret.

***steghide extract [-p "contrasenya"] -sf final.jpg [-xf secret.txt]*** : Extreu el contingut incrustat dins de "final.jpg" en un arxiu el nom del qual serà el que tenia l'arxiu que va ser incrustat (a no ser que se n'indiqui un altre diferent amb el paràmetre *-xf*). Es preguntarà la contrasenya indicada en crear el fitxer "final.jpg" per tal de procedir a fer l'extracció.

***steghide [-p "contrasenya"] info final.jpg*** : Mostra informació sobre el fitxer indicat (especialment, diu si hi conté o no alguna dada incrustada). Es preguntarà la contrasenya indicada en crear el fitxer "final.jpg" per accedir a les dades

**a)** Incrustar el missatge "Hola caracola" dins d'una imatge fent servir com a fotografia d'"encobriment" una imatge JPG qualsevol (el format PNG no és compatible!) de l'interior de la carpeta "/usr/share/backgrounds" i indicant una contrasenya de protecció.

**b)** Comprovar, amb el seu paràmetre *info* que, efectivament, el fitxer "jpg" obtingut conté alguna cosa

**c)** Comprovar, amb les comandes *hexdump -C* o *strings* que, efectivament, la incrustació del missatge "secret" ja no és visible (degut a la dispersació dels seus bytes)

**d)** Mostrar per pantalla el missatge secret contingut dins del fitxer "jpg"

**2.-a)** Instal·la el programa **OpenStego** (https://www.openstego.com) des dels repositoris oficials de la teva distribució i fes-lo servir (ja sigui de forma gràfica o bé des del terminal de comandes, consultant https://www.openstego.com/cmdline , com vulguis) per tornar a incrustar de nou el missatge qualsevol dins d'una imatge qualsevol (ara sí que pot ser PNG)

**b)** Compara, amb la comanda *hexdump -C* que el contingut binari de la imatge generada a l'apartat anterior no és igual que la imatge original ni tampoc que la imatge esteganogràfica generada a l'exercici anterior

**c)** Utilitza OpenStego per extreure el missatge "secret" introduït a l'apartat a). ¿Pots fer servir *steghide*?

**d)** ¿Què fa l'opció d'insertar "watermaks" d'OpenStego i per a què serveix això?

NOTA: Altres programes similar a "StegHide" o "OpenStego" són, per exemple, **Stegogo** (https://github.com/Ge0rg3/stegogo), **JSteg** (https://github.com/lukechampine/jsteg) o el més avançat de tots, **HStego** (https://github.com/daniellerch/hstego), la qual implementa les funcions HILL i J-UNIWARD, que són funcions de cost que indiquen les posicions de la imatge on és més òptim ocultar la informació (fins arribar a una determinada quantitat de bits límit, calculada per no ser detectada)

**3.-a)** Usa la web https://kaushalmeena.github.io/digi-cloak per fer la mateixa tasca que als exercicis anteriors: ocultar un missatge en una imatge i posteriorment extreure aquest missatge.

> NOTA: Una altra eina online similar és https://www.beautifyconverter.com/image-steganography.php i https://www.beautifyconverter.com/steganographic-decoder.php

**4.-**En una imatge de 24 bits (per píxel) amb una mida de 800x600 píxels, ¿quants bytes de "missatge secret" podem emmagatzemar-hi com a màxim si fem servir (només) el LSB de cada canal RGB de cada píxel?

**5.-**Prova algun (o tots!) dels següents programes curiosos:

**a) InfiniDrive** (https://github.com/nicomda/InfiniDrive): Genera imatges a partir de bytes (que poden ser els d'un executable, per exemple). El seu raonament de ser i el seu ús està explicat a https://www.flu-project.com/2021/02/almacenamiento-ilimitado-amazon-prime.html

**b) Pngrun** (https://github.com/djhworld/pngrun): Incrusta un executable als LSBs d'una imatge PNG i fa que aquesta sigui autoexecutable. El seu raonament de ser i el seu ús està explicat a https://djharper.dev/post/2020/12/26/executable-pngs

> NOTA: També es pot estudiar l'eina Javascript https://github.com/jklmnn/imagejs , que serveix per incrustar Javascript en una imatge. Això és interessant per implementar atacs de tipus XSS com el que es descriu al següent article: https://null-byte.wonderhowto.com/how-to/hack-forum-accounts-with-password-stealing-pictures-0179953

**c) MP3Stego** (https://github.com/fabienpe/MP3Stego) o **StegoWav** (https://github.com/LiquidFun/stegowav) o **AudioStego** (https://github.com/danielcardeenas/AudioStego): Eines que utilitzen fitxers d'àudio (WAV o MP3) com a mitjà d'encobriment

> NOTA: Una eina útil per detectar aquest tipus d'esteganografia és https://www.sonicvisualiser.org/download.html

**d) Cloakify** (https://github.com/asrabon/Cloakify-3): Genera un text esteganogràfic consistent en llistes de paraules "inòques" però que oculten en text real. En aquest sentit, una altra eina curiosa és **Chess-Steg-Cli** (https://github.com/Alheimsins/chess-steg-cli), la qual oculta el missatge transformant-lo en notació d'una partida d'escacs.

**e) Snow** (https://darkside.com.au/snow) o **Tweg** (https://github.com/fallais/tweg) o **Zwfp** (https://github.com/vedhavyas/zwfp): Eines que oculten visualment un missatge de text dins d'un altre mitjançant l'ús d'espais o línies en blanc i tabuladors. El seu raonament de ser i el seu ús està explicat a https://null-byte.wonderhowto.com/how-to/use-zero-width-characters-hide-secret-messages-text-even-reveal-leaks-0198692 i concretament un tutorial de l'eina Snow es pot trobar aquí: (https://delightlylinux.wordpress.com/2016/12/14/hide-text-in-text-files-using-stegsnow

> NOTA: Una explicació teòrica més detallada sobre aquesta tècnica d'ocultació es pot trobar a https://spectrum.ieee.org/hiding-information-in-plain-text o https://en.wikipedia.org/wiki/Null_cipher

**f) Stegowiper** (https://github.com/mindcrypt/stegowiper ): Elimina indiscriminadament tota la informació esteganogràfica que hi pugui haver en una imatge. La idea és evitar així la propagació de "malware" sota aqueta forma

NOTA: A més d'usar directament eines de terminal ja construïdes, també tenim la possibilitat de construir nosaltres les nostres pròpies eines fent servir llibreries especialitzades. Per exemple, en Javascript tenim https://github.com/zeruniverse/CryptoStego i en Python tenim https://stegano.readthedocs.io o https://github.com/RobinDavid/LSB-Steganography , entre altres (hi ha un tutorial a https://wiki.cedricbonhomme.org/security:steganography )
NOTA: Hi ha moltes més eines, recursos i trucs a les següents llistes:
https://github.com/DominicBreuker/stego-toolkit
https://book.hacktricks.xyz/crypto-and-stego/stego-tricks
https://github.com/mindcrypt/covertchannels-steganography