EXERCICIS CRIPTOGRAFIA (operació XOR):

L'operació XOR aplicat bit a bit entre una determinada informació ("el text pla") i una determinada clau secreta genera com a resultat una sèrie de bits que es poden considerar informació xifrada (pels motius que veurem a continuació). Recordem, abans, els detalls d'aquesta operació booleana: suposant que el missatge original és, en binari, aquest: $x_1x_2...x_n$, cada bit del missatge xifrat es calcularà així: $\mathbf{y_i = x_i\ XOR\ k_i}$, on k_i a la pràctica se sol generar en els sistemes electrònics per l'anomenat 'key stream generator' a partir d'un valor inicial aleatori (anomenat "seed") més una determinada funció matemàtica (que serveix per anar generant, a partir d'aquest valor inicial, tota la seqüència de bits següents). Es pot veure a la taula taula, que defineix l'operació XOR, que, en definitiva, el resultat obtingut és 0 sempre que els operands valguin igual i 1 si valen diferent:

$\mathbf{y_i}$	$\mathbf{x_i}$		$\mathbf{k_i}$
0	0		0
1	0	**XOR**	1
1	1		0
0	1		1

La gràcia d'usar l'operació XOR és que no només la clau és la mateixa per xifrar i desxifrar (és a dir, que és un algoritme simètric, cosa que no passa amb altres operacions bit-a-bit com AND o OR) sinó que el propi algoritme és el mateix en ambdues operacions: $\mathbf{x_i = y_i\ XOR\ k_i}$ A més, XOR no porta ròssec (i per tant, és computacionalment senzill i ràpid). D'altra banda, és difícil de trencar per força bruta si la clau és prou llarga (tot i que és fràgil davant l'anàlisi de freqüències, a no ser que la clau, aleatòria i no reutilitzable, fos d'igual llargària que el missatge -l'anomenat "one time pad"-: en aquest cas, l'operació XOR sí que seria indesxifrable).

Una altra raó important, a més de la seva simetricitat, de per què l'operació XOR és vàlida per la criptografia (mentre que altres com l'AND o el OR no), és perquè està "perfectament balancejat"; és a dir, donat un determinat bit 0 o 1 en el text en clar, la probabilitat de què en el text xifrat s'obtingui un 0 o 1 és del 50% (és aleatòria, doncs), com es pot comprovar a la taula anterior. En canvi, l'operació AND produeix tres 0 i només un 1 (així que un bit del text xifrat amb valor 1 informa al criptoanalista que aquest mateix bit al text en clar serà també 1) mentre que l'operació OR produeix tres 1 i només un 0 (així que un bit del text xifrat amb valor 0 denota que el bit corresponent al text en clar també és 0); en ambdós casos, aquesta informació "extra" podria ser de gran ajuda al criptoanalista perquè és fàcil veure que, degut al que s'acaba de descriure, aproximadament un 25% del missatge en clar el podrà obtenir "sense fer res".

> NOTA: En aquest sentit, cal notar que el resultat d'una operació AND sobre un número binari no podrà ser mai més gran que aquest (numèricament parlant); això ès fàcil veure-ho observant què passa amb els bits més significatius del número binari si se li aplica l'operació AND amb 1s i 0s. Igualment, cal notar que el resultat d'una operació OR sobre un número binari no podrà ser mai més petit que aquest.. En canvi, el resultat d'una operació XOR sobre un número binari pot ser qualsevol seqüència possible sense cap patró reconegut (això és degut a la difusió generada en tenir un 50% de probabilitat a l'hora d'obtenir un 0 o un 1, tal com s'ha explicat al paràgraf anterior), així que aquest resultat no ens aportarà cap informació que pugui ajudar al criptoanalista.

No obstant, l'operació XOR té un "problema" que cal tenir en compte, i és que també es compleix que $\mathbf{k_i = x_i\ XOR\ y_i}$ És a dir: tenint accés a un text cifrat i al seu text en clar corresponent, es pot deduir la clau fàcilment. I això és un perill si aquesta clau es reutiliza en altres ocasions.

1.-L'algoritme Vernam consisteix en aplicar l'operació XOR a cada bit que forma la representació binària numèrica de cada lletra d'un missatge en clar (és a dir, als valors A=0=00000, B=1=00001, C=2=00010, ...fins Z=26=11010) contra una clau formada per lletres del mateix alfabet que el missatge en clar i de longitud arbitrària (tot i que quan més llarga millor per evitar patrons, ja que si no arriba a la longitud del missatge en clar caldrà anar repetint-la); òbviament les lletres de la clau també caldrà que estiguin en representació binària per poder realitzar el càlcul.

> NOTA: Cal tenir en compte que si la representació numèrica del resultat d'haver transformat una lletra concreta superés el límit de 26 lletres de l'alfabet i es vol escriure igualment el missatge xifrat en format alfabètic, caldrà aplicar-li llavors l'operació mòdul 26 (*mod 26*) per tal de fixar el resultat per sota d'aquest límit i així poder-li assignar una lletra de l'alfabet. A https://es.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/what-is-modular-arithmetic i també a l'apartat 1 de http://www.criptored.upm.es/crypt4you/temas/criptografiaclasica/leccion5.html podreu trobar explicada l'operació "mòdul"

Per exemple, suposem que el missatge en clar és O A K i la clau és S O N. Per tant, el missatge xifrat serà:

O =14 =01110
S =18 =10010
O XOR S =11100 => 28 (*mod 26*) = 2 = **C**

A =0 =00000
O =14 =01110
A XOR O =01110 => 14 = **O**

K =10 =01010
N =13 =01101
K XOR N =00111 => 7 = **H**

Si volguéssim fer el desxifrat de C(+26) O H amb la mateixa clau (S O N), obtindrem el missatge en clar de nou si apliquem la mateixa operació XOR de nou:

C(+26) =28 =11100
S =18 =10010
C XOR S =01110 => 14 = **O**

O =14 =01110
O =14 =01110
O XOR O =00000 => 0 = **A**

H =7 =00111
N =13 =01101
H XOR N =01010 => 10 = **K**

a) Troba la tira binària corresponent a una paraula en clar qualsevol (que representarà el missatge que volem xifrar) però que tingui com a mínim sis lletres. Per fer això, però, en lloc de fer l'equivalència *lletra<->tira binària* utilitzada a l'exemple dels paràgrafs anterior fes servir la "Taula de Baudot" (https://ca.wikipedia.org/wiki/Codi_Baudot) :

Grup de bits	Grup de lletres
00011	A
11001	B
01110	C
01001	D
00001	E
01101	F
11010	G
10100	H
00110	I
01011	J
01111	K
10010	L
11100	M
01100	N
11000	O
10110	P
10111	Q
01010	R
00101	S
10000	T
00111	U
11110	V
10011	W
11101	X
10101	Y
10001	Z
01000	1
00010	2
11111	3
11011	4
00100	5
00000	6

b) Troba la tira binària corresponent a la cadena BABA (fent servir la mateixa "Taula de Baudot"), la qual representa la clau que farem servir per xifrar la paraula triada a l'apartat anterior

c) Obre la calculadora del Gnome, posa-la en el mode "Programador" i fes que la representació dels números sigui en format binari. Tot seguit aplica l'operació XOR entre cada seqüència de 5 bits que representa cada "lletra" de la tira binària corresponent a la paraula en clar triada al primer apartat amb cada seqüència de 5 bits que forma cada "lletra" de la tira corresponent a la clau BABA. És important que aquesta operació es faci "lletra a lletra" per a mantenir constància dels possibles 0 a l'esquerra que puguin anar apareixent, ja que si l'operació es fa d'un sol cop llavors podrien "desaparèixer" (amb l'error que ocasionaria en tot el procés).

> NOTA: Més enllà de la calculadora de Gnome, existeixen algunes webs que poden aplicar també operacions "bitwise" (és a dir, operacions AND, OR, XOR, NOT, etc sobre bits individuals). L'exemple més immediat és la web https://cryptii.com (el codi font de la qual es troba a https://github.com/cryptii/cryptii): si establim el primer i tercer quadre a "Bytes", omplim de contingut original (en format hexadecimal o binari) el primer, i establim el segon quadre a "Bitwise operation" per sel·leccionar llavors l'operació desitjada i també -a la caixa que apareix anomenada "Operand B (repeating)"- el valor hexadecimal que s'aplicarà (de forma repetida, si cal) sobre el contingut original, obtindrem el resultat al tercer quadre. Un altre exemple (tot i que només per l'operació XOR) seria el formulari web http://xor.pw D'altra banda, desgraciadament la calculadora de terminal *bc* , que també podria ser una bona alternativa a la calculador de Gnome, no és capaç de fer operacions "bitwise" (tal com es discuteix aquí: https://stackoverflow.com/questions/2954003/bitwise-operations-in-bc)

d) Torna a utilitzar la "Taula de Baudot" per "decodificar" el resultat obtingut en l'apartat anterior per tal de poder-lo expressar en forma de lletres alfabètiques

e) Passa a algun company de classe (com vulguis...via correu, escrit en un paper, etc) el missatge xifrat obtingut a l'apartat anterior. Aquest company haurà de fer el procés següent per trobar el missatge en clar original:

> **1.-**Transformar aquest missatge xifrat alfabètic en una tira binària (fent ús de la taula Baudot)
> **2.-**Transformar la clau (que és coneguda i la mateixa, BABA) també en una tira binària
> **3.-**Aplicar l'algoritme XOR entre les tires binàries anteriors (fent ús de la calculadora de Gnome)
> **4.-**Decodificar el resultat obtingut (fent ús de la taula Baudot) per expressar-lo en format alfabètic

2.-a) Descarrega't la imatge "secret.png" disponible a la web del centre. D'aquesta imatge (que si intentes obrir veuràs que no pots perquè no és reconeguda com una imatge vàlida ja que està xifrada) un "ocellet" ens ha dit vàries coses:

> -Que és de tipus PNG
> -Que està xifrada mitjançant XOR

Gràcies a la primera informació podem saber quina hauria de ser la seqüència de bytes per la què hauria de començar la imatge desxifrada, que no és una altra que el "magic number" assignat oficialment al format PNG (http://www.libpng.org/pub/png/spec/iso) . Aquest "magic number" és, concretament **89 50 4e 47 0d 0a 1a 0a**

> NOTA: Els "magic numbers" són seqüències de bits estandaritzades que estan integrades dins de tot fitxer (concretament, al seu inici) per identificar el tipus de fitxer del qual es tracta. Aquesta informació és molt més fiable que les extensions dels fitxers, completament manipulables per l'usuari. Hi ha una llista dels "magic numbers" més habitual a https://en.wikipedia.org/wiki/List_of_file_signatures o també https://www.garykessler.net/library/file_sigs.html

Comproba (fent simplement *hexdump -C secret.png*), però, que el començament del fitxer és **76 af b1 b8 f2 f5 e5 f5**

> NOTA: La comanda *hexdump* mostra el contingut binari "tal qual", sense interpretar, del fitxer que se li indiqui. Per defecte els bytes es mostren horitzontalment, en hexadecimal i agrupats per parelles; a l'esquerra de tot apareix també un nombre hexadecimal que indica la posició del primer byte (el de més a l'esquerra) de cadascuna de les línies de bytes mostrades (a mode de comptador de línies). El paràmetre *-C* serveix per mostrar, a més, una nova columna a la dreta de tot mostrant l'equivalència en ASCII de cada byte del fitxer, per ordre (i si no, un punt), per si aquesta fos de rellevància. Una altra comanda similar a *hexdump* que també es troba per defecte en els sistemes Linux és ***od***

b) Com que sabem que es compleix que $k_i = x_i$ *XOR* y_i , dels valors anteriors podem deduir que:

k_1 = 89 XOR 76
k_2 = 50 XOR af
k_3 = 4e XOR b1
k_4 = 47 XOR b8
k_5 = 0d XOR f2
...

Si tenim la sort de què la clau sigui més curta que la longitud del "magic number", podrem esbrinar-la per complet i llavors podrem aplicar l'operació $x_i = y_i$ *XOR* k_i per tal de desxifrar tota la imatge. El primer que has de fer, per tant, és trobar la clau mitjançant les operacions anteriors. Per fer-ho, torna a utilitzar la calculadora del Gnome (en "mode programació" i, en aquest cas, fent que la representació dels números sigui en format hexadecimal) i ves fent l'operació XOR per cada byte, un rera l'altre, per anar "descobrint" la clau byte a byte.

> NOTA: Com que en aquest cas les operacions XOR només afecten a un byte cada cop, també podries fer-les sense utilitzar la calculadora de Gnome sinó simplement executant la comanda següent per cada parella de bytes x_i i y_i fins trobar la clau utilitzada: *echo $((0x76 ^ 0x89))* És fàcil veure com a la comanda anterior s'indica que estem fent una operació XOR -amb l'operand "^"- de dos números escrits en hexadecimal; no obstant, el resultat obtingut vindria donat en decimal; si volguéssim obtenir-lo en hexadecimal, hauríem de fer servir la comanda *printf*, la qual ens permet especificar el format desitjat (mitjançant el modificador *%format*, en aquest cas seria %X, així): *printf "%X\n" $((0x76 ^ 0x89))*

c) A partir d'aquí, hauriess d'aplicar l'operació $x_i = y_i$ *XOR* k_i byte a byte al conjunt de tots els bytes del fitxer xifrat per obtenir la imatge desxifrada. No obstant, en aquest cas la calculadora de Gnome (o la web Cryptii o similars) no la podrem fer servir perquè el problema d'aquestes sol.lucions és que acabem tenint com a resultat una tira de números hexadecimals que representa el contingut de la foto desxifrada, sí, però no el podem guardar en forma de fitxer perquè el copiar-pegar es fa amb la seva representació ASCII i no pas amb el contingut binari subjacent. Per tant, el que haurem de fer és emprar una comanda específicament dissenyada per aplicar l'operació XOR directament sobre el contingut d'un fitxer (obtenint-ne un altre com a resultat), la comanda *xor*. No obstant, aquesta comanda no està disponible als repositoris oficials de les distribucions importants sinó que hauràs de descarregar el seu codi font i compilar-lo per obtenir l'executable pertinent. En concret, has de descarregar-te el fitxer "xor.c" (disponible a https://github.com/dirtbags/fluffy) des de https://raw.githubusercontent.com/dirtbags/fluffy/master/xor.c i executar la comanda següent per realitzar la compilació: *gcc -o xor.exe xor.c*

> NOTA: Una comanda similar, però dissenyada per aplicar l'operació XOR a fluxes d'entrada de bits (rebuts per canonada) en lloc de fitxers és https://github.com/mstrand/xcat

d) Utiliza la comanda *xor.exe* obtinguda al pas anterior (consulta el quadre següent, extret de la seva documentació oficial, per veure uns quants exemples d'ús) per obtenir un nou arxiu anomenat *foto.png* que es correspongui a la foto desxifrada i visualitza-la finalment amb un visor de fotos qualsevol. Comprova, a més (fent ara *hexdump -C foto.png*), que, efectivament i com no podia ser d'una altra manera, el "magic number" d'aquest fitxer es correspon al d'un fitxer de tipus PNG (*89 50 4e 47 0d 0a 1a 0a*)

*Consider the following example command, where "22" is the key (the value to manipulate the bits with) and the text is either the plaintext or ciphertext that will be manipulated: *echo "hello" | ./xor 22* This should display something like ~szzy Now try the reverse: *echo "~szzy" | ./xor 22*
*To use the *xor* command over all the content of a file, you should do something like this: *./xor 0x22 < filename.txt*
*Take note on the *xor* program that there is a big difference between "./xor 22 < file.txt" and "./xor 0x22 < file.txt". The 2nd one is a hex number while the 1st is decimal. You can indicate your value is hex by using the -x argument like this: *./xor -x 22 < file.txt*
*You can also have a multi value key which would look like this: *./xor -x 01 a0 22 f5 < file.txt*

e) Tant la web https://wiremask.eu/tools/xor-cracker com la comanda https://github.com/hellman/xortool utilitzen les mateixes tècniques per aconseguir els seus objectius. ¿Quins són?. Prova d'utilitzar la web per aconseguir la mateixa foto desxifrada aconseguida a l'apartat anterior.

Una alternativa a xifrar completament tot un fitxer (com per exemple una imatge) des del seu primer byte a l'últim (tal com vam veure a l'exercici anterior) és xifrar només el que seria el seu contingut efectiu: és a dir, deixar sense xifrar el que seria la capçalera on apareix el "magic number" del fitxer (juntament amb altres possibles metadades addicionals que hi puguin haver) per xifrar només els píxels pròpiament dits (en el cas de les imatges). D'aquesta manera, se seguiria tenint una imatge formalment vàlida (visible per qualsevol visor de fotografies) però mostrant un conjunt de píxels aleatoris sense cap sentit, semblans a simple soroll. És el que veurem en el proper exercici.

3.-a) Descarrega't les imatges "lemur.png" i "key.png" disponibles a la web del centre i visualitza-les: veuràs que ambdues semblen soroll, ja que la primera imatge ha sigut xifrada mitjançant l'operació XOR (però respectant l'estructura interna del format PNG per mantenir-la vàlida) amb una clau que resulta ser la segona imatge, la qual està formada, aquesta sí, per una tira de píxels aleatoris de la mateixa mida que la primera.

> NOTA: Aquestes imatges tenen una "profunditat de color" de 24 bits. Això significa que el color de cada píxel d'aquestes imatges ve definit per 3 bytes, el valor del qual indica respectivament la quantitat del color primari vermell (Red) que tindrà el color final, la quantitat de color primari verd (Green) i la quantitat de color primari blau (Blue). Existeixen altres profunditats de colors (per exemple, la d'1 bit només permet dos colors -blanc i negre-, la de 2 bits en permet quatre -blanc, negre, gris clar i gris fosc-, etc). No obstant, cal tenir en compte, però, que aquesta estructura és més aviat lògica: no està directament implementada així dins del contingut d'un fitxer d'imatge ja que, segons el format que tingui (PNG, JPG, etc), aquest contingut es trobarà comprimit de determinada manera i no seguirà aquest esquema de 24 bits per pixel. Aquesta és una raó més, a banda de la que s'explica al següent paràgraf, de per què no es pot fer un XOR directament als bytes del fitxer i ja està per desxifrar-lo

¿Com podem executar l'operació XOR entre "lemur.png" i "key.png"? Més enllà del fet que ara la clau és tot un fitxer i no una simple tira d'un pocs valors hexadecimals (i per tant, la comanda xor emprada a l'exercici anterior no serà adient) hem de tenir en compte que, tal com s'ha comentat, el procediment que utilitzem ha de ser prou "intel·ligent" per distingir les parts de les imatges que no estan xifrades (la capçalera amb els "magic numbers", per exemple) de les que sí; hem de pensar que, per exemple, l'operació XOR entre dos valors iguals dóna 0, així que si l'apliquéssim entre dos fitxers que fossin imatges des del seu començament, afectaríem a les seves dues capçaleres amb el resultat d'obtenir una capçalera plena de zeros (és a dir, un fitxer amb un format irreconeixible). Afortunadament, manipular imatges en general i, en concret, manipular-les juntament amb altres imatges (no només aplicant entre elles algoritmes criptogràfics com l'operació XOR sinó d'altres com les operacions AND, OR i moltíssimes més) és senzill si fem servir eines especialitzades en el processament digital de fotografies, com ara GMIC (https://gmic.eu), GraphicsMagick (http://www.graphicsmagick.org) o ImageMagick (https://imagemagick.org), entre altres.

b) Instal·la el paquet "gmic" en el teu ordinador i a continuació executa la comanda *gmic lemur.png key.png -blend xor -o fotofinal.png*. ¿Què n'obtens? ¿Què passa si, en canvi, com a valor del paràmetre "-blend" escrius *and*? ¿I si, tot i fer servir l'operació *xor*, utilitzes una altra imatge com a clau?

> NOTA: Tens la referència del paràmetre *-blend* aquí: https://gmic.eu/reference/blend.html

c) Dedueix, veient el resultat, què fa la comanda següent i en quin sentit es diferencia de la comanda utilitzada a l'apartat anterior: *gmic fotofinal.png xor 123* ¿Quin resultat obtens de fer, en canvi, *gmic fotofinal.png and 123* ? ¿I *gmic fotofinal.png or 123* ?

> NOTA: Tens la referència de l'operador *xor* aquí: https://gmic.eu/reference/xor.html
> NOTA: GraphicsMagick té el paràmetre *-operator* i ImageMagick els paràmetres *-evaluate* i *-fx* que son similars en funcionalitat als exemples vistos en aquest exercici amb Gmic. D'altra banda, si es vol tenir un control més programàtic de l'edició d'imatges, es poden fer servir llibreries especialitzades per fer totes aquestes tasques (i més), com ara OpenCV (https://opencv.org) o Pillow (https://python-pillow.org), entre moltes altres