Enginyeria Tècnica en Informàtica de Sistemes

Treball de Fi de Carrera

Inserció de marques inaudibles en ones d'àudio

Alumne: Raquel Buil Mur
Consultor: Antoni Martínez Ballesté
Universitat Oberta de Catalunya
Gener, 2005

Índex

1 Objectius

Es pretén implementar un sistema de detecció de còpia per a protegir el copyright de fitxers àudio digitals en format WAV. El sistema haurà de contenir dos algorismes bàsics: un per a inserir una marca d'aigua (watermark) en un fitxer i un per a recuperar la marca inserida en un fitxer mitjançant el primer algorisme. Ens basarem en una tècnica senzilla que és la substitució del bit menys significatiu (LSB).

Aquest sistema s'avaluarà pel que fa a robustesa enfront de manipulacions sobre el fitxer marcat. S'estudiaran els seus punts febles i es proposaran modificacions per a aconseguir millorar-lo. Intentarem utilitzar en primer lloc els codis correctors d'errors i més endavant la compressió MP3 com a eines per assolir aquesta millora.

Els objectius d'aquest treball es podrien resumir en els següents punts:

- Copsar la necessitat de la protecció del copyright dels productes digitals.
- Conèixer el format WAV per a material àudio.
- Entendre el concepte de marca d'aigua (watermarking) i les propietats que ha de tenir.
- Saber implementar el mètode de watermarking LSB.
- Saber avaluar la robustesa d'un sistema de watermarking.
- Entendre el concepte dels codis correctors d'errors i la seva utilitat en la inserció de marques d'aigua.
- Conèixer les característiques de la compressió MP3 i la seva possible aplicació en el disseny de sistemes de watermarking.

2 Introducció

Quan parlem de marca d'aigua ens ve al cap la marca que trobem en molts papers. Fa més de 700 anys, les “marques d’aigua” es feien servir a Itàlia per a indicar el molí i l'artesà que havia produït el paper, ja que hi havia una gran competitivitat en aquest mercat. Després, cap al segle XVIII, es van començar a fer servir com a mesura antifalsificadora en els diners i altres documents. Llavors va ser quan se’ls va donar aquest nom, probablement perquè les marques es semblaven als efectes de l’aigua sobre el paper.

El primer exemple d’una tecnologia similar al watermarking digital, és a dir les marques en fitxers informàtics tal com les estudiarem en aquest treball, és una patent presentada el 1954 per Emil Hembrooke per a identificar treballs musicals. L'analogia va inspirar l'ús del terme “watermarking digital”: sembla que el 1988, Komatsu i Tominaga són els primers que utilitzen pròpiament aquesta expressió.

L'objectiu de les marques d'aigua és proveir una protecció dels drets de l'autor (copyright) per als productes digitals, davant el gran creixement de la pirateria que es deu a la facilitat que hi ha per a aconseguir-ne còpies perfectes. Inserint un missatge dins del producte de manera que no es percebi i que no sigui fàcil d'eliminar, es poden detectar les còpies il·legals del material. Cap al 1995, l’interès en aquesta branca començà a créixer, motivat per la preocupació dels distribuïdors de música, pel·lícules i software, i avui dia és un dels temes candents dins de la recerca en l'àmbit de la seguretat informàtica. S'han proposat un gran nombre de sistemes; molts d'ells s'han aconseguit trencar. I, tot i que hi ha molts temes oberts per investigar, actualment ja es té una idea de què funciona en la pràctica i què no.

En aquest treball estudiarem i implementarem una tècnica senzilla de watermarking que consisteix bàsicament en inserir una tira de bits en un arxiu format WAV estèreo utilitzant marcatge temporal. Avaluarem la seva eficiència i veurem que en principi serà fàcil d’eliminar, però a continuació farem modificacions sobre el mètode amb l'objectiu de millorar-la.

3 Fonaments

3.1 Esquemes de protecció

Les xarxes de computadors ofereixen avui dia la possibilitat de comerciar amb articles en format electrònic (música, vídeo, dades,...) dintre del que es podria anomenar comerç electrònic pur, on totes les transaccions es realitzen a través de la xarxa. Aquesta distribució digital és molt atractiva per als consumidors, però planteja problemes per a protegir la propietat intel·lectual, ja que la pirateria i distribució il·legal és senzilla de dur a terme.

Hi ha dos tipus de solucions: la protecció de còpia (impedir amb mitjans tècnics que un usuari no autoritzat dupliqui un material) i la detecció de còpia (permetre que els propietaris identifiquin còpies il·legals del seu material). La primera s'ha vist que no és gaire efectiva i per això es dediquen més esforços a millorar la detecció. La idea en què es basen tots els esquemes de detecció és fer petites modificacions al contingut de manera que s'incorpora una marca. D'aquesta manera quan es copia el material es duplica la marca.

Trobem cada dia amb més freqüència que l'àudio, el vídeo i les imatges digitals estan dotats de marques imperceptibles que contenen informació de copyright o nombres de sèrie. S'han hagut de desenvolupar tècniques per a ocultar aquesta informació de manera que no sigui fàcil descobrir-la ni destruir-la, anomenades tècniques esteganogràfiques (la paraula prové del grec "steganos"="ocult o secret" i "graphia"="escriure o dibuixar").

També s'han trobat aplicacions a les marques visibles (les més similars a les del paper), principalment en el tractament d'imatges com a mesura per a prevenir l'ús comercial del producte digital. Però nosaltres ens centrarem en les marques ocultes.

De fet l'esteganografia i el que s'anomena "watermarking" o "marques d'aigua" són conceptes profundament relacionats, inclosos dintre del que s'anomena "ocultació de la informació", però lleugerament diferents. El watermarking té com a objectiu alterar un objecte de manera imperceptible inserint un missatge que fa referència a l'objecte. En l'esteganografia general el que importa és el missatge i es tria un objecte adequat per ocultar-lo de manera que la seva comunicació sigui invisible, però no hi ha cap relació objecte-missatge.

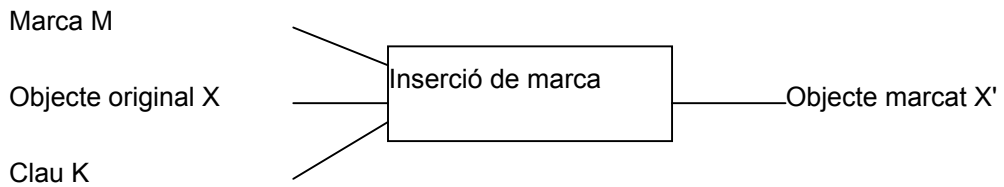
Per altra banda, l'esteganografia s'aplica típicament en comunicacions punt a punt entre dos subjectes, mentre que el watermarking s'utilitza sovint quan l'objecte està a disposició de subjectes que coneixen l'existència de les dades ocultes i poden tenir un interès en esborrar-les. Aquestes diferències en les aplicacions comporten que les propietats requerides siguin diferents, en particular la necessitat de resistència davant els atacs.

Quan parlem de "watermarking", la informació que s'insereix és un missatge de copyright, en principi el mateix per a tots els receptors o compradors. Hi ha també, però, aplicacions especials com el "fingerprinting" o "empremptatge", on el missatge és una identificació única del receptor o comprador d'un objecte concret. Les primeres marques permeten provar qui és l'autor del producte (extraient la informació inserida en el fitxer copiat) mentre que les segones són més potents perquè permeten identificar còpies il·legals. Però també són més complexes perquè requereixen propietats com la seguretat contra confabulació entre compradors (col·lisions) i plantegen altres problemes com el d'identificar al distribuïdor il·legal (traitor tracing).

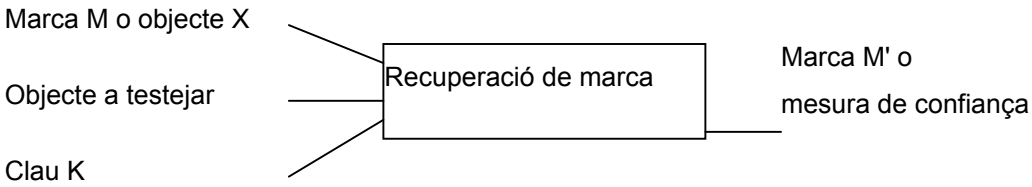
3.1.1 Principis i classificació dels mètodes de watermarking

Tots els mètodes de watermarking tenen en comú el fet que es descriuen a partir de dos algorismes: el d'inserció (embedding) i el d'extracció (reconstruction/recovery).

El procés d'inserció genèric a partir de l'objecte X, la marca M i una clau secreta K genera l'objecte marcat X'.



El procés d'extracció genèric a partir de l'objecte marcat X*, que pot haver patit modificacions, la clau K i potser altra informació (l'objecte X i/o la marca original M) obté la marca M o informació sobre si és present.



En funció de les dades originals que es necessiten i de les que s'obtenen en l'extracció es poden classificar els esquemes en tres tipus:

- privat (informed o nonblind): requereix l'objecte original X i la marca inserida i només informa de la presència o no de la marca
- semiprivat (semiblind): requereix el coneixement de la marca però no el de l'objecte original i determina si la marca és present o bé requereix el coneixement de l'objecte original però no de la marca i recupera aquesta
- públic (blind o oblivious): no requereix l'objecte original ni la marca inserida i recupera la marca

Segons l'aplicació que se'n vulgui obtenir sorgiran uns requeriments o altres i aquests definiran el tipus d'esquema necessari. Tot i així hi ha uns requeriments que són comuns a tots els esquemes, tot i que en unes aplicacions puguin ser més importants que en altres:

3.1.2 Requeriments

Imperceptibilitat:

Les modificacions que causa la inserció de la marca no han de ser perceptibles. L'objectiu és no introduir alteracions molestes indesitjables, i no reduir o destruir la qualitat i el valor comercial de les dades marcades.

Això implica que cal utilitzar algun criteri per a dissenyar l'algorisme de marca i també per a quantificar la distorsió produïda. Els mètodes han d'explotar les capacitats/incapacitats del sistema auditiu humà per tal de maximitzar l'energia de la marca amb la restricció de no superar el límit de perceptibilitat.

Una conseqüència d'aquest requeriment és que les mostres individuals que s'utilitzen per a inserir la marca es modifiquen només en una quantitat petita. I s'han de tenir en compte també els possibles processaments que aplicats a les dades puguin incrementar la visibilitat de la marca.

Un dels problemes que es planteja és com mesurar la distorsió introduïda. L'avaluació de la perceptibilitat es pot fer a través de tests subjectius que segueixin un protocol que descrigui clarament el procediment. Normalment es componen de dos passos: primer es demana ordenar de millor a pitjor uns conjunts amb distorsions i després es demana puntuar-los pel que fa a perceptibilitat (es pot utilitzar la puntuació ITUR Rec 500). Cal notar que individus amb diferent experiència poden generar diferents resultats. Aquests tipus de test són pràctics per a una avaluació final de qualitat d'una marca, però no són gaire útils en l'entorn de recerca i desenvolupament, on és preferible i més eficient utilitzar mètriques de distorsió quantitatives. Les més populars d'aquestes tècniques són la relació senyal soroll (SNR) i la relació senyal-soroll pic (PSNR).

Robustesa o seguretat pel venedor:

La marca ha de ser capaç de resistir una sèrie de possibles distorsions o atacs, s'ha de mantenir mentre el material segueixi essent útil, mentre no hagi perdut el seu valor. Les alteracions poden ser produïdes per un deteriorament accidental o per manipulació del fitxer, però el casos més perillosos són els atacs intencionats. Poden produir-se per part d'un usuari que té una còpia de l'article i hi aplica alguna distorsió imperceptible (compressió, aplicació de filtres, inclusió de soroll o noves marques). O poden realitzar-se entre diversos usuaris en el cas del fingerprinting, on cada còpia és diferent i per tant es pot obtenir informació per comparació, es podria revelar per exemple la posició d'alguns bits de la marca. En qualsevol cas, l'objectiu dels atacants serà eliminar o modificar la marca fent-la impossible d'identificar.

Es poden considerar dos grups de distorsions: el primer conté aquelles que consisteixen en soroll additiu, les segones són modificacions a la geometria temporal de les dades.

Aquesta propietat és potser la més important, perquè l'objectiu final de l'esquema és mantenir la marca dins de l'article. Però està barallada amb la imperceptibilitat ja que quan les modificacions per a inserir la marca són prou petites per a que no es detectin és possible que qualsevol petita manipulació les alteri. Com que per a solucionar aquest problema cal introduir la marca amb una certa redundància, també la taxa d'informació es veu afectada.

El fet és que la robustesa té grans implicacions en el disseny global del sistema de marcatge. No existeix actualment cap mètode de marcatge que resisteixi qualsevol tipus de distorsió, i potser no se'n trobarà cap. Els sistemes pràctics han de cercar un equilibri entre la robustesa, la invisibilitat i la taxa d'informació en funció de la seva aplicació.

Taxa d'informació:

És la mesura de la quantitat d'informació que es pot introduir en el material sense deteriorar-lo de manera que la seva qualitat quedi reduïda. És la possible longitud en bits de la marca. Ja hem vist que de cara a la robustesa es fa necessari replicar la marca per obtenir redundància i per tant es redueix la taxa d'informació.

Seguretat:

S'ha d'assumir (seguint el principi de Kerckhoff) que els algorismes d'inserció i extracció de la marca són públics, però que aquest fet no ajuda als atacants a detectar-la o eliminar-la. La seguretat recau en una clau K, que ha de ser aleatòria i independent, que ha de restar secreta per a assegurar la robustesa de l'esquema: és la informació mínima que ha de ser desconeguda per l'atacant, juntament amb l'objecte original X, evidentment. La clau no ha de ser transmesa en cap moment perquè només la necessita el venedor.

Seguretat per al comprador:

El venedor no ha de poder acusar falsament a un comprador de pirateria o distribució il·legal. L'esquema ha d'assegurar la identificació del propietari (o del comprador en el cas del fingerprinting). Cal demostrar de manera fiable la culpabilitat.

Anonimat:

S'aplica al fingerprinting i consisteix en que el comprador preservi la seva identitat quan fa una compra.

3.1.3 Productes watermarking comercial

Per tal de resoldre el problema de la pirateria estan apareixent en els últims anys diferents productes destinats a marcar els articles digitals, dels quals podem anomenar:

Digital Right Manager	Microsoft	Sistema per a la distribució via Internet de música, vídeos i altres formats de forma protegida.
AutoKey	Cognicity	Sistema de marcatge de copyright per a fitxers d'àudio.
Cryptolope	IBM	Sistema de gestió de copyrights electrònics.

Altres exemples de diferent programari així com companyies que s'hi dediquen es poden trobar a:

http://www.cl.cam.ac.uk/~fapp2/steganography/products.html.

3.1.4 Tècniques de watermarking

La manera de definir la operació d'inserció és la que determina de manera decisiva les propietats de la marca i també el mode d'extracció a utilitzar. S'han proposat moltes opcions que es poden descriure a partir d'una sèrie de decisions clau:

- com es seleccionen els punts on s'amagarà la informació, tenint en compte qüestions de seguretat i de percepció humana de la distorsió

- com es tria un domini per a realitzar l'operació d'inserció, que pot ser l'espai-temporal o bé un de transformat. La transformada discreta de Fourier, que permet controlar les freqüències del senyal, és útil per a fer una modulació de fase entre la marca i l'objecte, però s'utilitzen més algunes de les seves formes

derivades com la DCT (transformada discreta del cosinus). Aquesta resisteix millor els atacs de compressió, permet predir l'impacte sensitiu de la inserció i fer-la més ràpidament en el domini comprimit (dins d'un codificador JPEG/MPEG).

- com es dóna format al missatge abans d'ocultar-lo, introduint redundància o utilitzant codis correctors, amb l'objectiu d'incrementar la robustesa

- com es barreja el missatge amb l'objecte, posant una relació binària entre els bits del missatge i alguna característica dels de l'objecte

- com s'optimitza l'extracció, sabent que hi haurà operacions que no seran derivades de les d'inserció sinó específiques d'aquest procés

3.1.5 Avaluació de mètodes

Per a aconseguir una avaluació objectiva de l'eficiència d'un esquema de watermarking donat, cal que el provem sobre diferents conjunts de dades i utilitzant diferents claus i marques variables. Per no haver de considerar un nombre de paràmetres molt elevat, especialment si volem comparar diferents mètodes, el que es pot fixar és la quantitat d'informació a inserir, que dependrà de l'aplicació desitjada. D'aquesta manera reduirem les proves a marques de la mateixa mida i paràmetres de l'esquema que ens donin la capacitat necessària.

Ja hem vist que hi ha un compromís entre la perceptibilitat i la robustesa d'una marca. Per això haurem de considerar tant una propietat com l'altra en la nostra avaluació.

La robustesa es mesura normalment amb la taxa d'error de bit (proporció de bits extrets malament respecte al nombre total de bits inserits) després d'un atac o manipulació que no destrueixi el valor del producte. Però quantificar una característica tan important com la robustesa és fa difícil si parlem en termes tan variables com la qualitat que un usuari considerarà acceptable (que pot ser molt inferior a la que consideraria necessària el creador per a conservar la utilitat). S'hauria de determinar quin tipus i força d'atac (nivell de soroll, percentatge de compressió) s'ha de considerar en cada cas particular i imposar la condició de que un percentatge alt de bits de la marca es puguin recuperar després d'aquestes operacions. Alguns exemples de manipulacions possibles són: millorar el senyal, afegir soroll additiu i multiplicatiu (gaussià, uniforme, etc.), aplicar filtres (passa-altes, passa-baixes, lineal, no-lineal, etc.), comprimir amb pèrdua (JPEG, MPEG x, H.26x, etc.), retallar, trascodificar, fer dues conversions D/A i A/D successives, afegir noves marques, ...

La imperceptibilitat o qualitat del so es pot mesurar subjectivament o amb una mètrica de la distorsió introduïda per la inclusió de la marca. Una d'elles és utilitzar la relació senyal-soroll pic (PSNR), tot i que no és mesura de la qualitat percebuda per l'oient; s'han intentat fer millores que tinguin en compte dades perceptuals, però no hi ha un estàndard.

Per a analitzar els resultats obtinguts per a diferents claus i diferents fitxers, ens pot ser útil representar-los. Hi ha uns gràfics típics on es trien paràmetres fixes i variables per a comparar:

- Gràfic de robustesa segons la força de l'atac: mostra l'error de bit com a funció de la fortalesa de l'atac per a una qualitat donada. Permet la comparació directa de la robustesa i mostra el comportament general del mètode enfront dels atacs.
- Gràfic de robustesa versus qualitat: mostra la relació entre l'error de bit i la qualitat per a un atac fixat. Especialment útil per a determinar la mínima qualitat requerida per a obtenir una certa taxa d'error.
- Gràfic d'atac versus qualitat: mostra l'atac màxim com a funció de la qualitat per a una robustesa donada. Especialment útil si es dóna un rang de qualitats i cal avaluar la màxima distorsió permesa. També per a comparar diferents mètodes pel que fa a robustesa.

3.2 Àudio digital. Format WAV

Els sons en un ordinador difereixen dels que podem sentir en directe. Una ona sonora analògica es pot representar com una línia contínua amb un nombre infinit de valors d'amplitud possibles, però per a poder emmagatzemar aquestes dades en un suport digital són necessaris dos processos, el mostreig i la quantització, amb els quals s'obté un nombre finit de mostres que prenen un nombre finit de valors i així es poden representar per una seqüència de zeros i uns. Evidentment, el resultat no tindrà la mateixa informació que el so original, però adaptant els paràmetres utilitzats en els processos es pot aconseguir que la diferència no sigui audible.

El mostreig consisteix en triar uns punts concrets en el temps (normalment es prenen equidistants) dels quals prendrem el valor corresponent d'amplitud. Cadascuna de les mesures resultants és una mostra. S'anomena velocitat de mostreig al nombre de mostres preses per segon; les més comunes són 8000, 22050 i 44100 Hz.

La quantització consisteix en representar el valor d'amplitud de cada mostra amb un nombre finit de bits, normalment 8 o 16. En el primer cas podrem representar fins a 256 valors d'amplitud, en el segon fins a 65536.

De vegades, amb l'objectiu de crear efectes estèreo, els fitxers de so contenen diversos canals.

Hi ha diferents formats disponibles per a emmagatzemar dades àudio i cadascun ofereix certs avantatges. Aquests formats es poden organitzar en diferents categories: fitxers de música sintetitzada, fitxers de mostres... alguns d'ells es presenten en la següent taula:

Alguns formats d'àudio digital comuns (Fries and Fries 2000)		
Audio Type	**File Extension**	**Codec**
AIFF (Mac)	.aif, .aiff	Pulse code modulation (or other)
AU (Sun/Next)	.au	μ-law (or other)
CD audio (CDDA)	n/a	Pulse code modulation
MP3	.mp3	MPEG Audio Layer III
Windows Media Audio	.wma	Microsoft proprietary
QuickTime	.qt	Apple Computer proprietary
RealAudio	.ra, .ram	Real Networks proprietary
WAV	.wav	Pulse code modulation (or other)

Entre els fitxers de mostres de l'ona sonora analògica original hi ha el format WAV per PCs basats en Windows i el format MP3 que es basa en la tecnologia de compressió MPEG-1 Audio Layer 3.

El format WAV és l'estàndard de Microsoft i IBM i fins ara ha sigut el tipus de fitxer àudio més popular a Internet. Els fitxers en aquest format tenen l'extensió .wav. Els sistemes operatius Windows l'utilitzen pels sons bàsics del sistema. És un format adequat per a editar els fitxers de so: es poden eliminar defectes o afegir efectes amb un editor WAV. Actualment està essent substituït pels formats comprimits. Els fitxers WAV requereixen molt espai d'emmagatzematge (aprox. 10 Mb per minut amb mostres estèreo de 16-bit a 44.1kHz), però es poden convertir a aquests altres formats que ocupen menys espai per mitjà d'un procés de codificació.

El format WAV és una versió de la especificació RIFF (Resource Interchange File Format) genèrica de Microsoft per a fitxers multimèdia. És un format simple: les mostres d'àudio analògic es guarden en format PCM lineal sense compressió. Comença amb una capçalera de fitxer RIFF (12 bytes), segueix un "chunk" d'informació del format (24 bytes) i per acabar un "chunk" de dades de longitud variable que conté les mostres. Per mostres de 16 bits estèreo cada mostra ocupa 2 bytes (little-endian) i s'alternen les del canal esquerre i dret. Es pot variar la velocitat de mostreig, i es suporta el mostreig monaural o multi-canal, oferint diferents nivells de qualitat i de capacitat necessària.

BLOC descriptor del format RIFF:

0	ChunkID	big	4	Les lletres "RIFF" (0x52494646)
4	ChunkSize	little	4	36 + SubChunk2Size, més exactament: 4 + (8 + SubChunk1Size) + (8 +

				SubChunk2Size). Aquesta és la mida del de la resta del bloc, a partir d'aquest nombre. És la mida del fitxer en bytes menys 8 que són els dels camps no inclosos (ChunkID, ChunkSize)
8	Format	big	4	Les lletres "WAVE" (0x57415645)

BLOC que descriu el format del so:

12	Subchunk1ID	big	4	Les lletres "fmt " (0x666d7420)
16	Subchunk1Size	little	4	Sempre 0x10 (16) per PCM. És la mida de la resta del subbloc que segueix aquest nombre
20	AudioFormat	little	2	Sempre 0x01 (1) per PCM (quantització lineal) Altres valors indiquen algun tipus de compressió.
22	NumChannels	little	2	Nombre de canals: 1 per Mono, 2 per estèreo, etc.
24	SampleRate	little	4	Mostres per segon: probablement 8000 o 44100 (Hz)
28	ByteRate	little	4	Bytes per segon == SampleRate * BlockAlign
32	BlockAlign	little	2	Bytes entre mostres == NumChannels * BitsPerSample/8 Nombre de bytes d'una mostra incloent tots els canals. Què passa si no és un enter?
34	BitsPerSample	little	2	8 bits = 8, 16 bits = 16, etc.
	ExtraParamSize			Si és PCM, no existeix
	ExtraParams			Espai per paràmetres extra

BLOC que indica la longitud de les dades i les conté

36	Subchunk2ID	big	4	Les lletres "data" (0x64617461).
40	Subchunk2Size	little	4	== NumSamples * NumChannels * BitsPerSample/8 El nombre de bytes de dades. Es pot pensar com la mida de lectura del bloc que segueix aquest nombre.
44	Data	little	Subchunk2Size	Les dades de so.

Les mostres de 8 bits s'emmagatzemen com a bytes sense signe, en un rang de 0 a 255. Les mostres de 16 bits s'emmagatzemen com a enters amb signe en complement a 2, en un rang de -32768 a 32767.

3.3 Compressió MP3

Sense reducció de dades, els senyals d'àudio digitals consisteixen típicament en mostres de 16 bits gravades a una velocitat de mostreig de més del doble de l'ample de banda del senyal, per exemple 44.1 kHz pels CDs. Fent un petit càlcul (44100 mostres/s · 2 canals · 2 bytes/mostra · 60 s/min) veiem que es necessiten uns 10,6 MB per representar només un segon de música estèreo amb qualitat de CD. Això representa una gran despesa en espai de disc. A més a més, si pensem en baixar d'Internet un fitxer d'aquesta mida amb un mòdem de 56k connectat a 44k (un cas típic), la tasca ens portarà almenys 30 minuts.

Per a assolir qualitat de so acceptable amb menor quantitat de dades es poden utilitzar diferents esquemes de codificació, que minimitzen l'espai necessari per a les dades àudio. Aquests esquemes són, per tant, la tecnologia clau per a tenir aplicacions d'alta qualitat i baixa taxa de bits, com bandes sonores per jocs en CD-ROM, àudio en Internet, sistemes de difusió d'àudio digital i altres.

La compressió àudio sempre consisteix en dues parts. La primera, anomenada codificació, transforma les dades digitals àudio que resideixen, per exemple, en un fitxer WAVE, en una forma comprimida. Per a reproduir la tira de bytes resultant es necessita la segona part, anomenada descodificació, que l'expandeixi de nou a un fitxer WAVE abans de passar-lo a la targeta de so.

Els mètodes de compressió poden comportar pèrdua de dades (quan expandim el fitxer comprimit no obtindrem exactament la mateixa informació perquè s'han eliminat dades supèrflues) o no, però han de donar com a resultat un so pràcticament indistingible de l'original. Sempre s'obtindrà la millor qualitat amb el menor grau de compressió i viceversa. Els mètodes amb pèrdua són els que permeten un major grau de compressió, amb la qual cosa s'accepten les petites pèrdues; entre ells trobem els de la família MPEG.

Amb la codificació MPEG es poden reduir les dades de so fins a un factor de 12 (un factor de 4 el Layer 1, de 6-8 el Layer 2, de 10-12 el Layer 3) sense perdre qualitat de so, i factors de 24 o més encara mantenen una qualitat que és significativament millor que la que s'obté quan simplement es redueix la velocitat de mostreig i la resolució de les mostres.

MP3 és el membre més poderós de la família de codificació MPEG. Per a un nivell de qualitat fixat, requereix la menor taxa de bits - o per a una taxa de bits fixada, assoleix la millor qualitat. MP3 és acrònim de MPEG-1 Layer 3 (Moving Pictures Expert Group). És el sistema recomanat per la ITU-R (doc. BS.1115) per a l'aplicació en difusió (a taxes baixes de 60 kbit/s). Alguns exemples del seu rendiment:

Qualitat del so	Ample de banda	Mode	Taxa de bits	Grau de compressió
so telefònic	2.5 kHz	mono	8 kbps *	96:1
millor que SW	4.5 kHz	mono	16 kbps	48:1
millor que radio AM	7.5 kHz	mono	32 kbps	24:1
com radio FM	11 kHz	estèreo	56...64 kbps	26...24:1
CD	15 kHz	estèreo	112..128kbps	14..12:1

Un codificador MP3 elimina components del so que no són audibles per a la oïda humana. És un tipus de compressió amb pèrdua, cosa que vol dir que es perd informació quan s'utilitza, però la pèrdua és pràcticament imperceptible perquè el mètode de compressió controla que així sigui. Per a aconseguir-ho utilitza tècniques de codificació perceptual que exploten les propietats de la percepció de les ones per la oïda humana, les quals es van començar a desenvolupar el 1987 al IIS de Fraunhofer dintre del projecte EUREKA EU147 Digital Audio Broadcasting (DAB). En col·laboració estreta amb la Universitat de Erlangen (Prof. Dieter Seitzer), finalment van obtenir un algoritme molt potent que s'ha estandarditzat com a ISO-MPEG Audio Layer-3 (IS 11172-3 i IS 13818-3). Hi ha també una extensió que no és ISO de MPEG Layer-3 que el IIS de Fraunhofer utilitza per obtenir millor rendiment, "MPEG 2.5".

Un fitxer MPEG està construït a partir de parts més petites anomenades "frames", que són generalment parts independents que tenen la seva pròpia capçalera i la seva informació àudio. No hi ha capçalera del fitxer, de manera que es podria tallar una part qualsevol d'un fitxer MPEG (en principi tallant pels límits dels frames, però moltes aplicacions llegeixen també frames incomplets) i escoltar-la correctament. Però per fitxers MP3 això no és cert del tot, ja que la organització de les dades fa que els frames siguin interdependents.

Per a reproduir un fitxer MPEG cal llegir la capçalera de cada frame, ja que es poden utilitzar diferent taxes de bits segons el seu contingut (això és resultat de la compressió que redueix la taxa en els frames on això no perjudica la qualitat del so).

La capçalera d'un frame està formada per 4 bytes (32 bits):

11 bits	Sincronització de frame, de valor fix 11111111 111, serveix per a localitzar la capçalera
2 bits	Identificador de la versió MPEG (00=2.5, 10=2, 11=1)
2 bits	Descripció de la capa (01=III, 10=II, 11=I)
1 bit	Bit de protecció (0=hi ha CRC)
4 bits	Índex de taxa de bits (d'una taula on els valors estan en kbps i depenen de la versió i layer) Cada frame en pot tenir un de diferent, o el poden prendre de frames anteriors (només en

	Layer III)
2 bits	Índex de la freqüència de mostreig (d'una taula amb valors en Hz)
1 bit	lliure (per omplir)
1 bit	bit privat
2 bits	Modalitat de canal (00=estèreo, 01= joint estèreo, 10=canal dual, 11=mono)
2 bits	Extensió de modalitat (si l'anterior és 01)
1 bit	Copyright (0=no, 1=si)
1 bit	Original (0=còpia, 1=original)
2 bits	Èmfasis (00=cap, 01= 50/15ms, 11=CCIT J.17)

Pot haver-hi 2 bytes de CRC a continuació de la capçalera (són un bon mètode per a comprovar la validesa de la capçalera, ja que és probable que ens trobem els 11 bits de sincronització però no es tracti d'una capçalera). I a continuació ja venen les dades àudio.

Al final de les dades àudio hi pot haver un TAG de 128 bytes per a descriure el fitxer MPEG. Pot contenir informació sobre el títol (30), l'artista(30), àlbum(30), any de publicació(4) i n codi per al gènere(1), amb espai extra per a comentaris(30).

3.3.1 Implementació

La implementació de la compressió MP3 utilitza diversos algorismes matemàtics força complicats i costosos per tal de perdre només aquelles parts del so que són difícils de sentir fins i tot en el fitxer original i aconseguir que quedi més espai per a la informació important.

En MPEG Layer-3 s'utilitza un banc de filtres híbrid, format per un filtre polifase i una modificació de la transformada discreta del cosinus (MDCT). Es va triar així per raons de compatibilitat amb els seus predecessors, Layer-1 i Layer-2. La MDCT converteix els valors de les unitats de dades en una suma de funcions cosinus.

El model perceptual és el que determina la qualitat d'una implementació de la codificació. Utilitza un banc de filtres separat o combina el principal amb un càlcul de valors d'energia. La sortida del model són el límits de soroll acceptables per a cada partició, valors que si no són superats fan que el senyal comprimit sigui indistingible de l'original.

La solució comú per a quantificar i codificar en Layer-3 consisteix a quantificar amb menys precisió els valors més grans i després utilitzar codis de Huffman, que no comporten pèrdua ni afegeixen soroll, per a codificar els coeficients quantificats. La codificació Huffman assigna a cada símbol un valor binari en funció de la seva freqüència d'aparició de manera que els més usats tinguin codis més curts.

El procés de trobar el guany òptim i els factors d'escala per a un bloc, taxa i límit marcat pel model perceptual donats és fa normalment amb dos bucles imbricats fent una anàlisi per síntesi.

En el bucle interior (bucle de taxa) s'ajusta el guany global (taxa global de codificació) fins a obtenir un pas de quantització prou llarg que porta a valors quantificats menors en el cas que el nombre de bits que resulta de la operació de codificació (les taules Huffman assignen paraules més curtes als valors quantificats més petits, més freqüents) sobrepassi el nombre de bits disponibles per a codificar un bloc de dades donat

En el bucle exterior (bucle de control del soroll/distorsió) per a cada banda crítica s'aplica un factor d'escala, començant per un 1, i es va ajustant fins a obtenir un soroll de quantització que no sobrepassi el límit permès segons el model perceptual. Com que reduir el soroll implica augmentar el nombre de passos de quantització i per tant la taxa de bits, cal repetir el bucle interior per tal d'ajustar la taxa.

3.3.2 Codificadors: Lame

Hi ha molts codificadors MP3, alguns gratuïts i altres no. Alguns són ràpids però el resultat no és gaire bo i altres són lents però donen resultats amb molta qualitat. L'ideal seria un codificador gratuït, relativament ràpid i que doni excel·lents resultats, tot al mateix temps.

Mirant dins el món GNU hi ha un projecte, anomenat LAME (Lame Aint a Mp3 Encoder) que s'aproxima a aquests requeriments. Va ser desenvolupat originalment per Mike Cheng (http://www.uq.net.au/~zzmcheng) i ara és mantingut per Mark Taylor. El seu lloc web oficial és http://www.mp3dev.org/mp3/. Tenim accés al seu codi font, distribuït sota la LGPL (veure http://www.gnu.org). LAME utilitza el descodificador MPGLIB del paquet MPG123 escrit per Michael Hipp (http://www.mpg123.de) i alliberat sota la GPL.

Entre els altres codificadors hi ha el de Fraunhofer que és ràpid i molt bo, però no gratuït (http://www.iis.fhg.de/) i el de Xing Tech que és també ràpid però no gaire bo (http://www.xingtech.com/).

Lame pot codificar utilitzant 3 modes diferents: taxa de bit constant (CBR), taxa de bit mitjana (ABR) i taxa de bit variable (VBR). CBR és el mode per defecte i el més bàsic. La taxa de bit serà la mateixa per a totes les parts del fitxer. Això implica que la qualitat del fitxer resultant és variable, perquè hi haurà parts més difícils de codificar que d'altres i resultaran de qualitat més baixa. L'avantatge és que la mida del fitxer final es pot predir. Amb el mode ABR es tria una taxa de bits final i el codificador intenta mantenir aquest valor com a taxa mitjana mentre utilitza diferents taxes segons la necessitat de cada part. El resultat serà de major qualitat que en mode CBR i la mida mitjana del fitxer encara es podrà predir. Amb el mode VBR es tria la qualitat desitjada en una escala que va de 9 (menor qualitat/major distorsió) a 0 (major qualitat/menor distorsió). El codificador intenta mantenir aquesta qualitat en tot el fitxer triant el nombre òptim de bits a gastar en cada part del fitxer. L'avantatge principal és que podem especificar el nivell de qualitat, però no podem predir la mida final del fitxer.

Com que el grau (ratio) de compressió és una mesura poc manejable, els experts utilitzen el terme "taxa de bits" i la unitat kbps quan parlen de la força de la compressió. Denoten el nombre mitjà de bits que un segon de dades àudio ocuparà en el fitxer comprimit.

Els paràmetres més usuals per a cridar el programa Lame, i suficients per a la majoria d'usuaris, per a cridar el programa de compressió lame són els següents:

-b n	Taxa de bits fixa (modalitat CBR) o mínima (modalitat VBR) del resultat de la compressió; per MPEG1 entre 32 i 320, per defecte 128 kbps; per MPEG 2 entre 8 i 160, per defecte 80 kbps
--abr n	Activa la codificació ABR de n kbits, permetent l'ús de segments de diferents mides. El rang permès per a n és 8-320.
-V n	Activa el mode VBR o Variable Bit Rate i especifica la seva qualitat (0...9), 0 és la més alta
--decode	Utilitzar Lame només per descodificar, el fitxer d'entrada pot ser I,II,III (MP3)
-f	Tria algorisme de codificació ràpid amb qualitat baixa però raonable, equival a -q 7
-h	Qualitat alta amb algorisme de codificació una mica lent, equival a -q 2
-m mode	Mode (m)ono, (s)tereo o (j)oint estèreo
--longhelp	Ajuda: mostra una llista de totes les opcions possibles

Podem donar una idea superficial del funcionament intern de LAME 3.xx (de març del 2001). La part principal del codi es troba dins de la funció lame_encode_buffer(), que s'encarrega de gestionar el buffering i el remostreig, i que crida a la funció lame_encode_frame_mp3() per a cada frame. Dins d'aquesta funció es duen a terme els diferents passos de la codificació MP3:

l3psycho_anal()	Càlcul dels límits de soroll
mdct_sub()	Càlcul dels coeficients de la MDCT
iteration_loop()	Tria dels factors d'escala (per iteració) que determinen el model de soroll, i tria de les millors taules de Huffman per a una compressió sense pèrdua
format_bitstream	Construcció de la cadena de bits. Quan les dades i capçaleres estan completes, escriure-les a un buffer intern.
copy_buffer()	Copia del buffer intern de dades en el buffer mp3 de l'usuari

3.4 Audio-Watermarking

Tot i que la base teòrica de la protecció del copyright i de les marques d'aigua es comuna a tots els articles en format electrònic, la vessant tècnica més popular del watermarking estudia el cas de les imatges. Per això trobem en Internet moltes implementacions de watermarking per a imatges, però poques per a fitxers àudio.

L'àudio-watermarking consisteix en transmetre dades addicionals inaudibles juntament amb un material àudio dintre d'un canal de distribució existent. Investigacions recents han produït molts algorismes per a introduir i recuperar marques d'aigua en senyals d'àudio. La majoria de sistemes operen en el domini lineal (PCM Watermarking), uns quants són capaços d'introduir-les en material comprimit (Bitstream Watermarking).

Un factor a tenir en compte quan s'utilitzen tècniques esteganogràfiques sobre fitxers àudio és que el sistema auditiu humà (HAS) és més sensible (percep millor el soroll, el rang de potència i freqüències i les pertorbacions) a les transformacions que el visual (HVS). Per això un algorisme aplicat a material àudio pot tenir menor nivell d'imperceptibilitat que quan l'apliquem a imatges.

Parlarem una mica de les tècniques més comuns per a ocultar dades en senyals d'àudio. Recordem abans que dos dels objectius principals que s'han d'assolir són la transparència (el senyal marcat ha de ser indistingible de l'original per a la oïda humana) i la robustesa enfront de les operacions comuns de processament de àudio (per exemple: compressió mp3, filtrat passabaix, modificació de l'escala de temps, inframostreig).

La introducció de la marca en els dígits menys significatius de les mostres (l'alteració dels LSBs) és un dels algorismes més simples, una de les primeres tècniques estudiades en l'àrea de watermarking de dades digitals, tant d'imatges com d'àudio. Canviar aquests bits no produeix canvis detectables per la vista o la oïda, a no ser que podem comparar directament el fitxer original i el marcat; per tant la tècnica compleix el principi d'imperceptibilitat. Els seus principal avantatges són la gran capacitat (alta taxa d'informació comparada amb altres tècniques) i la baixa complexitat computacional (per tant, la rapidesa en el marcatge). El desavantatge evident és que la seva robustesa és molt baixa, ja que simples canvis aleatoris als LSBs (o posar-los tots a zero) anul·len les dades inserides. Altres transformacions menors de les dades (variació d'amplitud o filtrat lineal) també destrueixen la marca. És per tant un mètode fràgil i no resisteix molts dels atacs del StirMark. Per a propòsits esteganogràfics clàssics (codificar un missatge xifrat en un fitxer digital) en els que la taxa d'informació és més rellevant que la robustesa, es pot aplicar una tècnica senzilla com aquesta. Si el nostre objectiu és crear un esquema de watermarking o fingerprinting, però, hauríem de demanar més robustesa a la tècnica per tal de reduir la possibilitat d'atacs a la marca un cop distribuït el fitxer.

La codificació de fase és un mètode que aprofita el fet que el HAS és sensible als canvis en la fase d'un sistema àudio, però no a la fase absoluta. Per això es poden prendre segments del fitxer i inserir la informació en la fase inicial de cadascun, mentre que es mantenen les relacions entre les fases dintre del segment. Un oient mitjà no percebrà la diferència perquè hi ha poca distorsió del senyal..

La codificació Spread Spectrum es pot utilitzar amb material àudio de manera anàloga al cas de la imatge. Es genera un senyal pseudo-aleatori que té una distribució de freqüència uniforme i longitud de clau màxima (longitud de seqüència llarga no repetitiva). Aquesta senyal portadora és modulada per la informació a amagar (es multiplica per ± 1) i s'afegeix com a soroll blanc al senyal original. Per a descodificar s'utilitza la clau per a generar el vector aleatori que modula la portadora i es fa la correlació entre el senyal rebut i la portadora. És difícil eliminar el missatge completament sense destruir l'ona àudio perquè es distribueix en diferents bandes de freqüència. I el soroll introduït a cada banda és petit i, per tant, difícil de detectar.

La inserció d'eco en el senyal utilitza dos retards diferents per codificar els bits 0/1. S'ha demostrat que afegir eco amb una amplitud i retràs prou petits és imperceptible per als oients comuns. Aquesta tècnica trenca el senyal en segments que es desplacen en temps i en amplitud i s'afegeixen a l'original. El desplaçament temporal ve determinat pels bits de la informació a amagar (les proves mostren que ecos de 1,3ms i 0,3ms donen bons resultats en la descodificació amb audibilitat mínima). L'amplitud es pot ajustar per obtenir un eco més fort o més dèbil (amb amplitud menor de 0,5 molts pocs oients podien notar diferències en el senyal). Per a disminuir encara més la perceptibilitat, es difumina l'eco entre segments adjacents. Per extreure la marca, el senyal es trenca de nou en segments i s'examina la magnitud de l'autocepstrum del senyal. Es calcula fent $F^{-1}\{\ln F(x[n]))^2\}$, on F és la transformada de Fourier, i amb això es

transforma la convolució en una operació lineal, de complexitat molt menor, O(n log n). Com que el senyal té un eco en el segment examinat, l'espectre ho reflecteix en un pic que correspon al retràs de l'eco i l'autocorrelació converteix el retràs en constant i s'utilitza per descodificar la marca. Aquesta tècnica és la única que resisteix un atac de sincronització.

Es pot optar també per ocultar la informació en els coeficients d'una transformada del senyal, per exemple la DCT (domini freqüencial).

I una altra manera d'aconseguir que la marca sigui robusta és que el procés de marcatge treballi en conjunció amb la tecnologia de compressió. Quan codifiquem es pretén reduir la mida del fitxer eliminant les parts que no es perceben i quan inserim una marca ho volem fer de manera que no es percebi; per això es pot aprofitar el coneixement que tenim durant un procés de compressió de les dades perceptualment poc significatives per a decidir on inserim la marca. Per exemple, es pot introduir informació en les dades de so quan es fa la compressió MP3, en el bucle intern del procés de codificació. Aquest és el mètode utilitzat pel programa MP3Stego de Fabien A.P. Petitcolas, que pot ser utilitzat com a sistema de watermarking per a fitxer MP3. Si es descomprimeix el fitxer resultant i es torna a comprimir, la informació oculta s'esborra però es perd molta qualitat. El codi C complet i els executables (makefiles) estan disponibles a http://www.cl.cam.ac.uk/~fapp2/steganography/mp3stego/.

Com hem vist hi ha diverses tècniques disponibles per a codificar sobre àudio. La tècnica que triem dependrà del medi de transmissió i de l'aplicació que vulguem donar a la informació oculta. Per a esteganografia clàssica es dóna prioritat a la taxa d'informació per sobre de la robustesa i per tant es pot utilitzar una tècnica simple com la inserció LSB. Per a amagar marques d'aigua o fer fingerprinting, en canvi, es demana més robustesa, per reduir la possibilitat d'atacs sobre el missatge amagat (marca o fingerprint) després de la distribució del fitxer.

3.4.1 StirMark for Audio v0.2

Hem anomenat "atac" a qualsevol procés que pugui deteriorar una marca inserida en un fitxer. Però si cada desenvolupador d'esquemes de watermarking utilitza les seves pròpies eines de manipulació per a mesurar-ne la robustesa, difícilment podrem comparar els algorismes creats per diferents persones. StirMark és un intent de solució a aquest problema, una proposta d'entorn i material de test.

En la seva versió per a àudio, aquest programari ofereix una sèrie d'eines de manipulació de fitxers de so que efectuen diferents processos sobre el senyal amb l'objectiu d'esborrar o destruir la marca d'aigua inserida. Evidentment la manipulació redueix la qualitat del so, però es poden ajustar els paràmetres dels processos per tal d'obtenir distorsions imperceptibles.

Veiem a continuació una llista dels processos d'avaluació possibles:

Procés	Descripció
AddBrumm	Afegeix to buzz o de sinus al so.
AddDynNoise	Afegeix una part de soroll blanc dinàmic part a les mostres.
AddFFTNoise	Afegeix soroll blanc a les mostres en l'espai FFT.
AddNoise	Afegeix soroll blanc a les mostres. La unitat són els valors de mostra. El valor "0" no afegeix res i "32768" és la màxima distorsió.
AddSinus	Afegeix un senyal sinus al fitxer de so. Es pot inserir un senyal pertorbador en la banda de freqüència on es situa la marca.
Amplify	Canvia el volum del fitxer àudio. Per exemple el valor "100" no canvia l'amplificació i el valor "50" significa la meitat de volum.
BassBoost	Augmenta els baixos del fitxer de so (com la funció BassBoost d'alguns amplificadors).
Compressor	Actua com un compressor. Es pot augmentar o reduir el volum dels fragments silenciosos. La unitat per al límit és el decibel (dB). Si el valor és menor que "1", el compressor és un expansor i augmentarà el volum.
CopySample	Fa un procés com el del FlippSample però copiant les mostres entre elles.

CutSamples	Elimina mostres del fitxer àudio. Si el valor de ""Remove" és "10000" llavors s'eliminen "RemoveNumber" mostres cada "10000" mostres de manera periòdica.
Echo	Afegeix un eco al fitxer de so.
Exchange	Intercanvia dus mostres consecutives per a totes les mostres.
ExtraStereo	Augmenta la part estèreo part del fitxer. Si el fitxer no té part estèreo (és mono), llavors no fa cap efecte.
FFT_HLPassQuick	És com el RC-High- i el RC-LowPass, però ara en l'espai FFT.
FFT_Invert	Inverteix totes les mostres (part real i imaginària) en l'espai FFT.
FFT_RealReverse	Inverteix només la part real de la FFT.
FFT_Stat1	Avaluació estadística en l'espai FFT.
FFT_Test	Proves
FlippSample	Intercanvia mostres dintre del fitxer de so de manera periòdica. Permuta cada "Period" "FlippCount" mostres amb mostres que estan a distància "FlippDist". Important: Period > FlipDist > FlipCount!
Invert	Inverteix totes les mostres del fitxer àudio.
LSBZero	Posa tots els bits menys significatius (LSB) a "0" (zero).
Normalize	Normalitza l'amplificació al valor màxim.
Nothing	Aquest procés no fa res amb el fitxer àudio. La marca s'hauria de poder reconstruir. Si no, l'algorisme no ens serveix de res!
PitchScale	Fa una pitch scale
RC-HighPass	Simula un filtre passa-altes construït amb una resistència (R) i una capacitat (C).
RC-LowPass	Simula un filtre passa-baixes com el RC-HighPass.
Resampling	Canvia la velocitat de mostreig del fitxer de so.
Smooth	Suavitza les mostres. El nou valor de cada mostra depèn de les mostres anterior i posterior al punt de modificació.
Smooth2	Com Smooth, però les mostres veïnes influeixen de manera una mica diferent.
Stat1	Estadístic 1
Stat2	Estadístic 2
VoiceRemove	És l'oposat del ExtraStereo. Elimina la part mono del fitxer (on majoritàriament hi ha la veu). Si el fitxer no té part estèreo part (és mono) llavors s'elimina tot.
ZeroCross	És com un limitador. Si el valor d'una mostra és menor que el donat (límit), es posa a zero.
ZeroLength	Si el valor d'una mostra és "0" (zero exactament), s'insereixen més mostres amb aquest mateix valor (zero).
ZeroRemove	Elimina totes les mostres amb valor "0" (zero).

3.5 Codis correctors d'errors

És ben coneguda la tècnica d'afegir un bit de paritat a una cadena d'informació binària que ens proporciona una petita habilitat de tractament d'errors. Però amb aquest mètode estem limitats a detectar si hi ha error en un dels bits de la cadena: no podem esbrinar quin, no sabem si n'hi ha més d'un i no detectem res si n'hi ha un nombre parell . És possible afegir més redundància de manera que no només detectem la presència

d'errors sinó que també puguem dir on són els errors i fins i tot corregir-los; amb aquests objectius es creen els codis correctors.

El que és evident és que sempre hi haurà un conflicte entre una bona recuperació d'errors i la quantitat de redundància. Per això l'objectiu quan es dissenya un codi corrector és que mantingui una bona taxa d'informació i que pugui recuperar errors de manera eficient: aquest tipus de codis existeixen (pel teorema de codificació del canal de Shannon, 1948) però no són fàcils de trobar.

3.5.1 Codis de Hamming

Richard Hamming, company de Shannon als laboratoris Bell, va trobar la necessitat de corregir errors mentre feia la seva feina sobre ordinadors. Ja es feia servir la comprovació de paritat en els càlculs, però es va adonar que amb un patró de verificació de paritat una mica més complex es podien corregir els errors simples i també detectar els dobles. Aquesta troballa va iniciar el desenvolupament de la teoria de codis.

Tots els codis de Hamming tenen una distància de Hamming (mínim nombre de bits diferents entre dues paraules codi) de 3, es per això que, com ja hem dit, permeten corregir un error o detectar-ne dos. Els paràmetres d'un codi de Hamming són, normalment, de la forma (2^r-1, 2^r-1-r), per exemple tenim el codi (7,4) per a r=3. Per a generar un codi de Hamming amb paràmetres (n,k) es parteix d'una matriu de comprovació de paritat H que té la forma H = (P^t | I_{n-k}) on P es genera de manera que totes les columnes de H siguin linealment independents i no continguin el vector 0.

Veiem també un mètode alternatiu de construcció d'un codi de Hamming(7,4):

Tenim paraules de 4 bits de dades a les que afegim p dígits binaris de control que ocuparan les posicions iguals a les potències de 2 (1,2,4,...). D'aquí deduïm que el nombre d'aquests dígits de control ha de ser tal que 2^p≥4+p+1 i obtenim, per tant, que p=3.

Els dígits binaris del codi es poden anomenar b_7 b_6 b_5 b_4 b_3 b_2 b_1 . Llavors b_4 b_2 i b_1 són els dígits de control. Per a determinar el conjunt de posicions per als que comprova la paritat cadascun d'ells prenem les posicions que codificades en binari tenen un 1 en el dígit corresponent a la mateixa potència de 2:

Codificació de les posicions.		Controls de paritat
Posició	$b_4;b_2;b_1$	
1	0 0 1	
2	0 1 0	b_4: b_7 b_6 b_5 b_4 b_2: b_7 b_6 b_3 b_2 b_1: b_7 b_5 b_3 b_1
3	0 1 1	
4	1 0 0	
5	1 0 1	
6	1 1 0	
7	1 1 1	

Fent el càlcul d'aquests bits de paritat obtenim la següent llista de paraules codi:

	b_7 b_6 b_5 b_4 b_3 b_2 b_1	
0	0 0 0 0 0 0 0	
1	0 0 0 0 1 1 1	*Matriu correspondent de comprovació de paritat* *(de generadors)* *0 0 0 0 1 1 1* *0 0 1 1 0 0 1* *0 1 0 1 0 1 0* *1 0 0 1 0 1 1*
2	0 0 1 1 0 0 1	
3	0 0 1 1 1 1 0	
4	0 1 0 1 0 1 0	
5	0 1 0 1 1 0 1	
6	0 1 1 0 0 1 1	
7	0 1 1 0 1 0 0	
8	1 0 0 1 0 1 1	
9	1 0 0 1 1 0 0	
10	1 0 1 0 0 1 0	
11	1 0 1 0 1 0 1	
12	1 1 0 0 0 0 1	
13	1 1 0 0 1 1 0	
14	1 1 1 1 0 0 0	
15	1 1 1 1 1 1 1	

3.5.2 Aplicació al watermarking

Hi ha diverses referències que parlen de la possibilitat d'utilitzar codis correctors d'errors amb l'objectiu de millorar pel que fa a robustesa els algorismes bàsics de watermarking. La idea és natural si es compara el problema de resistència als atacs sobre les marques d'aigua amb el de la transmissió d'un senyal per un canal sorollós, on els codis correctors s'utilitzen àmpliament: el fitxer que conté la marca seria el canal i els atacs equivaldrien al soroll.

Fins i tot sembla que és més indicat fer servir codis correctors en el cas de les dades àudio que en el cas de les imatges, ja que en les primeres la marca es pot estendre pels diferents segments si és necessari.

Però els resultats obtinguts no són gaire bons degut a diversos motius. Per exemple, quan s'utilitza codificació del canal clàssica, el senyal sorollós amb el que ens enfrontem pot ser modelat en general com a soroll gaussià; en el cas d'aplicar-lo al watermarking el rang d'atacs que s'han de tenir en compte abraça sorolls de naturaleses molt diferents, no podem donar per suposat que tots són de tipus gaussià. Això implica una gran dificultat en el disseny del codi, que ha de ser molt compacte i ha de ser capaç de resistir tota una sèrie de diferents atacs. Aquestes dificultats fan que aquesta àrea d'estudi sigui oberta.

4 Sistema bàsic

Les instruccions per a assolir els objectius d'aquesta primera part del treball són les següents:

- Carregar un fitxer WAV estèreo en dos vectors: un pel canal dret i un per l'esquerre.
- Inserir una marca de n bits. Modificar el valor del bit de menys pes de la mostra equival a inserir un 1, no modificar-lo equival a inserir un 0. Fer el mateix als dos canals.
- Provar els següents atacs: addició de soroll, compressió MP3, fading de l'ona.

A continuació descrivim el disseny i la implementació i fem l'avaluació d'aquest primer sistema bàsic.

4.1 Lectura i escriptura dels fitxers WAV

Hem vist a la part de fonaments de la memòria quina és l'estructura dels fitxers WAV. Per a llegir un fitxer WAV, el que hauríem de fer és:

- llegir els primers 12 caràcters per a comprovar que és realment un fitxer WAV (conté “RIFF” en les 4 primeres posicions i “WAVE” en les posicions 9-12)

- llegir del bloc de format “fmt “ els paràmetres que ens interessin

- saltar els paràmetres extra que hi pugui haver en aquest bloc

- trobar el bloc de dades i llegir-les

- saltar tots els altres blocs, si n’hi ha

Decidim, però, no considerar el cas que les dades no estiguin en format PCM, i per això prescindim dels camps addicionals ExtraParamSize i ExtraParams que havíem vist en la descripció del format WAVE. D’aquesta manera podem prendre una estructura (WAV_HDR) de mida fixa (44 bytes) com a capçalera dels fitxers. Aquesta inclou la informació de fitxer RIFF (12), la informació de format (24) i la informació sobre les dades (8).

Aquests són els elements de la capçalera WAV_HDR:

```
struct WAV_HDR      {
  char    ChunkID[ 4]; // 4 – RIFF block      "RIFF"
  UWORD32 ChunkSize  ; // 4                   Filesize – 8
  char    Format[ 4] ; // 4                   "WAVE"
  char    SubChFID[4]; // 4 – format block    "fmt " -> fmt block
  UWORD32 SubChFSize ; // 4   leng of block   0x10 length of format block PCM
  UWORD16 AudioFmt   ; // 2 -\ block data     format tag 1=MS lineal PCM
  UWORD16 NumChannels; // 2  |                channels (1=mono 2=stereo)
  UWORD32 SampleRate ; // 4  |                samples/sec (8000 o 44100)
  UWORD32 ByteRate   ; // 4  |                bytes/sec (SampleRate*BlockAlign)
  UWORD16 BlockAlign ; // 2  |                bytes/block (all channels)
  UWORD16 BitsXSample; // 2 -/                bits/sample (8,16,...)
  char    SubChDID[4]; // 4 – data block     char "data"
  UWORD32 SubChDSize ; // 4 – leng of block  Filesize – Headersize

};
```

Fixem-nos que utilitzem el tipus char per a les característiques que es guarden en format big endian i tipus unsigned int (UWORD32 per long i UWORD16 per short) per les que tenen format little endian.

Per tal de treballar còmodament amb els fitxers WAV en la nostra aplicació ens convé definir una estructura tipus que agrupi tots els seus elements, a la que anomenem WAV_FILE. Està composada per l’estructura WAV_HDR i per un vector de vectors (ChannelData) que conté les dades de so. Aquest té tants vectors com canals, i a cadascun d'ells s'emmagatzemen les mostres que corresponen a un mateix canal. Hem decidit de moment permetre que el fitxer contingui un nombre de canals qualsevol, no necessàriament dos com es demana.

En l'estructura WAV_FILE incloem també alguns valors addicionals que caracteritzen el fitxer i que necessitem en el seu tractament. Es calculen a partir de les característiques que ja vénen donades en la capçalera (WAV_HDR), però treballem més còmodament si ja les tenim emmagatzemades en la mateixa estructura. Així la definició final de l’estructura queda com segueix:

```
struct WAV_FILE
{
  struct WAV_HDR *WAVHeader;
  int NumChannels;
  int ChannelLength;      // = SubChDSize / NumChannels
  int Samples;            // = ChannelLength / BytesPerSample
  int BytesPerSample;     // = BitsXSample/8
  char **ChannelData;
};
```

Per a llegir el fitxer WAV original, llegim en primer lloc la capçalera. Passant una variable amb l'estructura WAV_HDR, la funció read llegeix tants bytes com conté la capçalera i els mou a aquesta variable composada pels diferents camps que defineixen les característiques i el format del fitxer. Comprovem que els valors d'aquests camps corresponen efectivament als esperats en el cas d'un fitxer WAV i si no és així el programa finalitza amb un error.

La lectura anterior ens deixa al principi del bloc de mostres. Aquests bytes de dades del fitxer es llegeixen per l'ordre en què apareixen: ens anem trobant les mostres dels canals alternades (per exemple esquerre-dret-esquerre-dret-..., en el cas de 2 canals). Quan en llegim una, guardem els BytesXSample bytes que la componen en el vector que correspon al canal llegit. La següent mostra s'ha d'emmagatzemar en el canal següent, fins arribar al canal NumChannels i llavors hem de tornar a omplir el primer canal perquè hem acabat de llegir un bloc complet de mostres (corresponents a un mateix instant). Hem de repetir aquest cicle tantes vegades com la longitud en mostres d'un canal i així haurem llegit totes les mostres (en total ChannelLength*NumChannels=SubChDSize):

```
for (i=0; i<WAV->ChannelLength; i=i+WAV->BytesPerSample)
      for (k=0; k<WAV->NumChannels; k++)
            if (fread((WAV->ChannelData[k]+i),WAV->BytesPerSample,1,stream) < 1)
                  UnexpEOF();
```

Els avantatges que ens proporciona l'ús d'estructures són diversos. En primer lloc WAV_HDR ens facilita la lectura i escriptura en el fitxer, ja que la passem com a paràmetre de les funcions de la llibreria estàndard de C (read i write sobre fitxers binaris) en comptes de treballar amb els camps que la componen de manera individual, sense preocupar-nos del tipus i laa longitud d'aquests. També el fet que les característiques siguin ítems agrupats en l'estructura WAV_FILE simplifica el seu processament, i permet accedir-hi fent "wav.camp".

4.2 Disseny de l'algorisme

Ja hem vist anteriorment que la introducció de la marca en els dígits menys significatius de les mostres (el bit de menys pes és el menys significatiu) és una tècnica anomenada LSB, una de les més simples que es poden utilitzar dins de l'àrea del watermarking. La modificació d'aquests bits és difícilment perceptible per a la oïda humana.

Recordem que, quan hem repassat les tècniques de watermarking, hem vist que les propietats de la marca venen determinades de manera decisiva per la definició de la operació d'inserció. Veurem ara com podem prendre decisions en el disseny del nostre esquema LSB que ens porten a diferents opcions de marcatge.

La manera de barrejar el missatge amb l'objecte la tenim fixada. El missatge serà una cadena de bits i es tracta de modificar el valor del bit de menys pes d'una mostra quan hi volem inserir un 1 i deixar-lo igual per inserir un 0. Ho podem descriure com a una relació binària dient que la mostra modificada resulta de fer un XOR entre el byte menys significatiu de la mostra original i el byte 0x01 o 0x00, segons el bit a inserir.

Quan inserim un bit amb aquest mètode, el valor de la mostra queda alterat en una quantitat molt petita. Les mostres de 8 bits, que s'emmagatzemen com a bytes sense signe, tenen valors entre 0 i 255: canviar l'últim bit equival a sumar 1 o restar 1 al valor original (segons si aquest és parell o senar). Les mostres de 16 bits, que s'emmagatzemen com a enters amb signe en complement a 2, es mouen en un rang de –32768 a 32767: el canvi en el valor és també sumar o restar 1, fins i tot en els valors extrems no es produeix cap salt (0x0000=0 passa a 0x0001=1, 0x7FFF=32767 passa a 0x7FFE=32766, 0x8000 =-32768 passa a 0x8001=-32767 i 0xFFFF=-1 passa a 0xFFFE=-2). No ens hem de preocupar, doncs, ja que el nou valor de la mostra serà tan proper a l'original que no es percebrà la diferència.

Hi ha l'opció de crear una marca cega (blind watermark) si en comptes d'utilitzar aquest càlcul proposat inserim directament en la posició del bits menys significatiu el bit de la marca. Aquesta operació es pot descriure com un XOR entre el byte 0x01 o 0x00 i el resultat de fer un AND entre el byte menys significatiu de la mostra original i el byte NO(0x01), cosa que equival a posar a 0 el darrer bit. El benefici obtingut és que no necessitem el fitxer original per a recuperar la marca, i tenim llavors un esquema públic en comptes de semi-públic. La nostra aplicació contempla aquesta alternativa: només cal afegir la opció –b a la crida.

Ens plantegem també una altra opció, que és la d'utilitzar un altre dels bits poc significatius, no el de la darrera posició. El fet que les alteracions que es puguin fer sobre el fitxer marcat sense deteriorar-lo massa afectaran sobretot els darrers bits de les mostres, ens porta a pensar que si triem per a modificar un bit poc significatiu però que no sigui l'últim nivell podem millorar la robustesa de la marca, tot i que es tornarà més perceptible per al sistema auditiu. S'hauran d'avaluar els resultats obtinguts amb aquesta opció de l'aplicació (-l) per diferents nivells de bit.

La primera decisió important a prendre és en quines mostres amagarem la informació. Com a primera aproximació, podríem pensar en modificar els bits menys significatius de les n primeres mostres de manera que hauríem inserit els n bits de la marca tal com ens demanen. Tot i considerant que les mostres de cada canal d'un mateix bloc emmagatzemen el mateix bit (perquè fem la mateixa modificació a tots els canals), aquesta temptativa permetria que la longitud n de la marca fos el nombre de mostres per canal, de manera que assoliríem una gran capacitat. Però la robustesa seria la mínima, ja que un canvi que modifiqués la majoria de les mostres d'un mateix bloc ens impediria recuperar el bit corresponent de la marca. Parlem de "la majoria de les mostres" perquè quan volem recuperar un bit de la marca el que fem és calcular quin és el bit inserit a cada canal comparant el bit menys significatiu del fitxer marcat amb el del fitxer original (en el cas de la marca no cega, on el descodificador necessita les mostres originals), i mirant llavors quants bits d'aquest són 0 i quants són 1 podem decidir que el valor real de la marca és el més repetit. A aquesta tècnica que aprofita la redundància l'anomenem recuperació estadística.

Ens podem demanar si realment hem d'utilitzar totes les mostres del fitxer per a inserir la marca. Potser no és necessari, potser podem disminuir lleugerament la robustesa per aconseguir major imperceptibilitat (està clar és que quantes més mostres originals quedin intactes més imperceptible serà la marca). Si ens decidim per utilitzar un subconjunt de les mostres disponibles hem de tenir en compte el mètode per a triar-les tant en la inserció com en l'extracció de la marca.

Un mètode per seleccionar les mostres és fer-ho de manera aleatòria. Sabem que d'aquesta manera estem introduint soroll blanc (AWGN) de baixa potència, per tant sabem que serà poc perceptible. Haurem d'avaluar però si això fa més difícil de detectar i destruir la marca, més segura i robusta. Una versió d'aquesta selecció aleatòria s'activa amb la opció –d del programa.

La segona qüestió que es planteja és com es dóna format al missatge abans d'ocultar-lo, introduint-hi redundància amb l'objectiu d'incrementar el grau de robustesa. Si només afegim una vegada la marca ens trobem que fàcilment es pot malmetre fent petites modificacions al fitxer. La realitat és que normalment no necessitem marques molt llargues. Per això, sacrificant la capacitat podem aplicar una millora que consisteix en introduir la marca de manera repetida tantes vegades com sigui possible al llarg de tot el fitxer. Aquesta repetició aconsegueix un marcatge redundant.

Això sembla que presenta diversos avantatges: si es deterioren algunes parts del fitxer però queden intactes la majoria de les mostres que contenen un mateix bit de la marca podrem recuperar-lo estadísticament a partir d'aquestes; si el fitxer és tallat, podrem arribar a recuperar la marca sempre que el fragment de què disposem contingui almenys una de les repeticions d'aquesta.

Per últim hem de pensar com optimitzar l'extracció. Sabem que hi haurà operacions que no seran derivades de les d'inserció sinó específiques d'aquest procés, entre elles la recuperació estadística.

Donat que ens hem plantejat diferents opcions per al marcatge LSB, aquestes s'han volgut implementar com a versions del mateix programa. El resultat d'aquest treball, que és l'executable "marca", si és cridat amb diferents paràmetres, permet:

- posar la marca en diferents bits de la mostra per comprovar fins a quin nivell es manté la imperceptibilitat de la modificació (opció –l posició)

- utilitzar el bit del fitxer original (modificar-lo només quan el bit de la marca xifrada és 1) tal com es planteja en els objectius, o simplement reemplaçar el bit antic pel de la marca, amb la qual cosa ens podem estalviar l'ús del fitxer original en la recuperació de la marca (blind watermarking) (opció –b).

- xifrar la marca un sol cop (amb una seqüència pseudo-aleatòria d'1s i 0's) o xifrar cada repetició de la marca de manera diferent (opció –c)

- inserir la marca a totes les mostres (i així aprofitar tot l'espai disponible) o triar els punts (mostres) d'inserció aleatòriament de manera que es redueix la capacitat però augmenta la imperceptibilitat (opció –d)

4.3 Esquema de la inserció

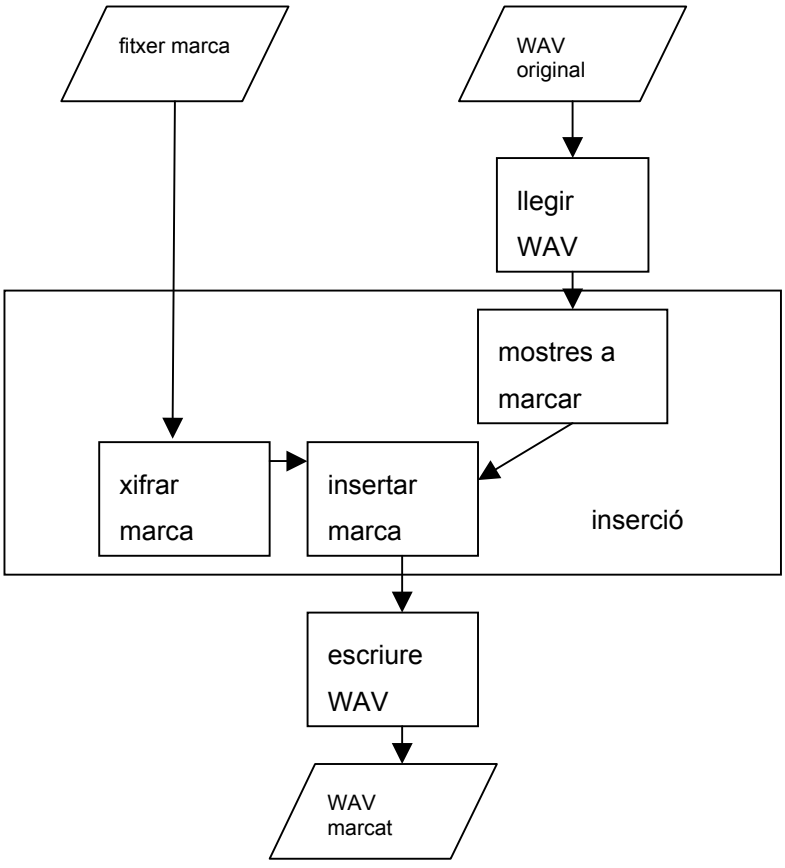
Per a implementar la funcionalitat de marcatge, el programa ha de rebre com a entrada el fitxer que volem marcar, la marca que hi volem introduir i una llavor per a generar la clau secreta (s'especifica amb la opció –s, però per defecte s'utilitza una llavor fixa). Com a sortida ens donarà un fitxer àudio marcat.

El fitxer que conté la marca es passa com a primer paràmetre del programa. Ha de contenir un byte per a cada bit d'informació: 0x00 per codificar un 0, 0x01 per codificar un 1. Aquest format ens facilita el treball posterior amb la marca. Si ens trobem en el cas d'haver d'inserir una cadena de caràcters (un codi de números i lletres o el nom complet del comprador) només caldrà transformar aquests en el seu codi binari i expandir-lo en bytes per tal de fer servir aquest mateix algorisme.

El fitxer que conté les dades àudio per a marcar el llegirem tal com hem vist al punt 4.1.

A continuació detallem els passos necessaris per a inserir la marca en el fitxer de so.

Esquema de l'algorisme d'inserció bàsic:



4.3.1 Determinar mostres a marcar:

Per defecte es marquen totes les mostres, però si s'utilitza l'opció –d es decideix per a cada mostra si es fa servir per a inserir o no a partir de la paritat d'un valor aleatori.

En aquest cas utilitzarem una seqüència pseudo-aleatòria que tants elements com mostres, generada per un PRNG (Pseudo-Random-Number-Generator), un algorisme generador de nombre aleatoris que per la seva mateixa naturalesa algorísmica no pot ser realment aleatori. Poden arribar a semblar aleatoris pel fet que produeixen una bona distribució dels valors (plana) i no hi ha correlació entre dos valors qualssevol,

però són totalment previsibles (generen sempre la mateixa seqüència de valors). Per a que el xifrat sigui segur la seqüència ha de ser generada donant un valor inicial o "llavor" al generador que sigui realment aleatori i desconegut per als possibles atacants: aquest valor el rebrà com a paràmetre d'entrada l'algorisme. Aquest valor ha de tenir com a mínim tants bits com els valors generats. Utilitzarem la funció rand() de la llibreria C estàndard, que proporciona valors aleatoris generats a partir de la llavor que se li passa amb la funció srand().

Amb la seqüència de les paritats obtinguda omplim un vector, que anomenem location[] que usarem en el moment de la inserció per a triar les mostres a modificar. Si no s'utilitza la opció –d, aquest vector té un 1 a totes les mostres per a indicar que totes poden ser utilitzades.

4.3.2 Xifrar la marca:

Cal xifrar la informació de la marca abans d'afegir-la al fitxer àudio per tal de fer més difícil que es trenqui la seguretat del sistema. Necessitem també una seqüència pseudo-aleatòria, en aquest cas de la mateixa longitud que la informació que incloem. La seqüència de valors generats la transformem en una seqüència de bits 1 i 0, prenent també com a criteri la paritat de cada valor (el resultat d'aplicar-li l'operació mòdul 2). La marca final que inserim en el fitxer és el resultat del XOR bit a bit de la informació que volem incloure i la seqüència pseudo-aleatòria generada d'aquesta manera.

Utilitzem aquí també la funció rand(). Podríem, per més seguretat, haver fet servir un generador criptogràficament segur. Aquest hauria hagut de ser, a més a més, imprevisible cap endavant (ha de ser impossible calcular el següent valor tot i conèixer l'algorisme i tots els valors anteriors) i cap enrera (impossible determinar la llavor coneixent tots els valors generats).

Aquest procés de xifrat es realitza en un pas previ a la inserció si ho indiquem amb la opció –o. Fent això totes les còpies de la marca que s'insereixen estaran xifrades de la mateixa manera. Però el comportament per defecte del procés és xifrar cada còpia de manera diferent, fent el XOR en el moment mateix de la inserció de cada bit amb un nou valor aleatori.

4.3.3 Inserir la marca:

Per tal de treballar amb el byte menys significatiu començarem pel primer byte de la primera mostra (el menys significatiu si s'emmagatzema en little endian) o per l'últim (el menys significatiu si s'emmagatzema en big endian) i després avançarem entre les mostres contingudes al fitxer fent salts de la longitud d'una mostra (BytesPerSample). D'aquesta manera, podríem treballar amb dades little endian o big endian, tot i que per al format WAV i altres formats PCM l'estàndard diu que s'utilitza sempre little-endian.

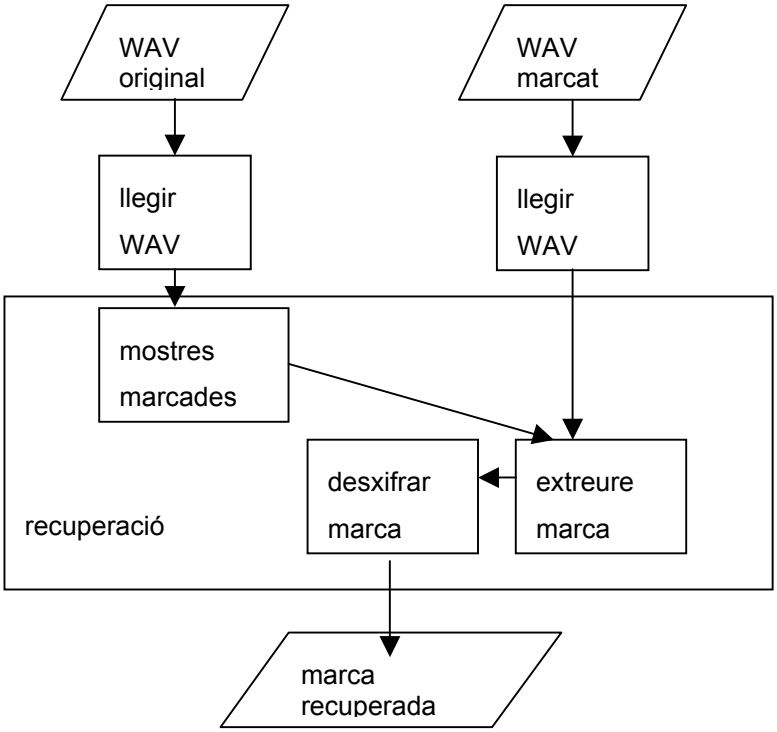
Ja hem vist que la inserció consisteix simplement en fer un XOR entre el byte menys significatiu de la mostra original i el byte 0x01 o 0x00, segons el bit a inserir. Recordem però que hem volgut donar la opció d'utilitzar un bit que no sigui l'últim: per això cal desplaçar el bit a inserir tantes posicions cap a l'esquerra com indiqui el nivell introduït a continuació del paràmetre –l (bit_level).

La marca s'introdueix de manera repetida fins a esgotar les mostres seleccionades per a modificar. Només cal tornar a començar pel primer bit de la marca cada cop que arribem al final, i això es pot indicar de manera senzilla fent la operació % (mòdul o residu) amb la longitud de la marca.

4.4 Esquema de la recuperació

Ara partim del fitxer marcat i volem recuperar la marca que s'hi ha inserit prèviament. Ens centrem en la versió no cega de l'algorisme (suposem que no utilitzem el paràmetre -b) per a descriure el procés. Per això considerem que en la crida al programa es proporciona el nom del fitxer marcat i el nom del fitxer original. Si utilitzem la versió cega, l'esquema és semblant però cal treballar amb unes dades WAV amb totes les mostres a 0 en comptes de les originals.

Esquema de l'algorisme de recuperació:



4.4.1 Determinar les mostres marcades

Es segueix el mateix procés utilitzat en la inserció per a omplir el vector location[] que indica les posicions de les mostres que han sigut utilitzades per a inserir bits de la marca. Hem de passar com a paràmetre la mateixa llavor que s'ha fet servir per a la inserció.

4.4.2 Extreure la marca

El bit inserit en una mostra es recupera comparant el bit menys significatiu del fitxer marcat amb el del fitxer original (o bé el bit de la posició indicada amb la opció –l si s'escau). Podem fer la comparació amb un XOR i el resultat ens indicarà directament quin és el bit amagat (un 1 si són diferents, un 0 si són iguals).

Per a poder portar a terme una recuperació estadística de la marca omplim en aquest moment un vector de comptadors d'aparicions d'uns on cada element correspon a una posició dins de la marca.

Un cop finalitzada la comparació de totes les mostres, revisarem aquest vector. Quan el valor d'un comptador sigui superior a la meitat de les repeticions de la marca deduirem que el bit d'aquesta posició és un 1 i en cas contrari suposarem que és 0.

4.4.3 Desxifrar la marca

Per a desxifrar hem de fer servir la mateixa llavor que en el procés d'inserció i així obtindrem la mateixa seqüència de nombres pseudo-aleatoris. Com en aquell procés, el desxifrat pot ser simultani al procés de recuperació (quan cada aparició de la marca es xifra de manera diferent) o bé posterior (quan haguem recuperat tota la marca xifrada).

4.5 Avaluació

Estudiarem ara l'eficiència de l'algorisme i n'analitzarem la robustesa davant dels atacs proposats.

Ja sabem que els principals avantatges de la tècnica LSB són l'alta taxa d'informació i la baixa complexitat computacional (per tant, la rapidesa en el marcatge). El desavantatge evident és que la seva robustesa és molt baixa, ja que simples canvis aleatoris als LSBs (o posar-los tots a zero) anul·len les dades inserides.

4.5.1 Robustesa

Obtenim el fitxer marcat a partir del fitxer original fent que el programa hi insereixi la marca emmagatzemada en un fitxer binari.

Per tal d'avaluar la robustesa de la marca executem el programa StirMark per a aquest fitxer marcat, i obtenim una sèrie de fitxers que resulten d'aplicar-hi diferents transformacions. El programa ens permet variar els paràmetres d'alguns dels atacs; podríem utilitzar aquesta possibilitat per tal que els fitxers obtinguts s'ajustin a una certa qualitat de so, però hem deixat els valors per defecte.

Hem d'intentar recuperar la marca a partir de cadascun dels fitxers alterats mitjançant l'algorisme d'extracció. Haurem de comptar en cada cas quin és el nombre de bits erronis en la marca recuperada, comparant-la amb la marca inserida. Els percentatges d'errors ens poden donar una idea aproximada de la resistència de la marca a cada atac aplicat.

Hauríem de repetir les mateixes proves per a diferents tipus de fitxers àudio: melodies d'un sol instrument, polifòniques, veu... I seria també interessant comprovar com poden influir factors com la llavor o les diferents longituds de marca en el percentatge d'errors produïts.

Fitxer original: vio_s.wav (melodia executada per un violoncel)

Inserció de la marca amb diferents algorismes:

./marca mark vio_b.wav vio_s.wav

./marca –d 541379852 mark vio_d.wav vio_s.wav

./marca –l 4 mark vio_4.wav vio_s.wav

Recuperació a partir dels fitxers alterats per StirMark:

./marca –x 96 mark_b vio_b_ATAC.wav vio_s.wav

./marca –x 96 -d 541379852 mark_d vio_d_ATAC.wav vio_s.wav

./marca –x 96 –l 4 mark_4 vio_4_ATAC.wav vio_s.wav

Comptem el percentatge de bits erronis entre la marca original i la recuperada per a alguns dels atacs:

ATAC (paràmetres)	% Bits erronis LSB simple vio_b.wav	% Bits erronis opció –d vio_d.wav	% Bits erronis opció vio_4.wav
nothing	0	0	0
addnoise_100	0,510417	0,468750	0,093750
amplify	0,677093	0,666666	0,593750
rc_highpass	0,406250	0,489583	0,479167
rc_lowpass	0,114583	0,1458	0
compressor	0	0	0
addsinus	0	0,156250	0
cutsamples	0,531250	0,468750	0,427083
addbrumm_100	1	1	0,927083
zerocross	0	0	0
zerolenght	0,5	0,520833	0,468750
zeroremove	0,5	0,510417	0,572917
Compressió MP3	0,583333	0,541667	0,645833

Transformacions menors de les dades (variació d'amplitud o filtrat) destrueixen la marca, no resisteix la majoria dels atacs del StirMark. Tampoc resisteix la compressió MP3. La distribució aleatòria de les mostres dóna uns resultats lleugerament millors en el cas de l'addició de soroll i l'amplificació.

La millora realment important és la que es produeix quan utilitzem el 4 bit menys significatiu per a inserir la marca i alterem el fitxer marcat afegint-hi soroll. En aquest cas el percentatge de bits erronis és molt petit i podríem considerar l'esquema com a vàlid. Una explicació a aquest fenomen podria ser el fet que el soroll afegit modifica només els bits per sota d'aquest 4rt.

En general, però, constatem que el mètode LSB és fràgil. Per a propòsits esteganogràfics clàssics (codificar un missatge xifrat en un fitxer digital) en els que la taxa d'informació és més rellevant que la robustesa, es podria aplicar una tècnica senzilla com aquesta. Però si el nostre objectiu és crear un esquema de watermarking o fingerprinting hauríem de demanar més robustesa a la tècnica per tal de reduir la possibilitat d'atacs a la marca un cop distribuït el fitxer.

4.5.2 Imperceptibilitat

Per a avaluar la imperceptibilitat de la marca inserida podem utilitzar el mètode subjectiu de fer que diferents persones ens responguin a un test, o bé podem mesurar la distorsió amb alguna mètrica objectiva, per exemple PSNR (disposem de codi que el calcula). Ens hem basat, però, en un primer moment, en la nostra agudesa auditiva i no hem trobat cap diferència entre els fitxers originals i els marcats amb l'algorisme LSB estàndard i amb la versió que distribueix aleatòriament la marca.

En el cas que utilitzem diferents posicions de bit per a fer la inserció, hem trobat un límit per a aquest nivell a partir del qual es comença a percebre soroll en el fitxer marcat, la posició quarta començant pel LSB. Fins a aquest valor (incloent-lo) no es detecten les diferències amb l'original.

4.5.3 Capacitat

El nombre de bits de marca que poden ser inserits és el mateix que el nombre de mostres que poden ser utilitzades per a modificar-ne el LSB. De fet, si treballem amb un fitxer estèreo, podríem inserir bits diferents al valor de cada canal de cada mostra i aconseguiríem doblar la capacitat.

En l'esquema LSB bàsic es poden utilitzar totes les mostres. En l'esquema amb distribució dels bits s'utilitzaran de mitjana aproximadament la meitat de les mostres. I el nombre de mostres d'un fitxer WAV és una quantitat molt gran (44100 per segon en molts casos).

Tant en un cas com en l'altre, si inserim aquest nombre de bits d'informació neta, no en queden per tenir redundància que proporcioni una certa robustesa a possibles alteracions. Com que en un cas pràctic no necessitarem marques tan llargues, optem per aprofitar la gran capacitat d'aquest esquema per a inserir moltes repeticions de la marca amb l'objectiu d'obtenir un mínim de robustesa.

En un fitxer WAV de 14 segons, que conté 626406 mostres, hi podem inserir per exemple una marca de 96 bits 6525 vegades. I, considerant que és estèreo, fins i tot podem aprofitar els dos canals (la nostra aplicació així ho fa) i tenir d'aquesta manera el doble de còpies.

4.5.4 Informació secreta

En qualsevol esquema de protecció del copyright se suposa que el fitxer original és secret, ja que de no ser així les còpies d'aquest fitxer no serien detectables pel fet de no estar marcades. El fet que els algorismes que hem presentat requereixin el fitxer original en l'extracció fan que ningú pugui obtenir la marca. Però en el cas que utilitzéssim una versió "cega" de l'algorisme, que convertiria l'esquema en públic, es faria completament necessari l'ús d'una clau per a xifrar o inserir la marca.

En l'esquema més senzill que hem estudiat, el coneixement de l'algorisme d'inserció ens permet detectar els bits de la marca, però el fet que estigui xifrat amb una clau que només coneix el propietari no ens permet conèixer el valor real de cada bit.

En el cas que la marca es distribueix aleatòriament en el fitxer tenim un altre nivell de secret, ja que les posicions on s'insereixen els bits no seran conegudes si no es fa pública la clau de distribució.

5 Primera millora

Els objectius en aquesta segona part són els següents:

- Emprar un codi corrector d'errors per a posar la marca de manera que es millori el sistema.

Partint del nombre de mostres que poden ser marcades sense degradar la qualitat (fins ara totes) i sabent que a cada mostra podem inserir un bit podem calcular la quantitat d'informació que podem inserir en un fitxer de so. Però ja hem vist aquesta xifra no es pot considerar com a informació neta perquè es fa necessària una certa redundància per obtenir robustesa, és a dir, per a assegurar la recuperació de la marca tot i els atacs.

En el moment en què hem repetit la marca tantes vegades com ens cabia dins del fitxer original, hem afegit molta redundància. Fent això i utilitzant un recompte del nombre d'ocurrències d'1 i 0 en cada posició per a descodificar es podria obtenir el missatge correcte fins i tot quan hi ha molts errors.

Però, a banda de repetir el missatge, podríem utilitzar un codi corrector que ens proporcioni més robustesa també a canvi de reduir la taxa d'informació. L'avantatge respecte a la simple repetició de la marca és la seva capacitat correctora.

La idea és codificar la marca abans d'inserir-la en el fitxer. S'obté una nova marca "codificada" que ocupa més espai però que conté la suficient redundància com per a permetre corregir algun dels seus bits quan resulten alterats. Quan vulguem recuperar aquesta marca ens trobarem potser que alguns dels bits recuperats no són correctes degut a les alteracions que ha patit el fitxer, però tindrem la possibilitat de descodificar-la i en aquest procés detectar i corregir alguns dels errors.

5.1 Implementació

Creem unes rutines de codificació que implementen un codi Hamming (7,4). Les paraules d'aquest codi són les següents:

{ 00000000, 00001111, 00010011, 00011100, 00100101, 00101010, 00110110, 00111001,

01000110, 01001001, 01010101, 01011010, 01100011, 01101100, 01110000, 01111111 }

En el nostre codi, però, treballem amb vectors de bytes que valen 0x00 o 0x01 (bits emmagatzemats en char) per tal que puguem cridar les funcions de codificació i descodificació passant com a paràmetres el vector amb la marca original (mark) i el vector amb la marca codificada (coded_mark).

La funció de codificació encode_block() pren les dades del vector d'entrada en trossos de 4 bits, els quals converteix en un nombre enter entre 0 i 16. Utilitza aquest enter per a localitzar la paraula codi (de Hamming_codes) corresponent a aquesta meitat de byte mitjançant la funció data_to_code(). La paraula codi són 8 bits que s'escriuen al vector de sortida.

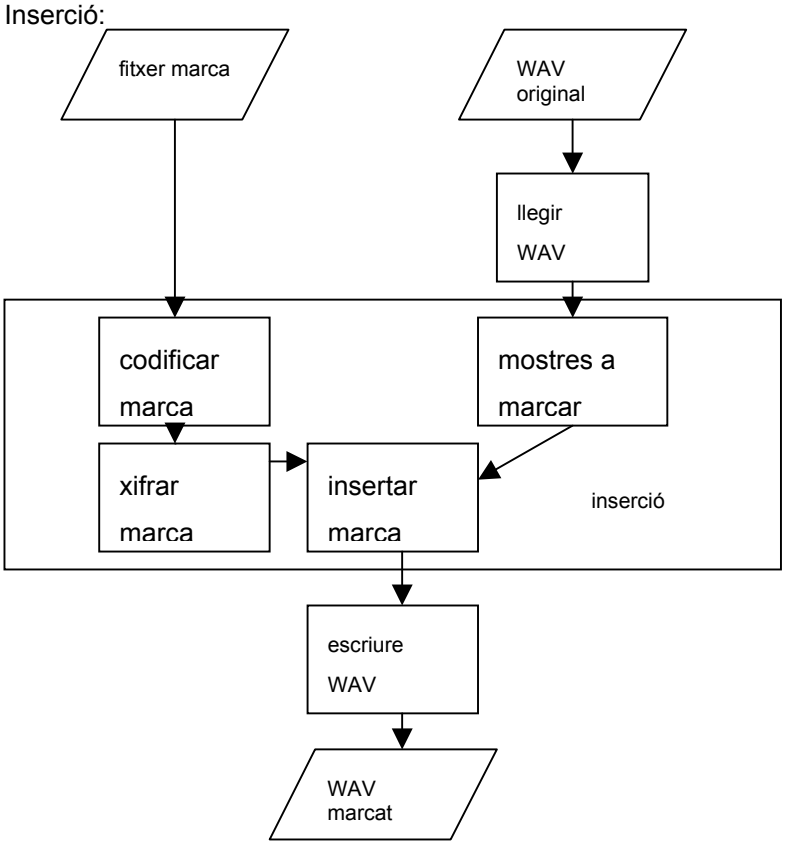
La funció de descodificació decode_block() pren les dades del vector d'entrada en trossos de 8 bits. Intenta localitzar el valor original de 4 bits que correspon a aquesta paraula codi mitjançant la funció code_to_data(). Considera el cas que la paraula llegida no es trobi entre les paraules codi vàlides (la funció retorna error), i en aquest cas el que fa és cridar la funció correct_code(). Allà es prova de corregir la paraula modificant un a un els seus bits i comprovant si cada paraula resultant és una paraula codi: si en troba una, la retorna. Llavors es calcula el valor original d'aquesta nova paraula, i s'escriuen els 4 bits en el vector de sortida.

Modifiquem les funcions d'inserció i recuperació de la marca per tal d'incloure la codificació i descodificació de la marca, respectivament.

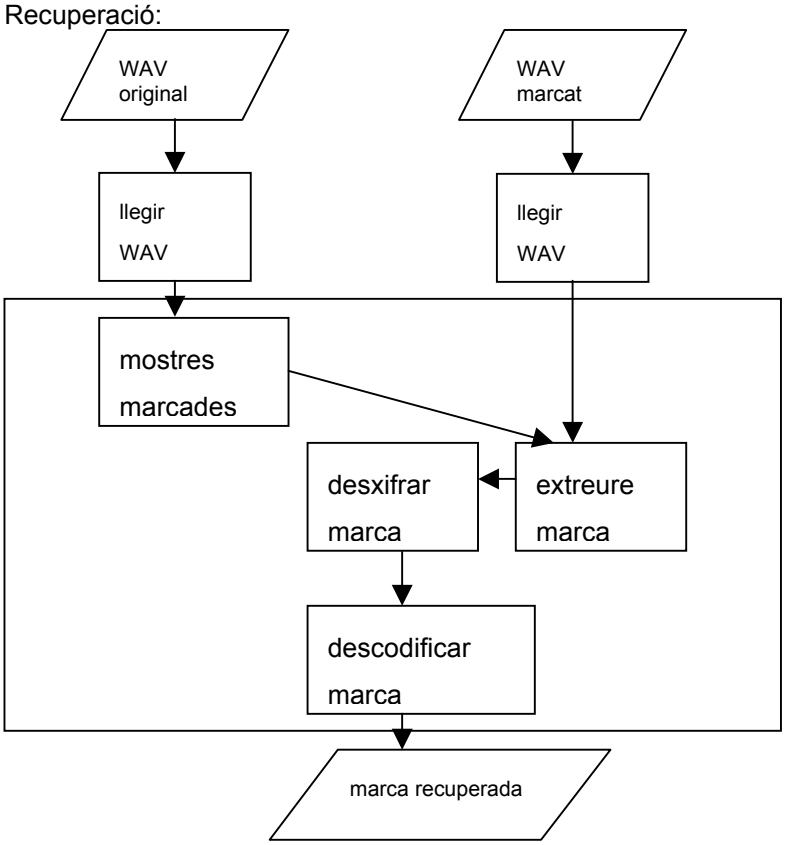
En la inserció farem l'operació de barrejar la marca amb les dades de so amb un nou vector "coded_mark" que omplim de bon principi amb la marca codificada obtinguda mitjançant la funció encode_block() o amb la marca tal com està al vector mark (en cas que no vulguem utilitzar la codificació). Observem com a l'esquema de l'algorisme hem afegit el pas de codificació abans del xifrat i la inserció de la marca.

En la recuperació estadística de la marca el vector on emmagatzemem la marca recuperada l'anomenem també coded_mark. En el moment que ja tenim la marca calculada la descodifiquem utilitzant la funció decode_block() sempre que indiquem que hem utilitzat la codificació en la inserció. A l'esquema de l'algorisme de recuperació hem afegit el pas de descodificació de la marca després de l'extracció i el desxifrat.

Inserció:



Recuperació:



Fixem-nos que el codi Hamming que hem triat fa que l'espai ocupat per les dades codificades sigui el doble del de les dades originals. Per això en el cas que vulguem utilitzar aquesta codificació per a inserir la marca hem de reservar per al vector "coded_mark" dues vegades la dimensió de la marca original. Altres codis que tenen més capacitat correctora d'errors incrementarien encara més l'espai utilitzat.

5.2 Avaluació

Repetim les proves que hem fet amb el sistema bàsic, ara utilitzant la opció de codificació (-e) del programa marca.

Resistència als atacs de StirMark i a la compressió:

Atac (paràmetres)	% Bits erronis opció -e	% Bits erronis opció –e –d	% Bits erronis opció –e –l 4
	vio_e.wav	vio_ed.wav	vio_e4.wav
nothing	0	0	0
addnoise_100	0,5	0,458333	0,020833
amplify	0,614583	0,479167	0,572917
rc_highpass	0,531250	0,583333	0,635417
rc_lowpass	0,281250	0,364583	0
compressor	0	0	0
addsinus	0	0,322917	1
cutsamples	0,427083	0,458333	0,520833
addbrumm_100	1	0,947917	0
zerocross	0	0	0
zerolenght	0,604167	0,5	0,531250
zeroremove	0,572917	0,572917	0,562500
Compressió LAME	0,500000		0,552083

Els atacs on es produeixen lleugeres millores són l'amplificació i l'addició de soroll. Aquest resultat concorda amb el fet que aquest tipus de codi corrector està pensat per a millorar la transmissió en presència de soroll.

Els resultats són encara dolents, no es pot donar com a vàlid un esquema amb taxes d'error tan altes. Només la combinació de la codificació amb la inserció en el quart bit permetria utilitzar l'algorisme, suposant que l'únic atac possible fos l'addició de soroll.

6 Segona millora

Objectius:

- Millorar el sistema usant la informació que ens dóna la compressió MP3.
- Implementació usant llibreries MP3 i anàlisi.

Hem vist que la tècnica LSB és molt fràgil i en particular no és robusta respecte la compressió MP3. Si amaguem una marca en un fitxer WAV, el convertim a format MP3 i després el descomprimim de nou a WAV, ens trobem que no podem recuperar la marca. Això és degut a que els bits menys significatius de les mostres són els més propensos a ser modificats per un algorisme de compressió amb pèrdua com el que es realitza en convertir a MP3.

La compressió amb pèrdua tendeix a eliminar les components d'alta freqüència perquè són les menys perceptibles i deixa les més baixes. Altres atacs afecten també majoritàriament les mostres insignificants perceptualment. Això interfereix amb la idea dels algorismes de marcat d'inserir el missatge en els punts on es percebi menys per produir menys distorsió, per no afectar la qualitat perceptual. S'ha suggerit, però, que les marques s'han de situar en components perceptualment significatives tot i les distorsions potencials per tal d'assegurar una certa robustesa.

El nostre mètode de codificació LSB és molt senzill i no té en compte la psicoacústica. Si poguéssim esbrinar i tenir en compte quines són les components perceptualment significatives dins de les dades de so en el moment de buscant una localització de la marca, podríem millorar la robustesa del sistema que és una de les qualitats importants.

La importància perceptual de les mostres es troba quan es treballa en el domini freqüencial del senyal. Hi ha esquemes de watermarking, com els basats en la transformada discreta del cosinus (DCT) i en les wavelets que determinen la localització de la marca amb tècniques perceptuals similars a les que utilitza la compressió MP3. Per això les marques inserides amb aquests mètodes resisteixen la compressió.

Implementar aquest tipus de tècniques que es mouen en el domini freqüencial comporta la utilització de rutines matemàtiques força complexes. Nosaltres el que intentarem serà detectar les mostres significatives perceptualment utilitzant el propi algorisme de compressió MP3 de manera tancada, sense haver d'entrar en els detalls de la seva implementació.

La idea per a enfortir la tècnica és triar com a mostres a modificar (mostres on podem inserir bits de la marca) aquelles que romanen iguals després de l'operació de compressió i descompressió MP3, que són precisament les més significatives perceptualment.

6.1 Implementació

Necessitem un codi que realitzi les operacions de compressió i descompressió MP3. Utilitzarem amb aquest objectiu:

- la funció de codificació lame_encode_buffer() continguda en la llibreria libmp3lame.a, que forma part del codi de l'aplicació Lame. Transforma dades WAV al format MP3.
- la funció de descodificació proporcionada per mpg123, que està inclosa en la mateixa llibreria. Fa el pas invers.

Obtenim el codi font de la llibreria (lame-3.96.1.tar.gz). Cal que el compilem segons les instruccions proporcionades per tal d'obtenir el fitxer libmp3lame.a.

Cal que incloguem el fitxer "lame.h" com a part del nostre projecte i que linkem estàticament a "libmp3lame.a". També podríem haver optat per instal·lar la llibreria compartida libmp3lame.so.

Es poden trobar descripcions de la interfície per al codificador MP3. A l'apèndix descrivim els passos que s'han de dur a terme i les funcions que s'han d'utilitzar per a aconseguir transformar les dades PCM d'un fitxer WAV en dades en format MP3.

No hem trobat una descripció clara de la interfície per a la descodificació. Però per a deduir la manera correcta d'utilitzar-la podem seguir les indicacions incloses en el fitxer capçalera "lame.h" i el codi font dels programes d'exemple "main.c" i "mp3rtp.c".

No arribarem a escriure en un fitxer les dades mp3 obtingudes amb la codificació, ja que només necessitem el resultat de tornar-les a descomprimir per tal de comparar-lo amb el fitxer WAV marcat i així determinar les mostres a utilitzar per a inserir la marca. El fitxer comprimit no ens interessa.

Haurem de descriure els dos passos de l'esquema de watermarking: la inserció i la reconstrucció. El primer determinarà en part el segon, per això ens centrem en primer lloc en on i com inserirem els bits de la marca.

6.1.1 Inserció de la marca

La localització de la marca es determina amb l'algorisme de compressió MP3, prenent les mostres que romanen iguals després de les operacions de compressió i descompressió. El resultat d'aquesta comparació s'introdueix en el vector location[], és a dir, es marquen amb un 1 les posicions corresponents a les mostres que no es modifiquen.

Tal com fèiem en la versió anterior del nostre programa, xifrem la marca abans d'inserir-la per tal que no pugui ser obtinguda sense la clau de xifrat. Els bits de la marca s'insereixen només en les mostres que es troben en les posicions indicades al vector de localització. Seguim intentant obtenir redundància replicant la marca tantes vegades com sigui possible (això dependrà del nombre de mostres que resultin adequades per a inserir bits de la marca). I hi ha la opció d'aplicar també el codi corrector d'errors.

6.1.2 Recuperació de la marca

Cal conèixer el fitxer de so original i la clau secreta de xifrat de la marca. No és necessari conèixer la marca. El procés es pot dividir en tres etapes. Primer determinem les mostres on estan amagats els bits de la marca tornant a calcular la diferència entre el fitxer original i el comprimit i omplint també el vector location[]. Després hem de deduir el valor que està inserit en cadascuna d'aquestes mostres, tenint en compte que el fitxer ha pogut patir alteracions. Per últim descodifiquem la marca trobada usant el codi corrector d'errors (si l'hem utilitzat) i el fet que hem replicat la marca. De fet l'esquema és pràcticament idèntic al que utilitzem en la primera millora, amb la única diferència de que la en la determinació de les mostres que s'utilitzen s'ha inclòs la comparació entre el fitxer original i el comprimit.

6.2 Avaluació

Mesurarem (de manera empírica) la seva capacitat, la seva imperceptibilitat i la seva robustesa segons els paràmetres utilitzats. Haurem de fer proves amb diferents paràmetres en la compressió MP3 per veure quina configuració ens dóna millors resultats de robustesa. S'haurien d'utilitzar diferents tipus de fitxers de so.

6.2.1 Robustesa

L'avaluarem utilitzant alguns atacs bàsics de StirMark for Audio: filtrat, retallat, compressió... després de comprovar que el nivell de degradació no sigui gaire alt. Considerarem que es supera un atac quan es pot recuperar més del 80% dels bits de la marca. La robustesa està estretament relacionada amb la capacitat perceptual de cada mostra modificada.

Fem una avaluació també utilitzant l'atac de compressió MP3 amb el programa Lame. Mostrem tot seguit una taula amb el percentatge de bits erronis obtingut en diferents proves:

Qualitat compressió	% Bits erronis opció –m 11	% Bits erronis opció –m 9	% Bits erronis opció –m 6
2	0,25	0,375	0,395833
default (3)	0,104167	0,208333	0,208333
7	0,552083	0,5624	0,541667

6.2.2 Capacitat

Vindrà donada pels paràmetres que utilitzem per a la compressió MP3. Mesurarem en cada prova el percentatge de mostres canviades (recordem que s'insereix un bit en cada mostra). El nombre de bits que poden ser inserits decreixerà quan el fitxer comprimit sigui més semblant a l'original, tot i que depèn de característiques intrínseques al senyal.

Mostrem una petita taula amb el nombre de mostres que s'utilitzen per a la inserció prenent diferents graus de compressió:

Ratio	Mostres
11	990
9	593
6	591

6.2.3 Imperceptibilitat

En una audició dels fitxers marcats no detectem la distorsió afegida per la inclusió de la marca.

7 Conclusions

7.1 Observacions finals

En aquest treball, després de fer un repàs dels esquemes de protecció del copyright d'articles electrònics, ens hem centrat bàsicament en les eines de watermarking per a material àudio.

Hem analitzat i implementat diversos esquemes de watermarking semi-públic per a fitxers àudio, tots basats en el mètode LSB, i hem avaluat les seves propietats de robustesa, capacitat i imperceptibilitat.

Les diferents modificacions que hem anat duent a terme a partir d'una implementació inicial senzilla del marcatge LSB ens han permès entendre per una banda la necessitat de redundància per a poder garantir una mínima robustesa del sistema, i per l'altra la importància que té la ubicació de la marca dintre del fitxer àudio.

El principal avantatge del mètode LSB és la baixa complexitat computacional, que és de l'ordre de O(n). També proporciona una alta capacitat d'inserció de bits de la marca I, a més a més, la modificació produïda per aquesta inserció no és audible. Però no compleix uns requeriments mínims de robustesa, ja que la majoria de manipulacions normals que no degraden perceptiblement el fitxer marcat impedeixen la recuperació de la marca. Per aquest motiu, i pel fet que en conèixer els punts d'inserció de la marca aquesta no és estadísticament invisible, no es podria utilitzar com a algorisme de watermarking. Podria ser útil, en canvi, per a propòsits esteganogràfics típics, per exemple ocultar dades en els fitxers àudio.

La distribució aleatòria dels bits de la marca millora l'esquema en un aspecte, que és fer desconeguts els punts d'inserció dels bits de la marca. Però la robustesa no incrementa, més aviat al contrari, ja que el fet de no utilitzar tota la capacitat del fitxer ens obliga a disminuir la redundància.

Hem comprovat que en inserir els bits en posicions més significatives de les mostres, es podia conservar la propietat d'imperceptibilitat (per a la oïda humana) fins a la quarta posició menys significativa. I hem vist que aquesta modificació de l'esquema millorava la resistència de la marca a alguns atacs, com el soroll additiu gaussià. I encara resultava més robusta si s'afegia una codificació prèvia a la inserció.

L'ús d'un codi corrector per a formatejar la marca abans de la inserció no només aporta redundància, sinó també capacitat correctora que és útil en el procés de recuperació. Però els resultats obtinguts no han sigut gaire significatius, degut potser a la tria d'un codi amb poca potència.

La darrera millora que hem volgut incorporar a l'esquema inicial ha estat l'ús de la compressió MP3 amb l'objectiu de seleccionar com a mostres a modificar les que tenen major importància perceptual. El benefici que volem obtenir és una major robustesa perquè aquestes mostres seran les que tenen més probabilitat de sobreviure als atacs. Però cal ajustar els paràmetres d'aquestes operacions per tal de conservar la imperceptibilitat de la marca i tenir prou capacitat.

7.2 Consideracions futures

Aquest treball podria ser un punt de partença per a una recerca més exhaustiva dintre del món del watermarking.

L'aplicació que hem obtingut és millorable en molts aspectes:

- En primer lloc, quan hem decidit implementar l'algorisme LSB de manera que no s'utilitzessin totes les mostres, la opció que hem triat per a distribuir-les ha sigut la més senzilla d'implementar, però no la millor. Es podrien haver provat altres sistemes, com el mètode d'intervals aleatoris en el que es determina aleatòriament la distància entre les mostres usades, o bé el mètode de permutacions pseudo-aleatòries, en el que es genera una seqüència d'índexs de les mostres i es van col·locant els bits a l'índex que toca (de manera que els bits no queden ni tan sols ordenats, però s'han de tenir en compte les col·lisions).
- En segon lloc, el codi corrector implementat és poc potent, i potser utilitzant-ne un altre que proporcioni més redundància es podrien millorar els resultats obtinguts. A més a més, podria ser interessant buscar altres tipus de codis correctors que poguessin aportar més robustesa davant els diferents tipus d'atacs.
- Per últim, la compressió MP3 s'ha utilitzat només per a decidir la ubicació dels bits de la marca, però es podria haver optat per una inserció diferent de la substitució LSB que aportés algun benefici extra. Es podria refinar l'ús de la compressió MP3 dissenyant un esquema més adient.

També hi ha punts que no hem tingut en compte com, per exemple, el tractament d'atacs que canviïn significativament el número de mostres. Caldria cercar idees aplicables a la recuperació de la marca en aquests casos.

En el cas dels fitxers estèreo com els que hem tractat, s'hauria de considerar la possibilitat d'utilitzar diferents tipus d'algorisme a cada canal de manera que un dels mètodes fos fort allà on l'altre fos feble. Per a fitxers amb més canals, com els Dolby Surround 5.1, potser es podrien millorar encara més els resultats.

L'esquema estudiat no necessita la marca en el procés de recuperació. Millorant la robustesa (la taxa de bits erronis hauria de ser del 0%) es podria utilitzar l'esquema per al fingerprinting Hem trobat que l'ús d'algun tipus de codi corrector d'errors ens ho permetria, ja que es podrien evitar col·lisions de dos compradors. Es podria analitzar aquesta possibilitat, definir un protocol, implementar-lo i avaluar-lo.

8 Referències i recursos

Bibliografia (llibres, articles, pàgines web, etc.) consultada durant la realització d'aquest treball:

8.1 Esteganografia i Watermarking

S. K. (editor) and F. P. (editor). *Information Hiding Techniques for Steganography and Digital Watermarking*. Artech House, 2001.

F.A.P., R.J. Anderson and Markus G. Kuhn. University of Cambridge Computer Laboratory, Security Group, Information Hiding - A Survey. http://www.cl.cam.ac.uk/~fapp2/publications/ ieee99-infohiding.pdf.F19 FA27 2F94 998D FDB5 DE3D F8B5 06E4 A169 4E46

F.A.P. Petitcolas and R.J. Anderson. Evaluation of copyright marking systems. In *Proceedings of IEEE Multimedia Systems' 99*, pages 574{579, 1999.

F.A.P. Petitcolas, R.J. Anderson, and M.G. Kuhn. Attacks on copyright marking systems. In *2nd Workshop on Information Hiding*, LNCS 1525, pages 218{238. Springer-Verlag, 1998.

Zsolt Szabo. Watermarks and Hidden Information in Communications: An Introductory Guide to Contemporary Methods. *GSEC Practical v1.4b*, (c) SANS Institute 2003

Steve Czerwinski, Richard Fromm, Todd Hodes. Computer Science Division, University of California, Berkeley. Digital Music Distribution and Audio Watermarking.

Ingemar J. Cox and Matt L. Miller. Electronic watermarking: the first 50 years.

Nedeljko Cvejic, Tapio Seppänen. Increasing Robustness of LSB Audio Steganography Using a Novel Embedding Method

P.Bassia and I.Pitas. Dept. Of Informatics, University of Thessaloniki. Robust audio watermarking in the time domain.

Dr. Mohammed Al-Mualla and Prof. Hussain Al-Ahmad (MCSP Research Group). Information Hiding: Steganography and Watermarking.

Jordi Herrera Joancomartí. Secure electronic commerce of multimedia contents over open networks.

J. Minguillon, J. Herrera-Joancomartí and D. Megías. Empirical Evaluation of a JPEG2000 Standard-based Robust Watermarking Scheme.

Anderson, R. & Petitcolas F. On the limits of steganography.

K. Weeks. Hiding in plain sight.

Fabien A. P. Petitcolas. The information hiding homepage digital watermarking & steganography http://www.petitcolas.net/fabien/steganography/

Neil F.Johnson. Steganography & Digital Watermarking -Information Hiding- http://www.jjtc.com/Steganography/

Gary C. Kessler. Computer & Digital Forensics Program, Champlain College. An Overview of Steganography for the Computer Forensics Examiner. *Forensic Science Communications, Juliol 2004.* http://www.garykessler.net/library/fsc_stego.html

StegoArchive.Com. Steganography Information, Software, and News to enhance your privacy: http://www.stegoarchive.com/

Digital Watermarking World: http://www.watermarkingworld.org/WMMLArchive/

Computer Forensics, Cybercrime and Steganography Resources: http://www.forensics.nl/steganography

8.2 WAV

Robert Shuler. Microsoft WAV Sound File Format. General RIFF File Background (http://www.wotsit.org/)

Audio Engineering Society. http://www.aes.org/

Software àudio per a Linux. http://linux-sound.org/

8.3 StirMark

http://www.petitcolas.net/fabien/watermarking/stirmark/

StirMark Benchmark for Audio (SMBA) http://amsl-smb.cs.uni-magdeburg.de/smfa/main.php

8.4 MP3 i Lame

Mark Noto. MP3Stego: Hiding Text in MP3 Files. SANS GIAC paper

Gabriel Bouvigne. MP3'Tech. http://www.mp3-tech.org/

Marshall Brain. How MP3 Files Work. http://www.glump.net/docs/How%20MP3%20Files%20Work/

Chern Lee. MP3 Audio. http://freebsd.active-venture.com/handbook/sound-mp3.html

Mpeg TV. AskMP3, The MP3 Portal by MPEG.ORG http://www.mpeg.org/MPEG/mp3.html

Predrag Supurovic. MPEG Audio Compression Basics. http://www.dv.co.yu/mpgscript/mpeghdr.htm

Digital Audio Systems. MP3 Encoder software. http://das.iocon.com/technical_enc.shtml

Davis Pan. Motorola Inc.A Tutorial on MPEG/Audio Compression. http://das.iocon.com/res/docs/pdf/mpegaud.pdf

Fraunhofer IIS. Audio & Multimedia. MPEG Audio Layer-3 http://www.iis.fhg.de/amm/techinf/layer3/index.html

The LAME Project (SourceForge.net) http://www.mp3dev.org/ o http://lame.sourceforge.net/ http://sourceforge.net/projects/lame/

Lame Manual Page. http://www.usinglinux.org/man/lame.1.html

Lame File List, http://www.leidinger.net/lame/doxy/html/files.html

Software: Sound: mpglib. http://www.really.demon.co.uk/Pages/normal/software/sound/mpglib/mpglib.html

8.5 Programació en C

C Coding Standard. http://www.ece.cmu.edu/~eno/coding/CCodingStandard.html

Llibreria GNU C. http://www.gnu.org/software/libc/manual/html_mono/libc.html

Eric Huss. The C Library Reference Guide. http://www.acm.uiuc.edu/webmonkeys/book/c_guide/

9 Apèndix

9.1 Ús de l'aplicació i exemples d'execució

9.1.1 Inserció

```
marca mark vio_b.wav vio_s.wav
```

Distribuint la marca
```
marca –d mark vio_d.wav vio_s.wav
```

Utilitzant un codi corrector d'errors:
```
marca –e mark vio_e.wav vio_s.wav
```

Utilitzant la compressió MP3:
```
marca –m 7 mark vio_m.wav vio_s.wav
```

9.1.2 Extracció

```
marca -x 96 mark_b vio_b.wav vio_s.wav
```

Distribuint la marca
```
marca –x 96 –d mark_d vio_d.wav vio_s.wav
```

Utilitzant un codi corrector d'errors:
```
marca –x 96 –e mark_e vio_e.wav vio_s.wav
```

Utilitzant la compressió MP3:
```
marca –x 96 –m 7 mark_m vio_m.wav vio_s.wav
```

9.2 Codi font

9.2.1 Algorisme d'inserció

```
/*-------------------------------------------------------------
  set_mark_on_WAV
    Embedding of a mark into a WAV file. Steps:
      location of samples to mofidy
      encryption
      encoding (error correcting code)
      insertion
  -------------------------------------------------------------*/
void set_mark_on_WAV(WAVFILE *WAV, char *mark, int mark_lng,
  int seed_encr, short blind, short bit_level, short one_key,
  short dist, int seed_dist, short ecc, short mp3, int ratio )
{

  int i,j,k,first;
  int mod_samples_cnt;
  int coded_mark_lng;
  char *coded_mark;
  char *location;
  char value;
  WAVFILE *MWAV;

  /* The LSbyte of each channel sample is the first one, supposing
    little endian or the last one, supposing big endian */
  switch (ENDIAN)   {
    case 'L': first = 0; break;
    case 'B': first = WAV->BytesPerSample - 1; break;
  }

  /* Vector location will contain marks for the samples to modify */
  location = malloc(WAV->ChannelLength * sizeof(char));
  if (location == NULL)   NoMemory();

  if (g_verbose) fputs("location of samples to modify\n",stderr);
```

```
mod_samples_cnt = 0;
if (dist) {
  /* Random samples */
  srand(seed_dist);
  for (i=first; i<WAV->ChannelLength; i=i+WAV->BytesPerSample) {
    location[i] = rand()%2;
    if (location[i]) mod_samples_cnt++;
  }
} else {
  if (mp3) {
    /* MP3 resistant samples */
    MWAV = init_WAV();
    MWAV->NumChannels = WAV->NumChannels;
    MWAV->ChannelLength = WAV->ChannelLength;
    init_channels(MWAV);
    compress_WAV(WAV,MWAV,ratio);
    for (i=first; i<WAV->ChannelLength; i=i+WAV->BytesPerSample) {
      location[i] = 1;
      for(k=0; k<WAV->NumChannels; k++) {
        for(j=-first; j<WAV->BytesPerSample-first; j++) {
          if (WAV->ChannelData[k][i+j] != MWAV->ChannelData[k][i+j]) {
            location[i] = 0;
          }
        }
      }
      if (location[i]) mod_samples_cnt++;
    }
  } else   {
     /* All samples */
    for (i=first; i<WAV->ChannelLength; i=i+WAV->BytesPerSample) {
      location[i] = 1;
      mod_samples_cnt++;
    }
  }
}
if (g_verbose)
  fprintf(stderr,"number of samples to modify: %d\n",mod_samples_cnt++);

if (g_verbose) fputs("encoding\n",stderr);
if (ecc)   {
  coded_mark_lng = mark_lng * 2;
  coded_mark = malloc(coded_mark_lng * sizeof(char));
  if (coded_mark == NULL)  NoMemory();
    encode_block(mark, coded_mark, mark_lng);
} else  {
  coded_mark_lng = mark_lng;
  coded_mark = mark;
}

if (g_verbose) fputs("encryption\n",stderr);
  srand(seed_encr);
  if (one_key)
    for (j=0; j<coded_mark_lng; j++)
      coded_mark[j] = coded_mark[j] ^ rand()%2;

if (g_verbose) fputs("insertion\n",stderr);
j=0;
if (blind)  {
  if (one_key)  {
    for (i=first; i<WAV->ChannelLength; i=i+WAV->BytesPerSample)
      if (location[i])  {
        value = coded_mark[j] << bit_level;
        for (k=0; k<WAV->NumChannels; k++)
          WAV->ChannelData[k][i] = ( WAV->ChannelData[k][i] &
                                          ~(1<<bit_level) ) ^ value;
          j=(j+1)%coded_mark_lng;
      }
  } else  {
    for (i=first; i<WAV->ChannelLength; i=i+WAV->BytesPerSample)
      if (location[i])  {
        value = coded_mark[j] << bit_level;
        value = value ^ ( rand()%2 << bit_level);
```

```
          for (k=0; k<WAV->NumChannels; k++)
            WAV->ChannelData[k][i] = ( WAV->ChannelData[k][i] &
                                          ~(1<<bit_level) ) ^ value;
            j=(j+1)%coded_mark_lng;
        }
    }
  } else  {
    if (one_key)  {
      for (i=first; i<WAV->ChannelLength; i=i+WAV->BytesPerSample)
        if (location[i])  {
          value = coded_mark[ j ] << bit_level;
          for (k=0; k<WAV->NumChannels; k++)
            WAV->ChannelData[k][i] = WAV->ChannelData[k][i] ^ value;
            j=(j+1)%coded_mark_lng;
        }
    } else  {
      for (i=first; i<WAV->ChannelLength; i=i+WAV->BytesPerSample)
        if (location[i]) {
          value = coded_mark[ j ] << bit_level;
          value = value ^ ( rand()%2 << bit_level);
          for (k=0; k<WAV->NumChannels; k++)
            WAV->ChannelData[k][i] = WAV->ChannelData[k][i] ^ value;
            j=(j+1)%coded_mark_lng;
        }
    }
  }
}
```

9.2.2 Algorisme de recuperació

```
/*-------------------------------------------------------------
  get_mark_from_WAV :
    Reconstruction of a mark embedded in a WAV file. Steps:
         location of modified samples
         statistical retrieval
         decoding
         decryption
 -------------------------------------------------------------*/
void get_mark_from_WAV(WAVFILE *WWAV, WAVFILE *OWAV, char *mark,
  int mark_lng, int seed_encr, short bit_level, short one_key,
  short dist, int seed_dist, short ecc, short mp3, int ratio)
{

  int i,j,k,first;
  int mod_samples_cnt;
  int half;
  int coded_mark_lng;
  char *coded_mark;
  char *location;
  char value;
  int *mark_ones_cnt;
  WAVFILE *MWAV;

  /* The LSbyte of each channel sample is the first one, supposing
    little endian or the last one, supposing big endian */
  switch(ENDIAN) {
    case 'L': first = 0; break;
    case 'B': first = WWAV->BytesPerSample - 1; break;
  }

  if (g_verbose) fputs("location of samples to modify\n",stderr);
  location = malloc(WWAV->ChannelLength * sizeof(char));
  if (location == NULL)  NoMemory();

  mod_samples_cnt = 0;
  if (dist){
    srand(seed_dist);
    for (i=first; i<WWAV->ChannelLength; i=i+WWAV->BytesPerSample){
      location[i] = rand()%2;
      if (location[i]) mod_samples_cnt++;
    }
  }else  {
    if (mp3)  {
```

```
    /* locate MP3 resistant samples */
      MWAV = init_WAV();
      MWAV->NumChannels = OWAV->NumChannels;
      MWAV->ChannelLength = OWAV->ChannelLength;
      init_channels(MWAV);
      compress_WAV(OWAV,MWAV,ratio);
      for (i=first; i<OWAV->ChannelLength; i=i+OWAV->BytesPerSample) {
        location[i] = 1;
        for(k=0; k<OWAV->NumChannels; k++)  {
          for(j=-first; j<OWAV->BytesPerSample-first; j++) {
            if (OWAV->ChannelData[k][i+j] != MWAV->ChannelData[k][i+j]) {
              location[i] = 0;
            }
          }
        }
        if (location[i]) mod_samples_cnt++;
      }
    } else  {
      for (i=first; i<WWAV->ChannelLength; i=i+WWAV->BytesPerSample) {
        location[i] = 1;
        mod_samples_cnt++;
      }
    }
  }
  if (g_verbose) fprintf(stderr,"number of modified samples: %d\n",mod_samples_cnt++);

  if (ecc) {
    coded_mark_lng = 2 * mark_lng;
    coded_mark = malloc(coded_mark_lng * sizeof(char));
    if (coded_mark == NULL)      NoMemory();
  } else  {
    coded_mark_lng = mark_lng;
    coded_mark = mark;
  }

  if (g_verbose) fputs("statistical retrieval...\n",stderr);
  srand(seed_encr);
  j=0;
  mark_ones_cnt = calloc(coded_mark_lng, sizeof(int));
  if (mark_ones_cnt == NULL)  NoMemory();
  if ( one_key )    {
    for (i=first; i<WWAV->ChannelLength; i=i+WWAV->BytesPerSample)
      if (location[i])  {
        value = 0x00;
        for(k=0; k<WWAV->NumChannels; k++)  {
          if ( ( ( ( ( WWAV->ChannelData[k][i] & (0x01<<bit_level) )
                    ^ ( OWAV->ChannelData[k][i] & (0x01<<bit_level) ) )
                     >> bit_level ) ^ value ) == 0x01)
            mark_ones_cnt[ j ]++;
        }
        j=(j+1)%coded_mark_lng;
      }
  } else  {
    for (i=first; i<WWAV->ChannelLength; i=i+WWAV->BytesPerSample)
      if (location[i])    {
        value = rand()%2;
        for(k=0; k<WWAV->NumChannels; k++) {
          if ( ( ( ( ( WWAV->ChannelData[k][i] & (0x01<<bit_level) )
                    ^ ( OWAV->ChannelData[k][i] & (0x01<<bit_level) ) )
                     >> bit_level ) ^ value ) == 0x01)
            mark_ones_cnt[ j ]++;
        }
        j=(j+1)%coded_mark_lng;
      }
  }

  /* Coded mark bit is the most frequent value in redundant locations */
  if (g_verbose) fputs("calculation\n",stderr);
  half = WWAV->NumChannels * mod_samples_cnt / ( coded_mark_lng * 2 );
  for (j=0; j<coded_mark_lng; j++)  {
    if ( mark_ones_cnt[j] > half)  {
      coded_mark[j] = 0x01;
    } else  {
```

```
      coded_mark[j] = 0x00;
    }
  }

  if (g_verbose) fputs("decryption\n",stderr);
  if ( one_key )    {
    for (j=0; j<coded_mark_lng; j++)
      coded_mark[j] = coded_mark[j] ^ rand()%2;
  }

  if (g_verbose) fputs("decoding\n",stderr);
  if (ecc)  {
    decode_block(coded_mark, mark, coded_mark_lng);
  } else  {
  }

  return;
}
```

9.2.3 Codi corrector d'errors

```
// HAMMING CODE.  Corrects 1 bit error.  This is a (7,4) code.

int CodeToData(unsigned char *code);
char *DataToCode(int data);

void CorrectCode(unsigned char *code);

// 7-bit Hamming code.
char HammingCodes[16][8] = {
       {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},
       {0x00,0x00,0x00,0x00,0x01,0x01,0x01,0x01},
       {0x00,0x00,0x00,0x01,0x00,0x00,0x01,0x01},
       {0x00,0x00,0x00,0x01,0x01,0x01,0x00,0x00},
       {0x00,0x00,0x01,0x00,0x00,0x01,0x00,0x01},
       {0x00,0x00,0x01,0x00,0x01,0x00,0x01,0x00},
       {0x00,0x00,0x01,0x01,0x00,0x01,0x01,0x00},
       {0x00,0x00,0x01,0x01,0x01,0x00,0x00,0x01},
       {0x00,0x01,0x00,0x00,0x00,0x01,0x01,0x00},
       {0x00,0x01,0x00,0x00,0x01,0x00,0x00,0x01},
       {0x00,0x01,0x00,0x01,0x00,0x01,0x00,0x01},
       {0x00,0x01,0x00,0x01,0x01,0x00,0x01,0x00},
       {0x00,0x01,0x01,0x00,0x00,0x00,0x01,0x01},
       {0x00,0x01,0x01,0x00,0x01,0x01,0x00,0x00},
       {0x00,0x01,0x01,0x01,0x00,0x00,0x00,0x00},
       {0x00,0x01,0x01,0x01,0x01,0x01,0x01,0x01}
};

/*-------------------------------------------------------------
  Return the data corresponding to a code vector
 -------------------------------------------------------------*/
int CodeToData(unsigned char *code)
{
       int i;

       for (i = 0; i < 16; i++) {
              if ( code[0] == HammingCodes[i][0] &&
                     code[1] == HammingCodes[i][1] &&
                     code[2] == HammingCodes[i][2] &&
                     code[3] == HammingCodes[i][3] &&
                     code[4] == HammingCodes[i][4] &&
                     code[5] == HammingCodes[i][5] &&
                     code[6] == HammingCodes[i][6] &&
                     code[7] == HammingCodes[i][7] )       {
                    // Got it!  Return the 4-bit data
                    return i;
              }
       }
       // Not a code!
       return -1;
}

/*-------------------------------------------------------------
```

```
  Return the code vector corresponding to a data
 ------------------------------------------------------------*/
char *DataToCode(int data)
{
	if (data >= 0 && data < 16) {
		return HammingCodes[data];
	}
	return 0;
}

/*------------------------------------------------------------
  Each 4 bytes are encoded in 8 bytes
 ------------------------------------------------------------*/
int EncodeBlock(char *inbuffer, char *outbuffer, int n)
{
	int i,j;
	char *poutbuffer;
	char *code;

	poutbuffer = outbuffer;
	for (i = 0; i < n; i = i+4) {
		code =
DataToCode(inbuffer[i]*8+inbuffer[i+1]*4+inbuffer[i+2]*2+inbuffer[i+3]);
		for (j=0; j<8;j++)
			poutbuffer[j] = code[j];
		poutbuffer = poutbuffer + 8;
	}
	return 0;
}

/*------------------------------------------------------------
  Try to correct errors in the code
 ------------------------------------------------------------*/
void CorrectCode(unsigned char *code)
{
	int n, i, mask;

	n = CodeToData(code);
	if (n == -1) {
		// Error!

		// Flip each bit and see if we can fix it!
		mask = 1;
		for (i = 0; i < 8; i++) {
			code[i] = code[i] ^ mask;
			n = CodeToData(code);
			if (n != -1) {
				// Corrected it!
				return;
			} else {
				code[i] = code[i] ^ mask;
			}
		}
		// Must not have worked...
		return;
	}
}

/*------------------------------------------------------------
  Each 8 bytes decode into 4 bytes
 ------------------------------------------------------------*/
int DecodeBlock(char *inbuffer, char *outbuffer, int n)
{
	int i;
	char *poutbuffer;

	int data;

	poutbuffer = outbuffer;
	for (i = 0; i < n; i = i+8) {
		data = CodeToData(inbuffer+i);
		if (data == -1) {
			// We got a problem.  Attempt to correct the code
```

```
                    CorrectCode(inbuffer+i);
                    data = CodeToData(inbuffer+i);
                    if (data == -1) {
                          data = 0;
                    } else {
//                        printf ("\n\rCorrected Code!");
                    }
              }

              poutbuffer[0] = (unsigned char)( (data >> 3 ) & 0x01);
              poutbuffer[1] = (unsigned char)( (data >> 2 ) & 0x01);
              poutbuffer[2] = (unsigned char)( (data >> 1 ) & 0x01);
              poutbuffer[3] = (unsigned char)( (data >> 0 ) & 0x01);
              poutbuffer = poutbuffer+4;
       }
       return 0;
}
```

9.2.4 Compressió/descompressió MP3

```
/*---------------------------------------------------------------
  We use functions from library libmp3lame, following the
  instructions on header file lame.h (lame-3.96.1.tar)
 ---------------------------------------------------------------*/
void compress_WAV(WAVFILE *WAV, WAVFILE *MWAV, int ratio)
{

  FILE *file;

  lame_global_flags *gf;
  unsigned char *mp3buf;
  int mp3buf_size;
  int mp3_lng;
  int rc;

  /* initialize encoder */
  gf=lame_init();

  /* Instead of calling lame_init_infile(), we set the values
  gf->samplerate, gf->num_channels, gf->num_samples from the wav file */
  if (g_verbose) fputs("lame set params\n", stderr);
  lame_set_in_samplerate(gf, (int)WAV->WAVHeader->SampleRate);
  lame_set_num_channels(gf, (int)WAV->NumChannels);
  lame_set_num_samples(gf, (unsigned long)WAV->Samples);

  MWAV->WAVHeader = WAV->WAVHeader;
  MWAV->WAVHeader->SampleRate = lame_get_in_samplerate(gf);
  MWAV->NumChannels = lame_get_num_channels(gf);
  MWAV->Samples = lame_get_num_samples(gf);
  MWAV->ChannelLength = WAV->ChannelLength;
  MWAV->BytesPerSample = WAV->BytesPerSample;

  /*
  internal algorithm selection.  True quality is determined by the bitrate
  but this variable will effect quality by selecting expensive or cheap algorithms.
  quality=0..9.  0=best (very slow).  9=worst.
  recommended:  2     near-best quality, not too slow
                5     good quality, fast
                7     ok quality, really fast
  */
//lame_set_quality(gf, 5);

  /* set one of brate compression ratio.  default is compression ratio of 11.  */
//lame_set_brate(gf, );
  lame_set_compression_ratio(gf, ratio);

  if (g_verbose) fputs("lame init params\n", stderr);
  /* Now that all the options are set, lame needs to analyze them and
      set some more options      */
    lame_init_params(gf);
```

```
  if (g_verbose) fputs("lame print config\n", stderr);
  /* Optional: print usefull information about options being used */
    lame_print_config(gf);

 /*
  * lame_encode_buffer handles all buffering, resampling and filtering for you.
  * input pcm data, output (maybe) mp3 frames.
  *
  * The required mp3buf_size can be computed from num_samples,
  * samplerate and encoding rate, but here is a worst case estimate:
  * mp3buf_size in bytes = 1.25*num_samples + 7200
  */

  mp3buf_size = 1.25 * WAV->Samples + 7200;
  mp3buf = (char *) malloc(mp3buf_size);
  if (mp3buf == NULL) NoMemory();

  if (g_verbose) fputs("lame encode buffer\n", stderr);

  rc = lame_encode_buffer(gf,(short int *)WAV->ChannelData[0],
       (short int *)WAV->ChannelData[1],WAV->Samples,mp3buf,mp3buf_size);

 /* Return code     number of bytes output in mp3buf. Can be 0
  *                 -1:  mp3buf was too small
  *                 -2:  malloc() problem
  *                 -3:  lame_init_params() not called
  *                 -4:  psycho acoustic problems
  */
  fprintf(stderr,"Bytes to mp3buf=%d, mp3buf_size=%d\n",rc,mp3buf_size);
  if (rc < 0)
    ErrorExit(ENCODING_MP3,"Error encoding into MP3 data\n");

  mp3_lng = rc;

 /* lame_encode_flush Will flush the intenal PCM buffers, padding with
  * 0's to make sure the final frame is complete, and then flush the
  * internal MP3 buffers, and thus may return a final few mp3 frames.
  * 'mp3buf' should be at least 7200 bytes long.
  * Will also write id3v1 tags (if any) into the bitstream
  */
  if (g_verbose) fputs("lame encode flush\n", stderr);

  rc = lame_encode_flush(gf, mp3buf, mp3buf_size);
  /* Return code = number of bytes output to mp3buf. Can be 0 */
  fprintf(stderr,"Bytes to mp3buf=%d, mp3buf_size=%d\n",rc,mp3buf_size);
  if (rc < 0)
    ErrorExit(ENCODING_MP3,"Error encoding into MP3 data\n");

  mp3_lng = mp3_lng + rc;

  /* Final call to free all remaining buffers */
  if (g_verbose) fputs("lame close\n", stderr);
  lame_close(gf);

#if 0
  /* Test MP3 file */
  file = fopen("file.mp3", "w+b");
  if ( (file == NULL) )   {
    ErrorExit(OPEN_ERROR, "Error opening MP3 file.\n");
  }
  else {
    rc = fwrite(mp3buf,1,mp3_lng,file);
    fprintf(stderr,"Bytes to file=%d\n",rc);
    fclose(file);
  }
#endif

  /* Initialize decoder */
  if (g_verbose) fputs("lame decode init\n", stderr);
  lame_decode_init();

 /*********************************************************************
```

```
  * Decode mp3 dada: input 1 mp3 frame, output (maybe) pcm data.
  * input:
  *    len          :  number of bytes of mp3 data in mp3buf
  *    mp3buf[len]  :  mp3 data to be decoded
  *****************************************************************/
  if (g_verbose) fputs("lame decode\n", stderr);
  rc = lame_decode(mp3buf, mp3_lng, (short int *)MWAV->ChannelData[0],
       (short int *)MWAV->ChannelData[1]);

 /* Return code:
  *    nout:  -1    : decoding error
  *            0    : need more data before we can complete the decode
  *           >0    : returned 'nout' samples worth of data in pcm_l,pcm_r
  *    pcm_l[nout]  : left channel data
  *    pcm_r[nout]  : right channel data
  */
  fprintf(stderr,"Bytes decoded=%d, mp3_lng=%d\n",rc,mp3_lng);
  if (rc < 0)
    ErrorExit(DECODING_MP3,"Error decoding MP3 data\n");

#if 0
  /* Test new WAV file */
  file = fopen("file.wav", "w+b");
  if ( (file == NULL) )   {
    ErrorExit(OPEN_ERROR, "Error opening WAV file.\n");
  }
  else {
    write_WAV(MWAV,file);
    fclose(file);
  }
#endif

  /* Cleanup call to exit decoder */
  if (g_verbose) fputs("lame decode exit\n", stderr);
  lame_decode_exit();

}
```

9.3 Guia d'estil

Versió editada de la LAME STYLEGUIDE, versió al mateix temps de la Linux Kernel Style Guide:

9.3.1 Punt 1: Indentació

Fer-la consistent, de 4 posicions. No utilitzar tabuladors (*).

9.3.2 Punt 2: Col·locar {}

Hi ha poques raons tècniques per a triar una estratègia o una altra, però la manera preferida, segons Kernighan i Ritchie, és posar la clau que obre al final de línia i la clau que tanca al principi, en una línia buida excepte quan continua la mateixa instrucció. Exceptuem el cas especial de les funcions (no es poden aniuar), que tindran la clau que obre al principi de la següent línia.

```
if (x is true) {          if (x == y) {          do {                    int function(int x)
    we do y                   ...                    body of do-loop     {
}                         } else {               } while (condition);        body of function
                          }                                                }
```

Aquesta estratègia minimitza el nombre de línies quasi buides, sense perdre llegibilitat. D'aquesta manera tenim més espai pels comentaris.

9.3.3 Punt 3: Nomenclatura

C és un llenguatge auster, per això la nomenclatura també ho hauria de ser. Cal triar noms fàcils d'escriure i d'entendre. Cal que els noms de les funcions i de les variables, sobretot les globals (que s'usaran només

si és estrictament necessari), siguin descriptius. Les variables locals han de tenir noms curts, però triats de manera que no confonguin.

9.3.4 Punt 4: Funcions

Cal documentar les funcions. Cal mantenir-les tan modulars com sigui possible, sense que sigui necessari posar limitacions artificials com la del nombre de línies.

9.3.5 Punt 5: Comentaris

Els comentaris són bons, però no cal explicar com funciona el codi; és millor escriure bé el codi de manera que el funcionament sigui obvi. Generalment voldrem que els comentaris expliquin què fa el codi en comptes de com. Els comentaris dins de les funcions no han de ser excessius, només els mínims, aquells que adverteixin d'algun punt particularment complex. És millor explicar en la capçalera de la funció què és el que fa i, si cal, el perquè.

9.4 Interfície per a codificar amb lame

1. (opcional) Obtenir el nombre de la versió del codificador, si ens interessa:

```
void get_lame_version(char *strbuf, size_t buflen, const char *prefix);
```

2. Establir el gestor de missatges d'error. Per defecte, LAME escriurà els missatges d'error a stderr utilitzant la funció vfprintf(). Això podria ser un problema per a les aplicacions GUI, per exemple, i llavors convindria assignar els nostres propis gestors de missatges: lame_set_errorf(gfp,error_handler_function); lame_set_debugf(gfp,error_handler_function); lame_set_msgf(gfp,error_handler_function);

3. Iniciar el codificador, posant tots els paràmetres al seu valor per defecte:

```
#include "lame.h"

lame_global_flags *gfp;

gfp = lame_init();
```

Per defecte (si no inicialitzem explícitament cap paràmetre) obtindrem un fitxer MP3 J-Stereo, 44.1khz, 128kbps CBR amb qualitat 5. Podem sobreescriure diversos paràmetres segons les nostres necessitats, per exemple:

```
lame_set_num_channels(gfp,2);

lame_set_in_samplerate(gfp,44100);

lame_set_brate(gfp,128);

lame_set_mode(gfp,1);

lame_set_quality(gfp,2);   /* 2=high  5 = medium  7=low */
```

A "lame.h" podem trobar la llista completa d'opcions (funcions lame_set_*()), però no totes estan documentades. Aquestes són experimentals, només per fer proves. Podrien ser esborrades en el futur.

4. Cal calcular més paràmetres de configuració interna que es basen en les dades anteriorment proporcionades, i també verificar que no hi hagi problemes. Cal comprovar que ret_code >= 0.

```
ret_code = lame_init_params(gfp);
```

5. Codificar algunes dades. A l'entrada tenim dades PCM, a la sortida obtindrem (suposadament) frames MP3. La següent rutina gestiona tot el buffering, el mostreig i el filtrat.

```
int lame_encode_buffer(lame_global_flags *gfp, short int leftpcm[], short int rightpcm[], int num_samples, char *mp3buffer,int mp3buffer_size);
```

A partir del nombre de mostres PCM en cada canal (num_samples, no és la suma del canal L i el R) i de la velocitat de mostreig i la taxa de codificació es pot calcular el paràmetre mp3buffer_size que es demana. Però també es pot fer una estimació més senzilla, del pitjor cas:

mp3buffer_size (en bytes) = 1.25*num_samples + 7200

La funció retorna el nombre de bytes que s'obtenen en el mp3buffer. Pot ser 0. Si és <0, vol dir que hi ha hagut algun error.

A banda d'aquesta, hi ha altres rutines per a diferents tipus de dades d'entrada (float, long, interleaved, etc). Per més detalls es pot consultar l'include "lame.h".

6. La següent funció buidarà els buffers i pot retornar alguns frames MP3 finals:

int lame_encode_flush(lame_global_flags *,char *mp3buffer, int mp3buffer_size);

El paràmetre mp3buffer hauria de ser almenys de 7200 bytes.

La funció retorna el nombre de bytes que s'obtenen al mp3buffer. Pot ser 0.

7. Escriure el tag Xing VBR/INFO al fitxer MP3:

void lame_mp3_tags_fid(lame_global_flags *,FILE* fid);

Això afegeix un frame MP3 vàlid que conté informació sobre el contingut que pot ser útil per a alguns usuaris. S'utilitza per CBR, ABR i VBR. La rutina provarà de rebobinar la cadena de sortida fins al principi. If això no és possible, (per exemple, si estem codificant i fent la sortida directament) s'ha de desactivar específicament el tag cridant la següent funció en el pas 3:

lame_set_bWriteVbrTag(gfp,0)

I després cridant lame_mp3_tags_fid() amb fid=NULL.

Si el rebobinat falla i no s'havia deshabilitat el tag, el primer frame mp3 de la cadena serà tot 0's.

8. Alliberar les estructures internes de dades:

void lame_close(lame_global_flags *);

9.5 Eines

- Implementem en llenguatge C els algorismes d'inserció i recuperació de la marca. Emprem l'entorn KDevelop com a eina de desenvolupament.
- Utilitzem el programa KHexEdit 0.8.5 (Espen Sand) per a tractar fitxers en format hexadecimal. Ens és útil per a crear els fitxers que contenen la marca i per a comparar aquests amb els resultats de l'extracció.
- Utilitzem StirMark for Audio com a programari d'avaluació de la capacitat i robustesa d'una marca sobre un fitxer àudio (Unzign, CheckMark o OptiMark)
- Utilitzem Lame v 3.96.1 com a compressor MP3 per avaluar la robustesa de les marques a aquest tipus de compressió.
- Creem un petit programa (count) per fer el càlcul del percentatge de bits erronis en les marques extretes. Simplement fa un recompte dels bits diferents als de la marca inserida i divideix pel nombre total de bits de la marca.
- Avaluem de manera subjectiva la perceptibilitat de les modificacions fetes sobre un fitxer àudio utilitzant la puntuació ITUR Rec 500 (5=imperceptible, 4=perceptible però no molest, 3=lleugerament molest, 2=molest, 1=molt molest).
- Utilitzem també la mètrica de distorsió quantitativa PSNR per a avaluar de manera objectiva el soroll introduït amb la inserció d'una marca (altres possibilitats eren MSE, SNR i NCC).

9.6 Planificació temporal

Les tasques a realitzar s'han distribuït en el temps seguint les indicacions del consultor:

- Primera setmana de novembre de 2004: Sistema bàsic (watermarking LSB en format WAV)
- Darrera setmana de novembre de 2004: Primera millora (ús de codis correctors)
- Darrera setmana de desembre de 2004: Segona millora (ús de la compressió MP3) i prememòria

Cadascun dels lliuraments anteriors representa una feina que pot descompondre en:

- cerca d'informació
- disseny de l'aplicació: processos, interfície, ...
- implementació partint del disseny a alt nivell
- avaluació i conclusions
- redacció de la part corresponent de la memòria