Simulacre Prova Final UF1 Seguretat

ALGORITMES DE FLUXE ("Xor")

1.-Fes servir l'algoritme Vernam (explicat detalladament en el primer PDF d'exercicis d'aquesta UF) per tal de desxifrar el missatge xifrat "UYF5YFJ" fent servir la Taula de Baudot i la clau "QWERTYU". Has de detallar cada pas com l'has fet manualment.

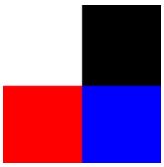
ALGORITMES DE BLOC (amb OpenSSL)

A l'exercici següent es demostrarà visualment perquè el mode ECB de l'algoritme AES no és vàlid per xifrar informació de forma segura. Per fer-ho farem servir una imatge que xifrarem en aquest mode i veurem (mai millor dit) el resultat.

NOTA: El format d'aquesta imatge, però, no pot ser "modern" (tal com PNG, JPG, GIF, etc) perquè aquests formats són molt sofisticats (és a dir, no són una simple seqüència de bytes RGB un darrera altre sinó que tenen estructures complexes dins del fitxer que implementen compressió, etc) i això significa que si xifréssim una imatge amb un format "modern" el resultat no seria visible en el visualitzador d'imatges perquè es mostraria un error en no trobar-se cap d'aquestes estructures de dades (i això faria perdre la "gràcia" de l'exercici). Així doncs, el format triat és el més senzill de tots, el PPM (https://en.wikipedia.org/wiki/Netpbm) : un format sense compressió i pràcticament sense metadades que, aquest sí, conté una seqüència de bytes RGB corresponents a cada píxel (hi ha un altre format similar, el PGM, que només és per imatges en escala de grisos). Concretament, una imatge en format PPM es pot construir amb un editor de text molt bàsicament perquè consisteix en:

*Una primera línia indicant el seu "magic number" que pot ser la cadena "P3" (si tot el contingut del fitxer està codificat en ASCII) o "P6" (si el contingut a partir de la quarta línia està codificat en binari -"raw"-)
*Una segona línia que conté dos números decimals (escrits en ASCII): el primer és l'amplada de la imatge (en número de píxels) i el segon, separat de l'anterior per un espai en blanc, és la seva alçada (en les mateixes unitats)
*Una tercera línia que és sempre la cadena "255", ja que indica el valor màxim que pot tenir cada color (R, G, B) d'un píxel qualsevol (com que cada color ve codificat en un byte, d'aquí ve que el valor màxim sigui "255")
*A partir d'aquí, si el "magic number" de la imatge és "P3", apareixen, o bé separats per espais o per un salt de línia, el valor decimal ASCII de cada color (per aquest ordre: R, G, B) de cada píxel de la imatge (començant pel píxel ubicat a la cantonada superior esquerra de la imatge i anant recorrent la mateixa fila fins passar a la fila de sota i així acabar finalment a la cantonada inferior dreta). En el cas que el "magic number" sigui "P6", el contingut representarà el mateix però estarà codificat en binari

Per exemple, una imatge amb quatre píxels (2x2) tal com la següent...



... vindria codificada així:

P3
2 2
255
255 255 255 0 0 0
255 0 0 0 0 255

Cal tenir en compte, d'altra banda, que entre la primera i segona línia és possible afegir un nombre indeterminat de línies que comencen amb el símbol "#"; aquestes línies representen comentaris arbitraris. De fet, a la imatge utilitzada al següent exercici hi consta una línia d'aquest tipus (concretament, la frase "Created by GIMP version 2.10.34 PNM plug-in")

2.-a) Descarrega't la imatge PPM (de tipus "P6") que t'haurà proporcionat el professor i separa la seva capçalera (és a dir, les quatre primeres línies del fitxer) dels valors RGB. Això ho hem de fer per no xifrar la capçalera (si ho féssim, la imatge xifrada no es visualitzaria perquè no seria reconeguda com una imatge vàlida en no tenir el "magic number" adient). És a dir, fes això:

head -n4 imatge.ppm > capçalerappm.txt
tail -n+5 imatge.ppm > valorsrgbppm.bin

aII) Xifra l'arxiu "valorsrgbppm.txt" fent servir la comanda *openssl* amb l'algoritme AES-256 i mode d'operació ECB i guarda el resultat en un fitxer que anomenaràs "valorsrgbppmxifrats.bin". De contrasenya pots indicar la que vulguis.

aIII) Torna a concatenar la capçalera (no xifrada) de la imatge PPM amb el contingut xifrat corresponent als valors RGB de la imatge per tal d'obtenir un altre cop la imatge completa. És a dir, fes això:

cat capçalerappm.txt valorsrgbppmxifrats.bin > imatgexifrada.ppm

b) Repassa la teoria per saber la resposta a la següent pregunta: ¿per què el mode ECB no és vàlid per xifrar informació de forma segura? Tot seguit, respón llavors aquesta altra pregunta: ¿per què la imatge xifrada resultant de fer l'apartat anterior demostra aquest fet (que el mode ECB no xifra convenientment)?

c) Repeteix l'apartat aII) però ara fent servir ara l'algoritme AES-256 però amb el mode d'operació CTR. Torna a concatenar la capçalera de la imatge PPM amb el resultat obtingut de la comanda anterior i visualitza amb un visor de fotografies el resultat final. ¿Quina diferència hi trobes ara amb la imatge obtinguda a l'apartat anterior i per què?

A l'exercici següent es demostrarà la maleabilitat d'alguns algoritmes de xifrat comuns. Però, ¿què significa *maleabilitat*? Molta gent pensa que xifrar les dades és suficient per mantenir-les segures perquè qualsevol modificació de les dades xifrades provocarà errors a l'hora de desxifrar o inconsistències a les dades (en el seu format o significat/semàntica) que alarmaran els usuaris (ja sigui humans o programes), i per tant es detectaran les manipulacions. Res més lluny de la realitat: el xifratge proporciona confidencialitat però no assegura la integritat o autenticitat de les dades.

Es diu que un algorisme de xifrat és mal·leable quan es poden introduir canvis en el text xifrat que provocaran alteracions dirigides (i perilloses) en el text desxifrat. La majoria d'algoritmes de xifratge convencionals de clau simètrica (tant de flux com de bloc) són mal·leables.

* En el cas dels algorismes de flux, canviant el bit N del text xifrat provoquem un canvi al bit N del text desxifrat. Per tant, l'atacant podria canviar parts del missatge xifrat al seu gust sense necessitat de conèixer la clau; només cal que conegui l'estructura de dades que s'està xifrant (és a dir, no el contingut, sinó el format). Per exemple, si sap que a la posició X d'un missatge hi ha un valor d'interès (un número enter, per exemple), li serà trivial convertir-lo a un número negatiu canviant un únic bit del missatge xifrat.

* En el cas dels algorismes de bloc, el canvi en un únic bit al bloc d'entrada (les cel·les del qual solen ser transformades, a més de per una clau K, per un conjunt de substitucions i permutacions segons el mode d'operació emprat) modifica tots els bits del bloc xifrat corresponent. La majoria de modes d'operació, en fer ús d'aquestes substitucions i permutacions que lliguen uns blocs amb altres, mitiguen fins a cert punt la mal·leabilitat del xifratge, (però no completament, com podem veure a la nota següent, que detalla alguns casos concrets quan es fa servir l'algoritme AES).

NOTA: A continuació es detalla el grau de maleabilitat de diferents modes d'operació:
- El mode ECB és totalment insegur perquè permet substituir fàcilment i de forma independent entre sí uns blocs xifrats per altres (sempre que es faci servir la mateixa clau), així com eliminar blocs del missatge xifrat.
- El mode CBC tampoc és immune: es pot demostrar que si provoquem un canvi al bit B del bloc xifrat N, en voler desxifrar-lo es destrossarà per complet el bloc desxifrat N i es provocarà un canvi al bit B del bloc desxifrat N+1 PERÒ no patirà, atenció, ni el bloc desxifrat N+2 o posteriors cap modificació. És a dir, l'error no es propaga i això el malfactor ho pot aprofitar per fer passar -més- desapercebut el canvi puntual
- En el mode CFB, al bloc N, el bit modificat canvia de valor i el bloc N+1 canvia completament (just al revés que a CBC) però cap bloc més pateix alteració.
- El mode CTR té exactament el mateix problema que els stream ciphers: la modificació del bit N del missatge xifrat provoca un canvi al bit N del missatge desxifrat.

Per tant, cal utilitzar HMACs per garantir la integritat o l'autenticitat de les dades xifrades (o bé un algorisme de tipus AE -com és l'AES amb el mode GCM-, el qual elimina el problema ja que quan s'intenta desxifrar amb aquest mode, l'algorisme genera un error si el text xifrat ha estat manipulat.

> NOTA: Teniu molts més detalls sobre aquest tema a https://sysfatal.github.io/maleable.html i, sobre tot, a l'interessantíssim article https://andrea.corbellini.name/2023/03/09/authenticated-encryption

3.-Descarrega't les dues imatges, anomenades "pagament1.bmp" i "pagament2.bmp", que t'haurà proporcionat el professor: ambdues representen ordres de pagament; la primera informa al departament de comptabilitat de la nostra empresa que ha de fer un pagament de 2000000€ a l'empresa ACME i la segona informa que ha de fer un altre pagament a un tal Enrique Soriano de 120€. Per assegurar que ningú pugui modificar aquestes imatges, als apartats següents se't demanarà que les xifris fent servir diferents modes d'operació de l'algoritme AES, però llegeix primer la "nota important" de sota abans de començar:

> NOTA IMPORTANT: A cadascun dels apartats següents, a la comanda *openssl* que hagis emprat caldrà que li afegeixis el paràmetre *-nosalt* per a què la KDF no faci servir cap sal (i, per tant, generi -de forma determinista- la mateixa clau per xifrar tant "pagament1.bmp" com "pagament2.bmp") o bé indicar directament amb el paràmetre *-K* la clau a utilitzar (en forma de tira hexadecimal de 16 bytes). D'aquesta manera, els arxius xifrats de "pagament1" i "pagament2" resultants tindran internament la mateixa clau; si no fos així, en voler desxifrar el fitxer final (que com veurem, és una "mescla" de "pagament1" i "pagament2"), hi hauria incoherències i l'exercici no funcionaria.

a) Xifra les dues imatges amb la comanda *openssl* fent servir el mode AES-128-ECB i guarda el resultat amb el nom de "pagament1.ecb" i "pagament2.ecb", respectivament.

aII) Xifra les dues imatges amb la comanda *openssl* fent servir el mode AES-128-CBC i guarda el resultat amb el nom de "pagament1.cbc" i "pagament2.cbc", respectivament (llegeix primer la nota següent, però!)

> NOTA: A la comanda *openssl* que executis, hauràs d'afegir (a més dels paràmetres habituals i els explicats a la "nota important" del principi de l'enunciat), el paràmetre extra *-iv* per indicar un determinat valor d'IV (al mateix per les dues imatges), com podria ser per exemple "11111111111111111111111111111111"

aIII) Xifra les dues imatges amb la comanda *openssl* fent servir el mode AES-128-CTR i guarda el resultat amb el nom de "pagament1.ctr" i "pagament2.ctr", respectivament (llegeix primer la nota següent, però!)

> NOTA: A la comanda *openssl* que executis, hauràs d'afegir (a més dels paràmetres habituals i els explicats a la "nota important" del principi de l'enunciat), el paràmetre extra *-iv* per indicar un determinat valor d'IV (al mateix per les dues imatges), com podria ser per exemple "11111111111111111111111111111111"

aIV) Xifra les dues imatges fent servir el mode AES-128-CTR i guarda el resultat amb el nom de "pagament1.ctr2" i "pagament2.ctr2", respectivament (llegeix primer la nota següent, però!)

> NOTA: A la comanda *openssl* que executis, hauràs d'afegir (a més dels paràmetres habituals i els explicats a la "nota important" del principi de l'enunciat), el paràmetre extra *-iv* per indicar un determinat valor d'IV. Al contrari que a l'apartat anterior, ara hauràs d'utilitzar un valor d'IV diferent en xifrar cadascuna de les dues imatges (per exemple, respectivament "11111111111111111111111111111111" i "22222222222222222222222222222222")

Ara imagina que les dues ordres de pagament anterior xifrades cauen en mans del tal Enrique Soriano i que aquest senyor intenta executar les comandes indicades als apartats següents. ¿Què és el que pretén fer en cada apartat? ¿Per què en alguns apartats aconsegueix el seu objectiu sense que no es noti res i ens altres sí que es nota quelcom (llegeix la primera nota de la teoria per tenir una pista)? En qualsevol cas fixa't, però, que Enrique no utilitza mai la contrasenya de xifrat/desxifrat (és a dir, les manipulacions que realitza les pot fer sense necessitat de conéixer aquesta contrasenya)

b) Primer intent. ¿La falsificació surt perfecta o, pel contrari, hi apareixen marques a la imatge desxifrada? (ho podràs esbrinar si següeixes el consell de la primera nota següent):

cp pagament2.ecb fake.ecb
dd if=pagament1.ecb of=fake.ecb count=3000000 bs=1 conv=notrunc

NOTA: Pots comprovar el resultat aconseguit simplement desxifrant amb *openssl* el fitxer "fake.ecb" (on hauràs de fer servir, tal com s'ha comentat a la nota anterior, o bé el paràmetre *-nosalt* o bé el paràmetre *-K* fent servir la mateixa clau de xifratge) per obtenir de nou una imatge visualitzable.
NOTA: El número 3000000 indicat a la comanda *dd* és el valor convenient que s'ha escollit tenint en compte la mida dels fitxers "ecb" per tal que en la sobreescriptura d'un "ecb" amb l'altre s'obtingui els resultats esperats. El paràmetre conv=notrunc és important per mantenir la mida al fitxer "falsificat" la mateixa mida dels fitxers xifrats originals ja que si no s'hi afegeix, en acabar la sobreescriptura el fitxer "falsificat" es truncaria, generant-se, doncs, malament.

bII) Segon intent ¿La falsificació surt perfecta o, pel contrari, hi apareixen marques a la imatge desxifrada? (ho podràs esbrinar si següeixes el consell de la primera nota següent):

cp pagament2.cbc fake.cbc
dd if=pagament1.cbc of=fake.cbc count=3000000 bs=1 conv=notrunc

NOTA: Pots comprovar el resultat aconseguit simplement desxifrant amb *openssl* el fitxer "fake.cbc" (on hauràs de fer servir, tal com ja s'ha comentat, o bé el paràmetre *-nosalt* o bé el paràmetre *-K* fent servir la mateixa clau de xifratge i també el mateix valor d'IV emprat en el xifratge) per obtenir de nou una imatge visualitzable.

bIII) Tercer intent ¿La falsificació surt perfecta o, pel contrari, hi apareixen marques a la imatge desxifrada? (ho podràs esbrinar si següeixes el consell de la primera nota següent):

cp pagament2.ctr fake.ctr
dd if=pagament1.ctr of=fake.ctr count=3000000 bs=1 conv=notrunc

NOTA: Pots comprovar el resultat aconseguit simplement desxifrant amb *openssl* el fitxer "fake.ctr" (on hauràs de fer servir, tal com ja s'ha comentat, o bé el paràmetre *-nosalt* o bé el paràmetre *-K* fent servir la mateixa clau de xifratge i també el mateix valor d'IV emprat en el xifratge) per obtenir de nou una imatge visualitzable.

bIV) Quart intent ¿La falsificació surt perfecta o, pel contrari, hi apareixen marques a la imatge desxifrada? (ho podràs esbrinar si següeixes el consell de la primera nota següent):

cp pagament2.ctr2 fake.ctr2
dd if=pagament1.ctr2 of=fake.ctr2 count=3000000 bs=1 conv=notrunc

NOTA: Pots comprovar el resultat aconseguit simplement desxifrant amb *openssl* el fitxer "fake.ctr2" (on hauràs de fer servir, tal com ja s'ha comentat, o bé el paràmetre *-nosalt* o bé el paràmetre *-K* fent servir la mateixa clau de xifratge i també el mateix valor d'IV emprat en el xifratge de "pagament1.ctr2" (perquè l'altra generarà una imatge invàlida, en carregar-se el seu "magic number") per obtenir de nou una imatge visualitzable.

OPENSSL

4.-a) Escriu un missatge en un arxiu de text i xifra'l simètricament amb *openssl* fent servir l'algoritme AES-256-CBC i una determinada "passphrase"

aII) Genera amb *openssl* (i fent servir la mateixa "passphrase") un altre fitxer que contingui el hash MD5-HMAC del missatge xifrat anterior

aIII) Escriu el valor de la "passphrase" utilitzada als apartats anteriors en un altre arxiu de text (sense indicar cap salt de línia al final!) i xifra aquest altre arxiu de text amb aquesta clau pública (pertanyent al professor).

-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAse5o8sQ1zrC3MBfTBsQ0
Awpwll31PW4jJ0Jwm5XIT0xbWp8vXFw5YwCLVGoXHyftAfY/4ynfdjagmN7aXOFO
Ep/GCr+NnvCCR3hF5UjQsCxgruOTaY6v7EMmJ7+pU1koW79l4f9jEdF9gRs9bWBm
8cd+7u7xUapUNWInimHUXryTeuWo4GhFJb0Cm/6xO/nkDtsM+FE44DONhsbojCMb
JkJJSI7sUIdLdvXdJcrANFXjzG96Val1B3btpsdsBOezK4XoWSm8bzP+RgIPlkez
Dk/a8H9hF3S8Fd6Und/E+1oGv7LnrIvPIQhtED2tHa8shlrRMJW7z+ud+/50bVJg
xQIDAQAB
-----END PUBLIC KEY-----

aIV) Envia al correu del professor el fitxer contenint el missatge secret (xifrat simètricament), el fitxer contenint el HMAC d'aquest missatge i el fitxer contenint la "passphrase" (xifrada asimètricament) emprada per generar els dos fitxers anteriors. Ell haurà de ser capaç llavors de desxifrar la "passphrase" amb la seva clau privada i, un cop coneguda, fer servir aquesta "passphrase" per comprovar primer la integritat del teu missatge (mitjançant la comparació del valor existent en el fitxer HMAC rebut amb el càlcul "in-situ" del valor HMAC del missatge xifrat rebut) i, si tot és correcte, desxifrar llavors aquest missatge xifrat rebut. ¿Quines són les comandes, doncs, que haurà d'executar?

5.-a) Explica què fan les següents comandes

*tar -czf - * | openssl enc -e -aes-128-cbc -out unfitxer.tar.gz*
openssl enc -d -aes-128-cbc -in unfitxer.tar.gz | tar -xzf -

> Pista: la combinació "f -" de la comanda *tar* , en el cas d'empaquetar, indica que el resultat de l'empaquetament no serà un fitxer sinó que s'enviarà a la sortida estàndard (canonada) i, en el cas de desempaquetar, indica que les dades a desempaquetar no provenen d'un fitxer sinó de l'entrada estàndard (canonada)

b) Explica per a què serviria aquest script (no cal que l'executis, només observant el seu codi ja hauries de saber què fa; haver fet l'exercici anterior també pot servir d'ajuda)

```
#/bin/bash
if [ $# -ne 1 ]; then
   echo "Has d'indicar la ruta d'una carpeta com a paràmetre"
   exit
fi
openssl pkeyutl -decrypt -inkey ./clau.pem -in "$1/key.enc" -out "$1/key.txt"
for i in $(find $1 -type f -name "*.encrypted")
do
   echo "Decrypting $1/$i"
   openssl enc -d -aes-256-cbc -pass file:"$1/key.txt" -in "$i" -out $1/$(basename "$i" .encrypted)
done
```

> NOTA: La comanda *basename* serveix per treure l'extensió (indicada com a segon paràmetre) dels noms dels fitxers indicats com a primer paràmetre

XIFRATGE DE CARPETES I FITXERS

6.- Escriu un missatge en un arxiu de text i xifra'l mitjançant *age* amb la següent clau pública (pertanyent al professor): *age1fej29ayd9jqurucj88v35n7fqksvswc49kv5yuk86wkknvzxyurq7rvn2f* Envia al correu del professor el fitxer xifrat; ell haurà de ser capaç llavors de desxifrar-lo amb la seva clau privada

La comanda *tle*, pertanyent al paquet "tlock" (https://github.com/drand/tlock) permet xifrar fitxers que només podran ser desxifrats (sense necessitat d'usar cap clau) passat un determinat temps.

> NOTA: El funcionament d'aquesta comanda es basa en la xarxa Drand (https://drand.love), un servei distribuït d'Internet que genera periòdicament nombres realment aleatoris de forma verificable. Teniu més informació sobre aquesta xarxa a https://drand.love/blog/2023/06/02/drand-explainer i també a https://drand.love/docs/timelock-encryption

7.-a) Executa les següents comandes per instal·lar el paquet "tlock" al teu sistema:

sudo apt install golang (a Ubuntu) o *sudo dnf install golang* (a Fedora)
git clone https://github.com/drand/tlock
cd tlock && go build cmd/tle/tle.go

A partir d'aquí, dins de la carpeta "tlock" podràs executar les següents comandes:

* Per obtenir un fitxer xifrat "fitxer.enc" (en format Base64 gràcies al paràmetre *-a*) a partir del fitxer original "fitxer.txt" que només podrà ser desxifrat després de passar 30 segons, caldria fer el següent: *tle -e -a -D 30s -o fitxer.enc fitxer.txt*

NOTA: En el moment de xifrar, per defecte s'utilitza el node de la xarxa Drand https://api.drand.sh . El node triat es pot canviar amb el paràmetre *-n* de la comanda *tle*, però si això es canvia, caldrà indicar llavors també el node triat explícitament a la comanda de desxifrat. Els nodes actualment disponibles són: https://api.drand.sh (US), https://api2.drand.sh (EU), https://api3.drand.sh/ (Asia) i https://drand.cloudflare.com (distribuït entre regions)
NOTA: La duració (és a dir, el valor indicat al paràmetre -D) es pot especificar amb un número i els següents sufixes: "ns", "us" , "ms", "s", "m", "h", "d", "M" i "y"

* Per desxifrar "fitxer.enc" (i obtenir un nou fitxer desxifrat "fitxernou.txt"...si no s'indiqués el paràmetre *-o* la sortida llavors seria la pantalla), sempre i quan ja hagi passat el temps estipulat en el moment de xifrar-lo, caldria fer el següent: *tle -d -o fitxernou.txt fitxer.enc*

aII) Escriu un missatge en un arxiu de text i xifra'l mitjançant *tle* de forma que no es pugui desxifrar fins d'aquí tres dies. Envia al correu del professor aquest fitxer xifrat

b) Explica què fan les següents comandes:

cat fitxer.txt | age -p | tle -e -a -D 30s -o fitxer.enc
cat fitxer.enc | tle -d | age -d -o fitxernou.txt

LUKS

8.-a) Prepara un fitxer de 100MB com a disc "loop" formatejat en Ext4 i fes que sigui de tipus LUKS. Munta'l, guarda-hi allà un fitxer de text amb un determinat missatge al seu interior i desmunta'l.

aII) Afegeix un segona clau al dispositiu LUKS anterior, la qual ha de consistir en un fitxer contenint una tira binària aleatòria de 10 bytes

aIII) Envia per correu al professor tant aquest fitxer-clau binari com el propi disc LUKS. Ell haurà de ser capaç de poder accedir, gràcies al fitxer-clau, a l'interior del dispositiu LUKS i llavors observar el missatge secret guardat al seu interior.

METADADES

9.-a) Tria una fotografia qualsevol i utilitza l'eina *exiftool* per afegir-li les metadades necessàries per geolocalitzar-la (falsament) Comprova que ho hagis aconseguit anant a https://tool.geoimgr.com

Pista: Les metadades concretes que hauràs d'afegir són quatre: *GPSLatitude*, *GPSLongitude*, *GPSLatitudeRef* i *GPSLongitudeRef*; es deixa com investigació esbrinar quins podrien ser els seus possibles valors

b) Tria un document PDF qualsevol i utilitza l'eina *exiftool* per modificar (o afegir, si no hi és) les metadades "Author" (persona), "Creator" (software) i "Title" amb el valor que vulguis. Comprova (amb *exiftool* de nou) que ho has fet bé

c) Tria un document ODT qualsevol i utilitza l'eina *exiftool* (o Mat2 o derivades) per eliminar totes les seves metadades possibles. ¿Quines són les metadades que no has pogut eliminar?

ESTEGANOGRAFIA

10.-a) Troba el missatge incrustat dins de la fotografia proporcionada pel professor emprant algun dels mètodes vistos a classe (comandes *strings/hexdump*, comandes *binwalk/dd*, comanda *exiftoo*l, edició digital)

b) Vés a https://stegonline.georgeom.net i puja-hi una imatge. Tot seguit, tria l'opció "Embed files/data" i, al formulari que t'apareix, indica que vols incrustar un determinat missatge secret en els darrers dos bits R, els darrers dos bit G i els darrers dos bits R de la imatge (¡però no modifiquis cap dels valors que per defecte ja apareixen triats als diferents desplegables presents al formulari!). Guarda la imatge resultant i envia-la per correu al professor. Ell haurà de ser capaç, fent servir l'opció "Extract files/data" de la mateixa web, d'obtenir el missatge ocult.

> NOTA: Com alternativa a la web anterior , es pot utilitzar la comanda Lsbsteg (https://github.com/adrg/lsbsteg), una petita aplicació Python que serveix també per incrustar missatges de text en imatges fent servir la tècnica LSB. D'altra banda, si es vol implementar "a mà" el mètode LSB i altres mitjançant codi propi Python, recomano la lectura de l'article https://daniellerch.me/stego/lab/intro/lsb-es

trobem força llibreries que ens permeten implementar diferents mètodes esteganogràfics en els nostres propis programes, com per e exemple

c) Utilitza la comanda *steghide* per ocultar un missatge secret dins d'un arxiu de so WAV qualsevol que tinguis al teu sistema. Fes servir com a contrasenya "1234" i envia el resultat al correu del professor. Ell haurà de poder obtenir aquest missatge secret sense problema

cII) Utilitza el programa Stegseek (https://github.com/RickdeJager/stegseek) per "crackejar" la contrasenya de l'arxiu esteganogràfic que has creat a l'apartat anterior. Es deixa com investigació com instal·lar i fer servir aquesta eina.

CARVING

11.-a) Descarrega't el fitxer "carving.raw" que t'haurà proporcionat el professor. Aquest fitxer binari conté dins el seu interior una fotografia. No obstant, Photorec no és capaç de trobar-la (ho pots comprovar) i la comanda *binwalk* és capaç de localitzar-la però no la pot extreure tampoc. Utilitza la comanda *dd* (amb la informació que et proporcioni *binwalk*) per extreure la fotografia

b) Descarrega't el fitxer "fat.dd" que t'haurà proporcionat el professor. Aquest fitxer representa la imatge d'una partició d'un determinat ordinador. Troba-hi, fent servir *photorec*, un full de càlcul XLS i un vídeo MOV

c) Descarrega't el fitxer "memdump.tar.xz" que t'haurà proporcionat el professor. Aquest fitxer representa una part d'una captura de la memòria RAM d'un ordinador en un determinat instant. Després de descomprimir-la, troba-hi, fent servir *bulk_extractor*, el número d'una tarja bancària guardat allà, sabent que comença per 3728.