# INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE INGENIERÍA MECÁNICA Y ELÉCTRICA
SECCIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN
UNIDAD ZACATENCO

# "MARCAS DE AGUA PARA ARCHIVOS EN FORMATO WAV"

**TESIS**
**Que para obtener el grado de:**
**MAESTRO EN CIENCIAS EN INGENIERÍA DE TELECOMUNICACIONES**

**PRESENTA:**
**ING. RENÉ MORENO TÉLLEZ**

**ASESOR:**
**M.C. MARCO ANTONIO ACEVEDO MOSQUEDA**

**México D.F.** **2005**

# Contenido

**CAPÍTULO 3 PRUEBAS REALIZADAS**



**CAPÍTULO 4 CONCLUSIONES**

**Índice de tablas**

**Capítulo 1**



**Capítulo 2**

## ÍNDICE DE FIGURAS

## LISTA DE SÍMBOLOS

| | |
|---|---|
| *a, b, c* | Elementos del campo matemático *F* |
| $C_1, C_2$ | Par de cifras significativas que se van a modificar de *M1 y M2* |
| $C_{11}, C_{22}$ | Nuevos valores de las cifras significativas $C_1$ y $C_2$ |
| *Cla* | Valores ASCII de la clave utilizada |
| *i, j, k, n* | Variables enteras |
| *e(x)* | Patrón de error |
| *F* | Campo matemático |
| *g(x)* | Polinomio generador de código de grado *n-k* |
| *G* | Matriz generadora de código |
| *GF(q)* | Campo de Galois finito |
| *In* | Número de bits que se van a ocultar |
| *Intervalo* | Vector que contiene las posiciones de las muestras disponibles |
| $I_1[n]$, $I_2[n]$ | Vectores que contienen las posiciones de las muestras para ocultar |
| *m* | Longitud de la clave utilizada |
| *Muestras* | Cantidad de muestras necesarias para ocultar los bits |
| *M1,M2* | Par de muestras elegidas por los índices $I_1[n]$, $I_2[n]$ |
| *P* | Promedio obtenido de $C_1, C_2$ |
| $Po_1$ | Posición inicial a partir de donde se oculta la información |
| $Pos_{vector}$ | Posición donde se localiza el vector de referencia $Vector_{ref}$ |
| *r(x)* | Vector de palabra de código recibida en el decodificador |
| *Separación* | Espacio entre cada muestra que se elige con respecto a la siguiente |
| *Si* | Símbolo del síndrome para localizar errores |
| $Vector_{ref}$ | Vector de valores en dominio wavelet para localizar la información |
| *v(x)* | Polinomio de mensaje que se va a codificar |
| $X_k$ | Localizadores de error |
| *y* | Tamaño del intervalo que se utiliza |
| $\sigma(x)$ | Polinomio localizador de errores |

## Resumen

La marca de agua es una técnica para ocultar una cantidad de información en archivos de vídeo, imagen o música. De esta forma se puede autentificar los derechos de autor o los cambios realizados no autorizados a la información original. Utilizando esta técnica no se prohíbe al usuario hacer uso de los archivos, pero permite demostrar la propiedad de la información ocultando ciertos datos en el trabajo original.

En esta tesis se presentan dos alternativas para introducir marcas de agua en archivos de audio en formato wav. La primera es el algoritmo Patchwork el cual es ampliamente conocido y la segunda es un método propuesto.

El método propuesto utiliza la cuantización de los archivos de audio en el dominio wavelet para introducir la marca de agua. Además, se emplea el código BCH para codificar la información de la marca de agua para evitar pérdida de información y crear marcas de agua robustas.

Las dos técnicas empleadas se implementan en Matlab. Se realizan pruebas de fidelidad y robustez para ambos métodos. Además, se utiliza un convertidor de formato de audio comercial, para demostrar cual de las dos técnicas es mas robusta contra ataques de cambio de formato.

## Abstract

We can hide information in music, video or image using a watermarking technique. In this way we know if the original information was modified without authorization or authenticate the copyrights. Using this technique the users can use the information, but hiding some data in the original work we can show the intellectual property.

This work shows two watermarking techniques in audio files. The first technique in the Patchwork algorithm and the second is a new method.

The new method uses the audio files quantization in the wavelet domain to introduce the watermarking. And to avoid loss of information and make robust watermarking we use the BCH code.

We implement both techniques in Matlab. Robustness and fidelity proves were made with both methods. To show which one of both techniques is more robustness; we use a commercial convert audio files.

## Objetivo General

Desarrollar un método para introducir marcas de agua robustas en archivos de audio en formato wav.

## Objetivos específicos.

- Desarrollar e implementar el algoritmo Patchwork para introducir marcas de agua en archivos con formato wav codificados a 8 ó 16 bits en el dominio wavelet.
- Desarrollar e implementar un método propuesto para introducir marcas de agua en archivos con formato wav codificados a 8 ó 16 bits en el dominio wavelet.
- Analizar el comportamiento de ambos algoritmos al usar un convertidor de formato comercial para cambios de codificación, formato CDA y formato MP3.

## Justificación

Las marcas de agua en los últimos de años han adquirido gran importancia debido al surgimiento de los medio digitales y a la Internet los cuales permiten tener acceso a todo tipo de información digital como imágenes, videos o música, toda esta información puede ser guardada y usada como se quiera sin el consentimiento del autor. Debido a este problema las marcas de agua surgen como una opción para proteger la información ya que permiten autentificar un trabajo y de esta forma poder proteger los derechos de autor.

Aunque existen diferentes algoritmos para introducir las marcas de agua, casi todos estos algoritmos están enfocados a imágenes, además casi en su totalidad usan la transformada de Fourier o Coseno Discreto. Teniendo en cuenta el problema de autentificación se decidió desarrollar un algoritmo que permita introducir marcas de agua en archivos de audio con formato wav y que sea capaz de resistir diferentes procesamientos y cambios de formato a los que son sometidos estos archivos. Uno de los métodos más utilizados es el algoritmo Patchwork. Esta técnica modifica la tercera cifra significativa de las muestras de los archivos para ocultar la marca de agua en el dominio wavelet. Basado en esta técnica se propone un nuevo método utilizando la cuantización de los archivos y la transformada wavelet. Se decidió utilizar la transformada wavelet Daubechies ya que las características de esta transformada permite manipular los archivos de una forma distinta dividiendo la señal de audio en dos señales una de frecuencias bajas y otra de frecuencias altas, para ocultar la marca y que esta sea capaz de resistir los diferentes ataques, como cambios de formato, y pueda ser recuperada en su totalidad.

# Introducción

En esta tesis se explica la implementación del algoritmo Patchwork y un método propuesto para introducir marcas de agua robustas en archivos de audio en formato wav. En el capítulo uno se muestra una introducción a la teoría de wavelets, al código BCH y algunas características de los archivos de audio cuando cambian de formato. Cuando a una archivo de audio se le cambia de formato, los dos principales problemas que se tienen son el escalamiento y la pérdida de muestras. Estos dos problemas son los que se solucionan en este trabajo de tesis.

En el capítulo 2 se presentan las dos técnicas para ocultar la marca de agua. Para el algoritmo Patchwork se muestran algunas pruebas donde se demuestra que no es conveniente ocultar la marca de agua en la tercera cifra. Y lo más conveniente es utilizar la segunda cifra para archivos en formato wav. Sin embargo, la marca de agua no se recupera cuando se realizan cambios de formato. También se propone un método para que la marca de agua sea más robusta.

Las pruebas se realizan en el capítulo 3. Las principales pruebas son cambio en la cuantización de los archivos, cambios en la frecuencia de muestreo, pérdida y escalamiento de muestras, compresión y cambios de formato CDA y MP3. Es este capítulo se demuestra que el algoritmo Patchwork no es eficiente para todas las pruebas. Mientras que el método propuesto recupera la marca de agua sin ningún problema.

Finalmente se muestran las  conclusiones de este trabajo así como el trabajo a futuro en el capítulo 4.

# Capítulo 1

## CONCEPTOS BÁSICOS

## 1.1 ESTADO DEL ARTE

Las marcas de agua en audio son un tema relativamente nuevo en nuestro país, ya que al realizar una investigación acerca de las diferentes técnicas existentes sobre patentes en este tema, los resultados indican que son prácticamente inexistentes, como se muestra en la tabla 1.1.

| **Estrategias** | **MÉXICO** | | |
|---|---|---|---|
| | Pat | Sol | Dis |
| Marca agua audio | 1 | 8 | 0 |

**Tabla 1.1** Cuadro de resultados

Sol: Solicitudes de patentes
Pat: Patentes concedidas
Dis: Diseños Industriales

La única patente en México es de KONINKLIJKE PHILIPS ELECTRONICS N.V., las solicitudes se muestran en la tabla 1.2.

| **Solicitante** | **Número de solicitudes** |
|---|---|
| KONINKLIJKE PHILIPS ELECTRONICS N.V.. | 4 |
| MATSUSHITA ELECTRIC INDUSTRIAL CO.,LTD. | 2 |
| MACROVISON CORPORATION | 2 |

**Tabla 1.2** Solicitudes de patentes

De estos resultados es claro que en México no es muy conocido el tema de la marca de agua como protección para los derechos de autor. También existe otro algoritmo bastante utilizado para insertar marcas de agua en audio, el algoritmo Patchwork, aunque inicialmente se creo para imágenes este algoritmo ha sufrido cambios para que se pueda utilizar en audio, y se pueda aplicar en diferentes dominios, como el dominio wavelet [10] o en dominio coseno [6].

En esta tesis se propone un método diferente para introducir marcas de agua en archivos de audio en formato wav, en dominio wavelet y utilizando los códigos BCH como protección contra posibles errores, este método es robusto contra cambios de formato y cambios de codificación En las siguientes secciones se explican cada una de las diferentes herramientas que se utilizan a lo largo de la tesis para lograr introducir la información en el archivo y poder recuperar la marca correctamente después de aplicar un convertidor de formato.

## 1.2 TEORÍA DE LA WAVELET

Una wavelet es una transformada que divide una señal *x[n]* en dos señales, una señal de frecuencias bajas *u* y una señal de frecuencias altas *v*, cada señal resultante tiene la mitad de la longitud de *x[n],* otra característica importante es que la señal *u* contiene la mayor cantidad de energía, comparada con la señal *v*, además al aplicar la wavelet se realizan desplazamientos de dos elementos con la señal *x[n]* como se muestra en el siguiente ejemplo [2].

*x[n]=[4 6 10 12 8 6 5 5]*

Para obtener la señal de frecuencias bajas *u* usando la Daubechies uno y ocupando sus coeficientes para frecuencias bajas *h,* mas adelante se explica la obtención de estos coeficientes. Se realiza lo siguiente.

$$h = \left( \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}} \right)$$

Para el primer elemento $u_1$ se realiza el producto punto de los dos primeros valores de *x[n]* con *h.*

$$u_1 = \left( \frac{4}{\sqrt{2}} + \frac{6}{\sqrt{2}} \right) = 7.071$$

Para el segundo elemento $u_2$ se desplazan dos unidades y se obtiene el siguiente valor.

$$u_2 = \left( \frac{10}{\sqrt{2}} + \frac{12}{\sqrt{2}} \right) = 15.556$$

Para el tercer elemento $u_3$ se desplazan dos unidades.

$$u_3 = \left( \frac{8}{\sqrt{2}} + \frac{6}{\sqrt{2}} \right) = 9.899$$

Para el cuarto elemento $u_4$ se desplazan dos unidades.

$$u_4 = \left( \frac{5}{\sqrt{2}} + \frac{5}{\sqrt{2}} \right) = 7.071$$

De esta forma se obtienen los elementos de *u,* hasta *u=N/2* (*N* es la longitud del vector) y la señal completa para las frecuencias bajas es:

*u = [7.071 15.556 9.899 7.071]*

Para la señal de frecuencias altas *v* se realiza el mismo procedimiento con los coeficientes para frecuencias altas *g* que se muestran a continuación.

$$g = \left( \frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}} \right)$$

La señal de frecuencias altas es la siguiente.

*v =[ -1.414  -1.414  1.414  0]*

La energía del vector *x* es de  446, la señal de frecuencias bajas *u* contiene la mayor cantidad de energía la cual es 440 mientras que la señal *v* de frecuencias altas contiene solo 6. Para el caso de la Daubechies dos usando el mismo vector se realiza lo siguiente.

Primero usando los coeficientes de frecuencias bajas para obtener *u* y realizando el producto punto por los coeficientes *h* y desplazándose de dos en dos. Los coeficientes *h* se muestran a continuación.

*h=[0.4829 0.8365 0.2241 –0.1294]*

Para $u_1$

$u_1$*=(0.4829 0.8365 0.2241 –0.1294)·(4 6 10 12)=7.6388*

Para $u_2$

$u_2$*=(0.4829 0.8365 0.2241 –0.1294)·(10 12 8 6)=15.8834*

Para $u_3$

$u_3$*=(0.4829 0.8365 0.2241 –0.1294)·(8 6 5 5)=9.3557*

Para $u_4$ es claro que los dos últimos elementos de *h* no tendrían valor con respecto a *x* para solucionar este problema se realiza un corrimiento de los dos primeros valores iniciales al final del vector para completar el producto punto, como se muestra a continuación.

$u_4$*=(0.4829 0.8365 0.2241 –0.1294)·(5 5 4 6)=6.717*

*u=[7.6388 15.8834 9.3557 6.717]*

Para encontrar *v* se realiza el mismo procedimiento y utilizando sus coeficientes de frecuencias altas *g*.

*g=[-0.1294 –0.2241 0.8365 –0.4829]*

Los valores de *v* son los siguientes.

*$v_1$=(-0.1294 –0.2241 0.8365 –0.4829)·(4 6 10 12)=0.708*

*$v_2$=(-0.1294 –0.2241 0.8365 –0.4829)·(10 12 8 6)=-0.1886*

*$v_3$=(-0.1294 –0.2241 0.8365 –0.4829)·(8 6 5 5)=-0.6118*

*$v_4$=(-0.1294 –0.2241 0.8365 –0.4829)·(5 5 4 6)=-1.389*

*v=[0.708 –0.1886 –0.6118 –1.389]*

La energía de la señal *u* es de 443 y la energía de la señal *v* es de 3. Entre más coeficientes contenga la wavelet se logra una mayor compactación de energía de la señal esto es importante cuando el objetivo es la compresión pero cuando se desea conocer mejor las características de la señal es mejor usar una wavelet con pocos coeficientes, en esta tesis el objetivo es ocultar información por lo tanto se decide usar la señal de frecuencias bajas *u* por que contiene la mayor cantidad de energía. La forma de obtener los coeficientes wavelet se muestra a continuación.

Suponga una señal continua que es muestreada para crear su versión discreta [1,3]:

$$x(n):\ n = 0,1,2,... \tag{1.1}$$

Cuya transformada de Fourier es más fácil de manejar por la transformada *z*:

$$X(z) = X\left(e^{i2\pi f}\right) = X\left(e^{j\omega}\right) = \sum_n x(n) z^{-n} \tag{1.2}$$

donde $\omega = 2\pi f$ y $z = e^{i\omega} = e^{i2\pi f}$

Si una señal continua tiene una banda bien limitada, tal como $f_0=1/2$ *Hz*, utilizando el criterio de Nyquist. Se debe muestrear a $F_T = 1$ *Hz*. La frecuencia angular de muestreo se define como $\Omega_T$:

$$\Omega_T = \omega_{saq} = 2\pi F_T = 2\pi \tag{1.3}$$

Suponiendo que la señal *x(n)* pasa por un filtro pasa bajas *H(z)* con una respuesta en magnitud como lo muestra la figura 1.1, con ancho de banda de *0* a $\pi/2$. De forma ideal:

$$\left|H(e^{j\omega})\right| = \begin{cases} 1 & 0 \le \omega \le \pi/2 \\ 0 & \pi/2 < \omega \le \pi \end{cases} \tag{1.4}$$

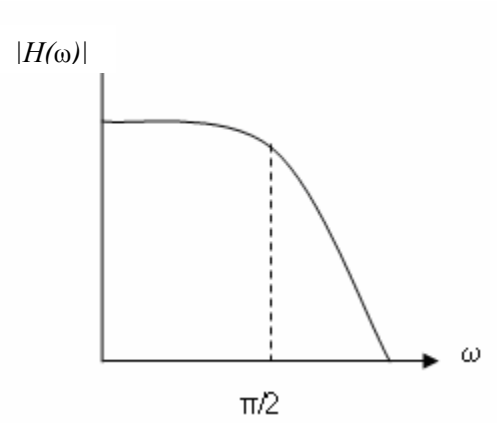



**Figura 1.1** Respuesta del filtro pasa bajas *H(z)*

En la figura 1.2 se muestra el filtro pasa bajas, *x(n)* es la señal discreta original, *X(z)* es la transformada *z* de *x(n)*, la respuesta al impulso *h(n)* y *H(z)* la función de transferencia. La salida del filtro de la señal discreta es *u'(n)* y la transformada z de la señal de salida es:

$$U'(z) = H(z)X(z) \tag{1.5}$$

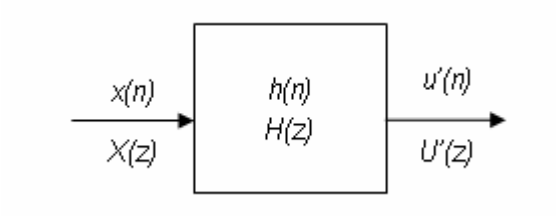



**Figura 1.2** Diagrama del filtro pasa bajas *H(z)*

En la figura 1.3 se muestra la respuesta en magnitud del filtro pasa altas *G(z)* con un ancho de banda de $\pi/2$ a $\pi$. De forma ideal:

$$\left|G\left(e^{j\omega}\right)\right| = \begin{cases} 0 & 0 \le \omega \langle \pi/2 \\ 1 & \pi/2 \le \omega \le \pi \end{cases} \tag{1.6}$$

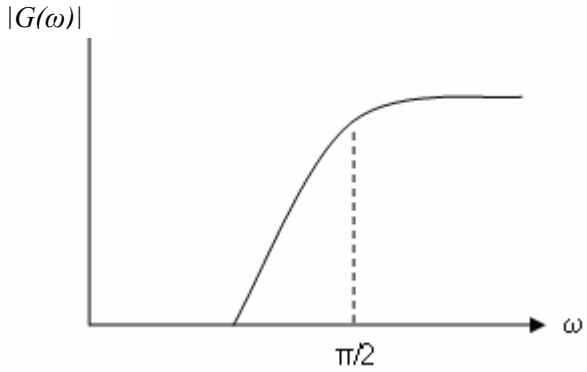



**Figura 1.3** Respuesta del filtro pasa altas *G(z)*

En la figura 1.4 se muestra que la señal original *x(n)* se hace pasar por un filtro pasa altas cuya respuesta al impulso es *g(n)*. La salida del filtro *V'(z)* es igual a la multiplicación de la función de transferencia *G(z)* y *X(z)*:

$$V'(z) = G(z)X(z) \tag{1.7}$$


**Figura 1.4** Diagrama de la respuesta del filtro pasa altas *G(z)* la señal de entrada *X(z)*

Las nuevas señales *v'(n)* y *u'(n)* también son señales de banda limitada y su ancho de banda es menor que la de x*(n)*. Esta acción permite decimar la señal antes de procesarla. Esto es posible ya que *v'(n)* y *u'(n)* tienen un ancho de banda efectivo de *f = 1/4 Hz* o $\omega = \pi/2$ rad/muestra ya que han sido filtradas. Ahora, como el ancho de banda de las señales *v'(n)* y *u'(n)* se ha reducido por un factor de dos, estas señales pueden ser decimadas por dos sin ningún traslape. Esto equivale a reducir la velocidad de muestreo y el número de muestras, por lo que la señal se ha comprimido.

Para muchas señales la información más importante se encuentra en la señal *u(n)* de frecuencias bajas, mientras que en la señal *v(n)* de altas frecuencias se encuentran los detalles o matices de la señal *x(n)*. Por ejemplo, en el caso de la voz humana, si se elimina las componentes con altas frecuencias, la voz suena diferente pero se sigue entendiendo el mensaje. En cambio, si lo que se elimina son las componentes de bajas frecuencias, el mensaje se vuelve irreconocible. Por eso el análisis Wavelet permite descomponer la señal en aproximaciones y detalles, a éste proceso se le conoce con el nombre de análisis wavelet. En la figura 1.5 se aprecia el proceso de descomposición de la señal *x(n)* en altas y bajas frecuencias. Las señales que resultan después de pasar el proceso de diezmado por un factor de dos son:

$$\begin{aligned} U(z) &= \frac{1}{2}\left[U'\left(\sqrt{z}\right) + U'\left(-\sqrt{z}\right)\right] \\ V(z) &= \frac{1}{2}\left[V'\left(\sqrt{z}\right) + V'\left(-\sqrt{z}\right)\right] \end{aligned} \tag{1.8}$$

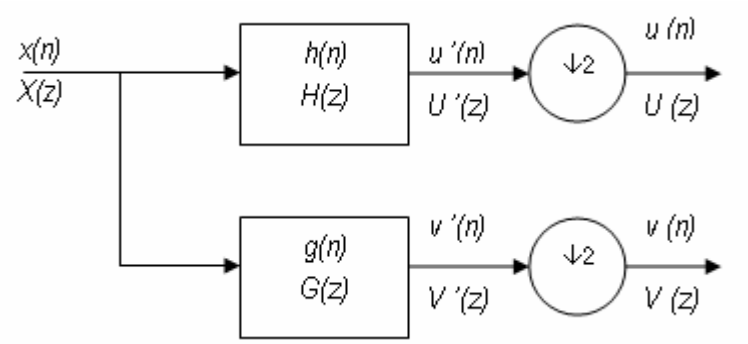



**Figura 1.5** Diagrama del proceso de la descomposición de la señal *x(n)* al aplicar una wavelet

Hasta ahora *u(n)* y *v(n)* tienen un ancho de banda mas pequeño que la señal original *x(n)*, por lo tanto puede ser muestreadas, cuantificadas, codificadas y transmitidas a una relación de bit mucho mayor que *x(n)* sin pérdidas en la información.

El problema ahora, es como reconstruir *x(n)*, a partir de *u(n)* y *v(n)* sin ninguna distorsión o traslape. Para hacer la reconstrucción se utiliza la expansión por un factor de 2 como se muestra en la figura 1.6.

$$U''(z) = U\left(z^2\right) = \frac{1}{2}\left[U'(z) + U'(-z)\right]$$
$$V''(z) = V\left(z^2\right) = \frac{1}{2}\left[V'(z) + V'(-z)\right] \qquad (1.9)$$

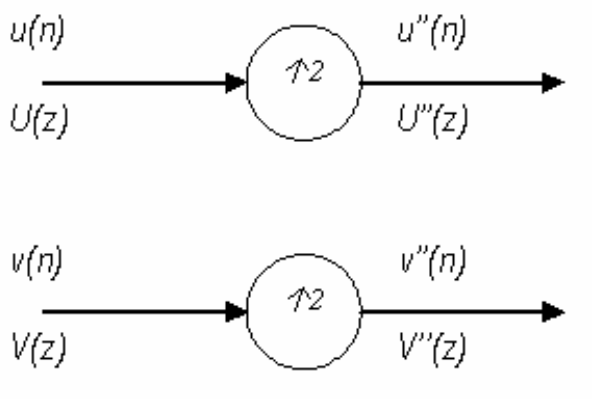



**Figura 1.6** Expansión de nuestras señales u(n) y v(n) para reconstruir nuestra señal original

En la figura 1.7 se muestra como las señales *u''(n)* y *v''(n)* pasan por los filtros pasa bajas *h'(n)* y pasa altas *g'(n)* respectivamente.

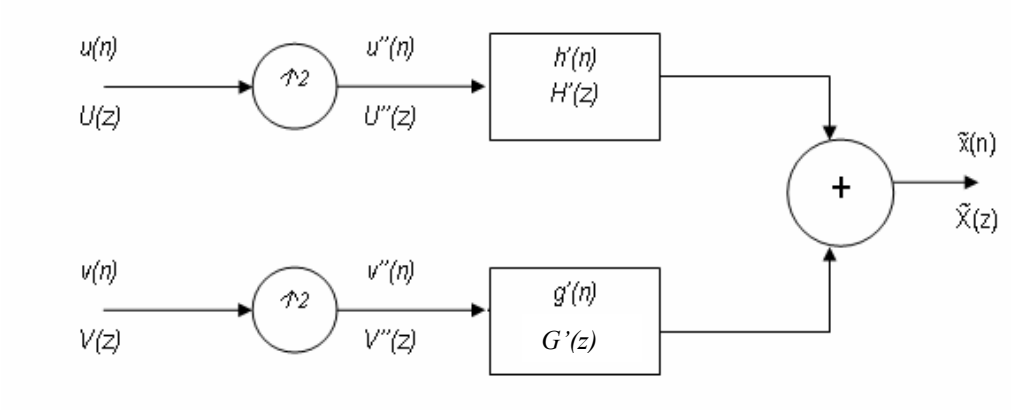



**Figura 1.7** Reconstrucción de la señal original

Por lo tanto

$$\begin{aligned}\tilde{X}(z) &= H'(z)U''(z) + G'(z)V''(z)\\ \tilde{X}(z) &= \frac{H'(z)}{2}\left[U'(z)+U'(-z)\right]+\frac{G'(z)}{2}\left[V'(z)+V'(-z)\right]\\ \tilde{X}(z) &= \frac{1}{2}\left[\left(U'(z)+U'(-z)\right)H'(z)+\left(V(z)+V'(-z)\right)G'(z)\right]\\ \tilde{X}(z) &= \frac{1}{2}\left[\left(H(z)X(z)+H(-z)X(-z)\right)H'(z)+\left(G(z)X(z)+G(-z)X(-z)\right)G'(z)\right]\end{aligned} \tag{1.10}$$

Factorizando *X(z)* y *X(-z)*.

$$\tilde{X}(z) = \frac{1}{2}\left[\left(H(z)H'(z)+G(z)G'(z)\right)X(z)+\left(H(-z)H'(z)+G(-z)G'(z)\right)X(-z)\right] \tag{1.11}$$

La señal procesada contiene la señal original *X(z)* y una señal traslapada *X(-z)*. Para obtener la señal original *X(z)* la ecuación 1.11 debe cumplir con:

$$H(-z)H'(z)+G(-z)G'(z)=0 \tag{1.12}$$

Para que la ecuación 1.12 sea igual a cero se puede elegir:

$$\begin{aligned}G(z) &= -z^{-N}H(-z^{-1}) = H'(-z)\\ G'(z) &= z^{-N}H'(-z^{-1}) = -H(-z)\end{aligned} \tag{1.13}$$

Sin perder generalidad y por conveniencia:

$$\begin{aligned}H'(z) &= z^{-N}H(z^{-1})\\ G'(z) &= z^{-N}G(z^{-1})\end{aligned} \tag{1.14}$$

El factor $z^{-N}$ garantiza la condición de causalidad de los filtros [esto es, *h'(n)=0* para *n<0* ]. De esta manera, sustituyendo la ecuación 1.13 y 1.14 en la ecuación 1.11:

$$\begin{aligned}\tilde{X}(z) &= \frac{1}{2}\left[H(z)H'(z)+G(z)G'(z)\right]X(z)\\ &= \frac{1}{2}\left[z^{-N}H(z)H(z^{-1})+z^{-N}G(z)G\left(z^{-1}\right)\right]X(z)\\ &= \frac{1}{2}\left[H(z)H(z^{-1})+H(-z^{-1})H(-z)\right]z^{-N}X(z)\end{aligned} \tag{1.15}$$

De la ecuación anterior se puede observar que:

$$H(z)H(z^{-1})+H(-z^{-1})H(-z)=2 \tag{1.16}$$

De esta manera se tendría una reconstrucción perfecta:

$$\tilde{X}(z)=z^{-N}X(z); \qquad \tilde{x}(n)=x(n-N) \tag{1.17}$$

La señal reconstruida esta desplazada *N* muestras. Se puede suponer que *H(z)* es un filtro FIR:

$$H(z)=h(0)+h(1)z^{-1}+\ldots+h(N)z^{-N} \tag{1.18}$$

De tal forma que la respuesta al impulso de los filtros *h'(n)*, *g(n)* y *g'(n)* esta dada por:

$$\begin{aligned}h'(n) &= h\left(N-n\right)\\ g(n) &= (-1)^n h(N-n)\\ g'(n) &= g\left(N-n\right)\end{aligned} \tag{1.19}$$

Los filtros *H(z)* y *G(z)* son llamados filtros espejo en cuadratura (QMF Quadrature Mirror Filters) debido a que tienen simetría alrededor del punto *π/2*, como se observa en la figura 1.8.

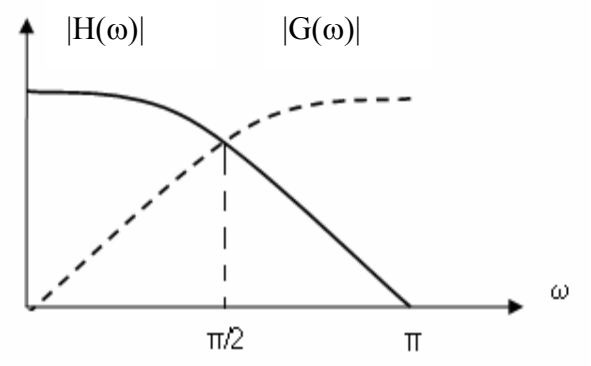



**Figura 1.8.** Respuesta en magnitud de los filtros espejo en cuadratura.

Se ha mencionado que se maneja filtros FIR, suponiendo un orden $N = 3$, sustituyendo en la ecuación 1.18

$$H(z) = \sum_{n=0}^{N=3} h(n)z^{-n} = h(0) + h(1)z^{-1} + h(2)z^{-2} + h(3)z^{-3} \tag{1.20}$$

Se pueden encontrar los coeficientes de los filtros *G(z)*, *G'(z)* y *H'(z)*, de las ecuaciones 1.13 y 1.14 se tiene:

$$\begin{aligned} G(z) &= -z^{-3}H(-z^{-1}) \\ &= -z^{-3}\left[h(0) - h(1)z^{+1} + h(2)z^{+2} + h(3)z^{+3}\right] \\ &= h(3) - h(2)z^{-1} + h(1)z^{-2} - h(0)z^{-3} \end{aligned} \tag{1.21}$$

$$\begin{aligned} G'(z) &= z^{-3}G(z^{-1}) = -H(-z) \\ &= -h(0) + h(1)z^{-1} - h(2)z^{-2} + h(3)z^{-3}, \end{aligned} \tag{1.22}$$

$$\begin{aligned} H'(z) &= z^{-N}H(z^{-1}) \\ &= h(3) + h(2)z^{-1} + h(1)z^{-2} + h(0)z^{-3} \end{aligned} \tag{1.23}$$

Por lo tanto, al encontrar los coeficientes del filtro *h(n)*, se pueden obtener la respuesta al impulso de los filtros *g(n)*, *g'(n)* y *h'(n)*. De la ecuación 1.16:

$$\begin{aligned} &\left[h(0) - h(1)z^{-1} + h(2)z^{-2} - h(3)z^{-3}\right] \times \left[h(0) - h(1)z + h(2)z^{2} - h(3)z^{3}\right] + \\ &\left[h(0) + h(1)z^{-1} + h(2)z^{-2} + h(3)z^{-3}\right] \times \left[h(0) + h(1)z + h(2)z^{2} + h(3)z^{3}\right] = 2 \end{aligned} \tag{1.24}$$

Desarrollando la ecuación 1.24 se tiene,

$$\begin{aligned} h^2(0) + h^2(1) + h^2(2) + h^2(3) &= 1 \quad \text{Para } z^0 \\ h(0)h(2) + h(1)h(3) &= 0 \quad \text{Para } z^2 \text{ y } z^{-2} \end{aligned} \tag{1.25}$$

Se necesitan dos ecuaciones mas para encontrar *H(z)*. Se sabe que *G(z)* es un filtro pasa altas, evaluando en $\omega=0$:

$$G\left(e^{j0}\right) = G(1) = 0 = H\left(e^{j\pi}\right) = H(-1) \tag{1.26}$$

de la ecuación 1.20;

$$
\begin{aligned}
H(z) &= h(0) + h(1)z^{-1} + h(2)z^{-2} + h(3)z^{-3} \\
H(-1) &= h(0) + h(1)\frac{1}{(-1)} + h(2)\frac{1}{(-1)^2} + h(3)\frac{1}{(-1)^3} \\
&= h(0) - h(1) + h(2) - h(3) \\
&= 0
\end{aligned}
\tag{1.27}
$$

Evaluando la ecuación 1.16 en ω = 0,

$$
\begin{aligned}
H(z)H(z^{-1}) + H(-z)H(-z^{-1}) &= 2 \\
H\left(e^{j\omega}\right)H\left(e^{-j\omega}\right) + H\left(-e^{j\omega}\right)H\left(-e^{-j\omega}\right) &= 2 \\
H(e^{j0})H\left(e^{-j0}\right) + H\left(-e^{j0}\right)H\left(-e^{-j0}\right) &= 2 \\
H(1)H(1) &= 2 \\
H(1)^2 &= 2 \\
H(1) = H\left(e^{j0}\right) &= \sqrt{2}
\end{aligned}
$$

$$
H(1) = h(0) + h(1) + h(2) + h(3) = \sqrt{2} \tag{1.28}
$$

De las ecuaciones 1.27 y 1.28 se puede observar que,

$$
h(0) + h(2) = \frac{1}{\sqrt{2}} \tag{1.29}
$$

$$
h(1) + h(3) = \frac{1}{\sqrt{2}} \tag{1.30}
$$

Las ecuaciones 1.25, 1.29 y 1.30 son linealmente dependientes, la pregunta es ¿Cómo encontrar la cuarta ecuación?, ya que sin esta cuarta ecuación no es posible encontrar una solución. Existen muchos métodos, en este caso se tomara $G^{*d}(1)=0$ [2], donde *d* es la *d*-ésima derivada de *G'* [1]. Igualando la derivada de *G'(1)* a cero.

$$
-0h(0) - 1h(1) + 2h(2) - 3h(3) = 0 \tag{1.31}
$$

Resolviendo el sistema de ecuaciones 1.25, 1.29, 1.30 y 1.31:

$$h(0) = \frac{1+\sqrt{3}}{4\sqrt{2}}$$
$$h(1) = \frac{3+\sqrt{3}}{4\sqrt{2}}$$
$$h(2) = \frac{3-\sqrt{3}}{4\sqrt{2}}$$
$$h(3) = \frac{1-\sqrt{3}}{4\sqrt{2}} \qquad (1.32)$$

Los coeficientes obtenidos corresponden a la wavelet Daubechies 4. Existen diferentes tipos de wavelet, la utilización de alguna wavelet en particular dependerá de la aplicación que se realice ya que entre mas elementos se usan la energía de la señal se compacta más. La compactación es útil cuando se desea comprimir. Si se usa un menor número de elementos permite conocer mejor las características de la señal que sé este usando. La forma de aplicar la wavelet es de una forma sencilla solo se realiza el producto punto de los coeficientes por los valores de una señal *x(n)* por ejemplo tenemos para la Daubechies 4.

Supóngase una señal discreta *x[n]* ,con un número *N* de muestras [2], donde *N* es un número par.

$$x[n] = (x_1, x_2, x_3, ..., x_N)$$

Para obtener la señal *u(n)* se ocupa los coeficientes encontrados en la ecuación 1.32:

$$h = (h_0, h_1, h_2, h_3)$$

De tal forma que el primer valor de *u(1)* es:

$$u(1) = x_1 h_0 + x_2 h_1 + x_3 h_2 + x_4 h_3$$

Para obtener el siguiente valor *u(2)* se realiza un desplazamiento de 2 unidades a la señal discreta *x[n]* y se hace repite procedimiento:

$$u(2) = x_3 h_0 + x_4 h_1 + x_5 h_2 + x_6 h_3$$

En forma general:

$$u(l) = x[n] \cdot h \quad donde \;\; l = 1,2,3,..., N/2 \qquad (1.33)$$

Así sé continua desplazándose de dos en dos unidades, hasta llegar al final de la señal al llegar al final de la señal ya no se tienen mas valores para los dos últimos coeficientes de la wavelet ya que se requieren *n+2* elementos de la señal original, lo que se debe hacer es un desplazamiento de los dos últimos coeficientes de la wavelet al inicio de la señal.

$$u(N/2) = x_{N-1}h_0 + x_N h_1 + x_1 h_2 + x_2 h_3 \tag{1.34}$$

De esta forma se aplica un nivel de wavelet a la señal original pero si se desea aplicar mas de un nivel a la señal lo que se hace es repetir el mismo procedimiento a la señal de frecuencias bajas *u(n)*. Esta señal tiene la mayor compactación de la energía que la señal *v(n)*.

Para obtener la señal *v(n)* se emplea la formula 1.19 y los coeficientes obtenidos son los siguientes:

$$\begin{aligned} g(0) &= \frac{1-\sqrt{3}}{4\sqrt{2}} \\ g(1) &= \frac{\sqrt{3}-3}{4\sqrt{2}} \\ g(2) &= \frac{3+\sqrt{3}}{4\sqrt{2}} \\ g(3) &= \frac{-1-\sqrt{3}}{4\sqrt{2}} \end{aligned} \tag{1.35}$$

Para obtener la señal *v(n)* se realiza el mismo procedimiento utilizado por la ecuación 1.33 pero usando los coeficientes *g*:

$$g = (g_1, g_2, g_3, g_4)$$

Y se repite el procedimiento anterior.

$$v(1) = x_1 g_0 + x_2 g_1 + x_3 g_2 + x_4 g_3$$

De la misma forma para *v(2)* se realiza un desplazamiento de 2 unidades a la señal discreta *x[n]* y se realiza lo mismo:

$$v(2) = x_3 g_0 + x_4 g_1 + x_5 g_2 + x_6 g_3$$

El desplazamiento al final se realiza de la misma forma que en la señal *u(n)*

$$v(N/2) = x_{N-1}g_0 + x_N g_1 + x_1 g_2 + x_2 g_3 \tag{1.36}$$

En forma general

$$v(l) = x \cdot g \quad donde \;\; l = 1,2,3,\ldots, N/2 \tag{1.37}$$

## 1.3 CÓDIGOS BCH (*n,k,t*)

Los códigos BCH(*n,k,t*) [4,5] son una subclase de códigos cíclicos ya que cada desplazamiento de un vector de código produce otro vector de código. En la tabla 1.3 se muestran algunos vectores código para el BCH(7,4,1) para ilustrar este desplazamiento, por ejemplo si el vector en color rojo desplaza su ultimo valor que es cero al inicio del vector produce otro vector de código que para este caso es el vector en color azul por esta razón se dice que cada desplazamiento de un vector código produce otro vector código.

| MENSAJE | VECTORES CÓDIGO |
|---|---|
| 0000 | 0000000 |
| 1000 | 1101000 |
| 0100 | 0110100 |
| 1100 | 1011100 |
| 0010 | 0011010 |

**Tabla 1.3** Algunos vectores código para el BCH(7,4,1)

Estos códigos BCH*(n,k,t)* son creados por un polinomio generador *g(x)* para construir una matriz generadora de código. El polinomio generador para un código especifico BCH(*n,k,t*) es un factor de $x^n+1$ donde:

*n = longitud del vector de código*
*k =longitud de la palabra a codificar*
*t = el número de errores que se pueden corregir*

El polinomio generador *g(x)* es un polinomio de grado (*n-k).*

$$g(x) = 1 + g_1 x + g_2 x^2 + \ldots + g_{n-k-1} x^{n-k-1} + x^{n-k} \tag{1.38}$$

La matriz generadora se construye de la siguiente forma:

$$G = \begin{vmatrix} g_0 & g_1 & g_2 & \ldots\ldots g_{n-k} & 0 & 0 & 0 & \ldots. & 0 \\ 0 & g_0 & g_1 & g_2 & \ldots\ldots & g_{n-k} & 0 & 0 & \ldots. & 0 \\ 0 & 0 & g_0 & g_1 & g_2 & \ldots\ldots & g_{n-k} & 0 & \ldots. & 0 \\ \cdot. & & & & & & & & & \\ . & & & & & & & & & \\ . & & & & & & & & & \\ 0 & 0 & \ldots\ldots\ldots & g_0 & g_1 & g_2 & \ldots\ldots\ldots & & & g_{n-k} \end{vmatrix}$$

Como un ejemplo, se usara el código BCH(15,7,2), se tiene que $n = 15$ entonces el polinomio generador es un factor de $x^{15+1}$.

Factorizar polinomios es complicado, además puede existir mas de un polinomio de grado ($n$-$k$). No todos los polinomios de grado ($n$-$k$) resultan ser eficientes en la creación de códigos, en la práctica ya existen polinomios generadores con un número determinado de errores para estos códigos para el caso del código BCH(15,7,2) el polinomio generador es el siguiente[4]:

$$g(x) = 1 + x^4 + x^6 + x^7 + x^8$$

La matriz generadora queda de la siguiente forma:

$$G = \begin{vmatrix} 1&0&0&0&1&0&1&1&1&0&0&0&0&0&0 \\ 0&1&0&0&0&1&0&1&1&1&0&0&0&0&0 \\ 0&0&1&0&0&0&1&0&1&1&1&0&0&0&0 \\ 0&0&0&1&0&0&0&1&0&1&1&1&0&0&0 \\ 0&0&0&0&1&0&0&0&1&0&1&1&1&0&0 \\ 0&0&0&0&0&1&0&0&0&1&0&1&1&1&0 \\ 0&0&0&0&0&0&1&0&0&0&1&0&1&1&1 \end{vmatrix}$$

Esta matriz generadora $G$ no esta en forma sistemática, la ventaja de una matriz sistemática es que los últimos $k$ dígitos de la derecha corresponden a la palabra que se va a codificar y los $n$-$k$ dígitos de la izquierda corresponden a los bits de verificación de paridad. Para convertir la matriz generadora en una matriz sistemática se realizan operaciones de suma y resta. La matriz generadora sistemática queda de la siguiente forma.

$$G = \begin{vmatrix} 1&0&0&0&1&0&1&1&1&0&0&0&0&0&0 \\ 1&1&0&0&1&1&1&0&0&1&0&0&0&0&0 \\ 0&1&1&0&0&1&1&1&0&0&1&0&0&0&0 \\ 1&0&1&1&1&0&0&0&0&0&0&1&0&0&0 \\ 0&1&0&1&1&1&0&0&0&0&0&0&1&0&0 \\ 0&0&1&0&1&1&1&0&0&0&0&0&0&1&0 \\ 0&0&0&1&0&1&1&1&0&0&0&0&0&0&1 \end{vmatrix}$$

Para convertir una palabra fuente en una palabra de código se multiplica cada palabra por cada una de las columnas de la matriz generadora. En esta tesis se utilizarán letras y números por esta razón se emplea el código BCH(15,7,2) ya que 7 bits resultan suficientes para desplegar todas las letras y números del código ASCII.

Se debe tener cuidado de que algunos caracteres al convertirse a binario solo ocupan 6 bits para resolver esto se le agrega un cero al final de la palabra para completar los 7 bits y poder usar correctamente el código. Se utilizará el valor binario del numero '9' para completar el ejemplo.

En el código ASCII él número 9 tiene asignado el valor de 57 y su valor en binario es el siguiente;

100111

Este valor solo ocupa 6 bits y se requieren 7 bits para completar la palabra para aplicar el código se le agrega un cero al final, este cero no afecta al carácter que se codifica y queda de la siguiente forma.

1001110

Después se procede a multiplicar este carácter por cada una de las columnas de la matriz generadora.

$$[1001110]\cdot\begin{vmatrix} 1&0&0&0&1&0&1&1&1&0&0&0&0&0&0\\ 1&1&0&0&1&1&1&0&0&1&0&0&0&0&0\\ 0&1&1&0&0&1&1&1&0&0&1&0&0&0&0\\ 1&0&1&1&1&0&0&0&0&0&0&1&0&0&0\\ 0&1&0&1&1&1&0&0&0&0&0&0&1&0&0\\ 0&0&1&0&1&1&1&0&0&0&0&0&0&1&0\\ 0&0&0&1&0&1&1&1&0&0&0&0&0&0&1 \end{vmatrix}$$

El vector código es el siguiente:

$$[010000011001110]$$

Es claro que los últimos 7 bits corresponden a la información original esta es la ventaja de utilizar una matriz generadora sistemática.

## 1.4 CAMPOS DE GALOIS

A continuación sé vera brevemente los campos de **Galois** [4,5] los cuales son importantes en el proceso de decodificación de los códigos BCH.

Un campo matemático *F*, es un sistema que tiene dos operaciones, ambas con inversas. Usualmente la adición y la multiplicación. Si *a, b* y *c* son elementos de *F*, esto es, *a, b, c* $\in F$ entonces se pueden definir mediante los siguientes postulados de la tabla 1.4:

| | Bajo la suma | Bajo la multiplicación |
|---|---|---|
| Cerradura | a + b ∈ f | a * b ∈F |
| Asociativa | (a + b) + c = a +( b + c) | (a*b)*c=a*(b*c) |
| Conmutativa | a+b=b+a | a*b=b*a |
| Identidad | a+0=a | a*1=a |
| Inversa | $a+a^{-1}=0$ | $a*a^{-1}=1$, a≠0 |
| Distributiva | a(b+c)=ab+ac | a(b+c)=ab+ac |

**Tabla 1.4** Postulados para el campo *F*

Un campo con un número finito de elementos, *q*, es llamado campo de Galois finito y es denotado por GF(*q*). En general los campo de Galois finitos existen cuando *q* es primo, o cuando $q=p^m$ donde *p* es primo y *m* es entero. El campo primo GF(*q*), con *q*>1 y *q* primo tendrá los elementos 0, 1, 1, ..., q-1, esto es el conjunto de todos los enteros modulo q, y las operaciones son de adición y multiplicación modulo *q*. Por ejemplo, el campo primo GF(5) tendrá los siguientes elementos (0, 1, 2, 3, 4) y 3+4=2 esto es con respecto al modulo 5. Similarmente sobre el mismo campo 2 * 3= 1 con modulo 5.

Es claro que el campo primo más simple es GF(2) con los elementos (0,1) y aritmética modulo 2. El campo GF($p^m$), con *p* primo y *m*>1 es llamado campo de extensión GF(*p*) y *p* es la característica. Los campos de extensión con característica 2 son particularmente relevantes en los códigos binarios BCH. Una característica atractiva es que las fuentes de datos son usualmente binarias y pueden representarse por GF($2^m$), otra característica es que su aritmética es sencilla. Para generar un campo GF($2^m$) se extienden los elementos 0,1 usando un elemento primitivo. Sí

$$\alpha \in GF(2^m)$$

Entonces el postulado de la cerradura bajo la multiplicación se tiene que $\alpha\,\alpha=\alpha^2$ , son también elementos de GF($2^m$), y se tiene:

$$GF\left(2^m\right)=\left\{0,1,\alpha,\alpha^2,\alpha^3,\ldots\right\} \tag{1.39}$$

Por definición un campo GF($2^m$) tiene que ser finito y esto se consigue asociando este a un polinomio $p(\alpha)$ de grado el cual es primitivo sobre GF(2). Existen tablas con estos polinomios primitivos.

Para el código BCH(15,7,2) que esta definido sobre GF($2^4$), el campo puede ser construido por el polinomio $p(x)=1+x+x^4$, en la tabla 1.5 se muestra la representación para este campo.

| Representación Potencia | Representación Polinomial | Representación vector |
|---|---|---|
| 0 | 0 | 0 0 0 0 |
| 1 | 1 | 1 0 0 0 |
| $\alpha$ | $\alpha$ | 0 1 0 0 |
| $\alpha^2$ | $\alpha^2$ | 0 0 1 0 |
| $\alpha^3$ | $\alpha^3$ | 0 0 0 1 |
| $\alpha^4$ | $1+\alpha$ | 1 1 0 0 |
| $\alpha^5$ | $\alpha+\alpha^2$ | 0 1 1 0 |
| $\alpha^6$ | $\alpha^2+\alpha^3$ | 0 0 1 1 |
| $\alpha^7$ | $1+\alpha+\alpha^3$ | 1 1 0 1 |
| $\alpha^8$ | $1+\alpha^2$ | 1 0 1 0 |
| $\alpha^9$ | $\alpha+\alpha^3$ | 0 1 0 1 |
| $\alpha^{10}$ | $1+\alpha+\alpha^2$ | 1 1 1 0 |
| $\alpha^{11}$ | $\alpha+\alpha^2+\alpha^3$ | 0 1 1 1 |
| $\alpha^{12}$ | $1+\alpha+\alpha^2+\alpha^3$ | 1 1 1 1 |
| $\alpha^{13}$ | $1+\alpha^2+\alpha^3$ | 1 0 1 0 |
| $\alpha^{14}$ | $1+\alpha^3$ | 1 0 0 1 |

**Tabla 1.5** Representación para GF($2^4$) generado por $p(x) = 1 + x + x^4$

## 1.5 DECODIFICACIÓN DE CÓDIGOS CÍCLICOS

Los códigos BCH [4,5] son atractivos por que son fáciles de codificar y especialmente por que existen algoritmos de decodificación algebraica eficientes basados en un polinomio de detección de errores.

Considere un código BCH(*n,k,t*) binario con corrección de '*t*' errores definido sobre GF($2^m$), esto es, $2^m$-1. Cuando se recibe una palabra de código esta es modelada como:

$$r(x) = v(x) + e(x) \tag{1.40}$$

*r(x): vector de código recibida*
*v(x): vector de código codificada*
*e(x): vector de error*

Como primer paso se debe buscar el síndrome para *r(x)*, que permite localizar los errores. Dado que $\alpha^j$=1,2,...,2t es raíz de *v(x)*, se tiene

$$r(\alpha^i) = v(\alpha^i) + e(\alpha^i) \tag{1.41}$$

$r(\alpha^j) = Si$

Donde *Si* es un símbolo de síndrome. De este resultado se observa que los símbolos síndrome se pueden encontrar como:

$Si=r(\alpha^i);$ i=1,2,…,2t (1.42)

Este es el primer paso en la decodificación BCH. Como un ejemplo una palabra de código BCH(15,7,2), se recibe en la entrada del decodificador como $r(x)=x+x^7+x^8+x^{11}$

Usando el campo de Galois ya desarrollado los símbolos síndromes correspondientes son:

$S1=r(\alpha)=\alpha+\alpha^7+\alpha^8+\alpha^{11}=\alpha+1+\alpha+\alpha^3+1+\alpha^2+\alpha+\alpha^2+\alpha^3=\alpha$

$S3= r(\alpha^3)= \alpha^3+\alpha^{21}+\alpha^{24}+\alpha^{33}=\alpha^3+\alpha^2+\alpha^3+\alpha+\alpha^3+\alpha^3=\alpha^5$

$S_{2k}=S^2_k \ \forall k$ $S=(\alpha, \alpha^2, \alpha^5, \alpha^4)$

El síndrome en la decodificación algebraica es usado para determinar los coeficientes de un polinomio localizador $\sigma(x)$ de errores.

Suponga para un $r(x)$ particular

$e(x)= x^8+x^{11}$

entonces se tiene que

$e(\alpha^j)=( \alpha^8)^i+(\alpha^{11})^i=x_2{}^i+x_1{}^i$

Donde $x_1=\alpha^{11}$ es un localizador de error, el cual denota que el símbolo recibido $r^{11}$ es erróneo, y $x^2=\alpha^8$ denota un error en $r^8$. Generalizando, el k-esimo error tendrá un localizador de la forma

$Xk=\alpha^j, \ \alpha^j \in GF(2^m), k=1,2,...,t$ (1.43)

Y denota un error en el símbolo $r_j$, j=0,1,...,n-1

Generalizando

$$S_i = e(\alpha^i) = \sum_{k=1}^{t} X_k^i, \qquad i=1,2,...,2t \tag{1.44}$$

Se podría resolver esta ecuación para los '*t*' valores de *Xk*, pero involucra ecuaciones no lineales, en la práctica se usa un polinomio localizador de error $\sigma(x)$. La importancia de $\sigma(x)$ es que transforma el problema en un conjunto de ecuaciones lineales y que sus raíces resultan ser localizadores de error *Xk*.

## 1.6 ALGORITMO DE DECODIFICACIÓN DIRECTA DE PETERSON Y BÚSQUEDA DE CHIEN

Esta es una aproximación directa para encontrar $\sigma(x)$, este método [4] resulta ineficiente para valores grandes de $t$ cuando se compara con técnicas iterativas como el algoritmo Berlekamp, pero en este caso $t=2$ por lo tanto el método es eficiente.

Se define a $\sigma(x)$ como

$$\sigma(x) = \prod_{k=1}^{t} (x + X_k) = x^t + \sigma_1 x^{t-1} + \sigma_2 x^{t-2} + .... + \sigma^t \qquad (1.45)$$

El localizador $X_k$ es una raíz de $\sigma(x)$. Dado que es una raíz se puede escribir de la siguiente forma

$$X^t_k + \sigma_1 X_k^{t-1} + \sigma_2 X_k^{t-2} + .... + \sigma_t = 0 \quad k=1,2,...,t \qquad (1.46)$$

Sustituyendo los términos de la suma tenemos

$$S^{t+j} + \sigma_1 S^{t+j-1} + ... + \sigma_t S^j = 0 \quad j=1,2,...,t \qquad (1.47)$$

Esta ecuación representa un conjunto de ecuaciones lineales que se pueden resolver para los coeficientes $\sigma i$, $i=1,2,...,t$ dado $\sigma(x)$. Además para códigos binarios:

$$S_{2k} = S_k^2, \forall k \qquad (1.48)$$

Esta ecuación indica que solo se necesitan valores impares. En la siguiente tabla 1.6 se muestran los coeficientes para $\sigma\ (x)$ de códigos binarios BCH con corrección de *'t'* errores.

| *t-errores* | *Coeficientes* |
|---|---|
| *1* | $\sigma_1 = s_1$ |
| *2* | $\sigma_1 = s_1$<br>$\sigma_2 = \frac{S_3 + S_1^3}{S_1}$ |
| *3* | $\sigma_1 = S_1$<br>$\sigma_2 = \frac{S_1^2 S_3 + S_5}{S_1^3 + S_3}$<br>$\sigma_3 = (S_1^3 + S_3) + S_1 \sigma_2$ |

**Tabla 1.6** Coeficientes para $\sigma^x$ para codigos BCH con *t* errores

Una vez que se tiene el polinomio localizador de error $\sigma(x)$ el siguiente paso es determinar raíces. Estas raíces serán los localizadores de error $Xk,\ k=1,2,...,t$ y cada localizador será de la forma dada en 1.43.

Una aproximación es intentar con cada una de las raíces posibles $\alpha^j$ en 1.46. La sustitución se da de la siguiente manera. Se reescribe 1.45 como:

$$\frac{\sigma(x)}{X^t} = 1 + \sigma_1 x^{-1} + ... + \sigma_t x^{-t}$$

Igualando a cero esta ecuación sobre GF($2^m$) se encuentran sus raíces.

$$\sum_{i-1}^{t} \sigma i_i x^{-1} = 1 \qquad (1.49)$$

Las raíces de 1.49 son de la forma $\alpha^j$, j=0,1,...n-1, esto es la raíz corresponde a un error $\alpha^j$ en el símbolo $r^j$. Suponga que primero se prueba si hay error en el símbolo $r_{n-1}$ sustituyendo $\alpha^{n-1}$ por $x$. Él termino $x^{-i}$ se reduce a

$$\alpha^{-i(n-1)} = \alpha^{ni}\alpha^j = \alpha^j$$

De forma similar, $\alpha^{n-2}$ se reduce a $\alpha^{2i}$ y $\alpha^{n-j}$ se reduce a $\alpha^{ji}$.

La ecuación 1.49 puede escribirse como la ecuación de búsqueda.

$$\sum_{i=1}^{t} \sigma_i \alpha^{ji} = 1, j = 1,2,...,n \qquad (1.50)$$

Para mostrar el procedimiento de la decodificación se realiza a continuación un ejemplo utilizando él número 9 que se codifico anteriormente. El número codificado y su polinomio de código se muestran a continuación.

010000011001110

$$x + x^7 + x^8 + x^{11} + x^{12} + x^{13}$$

Los bits 13 y 14 serán cambiados por su complemento y se aplicara el algoritmo para corregir estos bits de esta forma la palabra de código recibida es la siguiente.

010000011001101

$$r(x) = x + x^7 + x^8 + x^{11} + x^{12} + x^{14}$$

El primer paso que se realiza es calcular el síndrome del vector recibido *r(x)* utilizando la ecuación 1.44. y el campo de Galois de la tabla 1.3 se obtiene que los elementos del síndrome son los siguientes:

$$S_1 = r(\alpha) = \alpha + \alpha^7 + \alpha^8 + \alpha^{11} + \alpha^{12} + \alpha^{14} = \alpha^2$$

$$S_3 = r(\alpha^3) = \alpha^3 + \alpha^{21} + \alpha^{24} + \alpha^{33} + \alpha^{36} + \alpha^{42} = \alpha^8$$

Y utilizando la ecuación 1.48 se obtienen los elementos del síndrome restantes.

$$S_{2k} = S_k^2, \forall k$$

$$S = (\alpha^2, \alpha^4, \alpha^8, \alpha^8)$$

Usando la tabla 1.4 se encuentran los coeficientes para el polinomio localizador de error.

$$\sigma_1 = \alpha^2$$
$$\sigma_2 = \frac{\alpha^8 + \alpha^6}{\alpha^2} = \alpha^{12}$$

El polinomio localizador de error es el siguiente.

$$\sigma(x) = x^2 + \alpha^2 x + \alpha^{12}$$

Usando la fórmula 1.50 de la búsqueda de Chien y la tabla 1.3 se procede a buscar las raíces iguales a uno las cuales indican el bit o bits con error.

Para j=1, se tiene $\alpha^2\alpha + \alpha^{12}\alpha^2 = \alpha^3 + \alpha^{14} = 1$

Para j=2, se tiene $\alpha^2\alpha^2 + \alpha^{12}\alpha^4 = \alpha^4 + \alpha^{16} = 1$

Para j=3, se tiene $\alpha^2\alpha^3 + \alpha^{12}\alpha^6 = \alpha^5 + \alpha^{18} = \alpha^{11}$

Para j=4, se tiene $\alpha^2\alpha^4 + \alpha^{12}\alpha^8 = \alpha^6 + \alpha^{20} = \alpha^9$

Para j=5, se tiene $\alpha^2\alpha^5 + \alpha^{12}\alpha^{10} = \alpha^7 + \alpha^{22} = 0$

Entonces de los resultados se tiene $\alpha^{n-1} = \alpha^{14}$ y $\alpha^{n-2} = \alpha^{13}$ corresponden a los bits 13 y 14 que tienen error para corregirlos es necesario cambiarlos por su complemento y se corrige el bit correctamente quedando la palabra de código corregida de la siguiente forma.

$$x + x^7 + x^8 + x^{11} + x^{12} + x^{13}$$

## 1.7 PÉRDIDA DE MUESTRAS

Un problema importante que se debe considerar para la detección de una marca de agua es observar que sucede si el archivo recuperado ha perdido muestras. Este problema se presenta cuando se realizan pruebas de cambios de formato sobre los algoritmos empleados para ocultar la marca.

Una práctica frecuente es el cambiar de formato los archivos de audio wav, por ejemplo a MP3 y CDA. Para esto se puede utilizar cualquier convertidor de formato. Para cada formato utilizado el archivo con la marca pierde una cantidad distinta de muestras para mostrar este problema se convierte un archivo wav a estos distintos formatos y se muestran la cantidad de muestras recuperadas, para cada formato la figura 1.9 del archivo de prueba es el siguiente:

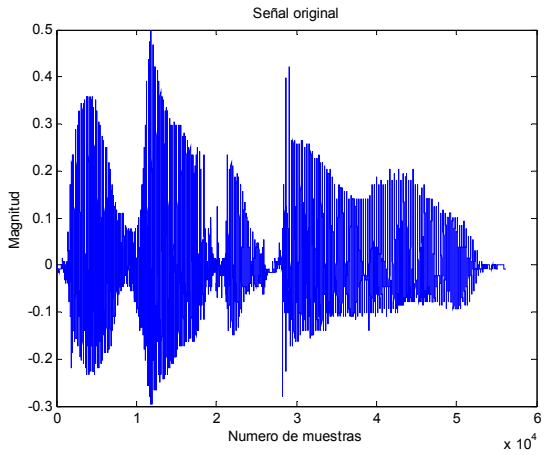



**Figura 1.9** Señal Original

Las características del archivo se muestran en la tabla 1.7:

| Archivos | Frecuencia | Codificación | Muestras | Energía |
|---|---|---|---|---|
| Archivo 1 | 44100 | 8 | 56200 | 1984.9 |

**Tabla 1.7** Características del archivo original

El archivo original wav se convierte a otro archivo wav con una frecuencia de 22050 y una codificación de 16 bits y se devuelve a su formato original y la tabla 1.8 se muestra que se pierden 16 muestras:

| Archivos | Frecuencia | Codificación | Muestras | Muestras Pérdidas |
|---|---|---|---|---|
| Archivo 1 | 44100 | 8 | 56194 | 16 |

**Tabla 1.8** Características tras modificar su frecuencia

Convirtiendo al formato CDA y al regresar al archivo a su formato original se tiene lo siguiente en la tabla 1.9:

| Archivos | Frecuencia | Codificación | Muestras | Muestras Aumentadas |
|---|---|---|---|---|
| Archivo 1 | 44100 | 8 | 126400 | 70200 |

**Tabla 1.9** Características después de modificar al formato CDA

De la tabla 1.9 las muestras agregadas al final del archivo recuperado que aparecen en este formato CDA en su mayor parte son ceros agregados al final del archivo y no afectan el sonido sin embargo si existe una pequeña modificación en la localización de la información.

Después de convertir el archivo al formato MP3 y de nuevo se restaura el archivo a su formato original los resultados se muestran en la tabla 1.10:

| Archivos | Frecuencia | Codificación | Muestras | Muestras Perdidas |
|---|---|---|---|---|
| Archivo 1 | 44100 | 8 | 52992 | 3208 |

**Tabla 1.10** Características después de cambiar al formato MP3

Para este formato es más evidente la pérdida de muestras pero también existe un retraso al inicio del archivo recuperado respecto al original como se muestra en la figura 1.10.

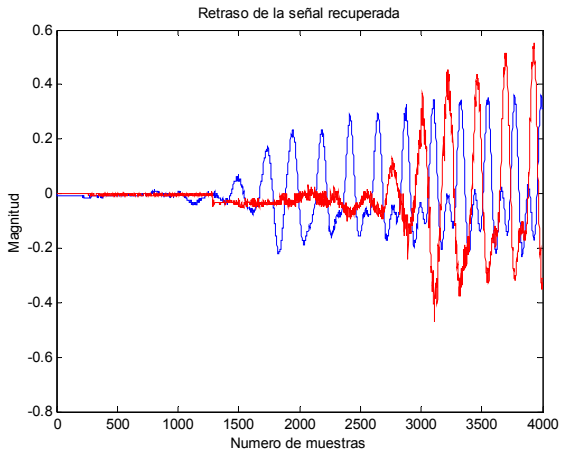



**Figura 1.10** Retraso de la señal roja recuperada comparada con la azul original

El retraso que ocurre en MP3 sucede también en el formato WAV y CDA pero en MP3 es más evidente, también es interesante como los niveles de cuantización del archivo original se modifican en cada formato utilizando el convertidor de formato a continuación se muestran diferentes figuras. La figura 1.11 corresponde al archivo original, la figura 1.12 corresponde al archivo después de convertirlo a un formato de 16 bits, la figura 1.13 corresponde a un formato de 8 bits pero se le modifico la frecuencia a 22050 Hz, la figura 1.14 corresponde a un cambio de 16 bits y una frecuencia de 22050 Hz, la figura 1.15 corresponde a un cambio al formato MP3 y la figura 1.16 a un cambio al formato CDA y después de convertir nuevamente al archivo a su formato original se muestran las distribuciones para cada formato (WAV, CDA, MP3) utilizando un convertidor de formato:

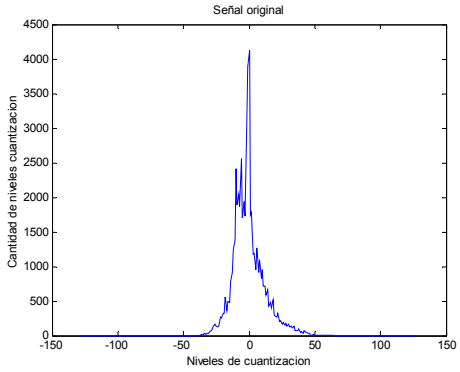



**Figura 1.11** Archivo Original

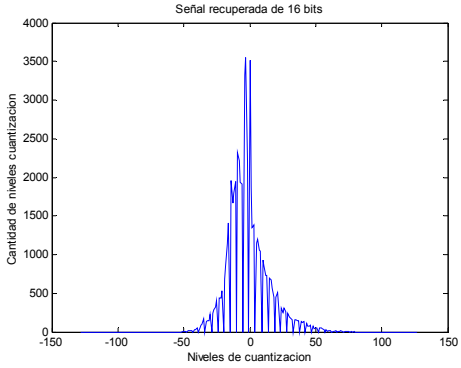



**Figura1.12** Archivo wav a 16 bits

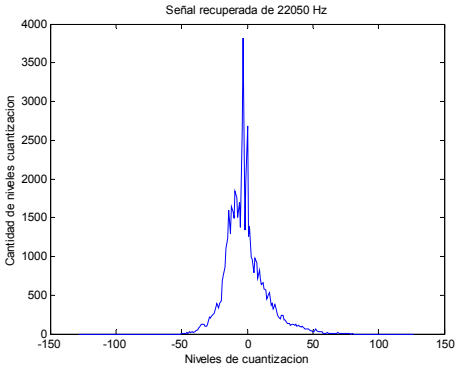



**Figura 1.13** Archivo wav a 22050 Hz

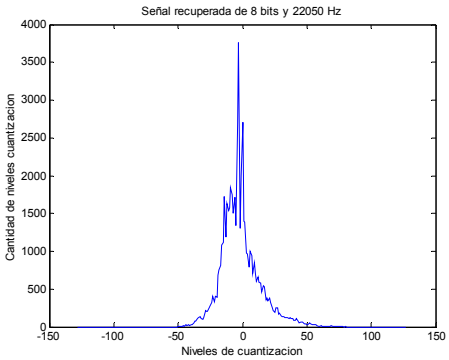



**Figura 1.14** Archivo a 8 bits y 22050 Hz

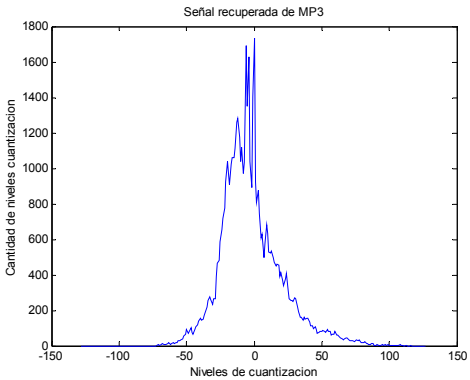



**Figura 1.15** Archivo recuperado de Mp3

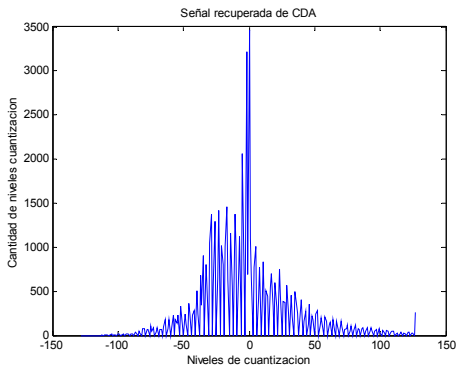



**Figura1.16** Archivo recuperado de CDA

Como se mostró en las figuras anteriores se requiere encontrar una forma para localizar un punto de referencia a partir del cual buscar la información que se oculta. Para resolver este problema lo primero que se hace es aplicar una wavelet a nuestro archivo usando el archivo de la figura 1.9 y aplicando una wavelet se muestra el resultado en la figura 1.17:

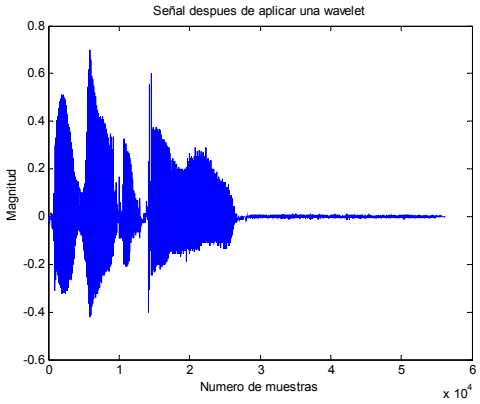



**Figura 1.17** Resultado de aplicar una wavelet

La primera mitad de la figura 1.17 corresponde a las frecuencias bajas en esta parte es donde se elige ocultar la marca ya que contiene las frecuencias bajas y la mayor cantidad de la energía del archivo original. El valor de referencia que se toma es el valor máximo absoluto el cual esta ubicado en la posición 5935 y su valor es de 0.6983 para obtener mejores resultados se usa un grupo de valores alrededor de este valor máximo por lo tanto el vector es de 5880-5979, la cantidad de elementos que se deben utilizar es de al menos 100 elementos ya que una cantidad menor no siempre dará un resultado correcto respecto a la posición que se esta buscando, este vector de referencia es el que se va a correlacionar con el archivo recuperado por medio de la siguiente fórmula [8]:

$$Zxy(l) = \frac{r_{xy}[l]}{\sqrt{r_{xx}[0]r_{yy}[0]}} \tag{1.51}$$

Usando esta fórmula y correlacionando este vector con la señal de frecuencia bajas *u* queda la siguiente gráfica:

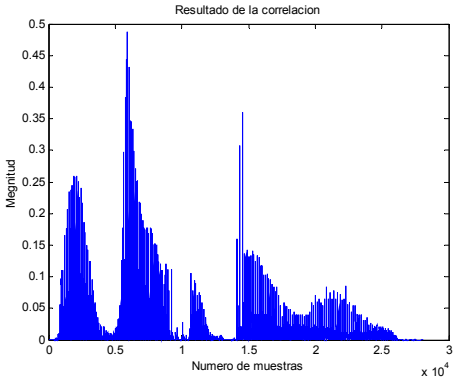



**Figura 1.18** Valores cuadráticos de la señal *u*

De la figura 1.18 se tiene que el valor máximo esta en la posición 5880 tal como el vector de referencia, usando esta técnica será capaz de saber cuantas muestras esta desplazada la señal y a partir de esto se empezará a buscar la información de la marca, como una recomendación los vectores de referencia dan mejores resultados cuando se usan de longitud de 100 a 200 elementos.

# Capítulo 2

## ALGORITMOS PARA LA INTRODUCCIÓN DE LA MARCA DE AGUA

En este capítulo se explican los dos algoritmos que se emplean para introducir la marca de agua. Primeramente se da una definición de la marca de agua y su relación con la estenografía, también se muestran las propiedades principales [7,8] que se deben cumplir para tener una marca de agua robusta. Posteriormente se explican los dos algoritmos, el primero de ellos es el algoritmo Patchwork el cual modifica una cifra de cada par de muestras seleccionadas de acuerdo con una clave para ocultar un bit de información. Además, se proporcionan algunas consideraciones que se deben tener en cuenta para lograr una marca de agua efectiva utilizando Patchwork. El segundo algoritmo es un método propuesto en esta tesis, el cual emplea la codificación de los archivos wav de 8 bits, para ocultar bits de información. A diferencia del Patchwork se emplean un par de muestras que se intercambian dependiendo del valor del bit a ocultar. La información que se oculta esta codificada con el código BCH(15,7,2). Se debe recordar que para ambos algoritmos se trabaja en el dominio wavelet.

### 2.1 MARCA DE AGUA

La marca de agua y la estenografía están relacionadas [7,8,9] ya que ambas técnicas tienen como objetivo ocultar información en cualquier archivo de video, imágenes o música, aunque ambas tienen sus diferencias, la estenografía usualmente se transmite entre dos partes, es decir, punto a punto. Además, su principal objetivo es comunicar un mensaje y no autentificar un trabajo como la marca de agua. En las técnicas de estenografía no se requiere una gran robustez y protección contra compresión o conversión de formato entre otros procesamientos que existen. Mientras que la marca de agua debe ser robusta contra ataques y si la información oculta es descubierta esta debe ser difícil de remover para el atacante.

Las marcas de agua han despertado un gran interés debido al surgimiento de los medios digitales y la facilidad de obtener información sin el consentimiento de su autor, principalmente por la Internet, este resulta ser un excelente medio para obtener música, imágenes y videos, para su posterior almacenamiento, copiado, modificación o simplemente usarlos sin autorización del propietario.

La primera tecnología usada para proteger la información es la criptografía, y probablemente este es el método más común de protección digital, ciertamente es un buen sistema, la información es cifrada por medio de una clave y para recuperar la información se requiere esta clave, de esta forma solo se pueden realizar copias si la persona conoce esta clave. Por desgracia si una persona no autorizada obtiene la clave y realiza copias, la criptografía no siempre puede distinguir si la copia es legítima o es una copia ilegal.

Aquí es donde surge la marca de agua que puede usarse en forma independiente o como un complemento a la criptografía, una de las principales aplicaciones de la marca de agua es la protección de derechos, el propietario introduce una cantidad de bits en su trabajo que permite comprobar que es de su propiedad y que cualquier otro usuario tiene una copia no autorizada. La información oculta por la marca de agua no debe ser removida o alterada durante el uso normal del archivo que puede ser cifrado, filtrado, comprimido, convertido de analógico-digital o viceversa, cambiado de formato, etc. La marca de agua puede ser diseñada para sobrevivir a estos procesos y otros más.

## 2.2 PROPIEDADES

La marca de agua debe cumplir ciertas propiedades entre las dos más importantes se encuentran [7,8]:

### Fidelidad

Este es uno de los requerimientos más importantes de la marca de agua independientemente en donde se este utilizando, esta se refiere a la similitud del trabajo original y la versión del trabajo con la marca.

Dos problemas están relacionados con esta propiedad, el primero es que la información ocultada causa que se degrade el trabajo, sin embargo, estos cambios no deben ser perceptibles ni visualmente (imagen) y tampoco auditivamente (música). El segundo problema es el procesamiento aplicado al trabajo con la marca de agua, esto puede poner al descubierto la información oculta, por ejemplo en una imagen protegida el escalamiento puede resaltar la información oculta. Para algunas aplicaciones se puede realizar un intercambio de fidelidad por robustez como en imágenes a escala de grises.

### Robustez

La marca de agua debe resistir la distorsión introducida por el procesamiento de datos ya sea que estos procesos son propios del medio en donde se usan los archivos o sean estos realizados intencionalmente, ejemplos de estas operaciones incluyen el filtrado espacial, compresión con pérdidas, distorsiones geométricas, etc. No existe un método que sea resistente a todos los procesos existentes, en la práctica al implementar el sistema se debe realizar un compromiso entre la robustez y los requerimientos de fidelidad.

Existen casos especiales de marcas de agua conocidas como marcas frágiles donde la robustez es irrelevante, al aplicar algún procesamiento de datos la marca se pierde, esta fragilidad tiene sus ventajas principalmente en la prueba de autentificación ya que si la marca fue detectada entonces puede asumirse que el trabajo no ha sido alterado.

## 2.3 ALGORITMO PATCHWORK

El algoritmo Patchwork [7,10] es bastante conocido y permite ocultar un mensaje en un archivo de música o imagen, en este trabajo se utilizarán archivos de música en formato wav. El algoritmo emplea una palabra clave con la que se genera una secuencia de números pseudoaleatorios, que indican las posiciones de las muestras del archivo que se van a utilizar para ocultar la información de la marca, por cada bit oculto se emplean dos muestras.

En su forma básica el algoritmo Patchwork funciona de la siguiente manera, una vez definida la palabra clave se eligen un par de cifras significativas ($C_1$, $C_2$) de las muestras elegidas *M1 y M2*. Se modifica la segunda o tercera cifra significativa de la magnitud de cada par de muestras, este trabajo se elige implementar el algoritmo en la segunda cifra significativa (no se utiliza la tercera cifra significativa por razones que más adelante se explican). Ambas cifras se suman y se obtiene su promedio *P*. El algoritmo Patchwork recomienda que a este promedio se le sume o reste 1, dependiendo del bit que se desea ocultar, si el bit a ocultar es un 1 lógico la nueva cifra significativa $C_{11}$ debe ser mayor que la nueva cifra significativa $C_{22}$. Para el caso de tener 0 lógico la cifra significativa $C_{11}$ debe ser menor a la cifra significativa $C_{22}$, como se muestra a continuación.

Primeramente se calcula el promedio de la siguiente forma:

$$P = \frac{C_1 + C_2}{2} \tag{2.1}$$

A continuación se calculan los nuevos valores $C_{11}$ y $C_{22}$ que reemplazaran a $C_1$ y $C_2$. Supóngase que se desea ocultar un bit con valor de 1,

$$\begin{aligned} C_{11} &= P + 1 \\ C_{22} &= P - 1 \end{aligned} \tag{2.2}$$

Supóngase que se desea ocultar un bit con valor de 0,

$$\begin{aligned} C_{11} &= P - 1 \\ C_{22} &= P + 1 \end{aligned} \tag{2.3}$$

En el proceso de extracción nuevamente se generan con la palabra clave las posiciones donde se encuentra la información y se revisa el par de cifras significativas de las muestras para saber que bit esta oculto de la siguiente forma.

*Sí* $C_{11} > C_{22}$

*bit=1*

*Sí* $C_{11}<C_{22}$

*bit=0*

Este es el algoritmo Patchwork [7,10] en su forma básica, a continuación se explicará la implementación y se discutirán los problemas por los cuales el algoritmo presenta errores en la recuperación de la marca y se le hicieron algunas modificaciones para su correcto funcionamiento y poder recuperar la información sin errores.

**Implementación del Algoritmo Patchwork**

A continuación se muestra el diagrama a bloques de todo el proceso para ocultar la información en el archivo original usando el algoritmo Patchwork en la figura 2.1, a continuación se explican los bloques.

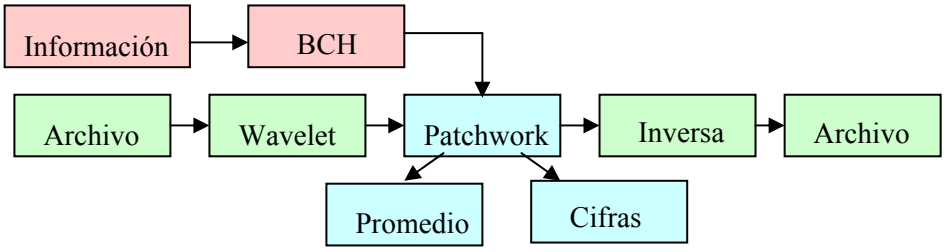
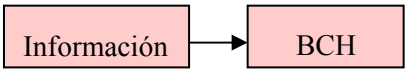



**Figura 2.1** Esquema de la introducción de la marca usando Patchwork

**Bloques de información**.

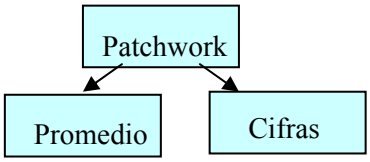



Estos dos bloques en color rojizo corresponden a la información que se oculta, el bloque de información es la marca que se va a ocultar, la cual se convierte a binario para posteriormente pasar al bloque de codificación BCH y codificar la información como protección contra posibles errores cuando se aplican los diferentes cambios de formato al archivo marcado.

**Bloques de la wavelet**


Estos cuatro bloques en verde indican que al archivo original se le aplica una wavelet para descomponer el archivo en dos señales, una de frecuencias bajas y una de frecuencias altas, y luego se aplica el algoritmo Patchwork únicamente a la señal de frecuencias bajas y posteriormente aplicar la wavelet inversa para devolver al archivo a su forma original pero con la marca ya dentro del archivo.

**Bloques del algoritmo Patchwork**




Los tres bloques en azul corresponden a la aplicación del algoritmo Patchwork después de aplicar la wavelet al archivo original y se ocupa la señal de frecuencias bajas la forma de ocultar la información con el algoritmo Patchwork se explica a continuación.

Como se menciono el primer paso es elegir la posición de los pares de muestras utilizando una clave. Esta palabra clave debe convertirse a código ASCII, posteriormente se suman todos sus valores y el resultado se multiplica por una constante. El resultado de esto será la posición $Po_1$. En este trabajo se propone multiplicar por dos, la elección de esta constante solo depende que la posición $Po_1$ no se encuentre dentro de las primeras muestras del archivo de audio ya que los archivos en formato wav empiezan con un silencio. A continuación se presenta un ejemplo utilizando la clave **'Hola'**.

La posición queda determinada de la siguiente forma:

$$Po_1 = 2\sum_{i=1}^{m} Cla(i) \qquad (2.4)$$

donde *Cla(i)* es la clave y los valores de cada carácter están en valor ASCII, la clave tiene una longitud de $m = 4$ caracteres y al aplicar la fórmula (2.4):

*Cla(i)* = 'Hola'=[72 111 108 97]

$$Po_1 = 2 \times 388 = 776$$

Después se determinan dos vectores $I_1[n]$, $I_2[n]$ que van a contener las posiciones del par de muestras que se utilizarán para guardar los bits correspondientes a la cantidad de información que se va a ocultar, para este ejemplo la información es '**Esime**', se utiliza el código BCH(15,7,2) explicado en el capítulo 1. Cada carácter es convertido a una palabra de código de 15 bits, es decir que la cantidad de bits a ocultar es de $In$ =75 bits que es la información total que se va a ocultar, pero cada bit requiere dos muestras por lo tanto.

$$Muestras = 2 \times 75 = 150$$

Es decir, que se necesitan 150 muestras, cada muestra elegida tiene una separación con respecto a la siguiente para dar mayor seguridad contra posibles ataques.

Esta separación entre muestras puede ser igual a la longitud *m* de la clave utilizada o puede ser cualquier otro que se desee, la forma de elegir los índices de las muestras se explica a continuación:

**Índice uno**

$$I_1[n] = Po_1 + (m \times (n-1)) \qquad (2.5)$$

*n=1 hasta In*

**Para el Índice dos se tiene**

$$I_{11}[n] = Po_1 + (m \times (n-1)) + 1 \qquad (2.6)$$

*n=1 hasta In*

$$I_2[n] = I_{11}[In - n - 1] \qquad (2.7)$$

*n=1 hasta In*

Para introducir la marca de agua en el par de muestras, se debe decidir cual cifra significativa va a ser alterada. En este trabajo se elige la segunda cifra. El promedio *P* de ambas cifras se redondea al entero superior, por ejemplo, tomando dos muestras *M1* y *M2*:

$M1[I_1[n]] = 0.1283$

$M2[I_2[n]] = 0.0548$

$$P = \frac{C_1 + C_2}{2}$$

$P = \frac{2+5}{2} \approx 4$ En esta tesis se redondea al entero superior

A continuación se muestra el algoritmo Patchwork con un ejemplo utilizando un archivo de prueba y se explican algunas consideraciones que se deben tener en cuenta con el algoritmo Patchwork para lograr su correcto funcionamiento. Se usará el siguiente archivo de la figura 2.2 para ilustrar estos problemas.

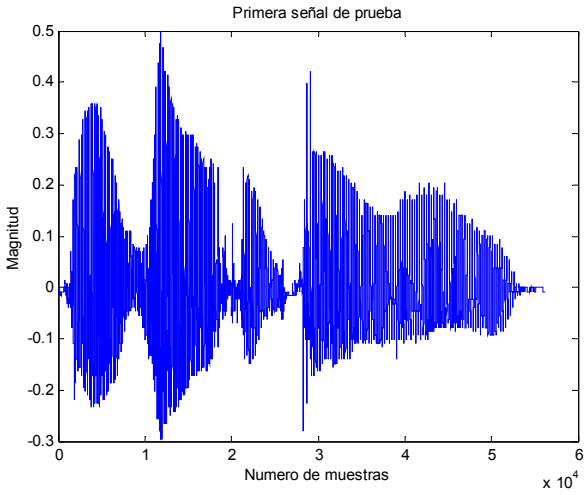



**Figura 2.2** Archivo uno de prueba

En la tabla 2.1 se muestran las características de esta señal

| **Archivos** | **Frecuencia** | **Codificación** | **Muestras** | **Energía** |
|---|---|---|---|---|
| Archivo 1 | 44100 | 8 | 56200 | 1984.9 |

**Tabla 2.1** Características de la señal original

El primer paso que se realiza es aplicar una wavelet al archivo original, la wavelet usada para este ejemplo es la Daubechies 3, la gráfica se muestra en la figura 2.3. La primera mitad corresponde a las frecuencias bajas que es la señal *u*, la segunda mitad corresponde a las frecuencias altas que es la señal *v*. Ambas señales se obtienen usando las fórmulas 1.33 y 1.37.

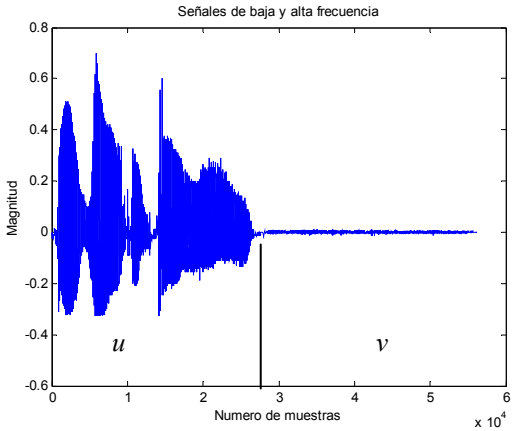



**Figura 2.3** Gráfica de las señales *u* y *v*

A continuación se debe encontrar la posición uno a partir de la cual se oculta la información, se usará la misma clave 'Hola' y recordando los resultados.

*Cla(i)* = 'Hola'=*[72 111 108 97]*

$$Po_1 = 2 \times 388 = 776$$

El siguiente paso es conocer cuenta información se va a ocultar para saber la cantidad de muestras de la señal de bajas frecuencias *u* que se van a ocupar. La información será la misma 'Esime' por lo tanto se requieren 150 muestras para ocultar la información. La matriz codificada usando el BCH(15,7,2) del capítulo 1, se muestra a continuación:

$$Matrizcodificada = \begin{vmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \end{vmatrix}$$

Lo siguiente es calcular los índices que se van a ocupar para elegir las muestras de nuestra señal *u* para ocultar la información usando las fórmulas 2.5 y 2.7.

$$I_1[1] = 776 + (4 \times 0) = 776$$
$$I_1[2] = 776 + (4 \times 1) = 780$$
. . .
. . .
. . .
$$I_1[75] = 776 + (4 \times 74) = 1072$$

Para el índice 2

$$I_2[1] = I_{11}[75] = 1073$$
$$I_2[2] = I_{11}[74] = 1069$$
. . .
. . .
. . .
$$I_2[75] = I_{11}[1] = 777$$

Para ocultar el primer bit se ocupara el par de muestras localizadas en la posición $I_1[1]=776$ e $I_2[1]=1073$ de nuestra señal *u*.

**Elección de la cifra**

Después de encontrar los índices, un problema importante es la elección de la cifra para ocultar la información esto va a determinar el éxito o fracaso del algoritmo la razón se explica mas adelante. Primeramente el archivo de la figura 2.1 es de un formato de 8 bits como se muestra en la tabla 2.1 este tipo de formato no permite que se modifique la tercera cifra como se muestra a continuación al ocultar la palabra 'Esime'.

Para ocultar el primer valor de la matriz codificada de información, el primer bit que se oculta es 1, por lo tanto usando los índices obtenidos y la señal *u,* se tiene el primer par de muestras que se van a modificar.

$$M1 = u(I_1[1]) = -0.0442$$

$$M2 = u(I_2[1]) = 0.0844$$

La cifra para ocultar la información será la tercera es decir que $C_1=4$ y $C_2=4$ y usando la fórmula 2.1 se tiene lo siguiente.

$$P = \frac{4+4}{2} = 4$$

Una vez obtenido el promedio las fórmulas que se tendrían que aplicar son 2.2 y 2.3 pero se aplicaran con una modificación y se reescriben de la siguiente forma:

$$C_{11} = P + 2$$
$$C_{22} = P - 1 \qquad (2.8)$$

$$C_{11} = P - 1$$
$$C_{22} = P + 2 \qquad (2.9)$$

La explicación del por que se realiza este cambio se explica mas adelante. El bit a ocultar es un 1 por lo tanto se usa la fórmula 2.8 y se tiene lo siguiente.

$$C_{11} = 4 + 2 = 6$$
$$C_{22} = 4 - 1 = 3$$

Y reemplazando el nuevo valor de la tercera cifra en las muestras *M1 y M2* los nuevos valores de $C_{11}$ y $C_{22}$

*M1=-0.0462*
*M2=0.0834*

De esta forma se oculta el primer bit, todo este procedimiento se repite para todos y cada uno de los bits que se ocultan. En la tabla 2.2 se muestran los resultados de ocultar la información de la columna uno con *M1 y M2* como las muestras originales que se eligen por los índices $I_1[n]$ y $I_2[n]$, sus valores modificados *Modificado1 y Modificado2*, así como también el resultado al recuperar la información de estas muestras.

| $I_1[n]$ | $I_2[n]$ | *M1* | *M2* | *Modificado1* | *Modificado2* | *Recuperado1* | *Recuperado2* |
|---|---|---|---|---|---|---|---|
| *776* | *1073* | *-0.0442* | *0 .0844* | *-0.0462* | *0.0834* | *-0.0442* | *0.0844* |
| *780* | *1069* | *-0.0935* | *-0.0238* | *-0.0925* | *-0.0258* | *-0.0935* | *-0.0238* |
| *784* | *1065* | *-0.0860* | *-0.1151* | *-0.0851* | *-0.1181* | *-0.0860* | *-0.1151* |
| *790* | *1061* | *-0.1547* | *-0.2006* | *-0.1517* | *-0.2046* | *-0.1547* | *-0.2006* |
| *794* | *1057* | *-0.1297* | *-0.2210* | *-0.1277* | *-0.2240* | *-0.1297* | *-0.2210* |

**Tabla 2.2** Valores originales, modificados y recuperados para la columna uno de información

De los resultados de la tabla 2.2 y comparando las columnas de las muestras originales y las recuperadas, es evidente que aun cuando se aplico el algoritmo Patchwork el cambio en la tercera cifra de las columnas *Modificado1* y *Modificado2* no se conserva.

Este problema es común para archivos wav en formato de 8 bits debido a que son poco flexibles a los cambios de valores en sus muestras ya que cuentan con un número muy limitado de niveles de cuantización de $2^7 = 128$ niveles o volúmenes de cuantización y un bit es utilizado para el signo, por lo tanto cualquier cambio que se realiza en la tercer cifra no se conserva para archivos de 8 bits. El mensaje recuperado es el siguiente:

*Bits con error = [1 2 6 7 9 10 12 13 15 16 19 20 23 24 25 30 32 33 35 37 39 41 44 46 47*

*48 49 52 53 54 56 57 58 59 60 62 66 68 69 72]*

*Mensaje = n%□Bo*

$$Matriz\,\mathrm{Re}cuperada = \begin{vmatrix} 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \end{vmatrix}$$

$$Matrizdecodificada = \begin{vmatrix} 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \end{vmatrix}$$

Es claro que el *Mensaje* se perdió ya que los cambios realizados por el Patchwork no se conservan y esto se muestra en la matriz recuperada de información y la matriz decodificada después de aplicar el BCH(15,7,2).

Para resolver este problema el algoritmo Patchwork solo se debe aplicar en la segunda cifra para archivos en formato de 8 bits. Aplicando nuevamente el algoritmo en la segunda cifra para el archivo de 8 bits, la primera columna queda oculta como se muestra en la tabla 2.3.

| $I_1$ | $I_2$ | *M1* | *M2* | *Modificado1* | *Modificado2* | *Recuperado1* | *Recuperado2* |
|---|---|---|---|---|---|---|---|
| *776* | *1073* | *-0.0442* | *0 .0844* | -0.0842 | 0.0544 | -0.0835 | 0.0483 |
| *780* | *1069* | *-0.0935* | *-0.0238* | -0.0535 | -0.0838 | -0.0531 | -0.0793 |
| *784* | *1065* | *-0.0860* | *-0.1151* | -0.0460 | -0.1751 | -0.0394 | -0.1733 |
| *790* | *1061* | *-0.1547* | *-0.2006* | -0.1247 | -0.2506 | -0.1189 | -0.2487 |
| ***794*** | ***1057*** | ***-0.1297*** | ***-0.2210*** | **-0.1497** | **-0.2110** | **-0.1388** | **-0.2237** |

**Tabla 2.3** Valores recuperados al utilizar la segunda cifra

De los resultados de la tabla 2.3 se tiene que aun cuando existen unos pequeños cambios se debe tener en cuenta que la diferencia de las cifras se conserva por lo tanto la información se recupera correctamente. Se comento que las fórmulas 2.2 y 2.3 no se aplican en su forma original en lugar de eso se aumento la diferencia entre ambas cifras la razón de esto se muestra en el renglón resaltado, es claro que la diferencia de la segunda cifra de *Modificado1* y *Modificado2* es de tres pero al recuperar la información la diferencia de ambas es de solo uno para prevenir este problema se opto en utilizar una diferencia ligeramente mayor aunque este error solo ocurre ocasionalmente.

El mensaje que se recupera y los bits con error son los siguientes:

*Bits con error =[]* *// No existe ningún bit con error*

*Mensaje = Esime*

$$Matrizrecuperada = \begin{vmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \end{vmatrix}$$

$$Matrizdecodificada = \begin{vmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \end{vmatrix}$$

Lo anterior solo se refiere a los archivos wav en formato de 8 bits estas consideraciones se deben tener en cuenta si el archivo utilizado tiene este tipo de formato.

A continuación se realizan las mismas pruebas pero usando un archivo de 16 bits, el archivo tiene las siguientes características que se muestran en la tabla 2.4 y en la figura 2.4.

| Archivos | Frecuencia | Codificación | Muestras | Energía |
|---|---|---|---|---|
| Archivo 4 | 44100 | 16 | 140000 | 11076 |

**Tabla 2.4** Características de la segunda señal de prueba

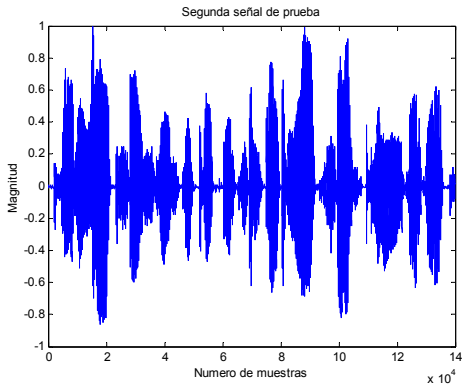



**Figura 2.4** Segundo archivo de prueba

Y se le aplica una wavelet Daubechies 3 y queda de la siguiente forma:

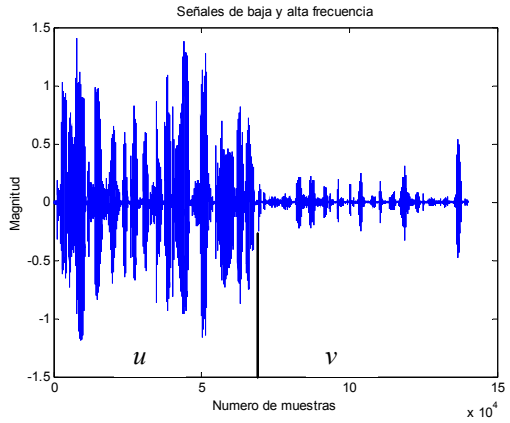



**Figura 2.5** Señales *u* y *v* obtenidas

Para simplificar se usará la misma clave 'Hola' y la información 'Esime' el procedimiento del algoritmo Patchwork es el mismo y la matriz codificada será la misma los índices son los mismos pero aquí se usa un archivo de 16 bits. Primero se oculta la información en la tercera cifra, por lo tanto para el primer par de muestras el promedio que se obtiene se tiene lo siguiente:

$$M1 = u(I_1[1]) = 0.0113$$
$$M2 = u(I_2[1]) = -0.0049$$

$P = \frac{1+4}{2} \approx 3$ En esta tesis se redondea al entero superior

El primer bit que se oculta es un uno por lo tanto se usa la fórmula 2.8 y se tiene lo siguiente:

$C_{11} = 3 + 2 = 5$
$C_{22} = 3 - 1 = 2$

Y reemplazando en la tercera cifra de *M1 y M2*

*M1=0.0150*
*M2=-0.0029*

El resto de los valores originales, modificados y recuperados se muestran en la tabla 2.5 para la primera columna:

| $I_1$ | $I_2$ | *M1* | *M2* | *Modificado1* | *Modificado2* | *Recuperado1* | *Recuperado2* |
|---|---|---|---|---|---|---|---|
| *776* | *1073* | 0.0113 | -0.0049 | 0.0150 | -0.0029 | 0.0150 | -0.0029 |
| *780* | *1069* | 0.0122 | -0.0054 | 0.0130 | -0.0064 | 0.0130 | -0.0064 |
| *784* | *1065* | 0 | -0.0113 | 0 | -0.0133 | 0 | -0.0133 |
| *790* | *1061* | 0 | -0.0113 | 0 | -0.0133 | 0 | -0.0133 |
| *794* | *1057* | 0 | 0 | 0.003 | 0 | 0.0030 | 0 |

**Tabla 2.5** Valores para el archivo de 16 bits

La información recuperada y los bits con error se muestran a continuación:

*Bits con error = []* *//No existe ningún bit con error*
*Mensaje = Esime*

Es claro que en archivos de 16 bits es fácil ocultar la información en la tercera cifra ocupando el algoritmo Patchwork y la información se recupera correctamente por que existe un mayor número de niveles de cuantización $2^{15} = 32768$ volúmenes y un bit para el signo.

Existe una forma muy sencilla de destruir la información oculta y es convertir el archivo de 16 bits a un formato de 8 bits y posteriormente a un formato de 16 bits para dejarlo en su formato original, en la tabla 2.6 se muestra que tras realizar el procedimiento la información se destruye y las cifras modificadas son devueltas a su valor original o un valor cercano del valor original.

| $I_1$ | $I_2$ | *M1* | *M2* | *Modificado1* | *Modificado2* | *Recuperado1* | *Recuperado2* |
|---|---|---|---|---|---|---|---|
| *776* | *1073* | 0.0113 | -0.0049 | 0.0150 | -0.0029 | 0.0110 | -0.0039 |
| *780* | *1069* | 0.0122 | -0.0054 | 0.0130 | -0.0064 | 0.0125 | -0.0084 |
| *784* | *1065* | 0 | -0.0113 | 0 | -0.0133 | 0 | -0.0110 |
| *790* | *1061* | 0 | -0.0113 | 0 | -0.0133 | 0 | -0.0110 |
| *794* | *1057* | 0 | 0 | 0.003 | 0 | 0 | -0.0001 |

**Tabla 2.6** Valores para el archivo de 16 bits después de haberlo convertido a un formato de 8 bits

La información recuperada es la siguiente:

*Bits con error=[ 1 5 9 10 11 12 14 16 18 19 22 25 28 32 34 35 37 38 39 40 41 42 43 44 45 47 54 55 63 64 65 66]*

*Mensaje = $Pxxp*

$$Matrizrecuperada = \begin{vmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{vmatrix}$$

$$Matrizdecodificada = \begin{vmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{vmatrix}$$

Aunque sé esta utilizando un código BCH(15,7,2) la cantidad de errores es demasiada en la matriz recuperada para que el código pueda corregirlos. Este problema se presenta debido al formato de 8 bits para que la información se conserve se requiere nuevamente ocultar la información en la segunda cifra, los resultados de la primera columna se presentan en la tabla 2.7 a continuación.

| $I_1$ | $I_2$ | *M1* | *M2* | *Modificado1* | *Modificado2* | *Recuperado1* | *Recuperado2* |
|---|---|---|---|---|---|---|---|
| *776* | *1073* | 0.0113 | -0.0049 | 0.0313 | -0.0049 | 0.0316 | 0.0038 |
| *780* | *1069* | 0.0122 | -0.0054 | 0.0022 | -0.0354 | 0.0026 | -0.0324 |
| *784* | *1065* | 0 | -0.0113 | 0 | -0.0313 | 0 | -0.0324 |
| *790* | *1061* | 0 | -0.0113 | 0 | -0.0313 | 0 | -0.0324 |
| *794* | *1057* | 0 | 0 | 0.0300 | 0 | 0.0298 | -0.0038 |

**Tabla 2.7** Valores para el archivo de 16 bits después de haberlo convertido a un formato de 8 bits pero usando la segunda cifra

La información recuperada es la siguiente:

*Bits con error = []* *// No existen errores*

*Mensaje = Esime*

$$Matrizrecuperada = \begin{vmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \end{vmatrix}$$

$$Matrizdecodificada = \begin{vmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \end{vmatrix}$$

De estos resultados es claro que la información oculta en las terceras cifras se modifica e incluso algunas en la segunda cifra. Esta es otra razón por la cual el algoritmo Patchwork no debe ser utilizado en la tercera cifra ya que cualquier información oculta se elimina y siempre se debe ocultar la información en la segunda cifra para lograr cierto grado de robustez.

**Puntos importantes sobre el promedio**

Cuando se aplica el algoritmo Patchwork se mencionó que se ocupan un par de cifras significativas y se calcula un promedio usando la fórmula 2.1 esto en la práctica puede generar ciertos problemas que se discutirán a continuación, primero se muestran los resultados de la tabla 2.7.

| $I_1$ | $I_2$ | *M1* | *M2* | *Modificado1* | *Modificado2* | *Recuperado1* | *Recuperado2* |
|---|---|---|---|---|---|---|---|
| *776* | *1073* | 0.0113 | -0.0049 | 0.0313 | -0.0049 | 0.0316 | 0.0038 |
| *780* | *1069* | 0.0122 | -0.0054 | 0.0022 | -0.0354 | 0.0026 | -0.0324 |
| *784* | *1065* | 0 | -0.0113 | 0 | -0.0313 | 0 | -0.0324 |
| *790* | *1061* | 0 | -0.0113 | 0 | -0.0313 | 0 | -0.0324 |
| *794* | *1057* | **0** | **0** | 0.0300 | 0 | 0.0298 | -0.0038 |

**Tabla 2.7** Valores para el archivo de 16 bits después de haberlo convertido a un formato de 8 bits pero usando la segunda cifra

Los valores *M1 y M2* marcados en negritas indican que estas muestras tienen un valor de cero, por lo tanto, al tratar de extraer una cifra y usar la fórmula 2.1 se tiene lo siguiente.

$$P = \frac{0+0}{2} = 0$$

El bit que se oculta es uno por lo tanto la fórmula que se utiliza es 2.8, es decir:

$$C_{11} = 0 + 2 = 2$$
$$C_{22} = 0 - 1 = -1$$

Es claro que es imposible reemplazar el valor de *-1* en la segunda cifra significativa y esto va a causar errores, la solución que se utiliza en esta tesis es verificar el promedio que se obtiene de las cifras significativas *C1 y C2* si el promedio es igual a cero este promedio se desecha y se le asigna el valor de uno.

$$Si \quad P = \frac{C1 + C2}{2} = 0 \Rightarrow P = 1$$

Calculando nuevamente los valores de $C_{11}$ *y* $C_{22}$

$$C_{11} = 1 + 2 = 3$$
$$C_{22} = 1 - 1 = 0$$

En este caso las muestras tienen un valor de cero, esto no necesariamente se tiene que cumplir, las muestras *M1 y M2* pueden tener un valor diferente de cero, pero las cifras significativas *C1 y C2* si pueden tener valor de cero para este problema se emplea la misma técnica.

Otro problema que ocurre con el promedio es cuando *C1 y C2* tienen los siguientes valores de *8 y 9* es decir que los promedios tendrán los siguientes valores.

$$P = \frac{8 + 8}{2} = 8 \qquad P = \frac{8 + 9}{2} \approx 9 \qquad P = \frac{9 + 9}{2} = 9$$

Las fórmulas 2.8 y 2.9 suman dos unidades a cada uno de estos valores de promedio esto claramente genera un error como se muestra a continuación con 2.8 para el promedio de menor valor.

$$C_{11} = 8 + 2 = 10$$
$$C_{22} = 8 - 1 = 7$$

Es evidente que $C_{11}$ no se puede reemplazar directamente, algo similar ocurre con los otros dos valores de promedio, para solucionar este problema se plantea lo siguiente.

$$Si \quad P = \frac{C1 + C2}{2} \geq 8 \Rightarrow P = 7$$

Al aplicar esta condición para el promedio el valor máximo que se puede obtener de $C_{11}$ *y* $C_{22}$ es nueve el cual puede ser reemplazado directamente sin problema alguno, estas consideraciones se deben tener en cuenta al implementar el Patchwork para su funcionamiento correcto ya que aun cuando se propone en esta tesis el código BCH(15,7,2) para corregir errores la meta principal de aplicar este código es corregir los errores producidos por cambios de formato, filtrado, compresión y no para este tipo de errores que fácilmente se pueden prevenir y corregir sin necesidad de usar el código.

## 2.4 MODIFICACION EN LA CANTIDAD y ESCALAMIENTO DE MUESTRAS

El punto que se va tratar en esta sección es muy importante ya que se explica que el algoritmo Patchwork es muy frágil en su implementación y la localización de las muestras puede ser alterada fácilmente sin causar un daño considerable al archivo con la marca oculta y que aún con los cambios realizados al algoritmo Patchwork y la implementación del código BCH(15,7,2) la información no se recupera en su totalidad, estos problemas se presentan principalmente al utilizar un convertidor de formato.

El algoritmo Patchwork para ocultar la información ocupa una posición de inicio $Po_1$ generada por una clave *Cla* y una par de índices $I_1[n]$ *e* $I_2[n]$ para elegir un par de muestras *M1 y M2,* para recuperar la información se generan nuevamente estas posiciones con la clave *Cla* pero el algoritmo no contempla posibilidad de que exista un aumento o pérdida de muestras esto puede suceder cuando se aplica un convertidor de formato al archivo con la marca oculta (en la sección 1.6 del capítulo 1 se mostró este problema) o también cuando alguien intencionalmente elimina o agrega valores al inicio del archivo, los archivos wav inician con un silencio y este cambio no altera la calidad del archivo. Como un ejemplo de lo anterior se usará el archivo de la figura 2.1 el cual contiene 56200 muestras intencionalmente se eliminaran las primeras 50 muestras, en la figura 2.6 se observa el retraso de 25 muestras la señal azul es la original y la señal roja la que contiene la marca oculta, es claro que la señal es la misma pero la información oculta se encuentra en otra parte, es decir 25 muestras antes.

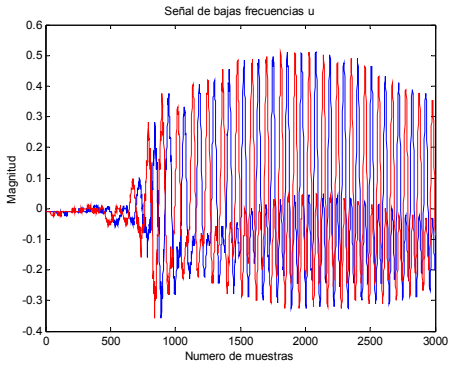



**Figura 2.6** Gráfica de la perdida de 50 muestras

La tabla 2.8 muestra los valores recuperados los que están marcados en negritas indican un valor equivocado. Cuando se eliminan 50 muestras es claro que estos valores se refieren a otros valores distintos ya que las posiciones de la información no se encuentran donde se ocultaron originalmente.

| $I_1$ | $I_2$ | *M1* | *M2* | *Modificado1* | *Modificado2* | *Recuperado1* | *Recuperado2* |
|---|---|---|---|---|---|---|---|
| *776* | *1073* | *-0.0442* | *0 .0844* | -0.0842 | 0.0544 | -0.0627 | 0.3127 |
| ***780*** | ***1069*** | ***-0.0935*** | ***-0.0238*** | **-0.0535** | **-0.0838** | **-0.0581** | **0.3333** |
| ***784*** | ***1065*** | ***-0.0860*** | ***-0.1151*** | **-0.0460** | **-0.1751** | **-0.1301** | **0.3111** |
| ***790*** | ***1061*** | ***-0.1547*** | ***-0.2006*** | **-0.1247** | **-0.2506** | **-0.0869** | **0.2747** |
| *794* | *1057* | *-0.1297* | *-0.2210* | -0.1497 | -0.2110 | -0.0670 | 0.2076 |

**Tabla 2.8** Valores recuperados cuando se eliminan 50 muestras

*Bits con error = [2 3 4 7 8 9 10 12 13 14 16 17 18 21 22 26 30 32 35 36 40 45 46 48 52 53 54 55 56 57 60 63 66 68 69 70 71 73 74]*

*Mensaje=/□ H*

De los resultados obtenidos se puede concluir que el algoritmo Patchwork no presenta una solución para obtener marcas de agua robustas en archivos de audio en formato wav. En esta tesis se propone usar un grupo de valores en dominio wavelet de la señal de bajas frecuencias *u* como se menciona en él capítulo 1 se utiliza para este caso el valor máximo absoluto y un grupo de valores alrededor de este valor máximo absoluto, en la señal de bajas frecuencias. En la figura 2.7 se muestra la señal de bajas frecuencias que se utilizará.

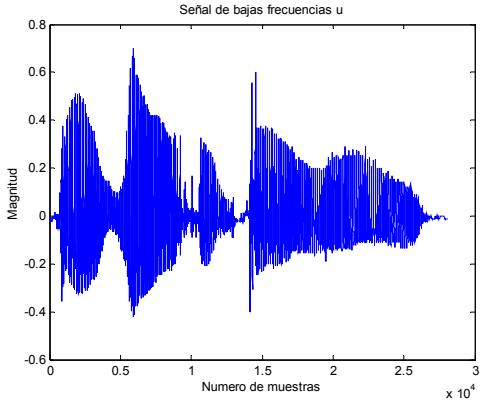



**Figura 2.7** Señal de frecuencias bajas

Para este archivo el vector de referencia ($Vector_{ref}$) contiene los valores de las posiciones 5800-5899, este vector tiene una posición inicial $Pos_{vector}$ que es el primer valor del vector y se correlaciona con la señal *u* que contiene la marca que se recupera usando la fórmula 1.51 y la posición donde se encuentre se le sumara a $Po_1$, el resto de la señal será utilizada para ocultar la información y es donde se aplica el algoritmo Patchwork, la cual se muestra en la figura 2.8. Al realizar esto la fórmula 2.4 queda de la siguiente forma.

$$Po_1 = 2\sum_{i=1}^{m} Cla(i) + Pos_{vector} \qquad (2.10)$$

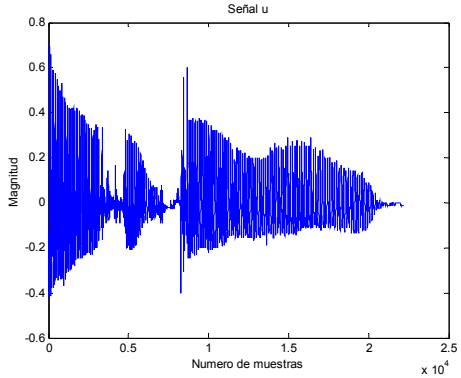



**Figura 2.8** Parte de la señal de frecuencias bajas donde se va a ocultar la información

A partir de esta correlación se tiene que el vector se encuentra las posiciones 5855-5954 por lo tanto la información se busca en el resto de la señal por medio de la clave *Cla* y se generan los índices para localizar la información.

| $I_1$ | $I_2$ | M1 | M2 | Modificado1 | Modificado2 | Recuperado1 | Recuperado2 |
|---|---|---|---|---|---|---|---|
| 6631 | 6928 | -0.0459 | 0.0235 | -0.0559 | 0.0235 | -0.0487 | 0.0198 |
| 6635 | 6924 | -0.0475 | 0.0420 | -0.0375 | 0.0620 | -0.0297 | 0.0634 |
| 6639 | 6920 | -0.0475 | 0.0578 | -0.0475 | 0.0778 | -0.0412 | 0.0703 |
| 6643 | 6916 | -0.0472 | 0.0773 | -0.0572 | 0.0873 | -0.0497 | 0.0773 |
| 6647 | 6912 | -0.0456 | 0.0747 | -0.0856 | 0.0547 | -0.0806 | 0.0460 |

**Tabla 2.9** Resultados recuperados al eliminar 50 muestras usando $V_{ref}$

*Bits con error = []* *// No hay error en los bits*

*Mensaje = Esime*

Pero un punto importante que se debe considerar es si el número de muestras pérdidas es par o impar, el caso anterior tiene una pérdida de muestras par, es muy importante ya que la wavelet se desplaza en dos muestras. A continuación se muestra el resultado obtenido tras eliminar solo una muestra del archivo 2 al inicio el vector de referencia es de 5800-5899, el vector de posición se localiza $Pos_{vector}=5800$.

*Bits con error=[1 2 5 6 7 9 12 13 15 17 23 24 25 27 29 30 35 38 39 40 41 42 44*
*48 50 52 55 56 61 63 65 69 71 72 74]*

*Mensaje=□6{ s*

Se puede observar que la información no se recupera, para solucionar este problema se deben realizar dos correlaciones, la primera se realiza con el archivo original y se aplica la wavelet y se correlaciona, la segunda se hace con una muestra desplazada del archivo original y se aplica la wavelet y se correlaciona, la máxima de ambas correlaciones es la que se toma usando la fórmula 1.51. Realizando ambas correlaciones se obtiene lo siguiente.

$R_{xx}=0.9951$
$Pos_{vector}=5800$

$R_{xx}=1$
$Pos_{vector}=5799$

Es claro cual de las dos correlaciones se va a utilizar por lo tanto la posición que se utiliza es 5799 en lugar de 5800 y lo recuperado es lo siguiente.

*Bits con error = []* *// No existen bits con error*

*Mensaje =Esime*

En los siguientes resultados se muestra que a pesar de los cambios realizados al algoritmo Patchwork, el algoritmo falla cuando se aplica un convertidor de formato y el código no es capaz de corregir los errores debido a su gran número. La prueba que se realiza es sencilla y solo consiste en convertir el archivo a un formato de 16 bits y posteriormente a su formato de 8 bits. En la figura 2.9 se muestran dos señales, la señal roja corresponde al archivo original y la azul al archivo recuperado, ambos archivos contienen la misma cantidad de muestras 56200 pero el archivo recuperado tiene un escalamiento, el vector de referencia $Vector_{ref}$ es el mismo y la posición del vector después de correlacionar con el recuperado se encuentra en $Pos_{vector}$=5800 es decir que las posiciones se encuentran en el mismo lugar y la información recuperada se muestra en la tabla 2.10.

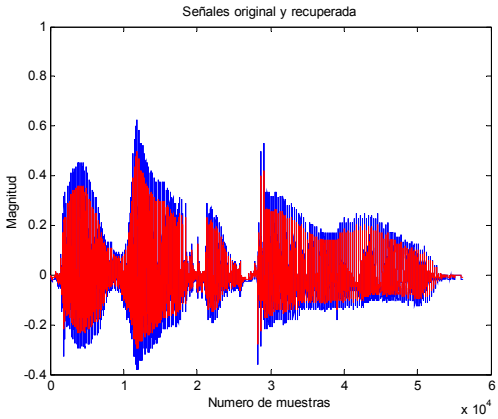



**Figura 2.9** Comparación de ambas señales

| $I_1$ | $I_2$ | M1 | M2 | Modificado1 | Modificado2 | Recuperado1 | Recuperado2 |
|---|---|---|---|---|---|---|---|
| 6631 | 6928 | -0.0459 | 0.0235 | -0.0559 | 0.0235 | -0.0601 | 0.0403 |
| 6635 | 6924 | -0.0475 | 0.0420 | -0.0375 | 0.0620 | -0.0470 | 0.0781 |
| 6639 | 6920 | -0.0475 | 0.0578 | -0.0475 | 0.0778 | -0.0511 | 0.0931 |
| 6643 | 6916 | -0.0472 | 0.0773 | -0.0572 | 0.0873 | -0.0767 | 0.0615 |
| 6647 | 6912 | -0.0456 | 0.0747 | -0.0856 | 0.0547 | -0.0721 | 0.0586 |

**Tabla 2.10** Resultados recuperados al cambiar al formato de 16 bits

*Bits con error = [ 4 6 7 8 9 12 13 14 15 16 17 18 19 23 25 26 27 28 29 32 33 36 40 42 43 47 50 51 53 55 57 58 59 60 61 62 63 67 68 69 70 71 72 73]*

*Mensaje =* □ □*DJ*

$$Matrizrecuperada = \begin{vmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \end{vmatrix}$$

$$Matrizdecodificada = \begin{vmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \end{vmatrix}$$

Existe un cambio de los valores originales con respecto a los recuperados. Se toman 10 valores del vector de referencia original y se encuentra el factor por el cual se tiene que multiplicar el elemento del vector de referencia para que tenga el mismo valor que el vector recuperado. Estos datos se muestran en la tabla 2.11.

| ***Posición*** | ***Vector$_{ref}$*** | ***Vector$_{rec}$*** | ***Factor*** |
|---|---|---|---|
| *5880* | *-0.1251* | *-0.1613* | *1.2888* |
| *5881* | *-0.0924* | *-0.1235* | *1.3367* |
| *5882* | *-0.0482* | *-0.0694* | *1.4380* |
| *5883* | *-0.0043* | *-0.0084* | *1.9387* |
| *5884* | *0.0417* | *0.0498* | *1.1931* |
| *5885* | *0.0753* | *0.0885* | *1.1752* |
| *5886* | *0.0975* | *0.1196* | *1.2266* |
| *5887* | *0.1189* | *0.1410* | *1.1858* |
| *5888* | *0.1215* | *0.1436* | *1.1818* |
| *5889* | *0.1219* | *0.1440* | *1.1812* |

**Tabla 2.11** Comparación de valores de ambos vectores

De los resultados se tiene que cada elemento se escala por un factor distinto esto indica que no es un cambio constante por lo tanto no se tiene forma de saber cual era el valor anterior y el algoritmo Patchwork falla ante este problema.

## 2.5 MÉTODO PROPUESTO

A continuación se presenta un método para introducir marcas de agua binarias en archivos de audio wav. Los archivos que se emplean tienen una frecuencia de 44100 Hz, además estos archivos pueden tener una codificación de 8 ó 16 bits. El método propuesto es robusto contra cambios de formato, frecuencia y compresión. La característica principal de esta técnica es que utiliza la cuantización de los archivos de audio wav en formato de 8 bits para introducir el bit a ocultar, esto se consigue intercambiando las muestras de un intervalo especifico, dependiendo si el bit que desea ocultar es un uno o un cero. Esta técnica se puede aplicar tanto a los archivos de 8 bits como a los archivos de 16 bits usando la misma técnica de cuantización de 8 bits y su diagrama a bloques se muestra en la figura 2.10.

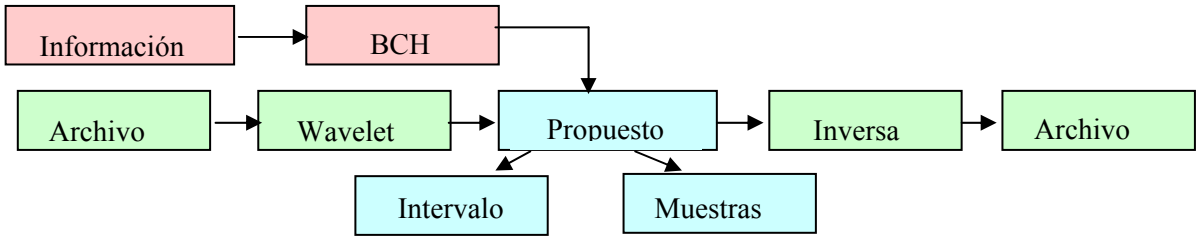
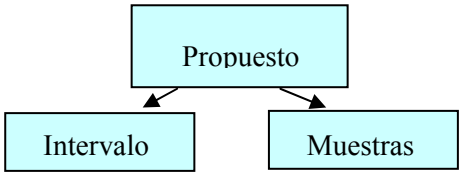



**Figura 2.10** Esquema para ocultar la información usando el método propuesto

Los bloques rojos y verdes realizan la misma función que en el algoritmo Patchwork.

**Bloques del método propuesto**




Estos bloques azules indican la forma en que trabaja el método propuesto, después de aplicar la descomposición wavelet se elige un intervalo de valores, este intervalo de valores se elige de tal forma que dañe lo menos posible al archivo marcado, posteriormente se oculta la información utilizando un par de muestras de referencia en el dominio wavelet que se intercambian dependiendo del bit que se oculta.

A continuación se recordara el problema de ocultar marcas de agua en archivos de audio modificando alguna cifra significativa, el principal problema es su debilidad contra cambios de formato ó escalamiento de las muestras. Por otro lado, también se debe considerar la pérdida de muestras, que como se vera mas adelante son diferentes para cada archivo, ya que cada archivo tiene características diferentes. Para observar este cambio en los archivos se convertirá un archivo usando un convertidor comercial. Las características del archivo que se usará se muestran en la tabla 2.12 y corresponden a la figura 2.1:

| Frecuencia | Bits | Número de muestras | Energía |
|---|---|---|---|
| 44100 | 8 | 56200 | 1984 |

**Tabla 2.12** Características del archivo de prueba

Primero se convertirá el archivo a una frecuencia de 22050 y 16 bits y después se convierte a su formato original usando el mismo convertidor y sé verán los cambios que sufre el archivo al pasar por este proceso;

| Frecuencia | Bits | Numero de muestras | Energía |
|---|---|---|---|
| 44100 | 8 | 56184 | 2054 |

**Tabla 2.13** Nuevas características del archivo de prueba recuperado

Comparando la tabla 2.12 y 2.13 se observa que se tiene una diferencia en el número de muestras para ser exacto se tienen 16 muestras menos también se observa que la cantidad de energía es mayor en el archivo recuperado a pesar de tener un menor número de muestras.

En las tablas 2.14 y 2.16 se muestran algunos valores de ambos archivos en el mismo intervalo usando la fórmula de correlación 1.51, para mostrar los cambios de sus valores, la forma de ubicar este intervalo es detectar el $Vector_{ref}$ de nuestra señal original $u$ que esta localizada en la posición 5800 y al correlacionar este vector con la señal recuperada se encuentra este valor en la posición 5801, es decir una posición después y en la tabla 2.14 se muestran algunos valores.

| **Posición** | **Magnitud original** | **Posición** | **Magnitud recuperada** |
|---|---|---|---|
| *5880* | -0.1251 | 5881 | -0.1613 |
| *5881* | -0.0924 | 5882 | -0.1235 |
| *5882* | -0.0482 | 5883 | -0.0696 |
| *5883* | -0.0043 | 5884 | -0.0073 |
| *5884* | 0.0417 | 5885 | 0.0435 |
| *5885* | 0.0753 | 5886 | 0.0885 |
| *5886* | 0.0975 | 5887 | 0.1190 |
| *5887* | 0.1189 | 5888 | 0.1446 |
| *5888* | 0.1215 | 5889 | 0.1462 |
| *5889* | 0.1219 | 5890 | 0.1434 |

**Tabla 2.14** Escalamiento de valores

De la tabla 2.14 es fácil observar que los valores fueron modificados, y también que los cambios no fueron hechos de una forma constante ya que cada valor tiene un incremento diferente. Como se vio anteriormente, la forma de realizar estos cambios es desconocida. También es lógico que modificando alguna cifra para ocultar la información el cambio de formato altera los valores y por lo tanto la información se perderá por lo cual el método de modificar alguna cifra no resulta adecuado para un convertidor de formato. Hasta ahora solo se ha visto que sucede cuando se usa un convertidor de formato para pasar el archivo wav a otro archivo wav, pero falta mostrar que sucede si se convierte este archivo a un formato como mp3 y nuevamente se convierte el archivo a su formato wav original. Como observa el la tabla 2.16 los valores son modificados a otras cantidades distintas, primero se muestran las características del archivo resultante después de realizar la conversión a mp3 y nuevamente a wav en la tabla 2.15:

| Frecuencia | Bits | Número de muestras | Energía |
|---|---|---|---|
| 44100 | 8 | 52992 | 3208 |

**Tabla 2.15** Características tras convertir a mp3

Comparando la tabla 2.12 y la tabla 2.15, es claro que la pérdida de muestras es mas que considerable al convertir de un formato wav a mp3, aquí se pierden 3208 muestras y la energía para este caso es de 2042. En la tabla 2.16 que los valores de las muestras en el mismo intervalo recordando que el $Vector_{ref}$ de la señal original esta en la posición 5800, para este caso la correlación nos indica que este valor se encuentra en la posición 6520.

| Posición | Magnitud original | Posición | Magnitud recuperada |
|---|---|---|---|
| *5880* | -0.1251 | 6520 | -0.2469 |
| *5881* | -0.0924 | 6521 | -0.2039 |
| *5882* | -0.0482 | 6522 | -0.1292 |
| *5883* | -0.0043 | 6523 | -0.0432 |
| *5884* | 0.0417 | 6524 | 0.0357 |
| *5885* | 0.0753 | 6525 | 0.1039 |
| *5886* | 0.0975 | 6526 | 0.1505 |
| *5887* | 0.1189 | 6527 | 0.1958 |
| *5888* | 0.1215 | 6528 | 0.2095 |
| *5889* | 0.1219 | 6529 | 0.2122 |

**Tabla 2.16** Nuevos valores tras convertir a mp3

Las tablas 2.14 y 2.16 muestran que en ambos casos los valores recuperados son mayores que los valores originales y que el grado de aumento en las muestras es distinto en cada caso. También la cantidad de muestras que se pierden es distinta para cada archivo, así como la energía en cada archivo recuperado. Por lo cual se tuvo que buscar otra forma de ocultar la información, en la figura 2.11 se muestran ambas señales tras aplicar el formato mp3, la recuperada en azul y se compara con la señal original en rojo, es evidente que se necesita una forma distinta de ocultar la marca y que esta pueda ser recuperada.

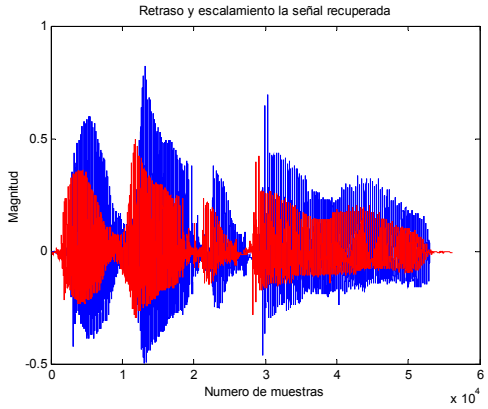



**Figura 2.11** Comparación después de aplicar mp3

## Codificación de 8 bits

Considerando los problemas anteriores se decidió ocultar la marca de una forma distinta previendo la pérdida de muestras y el cambio de valores, y es la siguiente usando los valores de cuantización de los archivos wav de 8 bits, pero estos valores de cuantización fueron obtenidos después de aplicar un nivel de una wavelet 'Daubechies'.

La forma de encontrar estos valores es la siguiente primero se aplica un nivel de descomposición una wavelet Daubechies para este caso se utilizo la wavelet 'Daubechies 1' ya que es la más sencilla de las 'Daubechies', la señal que se utiliza es la mostrada en la figura 2.2.

En la tabla 2.17 se muestra que sucede cuando se oculta la marca de agua en la tercera cifra, en la primera columna se muestran algunos valores después de aplicar una wavelet 'Db1' a la señal original, en la segunda columna se muestran los valores modificando la tercera cifra y en la tercera columna se muestran los valores recuperados. La finalidad de realizar esto es para recordar que tratar de ocultar una marca modificando esta cifra no es conveniente ya que la cuantización de 8 bits modifica los valores.

| ***Valores originales*** | ***Valores modificados*** | ***Valores recuperados*** |
|---|---|---|
| *-0.1768* | *-0.1788* | *-0.1768* |
| *-0.1878* | *-0.1868* | *-0.1878* |
| *-0.2320* | *-0.2350* | *-0.2320* |
| *-0.2541* | *-0.2561* | *-0.2541* |
| *-0.2099* | *-0.2089* | *-0.2099* |
| *-0.1878* | *-0.1858* | *-0.1878* |
| *-0.2320* | *-0.2370* | *-0.2320* |
| *-0.2645* | *-0.2662* | *-0.2652* |
| *-0.2486* | *-0.2486* | *-0.2486* |
| *-0.2375* | *-0.2365* | *-0.2372* |

**Tabla 2.17** Modificando cifras en formato de 8 bits

De la tabla 2.17 es claro que los valores modificados no se conservan ya que la cuantización de 8 bits nos modifica los valores ya que dentro de sus niveles no existen tales valores, observando este detalle se decidió encontrar los valores de cuantización de la siguiente forma se tomo una muestra y se le asigna el valor de cero luego se aplica la wavelet inversa y sé escribió el archivo a su formato de 8 bits nuevamente se abre el archivo y se le aplica un nivel de wavelet 'Db1' y verificando su valor para saber si este valor de cero cambio o fue conservado después se procedió a incrementar el valor de esta muestra hasta tener un cambio en la muestra recuperada. Como se ve en la tabla 2.18.

| **Valor original** | **Valor modificado** | **Valor recuperado** |
|---|---|---|
| 0 | -0.005 | 0 |
| 0 | -0.004 | 0 |
| 0 | -0.003 | 0 |
| 0 | -0.002 | 0 |
| 0 | -0.001 | 0 |
| 0 | 0 | 0 |
| 0 | 0.001 | 0 |
| 0 | 0.002 | 0 |
| 0 | 0.003 | 0 |
| 0 | 0.004 | 0 |
| 0 | 0.005 | 0 |
| 0 | 0.006 | 0.0110 |
| 0 | 0.007 | 0.0110 |

**Tabla 2.18** Localización de los niveles

De los resultados de la tabla 2.18 el intervalo de valores de –0.005 a 0.005 que se le fue incrementando a la muestra siempre es modificado a un valor de cero para el dominio wavelet.

Este sería el primer nivel de cuantización o volumen, para encontrar el resto de los volúmenes se tienen que seguir incrementando los valores, considere que los archivos emplean una cuantización que es simétrica además de los 8 bits que se utilizan se debe tener en consideración que 7 son para los niveles de cuantización y 1 es para el signo.

En la tabla 2.19 se muestran mas niveles de cuantización en dominio wavelet

| Intervalo | Valor |
|---|---|
| -0.005 – 0.005 | 0 |
| 0.006 – 0.015 | 0.0110 |
| 0.0166 – 0.027 | 0.0221 |
| 0.028 – 0.038 | 0.0331 |
| 0.039 – 0.049 | 0.0442 |
| 0.05 – 0.060 | 0.0552 |
| 0.061 – 0.071 | 0.0663 |

**Tabla 2.19** Niveles de cuantización

Estos niveles pueden ser encontrados combinando la wavelet Daubechies 1 y los niveles de cuantización del formato de 8 bits wav simétrico de la siguiente forma:

$$h(n) = \left( \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}} \right) \tag{2.11}$$

Y teniendo en cuenta que son 256 niveles, y recordando que son simétricos solo se requieren 128. Donde el nivel más bajo mayor que cero es:

$$\frac{1}{128} = 0.0078125 \tag{2.12}$$

Para realizar el respectivo nivel de la wavelet Daubechies 1 se requieren dos muestras por lo tanto 2.11 queda de la siguiente forma.

$$\left( \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}} \right) \cdot \left( \frac{i}{\sqrt{2}}, \frac{j}{\sqrt{2}} \right) = \left( \frac{i+j}{128\sqrt{2}} \right) \tag{2.13}$$

Con la fórmula 2.13 se obtienen los diferentes niveles para el caso de $i = 1$ y $j = 1$ se obtiene el primer nivel de cuantización en el dominio wavelet:

$$\left( \frac{2}{128\sqrt{2}} \right) = 0.01104$$

El cual corresponde al primer valor de la tabla 2.19. De los resultados de la tabla 2.19 es claro que no se pueden modificar tan fácilmente las cifras de las muestras cuando se esta trabajando en archivos de 8 bits.

Por otro lado si se utilizan archivos de 16 bits no se tendría ningún problema con modificar alguna cifra los cambios se conservan, pero se debe recordar un detalle importante un archivo de 16 bits puede ser convertido a 8 bits y después convertirlo a su formato original modificando sus valores.

**Implementación**

Para ocultar la información en el archivo se menciono que se va a utilizar la cuantización de los archivos wav de 8 bits y los niveles de cuantización fueron obtenidos después de aplicar una wavelet 'Daubechies' a las muestras del archivo. Lo primero que se debe hacer es aplicar un nivel de la wavelet. Se usará la señal de la figura 2.1 y se le aplica un nivel de la 'Daubechies'. Se ocultará la información en la señal de frecuencias bajas *u*, por cada bit que se oculta se requieren dos muestras, la Daubechies empleada es la 'Db3' y su gráfica se muestra en la figura 2.12:

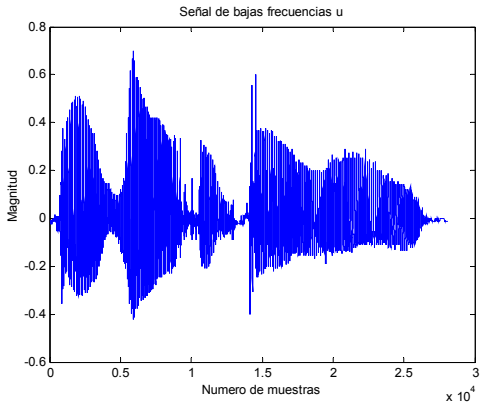



**Figura 2.12** Señal de bajas frecuencias *u*

Esta señal tiene la mitad de la longitud de la señal original, la energía de la señal original es de 1984, la subseñal de frecuencias bajas contiene una energía de 1982.3, es decir que en esta señal contiene más del 99% de la energía total, por esta razón se decide que se utilizará la señal de frecuencias bajas para ocultar la información. Una vez realizado esto, lo siguiente es determinar un intervalo de la señal donde se modificaran las muestras del intervalo para ocultar la información, se debe tener cuidado al elegir el intervalo para no alterar valores que contengan demasiada energía y al modificarlos puedan dañar al archivo que se utiliza.

En esta tesis se eligen las muestras cuya magnitud absoluta se encuentran dentro del intervalo de 0.105 a 0.193 después del vector de referencia. Así se hará para todos los archivos empleados ya que este grupo de muestras no altera en forma considerable la energía del archivo y siempre se tiene una buena cantidad de muestras disponibles para poder ocultar la información.

Se debe explicar un punto muy importante las posiciones de las muestras en este intervalo son aleatorias por lo tanto no llevan una secuencia y no se puede saber donde se localizan entonces estas posiciones deben ser almacenadas en una base de datos y una vez recibido el archivo aplicar la wavelet y buscar en estas posiciones. A continuación se muestran algunas posiciones para ilustrar lo anterior usando el archivo 1.

*Intervalo =[7 8 9 16 17 38 39 44 45 56 57 58 68*
*69 70 71 72 81 82 100 101 102 109 110 111 132*
*137 138 147 148 149 150 151 160 161 162 163 164 165]*

Aplicando este criterio se procede a emplear la señal de frecuencias bajas y determinar cuantas muestras se encuentran dentro del intervalo. La cantidad de muestras dentro de este intervalo es de 5077. Para este ejemplo se usará él número '9' para ilustrar el proceso.

Su valor en código ASCII es 57

Posteriormente se convierte a binario

1 0 0 1 1 1

Una vez aplicado esto se procede en codificar este vector con el BCH y nuestra cadena de bits queda de la siguiente forma.

0 1 0 0 0 0 0 1 1 0 0 1 1 1 0

Después de codificar la información se empieza a ocultarla en las muestras que se tienen dentro del intervalo, las muestras de referencia son 0.1215 y 0.1878 para ocultar el primer bit para este ejemplo es cero, la primera muestra será la menor 0.1215 y la segunda muestra será 0.1878 para ocultar un bit 1 se realiza lo contrario. Como se dispone de un gran número de muestras y solo se ocultan 15 bits se dispersa la información en todo el intervalo de acuerdo a la siguiente expresión.

$$Separacion = \frac{y}{2 \times I_n} \qquad (2.16)$$

*y* es el tamaño de *Intervalo*

Para el intervalo la separación es de 16 y el primer par de muestras originales son las siguientes:

*M1(Intervalo(1))=0.1857*

*M2(Intervalo(17))=0.1629*

Para el bit ‘0’ estas muestras se reemplazan de la siguiente forma

*M1(Intervalo(1))=0.1215*

*M2(Intervalo(17))=0.1878*

Este procedimiento se repite para todos los bits que se oculten en las muestras del archivo original de esta forma se asegura que la información soportara los cambios de formato, filtrado espacial y codificación.

## 2.6 DISPERSION DE LA INFORMACION

Al implementar los algoritmos y realizar las pruebas se utilizo un convertidor comercial, en este caso se debe tener en cuenta un detalle, el algoritmo busca un grupo de muestras en un intervalo especifico para ocultar la marca, estas muestras pueden estar en cualquier parte de la señal resultante después de aplicar la wavelet, como un ejemplo se muestra el siguiente caso. Usando la señal de la figura 2.1 se oculta la palabra ‘Politécnico’ y se aplica la wavelet inversa, a continuación se resta la señal original de la señal con marca su gráfica se muestra a continuación;

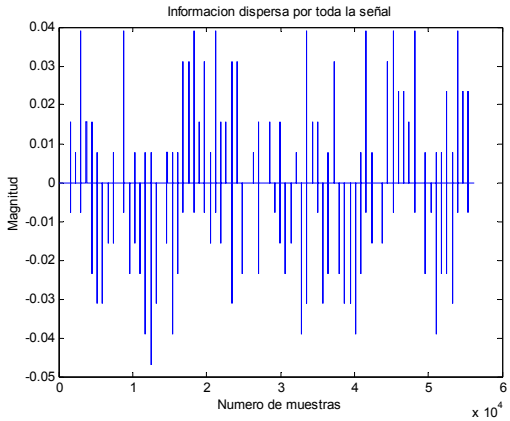



**Figura 2.13** Información dispersa

De la figura 2.13 se observa que la información esta dispersa por toda la señal, pero al tratar de recuperar la información después de aplicar un convertidor de formato el número de muestras del intervalo es diferente.

Al tratar de recuperar la información oculta se encuentra que las ultimas posiciones donde esta oculta la marca, ya no existen en el archivo recuperado. Esto se debe a que el archivo original tenia 56200 muestras y el archivo recuperado solo tiene 52992 muestras se han perdido mas de 3000 muestras. Y al aplicar la wavelet y tratar de encontrar la información de acuerdo a las posiciones donde se encuentra oculta la marca las ultimas posiciones ya no existen. Para solucionar este problema el algoritmo se utiliza de la siguiente forma.

Primero se toma un vector de referencia para localizar a partir de que punto empieza nuestra información oculta, después se ocupan las muestras siguientes a este punto de referencia excepto las últimas 4000 muestras para evitar que la pérdida de muestras afecten. Realizando esto se puede recuperar la información correctamente. Otra solución es ocultar la información unas 4000 muestras después del inicio la figura 2.14 muestra esta forma de ocultar la información:

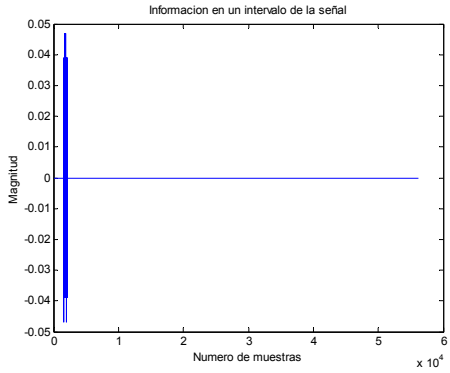



**Figura 2.14** Información Oculta

O también se puede duplicar la información para protegerla en caso de que se pierda una parte de la señal y la gráfica se muestra en 2.15:

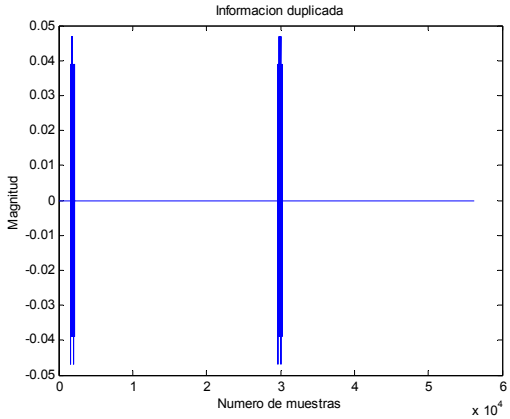



**Figura 2.15** Información duplicada

# Capítulo 3

## Pruebas Realizadas

Todas las pruebas que se realizaron utilizaron el algoritmo Patchwork y el método propuesto, y de los resultados obtenidos de las diferentes pruebas se realiza una comparación entre ambos métodos para encontrar cual es mejor. Para el algoritmo Patchwork se debe recordar que la información se oculto en la segunda cifra significativa.

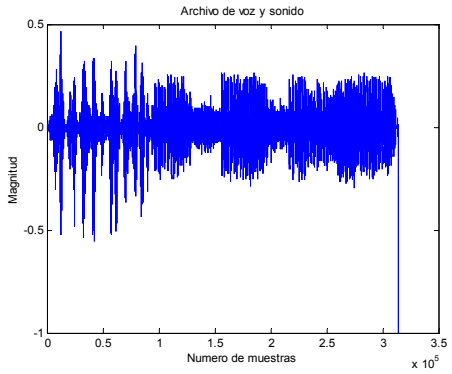



**Figura 3.1** Archivo 1 de voz y sonido

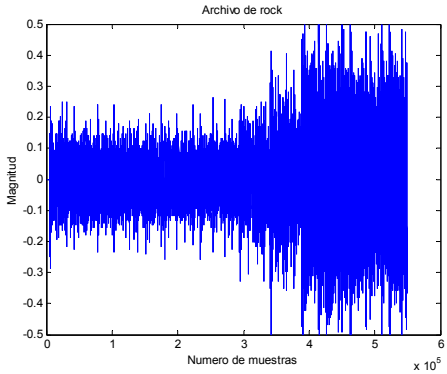



**Figura 3.2** Archivo 2 de música rock

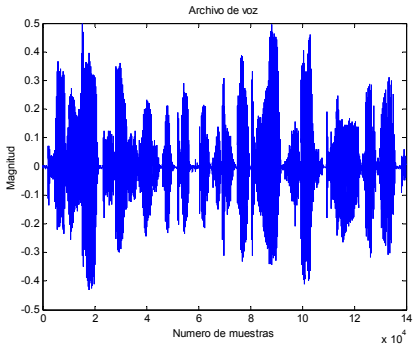



**Figura 3.3** Archivo 3 de voz

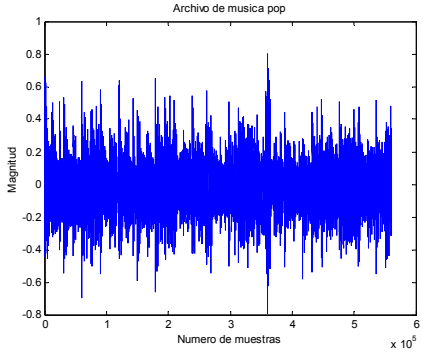



**Figura 3.4** Archivo 4 de música pop

En la tabla 3.1 se muestran algunas de las características de los archivos empleados. Los archivos utilizados en esta tesis se muestran en las figuras 3.1, 3.2, 3.3 y 3.4 se puede observar que son diferentes entre sí. La elección de estos archivos se debe a que contienen solo voz como la figura 3.3 y diferentes estilos de música como la figura 3.2 y 3.4. Así como voz y efectos de sonido como lo muestra la figura 3.1.

| Archivos | Frecuencia | Codificación | Muestras | Energía |
|---|---|---|---|---|
| Voz y Sonido | 44100 | 8 | 314304 | 1632.1 |
| Rock | 44100 | 8 | 550000 | 3991.7 |
| Voz | 44100 | 16 | 140000 | 1074.7 |
| Pop | 44100 | 16 | 561000 | 7770 |

**Tabla 3.1** Características de los archivos de prueba

## 3.1 Ocultar y recuperar la marca de agua

La primera prueba tiene como finalidad comprobar que ambos métodos pueden recuperar la información oculta. Se oculta la información con ambos métodos y se recupera con el proceso inverso y se comprueba que el resultado sea el archivo en su formato original. Si los métodos fallan en esta prueba no tiene sentido realizar pruebas más complicadas. Para ambos métodos se usa como clave la palabra 'Esime' y la información a ocultar es la palabra 'Zacatenco'. Las tablas de los resultados muestran los datos originales, los modificados y los recuperados así como el bit oculto correspondiente. En todas las pruebas realizadas los resultados se presentaran en este orden. Los bits ocultos corresponden a la primera columna de la matriz codificada de la palabra 'Zacatenco', la cual se muestra a continuación.

$$Matrizcodificada = \begin{vmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \end{vmatrix}$$

*Bits primera columna = [0 1 0 1 0 1 0 0 1]*

Primero se van a realizar las pruebas utilizando el algoritmo Patchwork y posteriormente el método propuesto.

### 3.1.1 Método Patchwork (Cifra 2)

Los resultados para el archivo uno, muestran que la información es recuperada correctamente sin ningún error en los bits, y la diferencia entre ambas cifras modificadas se mantiene después de aplicar la wavelet inversa y escribir el archivo, como se mostrará en los resultados a continuación.

*Bits con error =[]* *// No hay bits con error*
*Mensaje = 'Zacatenco'*

| *Ind1* | *Ind2* | *M1* | *M2* | *Mod1* | *Mod2* | *Rec1* | *Rec2* | *Bit* |
|---|---|---|---|---|---|---|---|---|
| *6798* | *7469* | *-0.0312* | *0* | *-0.0112* | *0.0400* | *-0.0098* | *0.0365* | *0* |
| *6803* | *7464* | *-0.0205* | *-0.0004* | *-0.0305* | *0* | *-0.0296* | *-0.0015* | *1* |
| *6808* | *7459* | *-0.0664* | *0.0125* | *-0.0364* | *0.0625* | *-0.0261* | *0.0596* | *0* |
| *6813* | *7454* | *0.0138* | *-0.0203* | *0.0438* | *-0.0103* | *0.0406* | *-0.0104* | *1* |
| *6818* | *7449* | *-0.0201* | *-0.0331* | *-0.0201* | *-0.0531* | *-0.0201* | *-0.0519* | *0* |
| *6823* | *7444* | *0.0549* | *-0.0446* | *0.0749* | *-0.0446* | *0.0728* | *-0.0409* | *1* |
| *6828* | *7439* | *0.0810* | *-0.0320* | *0.0510* | *-0.0820* | *0.0504* | *-0.0838* | *0* |
| *6833* | *7434* | *-0.0572* | *-0.0224* | *-0.0372* | *-0.0624* | *-0.0277* | *-0.0573* | *0* |
| *6838* | *7429* | *0.0485* | *-0.0096* | *0.0485* | *-0.0196* | *0.0386* | *-0.0122* | *1* |

**Tabla 3.2** Muestras recuperadas del archivo 1 usando Patchwork

Los bits de la columna recuperados son los siguientes

*Bits primera columna = [0 1 0 1 0 1 0 0 1]  // No hay bits con error*

Los cuales corresponden a la primera columna de la matriz codificada.

Para el archivo dos también se tiene que la información recuperada esta correcta y no se presenta ningún bit con error.

*Bits con error=[]  // No hay bits con error*
*Mensaje = 'Zacatenco'*

| *Ind1* | *Ind2* | *M1* | *M2* | *Mod1* | *Mod2* | *Rec1* | *Rec2* | *Bit* |
|---|---|---|---|---|---|---|---|---|
| *6798* | *7469* | *-0.0353* | *0.0308* | *-0.0253* | *0.0508* | *-0.0177* | *0.0485* | *0* |
| *6803* | *7464* | *-0.0590* | *-0.0701* | *-0.0890* | *-0.0501* | *-0.0778* | *-0.0513* | *1* |
| *6808* | *7459* | *-0.0014* | *-0.0750* | *-0.0314* | *-0.0650* | *-0.0293* | *-0.0600* | *0* |
| *6813* | *7454* | *-0.0227* | *-0.0349* | *-0.0527* | *-0.0249* | *-0.0526* | *-0.0213* | *1* |
| *6818* | *7449* | *-0.1093* | *-0.0439* | *-0.1193* | *-0.0439* | *-0.1093* | *-0.0376* | *0* |
| *6823* | *7444* | *-0.1257* | *0.0333* | *-0.1557* | *0.0233* | *-0.1519* | *0.0234* | *1* |
| *6828* | *7439* | *-0.0271* | *-0.0002* | *-0.0071* | *-0.0302* | *0.0016* | *-0.0315* | *0* |
| *6833* | *7434* | *0.0957* | *-0.0343* | *0.0557* | *-0.0843* | *0.0464* | *-0.0828* | *0* |
| *6838* | *7429* | *0.1121* | *0.0340* | *0.1421* | *0.0140* | *0.1412* | *0.0121* | *1* |

**Tabla 3.3** Muestras recuperadas del archivo 2 usando Patchwork

Los bits de la columna recuperados son los siguientes

*Bit primera columna =[ 0 1 0 1 0 1 0 0 1]  // No hay bits con error*

Los archivos 1 y 2 están codificados a 8 bits, por esta razón las cifras varían ligeramente de sus valores modificados ya que la codificación de 8 bits altera estos valores al aplicar la wavelet inversa y escribir el archivo pero la diferencia entre ambas cifras se mantiene y se recupera la información correctamente. Este detalle no ocurre para archivos de 16 bits como se muestra en los resultados que se presentan a continuación.

En el archivo 3 tampoco se tienen bits con error y la información se recupera correctamente

*Bits con error=[]* *// No hay bits con error*
*Mensaje='Zacatenco'*

| ***Ind1*** | ***Ind2*** | ***M1*** | ***M2*** | ***Mod1*** | ***Mod2*** | ***Rec1*** | ***Rec2*** | ***Bit*** |
|---|---|---|---|---|---|---|---|---|
| *6798* | *7469* | *-0.0172* | *-0.0014* | *-0.0072* | *-0.0314* | *-0.0072* | *-0.0314* | *0* |
| *6803* | *7464* | *-0.0277* | *0.0146* | *-0.0477* | *0.0146* | *-0.0477* | *0.0146* | *1* |
| *6808* | *7459* | *-0.0048* | *0.0200* | *-0.0048* | *0.0300* | *-0.0048* | *0.0300* | *0* |
| *6813* | *7454* | *-0.0226* | *-0.0213* | *-0.0426* | *-0.0113* | *-0.0426* | *-0.0113* | *1* |
| *6818* | *7449* | *-0.0054* | *0.0017* | *-0.0054* | *0.0317* | *-0.0054* | *0.0317* | *0* |
| *6823* | *7444* | *-0.0291* | *-0.0027* | *-0.0391* | *-0.0027* | *-0.0391* | *-0.0027* | *1* |
| *6828* | *7439* | *-0.0309* | *-0.0250* | *-0.0209* | *-0.0550* | *-0.0209* | *-0.0550* | *0* |
| *6833* | *7434* | *-0.0521* | *-0.0012* | *-0.0221* | *-0.0512* | *-0.0221* | *-0.0512* | *0* |
| *6838* | *7429* | *-0.0172* | *0.0039* | *-0.0372* | *0.0039* | *-0.0372* | *0.0039* | *1* |

**Tabla 3.4** Muestras recuperadas del archivo 3 usando Patchwork

Los bits de la columna recuperados son los siguientes

*Bits primera columna =[ 0 1 0 1 0 1 0 0 1]* *// No hay bits con error*

En el archivo 4 tampoco se tienen bits con error y la información se recupera correctamente

*Bit=[]* *// No hay bits con error*
*Mensaje='Zacatenco'*

| ***Ind1*** | ***Ind2*** | ***M1*** | ***M2*** | ***Mod1*** | ***Mod2*** | ***Rec1*** | ***Rec2*** | ***Bit*** |
|---|---|---|---|---|---|---|---|---|
| *6798* | *7469* | *0.1468* | *-0.0695* | *0.1468* | *-0.0795* | *0.1468* | *-0.0795* | *0* |
| *6803* | *7464* | *0.0060* | *-0.1788* | *0.0660* | *-0.1388* | *0.0660* | *-0.1388* | *1* |
| *6808* | *7459* | *0.0821* | *-0.1754* | *0.0621* | *-0.1954* | *0.0621* | *-0.1954* | *0* |
| *6813* | *7454* | *0.0919* | *-0.1375* | *0.0819* | *-0.1575* | *0.0819* | *-0.1575* | *1* |
| *6818* | *7449* | *0.1669* | *-0.1081* | *0.1269* | *-0.1581* | *0.1269* | *-0.1581* | *0* |
| *6823* | *7444* | *0.0854* | *-0.0792* | *0.0954* | *-0.0692* | *0.0954* | *-0.0692* | *1* |
| *6828* | *7439* | *0.1978* | *-0.0722* | *0.1678* | *-0.0922* | *0.1678* | *-0.0922* | *0* |
| *6833* | *7434* | *0.1978* | *-0.0515* | *0.1678* | *-0.0915* | *0.1678* | *-0.0915* | *0* |
| *6838* | *7429* | *0.0726* | *-0.0974* | *0.0926* | *-0.0674* | *0.0926* | *-0.0674* | *1* |

**Tabla 3.5** Muestras recuperadas del archivo 4 usando el algoritmo Patchwork

Los bits correspondientes a la primera columna recuperados son los siguientes

*Bits primera columna=[ 0 1 0 1 0 1 0 0 1]* *// No hay bits con error*

### 3.1.2 Método Propuesto

Con el método propuesto se realizaron las mismas pruebas y los resultados son los siguientes.

La información recuperada para el archivo uno es la siguiente.

*Bits con error= [ ]* *// No existen bits con error*
*Mensaje='Zacatenco'*

| ***Ind1*** | ***Ind2*** | ***M1*** | ***M2*** | ***Mod1*** | ***Mod2*** | ***Rec1*** | ***Rec2*** | ***Bit*** |
|---|---|---|---|---|---|---|---|---|
| *5810* | *5922* | *0.1751* | *0.1397* | *0.1215* | *0.1875* | *0.1195* | *0.1889* | *0* |
| *6036* | *6160* | *0.1352* | *0.1279* | *0.1875* | *0.1215* | *0.1844* | *0.1216* | *1* |
| *6345* | *6523* | *0.1076* | *0.1715* | *0.1215* | *0.1875* | *0.1201* | *0.1903* | *0* |
| *6637* | *6762* | *-0.1067* | *-0.1082* | *-0.1875* | *-0.1215* | *-0.1846* | *-0.1207* | *1* |
| *6976* | *9684* | *0.1362* | *-0.1556* | *0.1215* | *-0.1875* | *0.1174* | *-0.1854* | *0* |
| *9974* | *10225* | *-0.1114* | *-0.1111* | *-0.1875* | *-0.1215* | *-0.1857* | *-0.1210* | *1* |
| *10314* | *10349* | *-0.1102* | *-0.1516* | *-0.1215* | *-0.1875* | *-0.1201* | *-0.1902* | *0* |
| *10551* | *11600* | *0.1161* | *0.1273* | *0.1215* | *0.1875* | *0.1224* | *0.1891* | *0* |
| *11767* | *11938* | *-0.1067* | *0.1688* | *-0.1875* | *0.1215* | *-0.1847* | *0.1196* | *1* |

**Tabla 3.6** Muestras recuperadas para el archivo uno

Los bits correspondientes a la columna uno se recuperan correctamente

*Bits primera columna =[0 1 0 1 0 1 0 0 1]* *// No hay bits con error*

La información recuperada en el archivo 2 es la siguiente.

*Bits con error=[]* *//No hay bits con error*
*Mensaje='Zacatenco'*

| ***Ind1*** | ***Ind2*** | ***M1*** | ***M2*** | ***Mod1*** | ***Mod2*** | ***Rec1*** | ***Rec2*** | ***Bit*** |
|---|---|---|---|---|---|---|---|---|
| *5881* | *6497* | *-0.1218* | *0.1227* | *-0.1215* | *0.1875* | *-0.1218* | *0.1907* | *0* |
| *6858* | *6944* | *-0.1297* | *-0.1085* | *-0.1875* | *-0.1215* | *-0.1853* | *-0.1210* | *1* |
| *7189* | *7298* | *-0.1430* | *0.1117* | *-0.1215* | *0.1875* | *-0.1243* | *0.1860* | *0* |
| *7603* | *7924* | *0.1236* | *-0.1358* | *0.1875* | *-0.1215* | *0.1916* | *-0.1233* | *1* |
| *8557* | *8759* | *0.1306* | *-0.1544* | *0.1215* | *-0.1875* | *0.1207* | *-0.1842* | *0* |
| *8931* | *9070* | *0.1730* | *-0.1808* | *0.1875* | *-0.1215* | *0.1855* | *-0.1227* | *1* |
| *9136* | *9354* | *-0.1521* | *0.1241* | *-0.1215* | *0.1875* | *-0.1223* | *0.1921* | *0* |
| *9447* | *9837* | *0.1265* | *-0.1620* | *0.1215* | *-0.1875* | *0.1202* | *-0.1907* | *0* |
| *10111* | *10263* | *-0.1146* | *0.1311* | *-0.1875* | *0.1215* | *-0.1890* | *0.1212* | *1* |

**Tabla 3.7** Muestras recuperadas del archivo 2 usando método Propuesto

En el archivo 3 tampoco se tienen bits con error

*Bits con error=[]* *// No hay bits con error*
*Mensaje='Zacatenco'*

| ***Ind1*** | ***Ind2*** | ***M1*** | ***M2*** | ***Mod1*** | ***Mod2*** | ***Rec1*** | ***Rec2*** | ***Bit*** |
|---|---|---|---|---|---|---|---|---|
| *5801* | *5851* | *-0.1727* | *0.1680* | *-0.1215* | *0.1875* | *-0.1215* | *0.1875* | *0* |
| *5902* | *5943* | *0.1466* | *-0.1547* | *0.1875* | *-0.1215* | *0.1875* | *-0.1215* | *1* |
| *5999* | *6047* | *-0.1530* | *0.1734* | *-0.1215* | *0.1875* | *-0.1215* | *0.1875* | *0* |
| *6080* | *6140* | *-0.1541* | *-0.1624* | *-0.1875* | *-0.1215* | *-0.1875* | *-0.1215* | *1* |
| *6202* | *6265* | *-0.1487* | *-0.1287* | *-0.1215* | *-0.1875* | *-0.1215* | *-0.1875* | *0* |
| *6314* | *6382* | *0.1408* | *0.1085* | *0.1875* | *0.1215* | *0.1875* | *0.1215* | *1* |
| *6470* | *6540* | *-0.1610* | *-0.1821* | *-0.1215* | *-0.1875* | *-0.1215* | *-0.1875* | *0* |
| *6608* | *6674* | *-0.1760* | *-0.1855* | *-0.1215* | *-0.1875* | *-0.1215* | *-0.1875* | *0* |
| *6741* | *6812* | *-0.1560* | *-0.1360* | *-0.1875* | *-0.1215* | *-0.1875* | *-0.1215* | *1* |

**Tabla 3.8** Muestras recuperadas para el archivo 3 usando el método propuesto

Los bits de la primera columna recuperados son los siguientes

*Bits primera columna=[0 1 0 1 0 1 0 01]* *// No hay bits con error*

En el archivo 4 no se tienen bits con error.

*Bits con error=[]* *// No hay bits con error*
*Mensaje2='Zacatenco'*

| ***Ind1*** | ***Ind2*** | ***M1*** | ***M2*** | ***Mod1*** | ***Mod2*** | ***Rec1*** | ***Rec2*** | ***Bit*** |
|---|---|---|---|---|---|---|---|---|
| *5803* | *6000* | *0.1897* | *0.1309* | *0.1215* | *0.1875* | *0.1215* | *0.1875* | *0* |
| *6228* | *6431* | *0.1604* | *-0.1453* | *0.1875* | *-0.1215* | *0.1875* | *-0.1215* | *1* |
| *6618* | *6855* | *0.1138* | *-0.1263* | *0.1215* | *-0.1875* | *0.1215* | *-0.1875* | *0* |
| *7019* | *7239* | *-0.1905* | *0.1053* | *-0.1875* | *0.1215* | *-0.1875* | *0.1215* | *1* |
| *7420* | *7645* | *-0.1120* | *0.1593* | *-0.1215* | *0.1875* | *-0.1215* | *0.1875* | *0* |
| *7856* | *8132* | *-0.1199* | *0.1710* | *-0.1875* | *0.1215* | *-0.1875* | *0.1215* | *1* |
| *8306* | *8499* | *-0.1618* | *0.1437* | *-0.1215* | *0.1875* | *-0.1215* | *0.1875* | *0* |
| *8677* | *8911* | *-0.1339* | *0.1395* | *-0.1215* | *0.1875* | *-0.1215* | *0.1875* | *0* |
| *9001* | *9158* | *0.1481* | *-0.1242* | *0.1875* | *-0.1215* | *0.1875* | *-0.1215* | *1* |

**Tabla 3.9** Muestras recuperadas para el archivo 4 usando el método propuesto

Los bits de la primera columna son los siguientes

*Bits primera columna=[0 1 0 1 0 1 0 01]* *// No hay bits con error*

En la tabla 3.10 se muestran los resultados finales de ambos métodos como se puede ver de los resultados ambos métodos son eficientes. La información se recupera correctamente después de aplicar la wavelet inversa y escribir el archivo en su forma original. La correlación es cercana a la unidad indicando la información oculta por cualquiera de los dos métodos no daña al archivo y pueden utilizarse sin ningún problema.

| *Archivo* | *Energía Original* | *Patchwork* | *Propuesto* | *Corr-Patch* | *Corr-Prop* | *Bits con error* |
|---|---|---|---|---|---|---|
| ***Archivo 1*** | ***1632.1*** | ***1631.3*** | ***1632.5*** | ***1.0000*** | ***0.9999*** | ***0*** |
| ***Archivo 2*** | ***3991.7*** | ***3991.8*** | ***3993.5*** | ***1.0000*** | ***0.9999*** | ***0*** |
| ***Archivo 3*** | ***1074.7*** | ***1074.7*** | ***1075.5*** | ***1.0000*** | ***1.0000*** | ***0*** |
| ***Archivo 4*** | ***7770*** | ***7770*** | ***7770.8*** | ***1.0000*** | ***0.9999*** | ***0*** |

**Tabla 3.10** Resultados comparativos de la correlación y bits con error

## 3.2 Convirtiendo cada archivo a una frecuencia de 22050 Hz y una codificación de 8 ó 16

En esta prueba los archivos de audio se les modifica su frecuencia de muestreo de 44100 a 22050 Hz. Además, los archivos cuantizados en 8 bits se convirtieron a 16 bits, mientras que los archivos cuantizados en 16 bits se convirtieron a 8 bits. La finalidad de esta prueba es demostrar que en ambos métodos la información se recupera correctamente y que pueden ser regresados a su formato original.

### 3.2.1 Usando el algoritmo Patchwork

Para el archivo uno no se tienen bits con error por lo cual la información se recupera correctamente

*Bits con error =[]* *// No hay bits con error*

*Mensaje='Zacatenco'*

| ***Ind1*** | ***Ind2*** | ***M1*** | ***M2*** | ***Mod1*** | ***Mod2*** | ***Rec1*** | ***Rec2*** | ***Bit*** |
|---|---|---|---|---|---|---|---|---|
| *6798* | *7469* | *-0.0312* | *0* | *-0.0112* | *0.0400* | *-0.0098* | *0.0365* | *0* |
| *6803* | *7464* | *-0.0205* | *-0.0004* | *-0.0305* | *0* | *-0.0296* | *-0.0015* | *1* |
| *6808* | *7459* | *-0.0664* | *0.0125* | *-0.0364* | *0.0625* | *-0.0261* | *0.0596* | *0* |
| *6813* | *7454* | *0.0138* | *-0.0203* | *0.0438* | *-0.0103* | *0.0406* | *-0.0104* | *1* |
| *6818* | *7449* | *-0.0201* | *-0.0331* | *-0.0201* | *-0.0531* | *-0.0201* | *-0.0519* | *0* |
| *6823* | *7444* | *0.0549* | *-0.0446* | *0.0749* | *-0.0446* | *0.0728* | *-0.0409* | *1* |
| *6828* | *7439* | *0.0810* | *-0.0320* | *0.0510* | *-0.0820* | *0.0504* | *-0.0838* | *0* |
| *6833* | *7434* | *-0.0572* | *-0.0224* | *-0.0372* | *-0.0624* | *-0.0277* | *-0.0573* | *0* |
| *6838* | *7429* | *0.0485* | *-0.0096* | *0.0485* | *-0.0196* | *0.0386* | *-0.0122* | *1* |

**Tabla 3.11** Muestras recuperadas del archivo 1 modificando la frecuencia y la codificación

Los bits correspondientes a la primera columna son los siguientes

*Bits primera columna=[0 1 0 1 0 1 0 0 1]* *// No hay bits con error*

Para el archivo 2 la información recuperada es la siguiente.

*Bits con error=[]* *// No hay bits con error*

*Mensaje='Zacatenco'*

| ***Ind1*** | ***Ind2*** | ***M1*** | ***M2*** | ***Mod1*** | ***Mod2*** | ***Rec1*** | ***Rec2*** | ***Bit*** |
|---|---|---|---|---|---|---|---|---|
| *6798* | *7469* | *-0.0353* | *0.0308* | *-0.0253* | *0.0508* | *-0.0177* | *0.0485* | *0* |
| *6803* | *7464* | *-0.0590* | *-0.0701* | *-0.0890* | *-0.0501* | *-0.0778* | *-0.0513* | *1* |
| *6808* | *7459* | *-0.0014* | *-0.0750* | *-0.0314* | *-0.0650* | *-0.0293* | *-0.0600* | *0* |
| *6813* | *7454* | *-0.0227* | *-0.0349* | *-0.0527* | *-0.0249* | *-0.0526* | *-0.0213* | *1* |
| *6818* | *7449* | *-0.1093* | *-0.0439* | *-0.1193* | *-0.0439* | *-0.1093* | *-0.0376* | *0* |
| *6823* | *7444* | *-0.1257* | *0.0333* | *-0.1557* | *0.0233* | *-0.1519* | *0.0234* | *1* |
| *6828* | *7439* | *-0.0271* | *-0.0002* | *-0.0071* | *-0.0302* | *0.0016* | *-0.0315* | *0* |
| *6833* | *7434* | *0.0957* | *-0.0343* | *0.0557* | *-0.0843* | *0.0464* | *-0.0828* | *0* |
| *6838* | *7429* | *0.1121* | *0.0340* | *0.1421* | *0.0140* | *0.1412* | *0.0121* | *1* |

**Tabla 3.12** Muestras recuperadas del archivo 2 modificando la frecuencia y la codificación

Para el archivo 3 no se presentan bits con error.

*Bits con error=[]* *// No hay bits con error*
*Mensaje = 'Zacatenco'*

| ***Ind1*** | ***Ind2*** | ***M1*** | ***M2*** | ***Mod1*** | ***Mod2*** | ***Rec1*** | ***Rec2*** | ***Bit*** |
|---|---|---|---|---|---|---|---|---|
| *6798* | *7469* | *-0.0172* | *-0.0014* | *-0.0072* | *-0.0314* | *0.0003* | *-0.0264* | *0* |
| *6803* | *7464* | *-0.0277* | *0.0146* | *-0.0477* | *0.0146* | *-0.0437* | *0.0082* | *1* |
| *6808* | *7459* | *-0.0048* | *0.0200* | *-0.0048* | *0.0300* | *-0.0021* | *0.0258* | *0* |
| *6813* | *7454* | *-0.0226* | *-0.0213* | *-0.0426* | *-0.0113* | *-0.0419* | *-0.0101* | *1* |
| *6818* | *7449* | *-0.0054* | *0.0017* | *-0.0054* | *0.0317* | *-0.0017* | *0.0346* | *0* |
| *6823* | *7444* | *-0.0291* | *-0.0027* | *-0.0391* | *-0.0027* | *-0.0308* | *0.0053* | *1* |
| *6828* | *7439* | *-0.0309* | *-0.0250* | *-0.0209* | *-0.0550* | *-0.0208* | *-0.0533* | *0* |
| *6833* | *7434* | *-0.0521* | *-0.0012* | *-0.0221* | *-0.0512* | *-0.0221* | *-0.0506* | *0* |
| *6838* | *7429* | *-0.0172* | *0.0039* | *-0.0372* | *0.0039* | *-0.0314* | *-0.0029* | *1* |

**Tabla 3.13** Muestras recuperadas del archivo 3 modificando la frecuencia y la codificación

Los bits de la primera columna son los siguientes

*Bits primera columna=[0 1 0 1 0 1 0 0 1]* *// No hay bits con error*

Para el archivo 4 la información recuperada es la siguiente.

*Bits con error=[]* *// No hay bits con error*

*Mensaje =’Zacatenco’*

| *Ind1* | *Ind2* | *M1* | *M2* | *Mod1* | *Mod2* | *Rec1* | *Rec2* | *Bit* |
|---|---|---|---|---|---|---|---|---|
| *6798* | *7469* | *0.1468* | *-0.0695* | *0.1468* | *-0.0795* | *0.1352* | *-0.0690* | *0* |
| *6803* | *7464* | *0.0060* | *-0.1788* | *0.0660* | *-0.1388* | *0.0584* | *-0.1281* | *1* |
| *6808* | *7459* | *0.0821* | *-0.1754* | *0.0621* | *-0.1954* | *0.0616* | *-0.1905* | *0* |
| *6813* | *7454* | *0.0919* | *-0.1375* | *0.0819* | *-0.1575* | *0.0810* | *-0.1517* | *1* |
| *6818* | *7449* | *0.1669* | *-0.1081* | *0.1269* | *-0.1581* | *0.1192* | *-0.1515* | *0* |
| *6823* | *7444* | *0.0854* | *-0.0792* | *0.0954* | *-0.0692* | *0.0919* | *-0.0598* | *1* |
| *6828* | *7439* | *0.1978* | *-0.0722* | *0.1678* | *-0.0922* | *0.1618* | *-0.0894* | *0* |
| *6833* | *7434* | *0.1978* | *-0.0515* | *0.1678* | *-0.0915* | *0.1629* | *-0.0920* | *0* |
| *6838* | *7429* | *0.0726* | *-0.0974* | *0.0926* | *-0.0674* | *0.0874* | *-0.0625* | *1* |

**Tabla 3.14** Muestras recuperadas del archivo 4 modificando la frecuencia y la codificación

Los bits de la primera columna son los siguientes

*Bits primera columna=[0 1 0 1 0 1 0 0 1]  // No hay bits con error*

## 3.2.2 Método Propuesto

**Archivo 1**

Para el archivo 1 no se tienen bits con error.

*Bits con error=[]  // No hay bits con error*
*Mensaje=’Zacatenco’*

| *Ind1* | *Ind2* | *M1* | *M2* | *Mod1* | *Mod2* | *Rec1* | *Rec2* | *Bit* |
|---|---|---|---|---|---|---|---|---|
| *5810* | *5922* | *0.1751* | *0.1397* | *0.1215* | *0.1875* | *0.1195* | *0.1889* | *0* |
| *6036* | *6160* | *0.1352* | *0.1279* | *0.1875* | *0.1215* | *0.1844* | *0.1216* | *1* |
| *6345* | *6523* | *0.1076* | *0.1715* | *0.1215* | *0.1875* | *0.1201* | *0.1903* | *0* |
| *6637* | *6762* | *-0.1067* | *-0.1082* | *-0.1875* | *-0.1215* | *-0.1846* | *-0.1207* | *1* |
| *6976* | *9684* | *0.1362* | *-0.1556* | *0.1215* | *-0.1875* | *0.1174* | *-0.1854* | *0* |
| *9974* | *10225* | *-0.1114* | *-0.1111* | *-0.1875* | *-0.1215* | *-0.1857* | *-0.1210* | *1* |
| *10314* | *10349* | *-0.1102* | *-0.1516* | *-0.1215* | *-0.1875* | *-0.1201* | *-0.1902* | *0* |
| *10551* | *11600* | *0.1161* | *0.1273* | *0.1215* | *0.1875* | *0.1224* | *0.1891* | *0* |
| *11767* | *11938* | *-0.1067* | *0.1688* | *-0.1875* | *0.1215* | *-0.1847* | *0.1196* | *1* |

**Tabla 3.15** Muestras recuperadas del archivo 1 modificando la frecuencia y la codificación

Los bits de la primera columna son

*Bits primera columna=[0 1 0 1 0 1 0 0 1]  // No hay bits con error*

Para el archivo 2 lo recuperado es lo siguiente.

*Bits con error=[]* *// No hay bits con error*

*Mensaje='Zacatenco'*

| ***Ind1*** | ***Ind2*** | ***M1*** | ***M2*** | ***Mod1*** | ***Mod2*** | ***Rec1*** | ***Rec2*** | ***Bit*** |
|---|---|---|---|---|---|---|---|---|
| *5881* | *6497* | *-0.1218* | *0.1227* | *-0.1215* | *0.1875* | *-0.1218* | *0.1907* | *0* |
| *6858* | *6944* | *-0.1297* | *-0.1085* | *-0.1875* | *-0.1215* | *-0.1853* | *-0.1210* | *1* |
| *7189* | *7298* | *-0.1430* | *0.1117* | *-0.1215* | *0.1875* | *-0.1243* | *0.1860* | *0* |
| *7603* | *7924* | *0.1236* | *-0.1358* | *0.1875* | *-0.1215* | *0.1916* | *-0.1233* | *1* |
| *8557* | *8759* | *0.1306* | *-0.1544* | *0.1215* | *-0.1875* | *0.1207* | *-0.1842* | *0* |
| *8931* | *9070* | *0.1730* | *-0.1808* | *0.1875* | *-0.1215* | *0.1855* | *-0.1227* | *1* |
| *9136* | *9354* | *-0.1521* | *0.1241* | *-0.1215* | *0.1875* | *-0.1223* | *0.1921* | *0* |
| *9447* | *9837* | *0.1265* | *-0.1620* | *0.1215* | *-0.1875* | *0.1202* | *-0.1907* | *0* |
| *10111* | *10263* | *-0.1146* | *0.1311* | *-0.1875* | *0.1215* | *-0.1890* | *0.1212* | *1* |

**Tabla 3.16** Muestras recuperadas del archivo 2 modificando la frecuencia y la codificación

Los bits recuperados son los siguientes

*Bits primera columna=[0 1 0 1 0 1 0 0 1]* *// No hay bits con error*

Para el archivo 3 tampoco se tienen bits con error.

*Bits con error=[]* *// No hay bits con error*

*Mensaje='Zacatenco'*

| ***Ind1*** | ***Ind2*** | ***M1*** | ***M2*** | ***Mod1*** | ***Mod2*** | ***Rec1*** | ***Rec2*** | ***Bit*** |
|---|---|---|---|---|---|---|---|---|
| *5801* | *5851* | *-0.1727* | *0.1680* | *-0.1215* | *0.1875* | *-0.1215* | *0.1875* | *0* |
| *5902* | *5943* | *0.1466* | *-0.1547* | *0.1875* | *-0.1215* | *0.1875* | *-0.1215* | *1* |
| *5999* | *6047* | *-0.1530* | *0.1734* | *-0.1215* | *0.1875* | *-0.1215* | *0.1875* | *0* |
| *6080* | *6140* | *-0.1541* | *-0.1624* | *-0.1875* | *-0.1215* | *-0.1875* | *-0.1215* | *1* |
| *6202* | *6265* | *-0.1487* | *-0.1287* | *-0.1215* | *-0.1875* | *-0.1215* | *-0.1875* | *0* |
| *6314* | *6382* | *0.1408* | *0.1085* | *0.1875* | *0.1215* | *0.1875* | *0.1215* | *1* |
| *6470* | *6540* | *-0.1610* | *-0.1821* | *-0.1215* | *-0.1875* | *-0.1215* | *-0.1875* | *0* |
| *6608* | *6674* | *-0.1760* | *-0.1855* | *-0.1215* | *-0.1875* | *-0.1215* | *-0.1875* | *0* |
| *6741* | *6812* | *-0.1560* | *-0.1360* | *-0.1875* | *-0.1215* | *-0.1875* | *-0.1215* | *1* |

**Tabla 3.17** Muestras recuperadas del archivo 3 modificando la frecuencia y la codificación

Los bits de la columna uno son los siguientes

*Bits primera columna=[0 1 0 1 0 1 0 0 1]* *// No hay bits con error*

Para el archivo 4 no se tienen bits con error.

*Bits con error=[]* *// No hay bits con error*
*Mensaje='Zacatenco'*

| ***Ind1*** | ***Ind2*** | ***M1*** | ***M2*** | ***Mod1*** | ***Mod2*** | ***Rec1*** | ***Rec2*** | ***Bit*** |
|---|---|---|---|---|---|---|---|---|
| *5803* | *6000* | *0.1897* | *0.1309* | *0.1215* | *0.1875* | *0.1215* | *0.1875* | *0* |
| *6228* | *6431* | *0.1604* | *-0.1453* | *0.1875* | *-0.1215* | *0.1875* | *-0.1215* | *1* |
| *6618* | *6855* | *0.1138* | *-0.1263* | *0.1215* | *-0.1875* | *0.1215* | *-0.1875* | *0* |
| *7019* | *7239* | *-0.1905* | *0.1053* | *-0.1875* | *0.1215* | *-0.1875* | *0.1215* | *1* |
| *7420* | *7645* | *-0.1120* | *0.1593* | *-0.1215* | *0.1875* | *-0.1215* | *0.1875* | *0* |
| *7856* | *8132* | *-0.1199* | *0.1710* | *-0.1875* | *0.1215* | *-0.1875* | *0.1215* | *1* |
| *8306* | *8499* | *-0.1618* | *0.1437* | *-0.1215* | *0.1875* | *-0.1215* | *0.1875* | *0* |
| *8677* | *8911* | *-0.1339* | *0.1395* | *-0.1215* | *0.1875* | *-0.1215* | *0.1875* | *0* |
| *9001* | *9158* | *0.1481* | *-0.1242* | *0.1875* | *-0.1215* | *0.1875* | *-0.1215* | *1* |

**Tabla 3.18** Muestras recuperadas del archivo 1 modificando la frecuencia y la codificación

Los bits de la columna uno son los siguientes

*Bits primera columna=[0 1 0 1 0 1 0 0 1]* *// No hay bits con error*

La siguiente tabla muestra los resultados finales de ambos métodos, de los resultados se concluye que los dos métodos son efectivos y la información sobrevive para este tipo de prueba más adelante se mostrará que no sucede lo mismo usando un convertidor comercial. La correlación entre ambos archivos sigue siendo cercana a la unidad y no se tienen ningún bit con error.

| ***Archivo*** | ***Energía Original*** | ***Patchwork*** | ***Propuesto*** | ***Corr-Patch*** | ***Corr-Prop*** | ***Bits con error*** |
|---|---|---|---|---|---|---|
| ***Archivo 1*** | ***1632.1*** | ***1631.3*** | ***1632.5*** | ***1.0000*** | ***0.9999*** | ***0*** |
| ***Archivo 2*** | ***3991.7*** | ***3991.8*** | ***3993.5*** | ***1.0000*** | ***0.9999*** | ***0*** |
| ***Archivo 3*** | ***1074.7*** | ***1074.7*** | ***1075.5*** | ***1.0000*** | ***1.0000*** | ***0*** |
| ***Archivo 4*** | ***7770*** | ***7770*** | ***7770.8*** | ***1.0000*** | ***0.9999*** | ***0*** |

**Tabla 3.19** Resultados de ambos algoritmos

## 3.3 Aplicando un convertidor comercial

Las tres últimas pruebas que se realizan son las siguientes. Al igual que en la prueba 2 a los archivos se les modificará se frecuencia de muestreo de 44100 a 22050 Hz. Además los archivos cuantizados en 8 bits y viceversa. Se realizaran en el archivo de audio convertirlo de formato wav a mp3 y regresarlo a su formato original. Se realizará otro cambio de formato de wav a CDA y de nuevo a wav.se emplean dos convertidores comerciales el Roxio y Pinnacle. Se debe explicar que el algoritmo Patchwork ya no es adecuado para estas pruebas ya que los convertidores comerciales escalan los valores de las muestras, la codificación empleada por estos convertidores es desconocida y modificar una cifra ya resulta poco eficiente.

Como se verá a continuación en el archivo dos, el cual fue convertido a un archivo wav de una frecuencia de 22050 Hz y 16 bits para después con el mismo convertidor regresarlo a su formato de 44100 Hz y 8 bits. Se debe tener en cuenta que debido a la pérdida de muestras se toman los valores de de las posiciones 5800 a 5999 como un vector de referencia para localizar el punto a partir del cual localizar la información. Los resultados de la primera columna se presentan a continuación, los valores *Pin1 y Pin2 se refieren a Pínnacle y Rox1 y Rox2 a Roxio, Pin1 y Rox1 a M1 y Pin2 y Rox2 a M2,* los valores en negritas son valores con error:

**Archivo 2**

| ***Ind1*** | ***Ind2*** | ***M1*** | ***M2*** | ***Mod1*** | ***Mod2*** | ***Pin1*** | ***Pin2*** | ***Bit*** | ***Rox1*** | ***Rox2*** | ***Bit*** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *6798* | *7469* | *-0.0353* | *0.0308* | *-0.0253* | *0.0508* | *-0.0451* | *0.0870* | *0* | *-0.0290* | *0.0482* | *0* |
| *6803* | *7464* | *-0.0590* | *-0.0701* | *-0.0890* | *-0.0501* | *-0.1592* | *-0.1285* | *1* | *-0.0768* | *-0.0631* | *1* |
| *6808* | *7459* | *-0.0014* | *-0.0750* | *-0.0314* | *-0.0650* | ***-0.0488*** | ***-0.1371*** | ***1*** | *-0.0199* | *-0.0539* | *0* |
| *6813* | *7454* | *-0.0227* | *-0.0349* | *-0.0527* | *-0.0249* | *-0.0895* | *-0.0822* | *1* | *-0.0502* | *-0.0405* | *1* |
| *6818* | *7449* | *-0.1093* | *-0.0439* | *-0.1193* | *-0.0439* | *-0.2290* | *-0.0945* | *0* | *-0.1156* | *-0.0439* | *0* |
| *6823* | *7444* | *-0.1257* | *0.0333* | *-0.1557* | *0.0233* | ***-0.3056*** | ***0.0415*** | ***0*** | *-0.1437* | *0.0255* | *1* |
| *6828* | *7439* | *-0.0271* | *-0.0002* | *-0.0071* | *-0.0302* | *-0.0226* | *-0.0653* | *0* | *-0.0123* | *-0.0307* | *0* |
| *6833* | *7434* | *0.0957* | *-0.0343* | *0.0557* | *-0.0843* | *0.1090* | *-0.1480* | *0* | *0.0559* | *-0.0747* | *0* |
| *6838* | *7429* | *0.1121* | *0.0340* | *0.1421* | *0.0140* | ***0.2233*** | ***0.0346*** | ***0*** | ***0.1246*** | ***0.0368*** | ***0*** |

**Tabla 3.20** Muestras recuperadas del archivo 2 al aplicar un convertidor comercial

El vector de referencia se encuentra en la posición 5801. Se tienen 53 bits con error usando el Pinnacle algunos de los cuales se muestran a continuación.

*Bits con error=[3 6 9 13 15 18 19 20 21 22 23 26 27 28 29 33 39 40 42 44 45 52 55 58 59]*

La secuencia de bits introducidos es la siguiente:

*Bits primera columna original=[0 1 0 1 0 1 0 0 1]*

Y la recuperada es la siguiente

*Bits primera columna recuperada=[0 1 1 1 0 0 0 0 0]*

Y usando el convertidor Roxio el vector de referencia se encuentra en la posición 5802, y se tiene lo siguiente.

Bits con error = [9 17 21 34 50 60 65 72 80 102 110 116 119 124 127 128 130 134]

*Bits primera columna recuperada=[0 1 0 1 0 1 0 0 0]*

Es claro que el algoritmo Patchwork no funciona cuando se utilizan convertidores. Por estas razones no se utiliza para el resto de las pruebas. De aquí en adelante todos los archivos, se usa el vector de referencia de las posiciones 5800-5999 en el dominio wavelet para localizar la información.

## 3.3.1 Convirtiendo los archivos usando un convertidor comercial a una frecuencia de 22050 y cambiando su codificación a 8 o 16 bits.

En el archivo uno el vector de referencia para Pinnacle fue localizado en la posición 5801 y para Roxio en 5802 y lo recuperado es lo siguiente

*Bits con error Pinnacle=[]* *// No hay bits con error*
*Mensaje='Zacatenco'*

*Bits con error Roxio=[]* *// No hay bits con error*
*Mensaje='Zacatenco'*

| ***Ind1*** | ***Ind2*** | ***M1*** | ***M2*** | ***Mod1*** | ***Mod2*** | ***Pin1*** | ***Pin2*** | ***Bit*** | ***Rox1*** | ***Rox2*** | ***Bit*** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *5810* | *5922* | *0.1751* | *0.1397* | *0.1215* | *0.1875* | *0.1765* | *0.2274* | *0* | *0.1233* | *0.1813* | *0* |
| *6036* | *6160* | *0.1352* | *0.1279* | *0.1875* | *0.1215* | *-0.2202* | *0.1462* | *1* | *0.1702* | *0.1352* | *1* |
| *6345* | *6523* | *0.1076* | *0.1715* | *0.1215* | *0.1875* | *0.1429* | *0.2191* | *0* | *0.1188* | *0.1837* | *0* |
| *6637* | *6762* | *-0.1067* | *-0.1082* | *-0.1875* | *-0.1215* | *-0.2271* | *0.1481* | *1* | *-0.1623* | *-0.1196* | *1* |
| *6976* | *9684* | *0.1362* | *-0.1556* | *0.1215* | *-0.1875* | *-0.1700* | *0.2073* | *0* | *0.1248* | *-0.1897* | *0* |
| *9974* | *10225* | *-0.1114* | *-0.1111* | *-0.1875* | *-0.1215* | *0.2055* | *-0.1479* | *1* | *-0.1636* | *-0.1189* | *1* |
| *10314* | *10349* | *-0.1102* | *-0.1516* | *-0.1215* | *-0.1875* | *-0.1730* | *-0.2266* | *0* | *-0.1190* | *-0.1818* | *0* |
| *10551* | *11600* | *0.1161* | *0.1273* | *0.1215* | *0.1875* | *0.1486* | *-0.2241* | *0* | *0.1095* | *0.1806* | *0* |
| *11767* | *11938* | *-0.1067* | *0.1688* | *-0.1875* | *0.1215* | *-0.2023* | *0.1505* | *1* | *-0.1715* | *0.1288* | *1* |

**Tabla 3.21** Muestras recuperadas del archivo 1 aplicando un convertidor de formato comercial

En el archivo dos el vector de referencia para Pinnacle fue localizado en la posición 5801 y 5802 para Roxio lo recuperado es lo siguiente:

*Bits con error Pinnacle=[]* *// No hay bits con error*
*Mensaje='Zacatenco'*

*Bits con error Roxio=[]* *// No hay bits con error*
*Mensaje='Zacatenco'*

| *Ind1* | *Ind2* | *M1* | *M2* | *Mod1* | *Mod2* | *Pin1* | *Pin2* | *Bit* | *Rox1* | *Rox2* | *Bit* |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *5881* | *6497* | *-0.1218* | *0.1227* | *-0.1215* | *0.1875* | *-0.1498* | *0.2062* | *0* | *-0.1270* | *0.1692* | *0* |
| *6858* | *6944* | *-0.1297* | *-0.1085* | *-0.1875* | *-0.1215* | *-0.2189* | *-0.1489* | *1* | *-0.1755* | *-0.1191* | *1* |
| *7189* | *7298* | *-0.1430* | *0.1117* | *-0.1215* | *0.1875* | *-0.1663* | *0.1952* | *0* | *-0.1306* | *0.1584* | *0* |
| *7603* | *7924* | *0.1236* | *-0.1358* | *0.1875* | *-0.1215* | *0.2081* | *-0.1657* | *1* | *0.1700* | *-0.1222* | *1* |
| *8557* | *8759* | *0.1306* | *-0.1544* | *0.1215* | *-0.1875* | *0.1527* | *-0.2232* | *0* | *0.1165* | *-0.1750* | *0* |
| *8931* | *9070* | *0.1730* | *-0.1808* | *0.1875* | *-0.1215* | *0.2231* | *-0.1840* | *1* | *0.1800* | *-0.1429* | *1* |
| *9136* | *9354* | *-0.1521* | *0.1241* | *-0.1215* | *0.1875* | *-0.1694* | *0.2079* | *0* | *-0.1310* | *0.1643* | *0* |
| *9447* | *9837* | *0.1265* | *-0.1620* | *0.1215* | *-0.1875* | *0.1504* | *-0.2320* | *0* | *0.1210* | *-0.1833* | *0* |
| *10111* | *10263* | *-0.1146* | *0.1311* | *-0.1875* | *0.1215* | *-0.2166* | *0.1528* | *1* | *-0.1732* | *0.1233* | *1* |

**Tabla 3.22** Muestras recuperadas del archivo 2 aplicando un convertidor de formato comercial

En el archivo tres el vector de referencia para Pinnacle fue localizado en la posición 5801 y 5802 para Roxio lo recuperado es:

*Bits con error Pinnacle=[]* *// No hay bits con error*
*Mensaje='Zacatenco'*

*Bits con error Roxio=[]* *// No hay bits con error*
*Mensaje='Zacatenco'*

| *Ind1* | *Ind2* | *M1* | *M2* | *Mod1* | *Mod2* | *Pin1* | *Pin2* | *Bit* | *Rox1* | *Rox2* | *Bit* |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *5801* | *5851* | *-0.1727* | *0.1680* | *-0.1215* | *0.1875* | *-0.1537* | *0.2372* | *0* | -0.1430 | 0.1770 | *0* |
| *5902* | *5943* | *0.1466* | *-0.1547* | *0.1875* | *-0.1215* | *0.2372* | *-0.1537* | *1* | 0.1711 | -0.1288 | *1* |
| *5999* | *6047* | *-0.1530* | *0.1734* | *-0.1215* | *0.1875* | *-0.1537* | *0.2372* | *0* | -0.1323 | 0.1802 | *0* |
| *6080* | *6140* | *-0.1541* | *-0.1624* | *-0.1875* | *-0.1215* | *-0.2372* | *-0.1537* | *1* | -0.1860 | -0.1332 | *1* |
| *6202* | *6265* | *-0.1487* | *-0.1287* | *-0.1215* | *-0.1875* | *-0.1537* | *-0.2372* | *0* | -0.1379 | -0.1776 | *0* |
| *6314* | *6382* | *0.1408* | *0.1085* | *0.1875* | *0.1215* | *0.2372* | *0.1537* | *1* | 0.1698 | 0.1037 | *1* |
| *6470* | *6540* | *-0.1610* | *-0.1821* | *-0.1215* | *-0.1875* | *-0.1537* | *-0.2372* | *0* | -0.1439 | -0.1978 | *0* |
| *6608* | *6674* | *-0.1760* | *-0.1855* | *-0.1215* | *-0.1875* | *-0.1537* | *-0.2372* | *0* | -0.1507 | -0.1930 | *0* |
| *6741* | *6812* | *-0.1560* | *-0.1360* | *-0.1875* | *-0.1215* | *-0.2372* | *-0.1537* | *1* | -0.1924 | -0.1378 | *1* |

**Tabla 3.23** Muestras recuperadas del archivo 3 aplicando un convertidor de formato comercial

En el archivo cuatro el vector de referencia para Pinnacle fue localizado en la posición 5801 y 5802 para Roxio la información recuperada es la siguiente:

*Bits con error Pinnacle=[1 127 129]*
*Mensaje='Zacatenco'*

*Bits con error Roxio=[2 74 43 60 77 122]*
*Mensaje='Zacatenco'*

| *Ind1* | *Ind2* | *M1* | *M2* | *Mod1* | *Mod2* | *Pin1* | *Pin2* | *Bit* | *Rox1* | *Rox2* | *Bit* |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *5803* | *6000* | *0.1897* | *0.1309* | *0.1215* | *0.1875* | ***0.1796*** | ***0.1766*** | ***1*** | 0.1517 | 0.1643 | *0* |
| *6228* | *6431* | *0.1604* | *-0.1453* | *0.1875* | *-0.1215* | *0.1808* | *0.1524* | *1* | **0.1514** | **-0.1557** | ***0*** |
| *6618* | *6855* | *0.1138* | *-0.1263* | *0.1215* | *-0.1875* | *-0.1685* | *-0.2392* | *0* | 0.1282 | -0.1614 | *0* |
| *7019* | *7239* | *-0.1905* | *0.1053* | *-0.1875* | *0.1215* | *-0.2236* | *-0.1718* | *1* | -0.2190 | 0.1272 | *1* |
| *7420* | *7645* | *-0.1120* | *0.1593* | *-0.1215* | *0.1875* | *-0.1489* | *0.2114* | *0* | -0.1467 | 0.1690 | *0* |
| *7856* | *8132* | *-0.1199* | *0.1710* | *-0.1875* | *0.1215* | *-0.2127* | *-0.1710* | *1* | -0.2244 | 0.0977 | *1* |
| *8306* | *8499* | *-0.1618* | *0.1437* | *-0.1215* | *0.1875* | *0.1472* | *0.1637* | *0* | -0.1492 | 0.1467 | *0* |
| *8677* | *8911* | *-0.1339* | *0.1395* | *-0.1215* | *0.1875* | *-0.1452* | *0.2428* | *0* | -0.1357 | 0.1710 | *0* |
| *9001* | *9158* | *0.1481* | *-0.1242* | *0.1875* | *-0.1215* | *0.2388* | *-0.2128* | *1* | 0.1676 | -0.1430 | *1* |

**Tabla 3.24** Muestras recuperadas del archivo 4 aplicando un convertidor de formato comercial

De los resultados de la tabla 3.25 se muestra que el método propuesto es efectivo ya que no se presentan errores en la información recuperada *Rec-Pin y Rec-Rox* se refieren a la cantidad de muestras en el archivo recuperado, *E-Pin y E-Rox* a su energía y *C-Pin y C-Rox* a su correlación, es claro que la energía es diferente en cada convertidor ya que cada muestra es modificada aunque no es de una forma constante pero a pesar de esto el método es efectivo, también es claro que la pérdida de muestras es 16 para todos los archivos en Pinnacle y se añaden 5 más en el Roxio.

| *Archivos* | *Muestras* | *Rec-Pin* | *Rec-Rox* | *Per-Pin* | *Per-Rox* | *E -Pin* | *E-Rox* | *C-Pin* | *C-Rox* |
|---|---|---|---|---|---|---|---|---|---|
| *Archivo 1* | *314304* | *314288* | *314309* | *-16* | *+5* | *2510* | *1681.9* | *0.9983* | *0.9970* |
| *Archivo 2* | *550000* | *549984* | *550005* | *-16* | *+5* | *6122* | *3999.2* | *0.9925* | *0.9967* |
| *Archivo3* | *140000* | *139984* | *140005* | *-16* | *+5* | *1688* | *1109.8* | *0.9973* | *0.9925* |
| *Archivo 4* | *561000* | *560984* | *561005* | *-16* | *+5* | *11939* | *7752.5* | *0.9880* | *0.9882* |

**Tabla 3.25** Resultados de aplicar ambos convertidores

## 3.3.2 CONVIRTIENDO A FORMATO MP3 A 128 KBPS

Para esta prueba cada archivo es convertido a un formato Mp3 a 128 Kbps ya que este formato es bastante popular y el grado de compresión es considerable y sirve como un buen parámetro para saber el grado de robustez del método propuesto.

Para el archivo uno el vector de referencia se localiza en la posición 6440 para Pinnacle y 7193 para Roxio la información recuperada es la siguiente.

*Bits con error Pinnacle=[]* *// No hay bits con error*
*Mensaje='Zacatenco'*

*Bits con error Roxio=[1 16 26 33 58 88 94]*
*Mensaje='Zacatenco'*

| *Ind1* | *Ind2* | *M1* | *M2* | *Mod1* | *Mod2* | *Pin1* | *Pin2* | *Bit* | *Rox1* | *Rox2* | *Bit* |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *5810* | *5922* | *0.1751* | *0.1397* | *0.1215* | *0.1875* | *0.3213* | *0.3654* | *0* | ***0.1218*** | ***0.1198*** | ***1*** |
| *6036* | *6160* | *0.1352* | *0.1279* | *0.1875* | *0.1215* | *-0.3412* | *0.1747* | *1* | *0.1468* | *0.1096* | *1* |
| *6345* | *6523* | *0.1076* | *0.1715* | *0.1215* | *0.1875* | *0.2405* | *0.3036* | *0* | *0.1135* | *0.1782* | *0* |
| *6637* | *6762* | *-0.1067* | *-0.1082* | *-0.1875* | *-0.1215* | *-0.3306* | *0.2094* | *1* | *-0.1706* | *-0.1237* | *1* |
| *6976* | *9684* | *0.1362* | *-0.1556* | *0.1215* | *-0.1875* | *-0.2488* | *0.3164* | *0* | *0.1329* | *-0.1585* | *0* |
| *9974* | *10225* | *-0.1114* | *-0.1111* | *-0.1875* | *-0.1215* | *0.3080* | *-0.1776* | *1* | *-0.1435* | *-0.1321* | *1* |
| *10314* | *10349* | *-0.1102* | *-0.1516* | *-0.1215* | *-0.1875* | *-0.2404* | *-0.3492* | *0* | *-0.1212* | *-0.1803* | *0* |
| *10551* | *11600* | *0.1161* | *0.1273* | *0.1215* | *0.1875* | *0.2414* | *-0.2772* | *0* | *0.0999* | *0.1386* | *0* |
| *11767* | *11938* | *-0.1067* | *0.1688* | *-0.1875* | *0.1215* | *-0.3138* | *0.1969* | *1* | *-0.1979* | *0.1160* | *1* |

**Tabla 3.26** Muestras recuperadas en el archivo uno al aplicar el formato Mp3

Para el archivo dos el vector de referencia en Pinnacle esta localizado en la posición 6440 y 7193 para Roxio y se tiene:

*Bits con error Pinnacle=[]* *// No hay bits con error*
*Mensaje='Zacatenco'*

*Bits con error Roxio=[16 41 53 60 82 115]*
*Mensaje='Zacatenco'*

| *Ind1* | *Ind2* | *M1* | *M2* | *Mod1* | *Mod2* | *Pin1* | *Pin2* | *Bit* | *Rox1* | *Rox2* | *Bit* |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *5881* | *6497* | *-0.1218* | *0.1227* | *-0.1215* | *0.1875* | *-0.2019* | *0.3145* | *0* | -0.1266 | 0.1749 | *0* |
| *6858* | *6944* | *-0.1297* | *-0.1085* | *-0.1875* | *-0.1215* | *-0.3429* | *-0.2182* | *1* | -0.1726 | -0.1222 | *1* |
| *7189* | *7298* | *-0.1430* | *0.1117* | *-0.1215* | *0.1875* | *-0.2500* | *0.3034* | *0* | -0.1324 | 0.1402 | *0* |
| *7603* | *7924* | *0.1236* | *-0.1358* | *0.1875* | *-0.1215* | *0.3103* | *-0.2409* | *1* | 0.1570 | -0.1239 | *1* |
| *8557* | *8759* | *0.1306* | *-0.1544* | *0.1215* | *-0.1875* | *0.2124* | *-0.3179* | *0* | 0.1128 | -0.1753 | *0* |
| *8931* | *9070* | *0.1730* | *-0.1808* | *0.1875* | *-0.1215* | *0.3279* | *-0.2500* | *1* | 0.1715 | -0.1108 | *1* |
| *9136* | *9354* | *-0.1521* | *0.1241* | *-0.1215* | *0.1875* | *-0.2377* | *0.2925* | *0* | -0.1059 | 0.1527 | *0* |
| *9447* | *9837* | *0.1265* | *-0.1620* | *0.1215* | *-0.1875* | *0.2172* | *-0.3392* | *0* | 0.1064 | -0.1641 | *0* |
| *10111* | *10263* | *-0.1146* | *0.1311* | *-0.1875* | *0.1215* | *-0.3419* | *0.2067* | *1* | -0.1620 | 0.0913 | *1* |

**Tabla 3.27** Muestras recuperadas en el archivo dos al aplicar el formato Mp3

Para el archivo tres el vector de referencia se localiza en la posición 6440 para Pinnacle y 7193 para Roxio la información recuperada es la siguiente.

*Bits con error Pinnacle=[13 14 31 96]*
*Mensaje='Zacatenco'*

*Bits con error Roxio=[17 24 56 57 82 131]*
*Mensaje='Zacatenco'*

| *Ind1* | *Ind2* | *M1* | *M2* | *Mod1* | *Mod2* | *Pin1* | *Pin2* | *Bit* | *Rox1* | *Rox2* | *Bit* |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *5801* | *5851* | *-0.1727* | *0.1680* | *-0.1215* | *0.1875* | *-0.2316* | *0.3224* | *0* | *-0.1311* | *0.1809* | *0* |
| *5902* | *5943* | *0.1466* | *-0.1547* | *0.1875* | *-0.1215* | *0.3130* | *-0.2460* | *1* | *0.1770* | *-0.1191* | *1* |
| *5999* | *6047* | *-0.1530* | *0.1734* | *-0.1215* | *0.1875* | *-0.2397* | *0.3261* | *0* | *-0.1305* | *0.2007* | *0* |
| *6080* | *6140* | *-0.1541* | *-0.1624* | *-0.1875* | *-0.1215* | *-0.3479* | *-0.2270* | *1* | *-0.1870* | *-0.1273* | *1* |
| *6202* | *6265* | *-0.1487* | *-0.1287* | *-0.1215* | *-0.1875* | *-0.2297* | *-0.3324* | *0* | *-0.1322* | *-0.1822* | *0* |
| *6314* | *6382* | *0.1408* | *0.1085* | *0.1875* | *0.1215* | *0.3498* | *0.2284* | *1* | *0.1766* | *0.1164* | *1* |
| *6470* | *6540* | *-0.1610* | *-0.1821* | *-0.1215* | *-0.1875* | *-0.2427* | *-0.3414* | *0* | *-0.1419* | *-0.1959* | *0* |
| *6608* | *6674* | *-0.1760* | *-0.1855* | *-0.1215* | *-0.1875* | *-0.2493* | *-0.3376* | *0* | *-0.1377* | *-0.1739* | *0* |
| *6741* | *6812* | *-0.1560* | *-0.1360* | *-0.1875* | *-0.1215* | *-0.3199* | *-0.2208* | *1* | *-0.1504* | *-0.1161* | *1* |

**Tabla 3.28** Muestras recuperadas en el archivo tres al aplicar el formato Mp3

Para el archivo cuatro el vector de referencia se localiza en la posición 6440 para Pinnacle y 7193 para Roxio la información recuperada es la siguiente.

*Bits con error Pinnacle=[1 32 100 109 125]*
*Mensaje='Zacatenco'*

*Bits con error Roxio=[6 7 9 19 30 109]*
*Mensaje='Zacatenco'*

| *Ind1* | *Ind2* | *M1* | *M2* | *Mod1* | *Mod2* | *Pin1* | *Pin2* | *Bit* | *Rox1* | *Rox2* | *Bit* |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *5803* | *6000* | *0.1897* | *0.1309* | *0.1215* | *0.1875* | ***0.3606*** | ***0.2689*** | ***1*** | *0.1497* | *0.1591* | *0* |
| *6228* | *6431* | *0.1604* | *-0.1453* | *0.1875* | *-0.1215* | *0.2929* | *0.1832* | *1* | *0.1362* | *0.0994* | *1* |
| *6618* | *6855* | *0.1138* | *-0.1263* | *0.1215* | *-0.1875* | *-0.1157* | *-0.4146* | *0* | *-0.1078* | *-0.1158* | *0* |
| *7019* | *7239* | *-0.1905* | *0.1053* | *-0.1875* | *0.1215* | *-0.3560* | *-0.2344* | *1* | *-0.1609* | *-0.0860* | *1* |
| *7420* | *7645* | *-0.1120* | *0.1593* | *-0.1215* | *0.1875* | *-0.1793* | *0.2527* | *0* | *-0.1163* | *0.1890* | *0* |
| *7856* | *8132* | *-0.1199* | *0.1710* | *-0.1875* | *0.1215* | *-0.3060* | *-0.1569* | *1* | ***-0.1240*** | ***-0.1392*** | ***0*** |
| *8306* | *8499* | *-0.1618* | *0.1437* | *-0.1215* | *0.1875* | *0.2152* | *0.3082* | *0* | ***0.1301*** | ***0.1222*** | ***1*** |
| *8677* | *8911* | *-0.1339* | *0.1395* | *-0.1215* | *0.1875* | *-0.0767* | *0.3110* | *0* | *-0.0544* | *0.2092* | *0* |
| *9001* | *9158* | *0.1481* | *-0.1242* | *0.1875* | *-0.1215* | *0.4277* | *-0.2400* | *1* | ***0.1450*** | ***-0.1647*** | ***0*** |

**Tabla 3.29** Muestras recuperadas en el archivo uno al aplicar el formato Mp3

De los resultados de la tabla 3.30 se tiene que la pérdida de muestras es diferente para los archivos usando cada convertidor, y en cada convertidor también cada archivo pierde una cantidad de muestras distintas, también la energía aumenta con Pinnacle y disminuye con Roxio, por estas razones se considera que el formato Mp3 es la prueba más importante, ya que a pesar de los diferentes cambios que sufren los archivos la calidad de audio se mantiene, aunque numéricamente es claro que hubo cambios. Aún con todos los problemas la información se recupera correctamente a pesar de tener varios bits con error, indicando que el método propuesto es robusto a este cambio de formato.

| *Archivos* | *Muestras* | *Rec-Pin* | *Rec-Rox* | *Per-Pin* | *Per-Rox* | *E -Pin* | *E-Rox* | *C-Pin* | *C-Rox* |
|---|---|---|---|---|---|---|---|---|---|
| *Archivo 1* | *314304* | *311040* | *313345* | *-3264* | *-959* | *2800* | *1448.3* | *0.9598* | *0.9519* |
| *Archivo 2* | *550000* | *547200* | *549505* | *-2800* | *-495* | *7200.5* | *3721* | *0.9695* | *0.9619* |
| *Archivo3* | *140000* | *138240* | *139393* | *-1760* | *-607* | *2414.2* | *1043* | *0.9888* | *0.9893* |
| *Archivo 4* | *561000* | *558720* | *559873* | *-2280* | *-1127* | *13039* | *7498* | *0.9744* | *0.9823* |

**Tabla 3.30** Comparación de las muestras pérdidas por los convertidores

## 3.3.3 CONVIRTIENDO A FORMATO CDA

El vector de referencia para el archivo uno se localiza en la posición 5785 para Pinnacle y 7178 para Roxio la información recuperada es la siguiente.

*Bits con error Pinnacle=[]* *// No hay bits con error*
*Mensaje='Zacatenco'*

*Bits con error Roxio=[24 123]*
*Mensaje='Zacatenco'*

| ***Ind1*** | ***Ind2*** | ***M1*** | ***M2*** | ***Mod1*** | ***Mod2*** | ***Pin1*** | ***Pin2*** | ***Bit*** | ***Rox1*** | ***Rox2*** | ***Bit*** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *5810* | *5922* | *0.1751* | *0.1397* | *0.1215* | *0.1875* | *0.1987* | *0.3155* | *0* | *0.3600* | *0.4783* | *0* |
| *6036* | *6160* | *0.1352* | *0.1279* | *0.1875* | *0.1215* | *0.3126* | *-0.2125* | *1* | *0.4986* | *0.3693* | *1* |
| *6345* | *6523* | *0.1076* | *0.1715* | *0.1215* | *0.1875* | *0.2009* | *-0.3229* | *0* | *0.3184* | *0.5264* | *0* |
| *6637* | *6762* | *-0.1067* | *-0.1082* | *-0.1875* | *-0.1215* | *-0.3253* | *0.2001* | *1* | *-0.5645* | *-0.3147* | *1* |
| *6976* | *9684* | *0.1362* | *-0.1556* | *0.1215* | *-0.1875* | *-0.2111* | *0.3123* | *0* | *0.3394* | *-0.5370* | *0* |
| *9974* | *10225* | *-0.1114* | *-0.1111* | *-0.1875* | *-0.1215* | *-0.3239* | *0.1991* | *1* | *-0.4316* | *-0.4239* | *1* |
| *10314* | *10349* | *-0.1102* | *-0.1516* | *-0.1215* | *-0.1875* | *-0.2115* | *0.3134* | *0* | *-0.3709* | *-0.5604* | *0* |
| *10551* | *11600* | *0.1161* | *0.1273* | *0.1215* | *0.1875* | *-0.2095* | *0.3104* | *0* | *0.3407* | *0.5414* | *0* |
| *11767* | *11938* | *-0.1067* | *0.1688* | *-0.1875* | *0.1215* | *0.3117* | *-0.2137* | *1* | *-0.5583* | *0.3072* | *1* |

**Tabla 3.31** Muestras recuperadas del archivo 1 cambiando al formato CDA

El vector de referencia para el archivo dos se localiza en la posición 5785 para Pinnacle y 7178 para Roxio la información recuperada es la siguiente.

*Bits con error Pinnacle=[]* *// No hay bits con error*
*Mensaje='Zacatenco'*

*Bits con error Roxio=[]* *// No hay bits con error*
*Mensaje='Zacatenco'*

| ***Ind1*** | ***Ind2*** | ***M1*** | ***M2*** | ***Mod1*** | ***Mod2*** | ***Pin1*** | ***Pin2*** | ***Bit*** | ***Rox1*** | ***Rox2*** | ***Bit*** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *5881* | *6497* | *-0.1218* | *0.1227* | *-0.1215* | *0.1875* | -0.3136 | 0.4751 | *0* | *-0.3031* | *0.3890* | *0* |
| *6858* | *6944* | *-0.1297* | *-0.1085* | *-0.1875* | *-0.1215* | 0.4735 | -0.3148 | *1* | *-0.4893* | *-0.2483* | *1* |
| *7189* | *7298* | *-0.1430* | *0.1117* | *-0.1215* | *0.1875* | -0.3179 | 0.4746 | *0* | *-0.3118* | *0.4107* | *0* |
| *7603* | *7924* | *0.1236* | *-0.1358* | *0.1875* | *-0.1215* | -0.4893 | -0.3170 | *1* | *0.4385* | *-0.2809* | *1* |
| *8557* | *8759* | *0.1306* | *-0.1544* | *0.1215* | *-0.1875* | -0.3187 | -0.4878 | *0* | *0.2871* | *-0.4430* | *0* |
| *8931* | *9070* | *0.1730* | *-0.1808* | *0.1875* | *-0.1215* | 0.4731 | 0.3062 | *1* | *0.4444* | *-0.2780* | *1* |
| *9136* | *9354* | *-0.1521* | *0.1241* | *-0.1215* | *0.1875* | -0.3118 | -0.4855 | *0* | *-0.2532* | *0.4324* | *0* |
| *9447* | *9837* | *0.1265* | *-0.1620* | *0.1215* | *-0.1875* | -0.3205 | -0.4897 | *0* | *0.2267* | *-0.4613* | *0* |
| *10111* | *10263* | *-0.1146* | *0.1311* | *-0.1875* | *0.1215* | -0.4858 | -0.3138 | *1* | *-0.4016* | *0.2803* | *1* |

**Tabla 3.32** Muestras recuperadas del archivo dos cambiando al formato CDA

El vector de referencia para el archivo tres se localiza en la posición 5785 para Pinnacle y 7178 para Roxio la información recuperada es la siguiente.

*Bits con error Pinnacle=[]* *// No hay bits con error*
*Mensaje='Zacatenco'*

*Bits con error Roxio=[19 27]*
*Mensaje='Zacatenco'*

| ***Ind1*** | ***Ind2*** | ***M1*** | ***M2*** | ***Mod1*** | ***Mod2*** | ***Pin1*** | ***Pin2*** | ***Bit*** | ***Rox1*** | ***Rox2*** | ***Bit*** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *5801* | *5851* | *-0.1727* | *0.1680* | *-0.1215* | *0.1875* | *-0.2893* | *0.4433* | *0* | *-0.2917* | *0.4784* | *0* |
| *5902* | *5943* | *0.1466* | *-0.1547* | *0.1875* | *-0.1215* | *-0.4376* | *-0.2873* | *1* | *0.4699* | *-0.3201* | *1* |
| *5999* | *6047* | *-0.1530* | *0.1734* | *-0.1215* | *0.1875* | *-0.2953* | *0.4317* | *0* | *-0.3005* | *0.4649* | *0* |
| *6080* | *6140* | *-0.1541* | *-0.1624* | *-0.1875* | *-0.1215* | *0.4454* | *-0.2937* | *1* | *-0.4649* | *-0.2879* | *1* |
| *6202* | *6265* | *-0.1487* | *-0.1287* | *-0.1215* | *-0.1875* | *0.2771* | *-0.4360* | *0* | *-0.2894* | *-0.4567* | *0* |
| *6314* | *6382* | *0.1408* | *0.1085* | *0.1875* | *0.1215* | *0.4265* | *-0.2934* | *1* | *0.4513* | *0.3171* | *1* |
| *6470* | *6540* | *-0.1610* | *-0.1821* | *-0.1215* | *-0.1875* | *-0.2901* | *0.4467* | *0* | *-0.3291* | *-0.5205* | *0* |
| *6608* | *6674* | *-0.1760* | *-0.1855* | *-0.1215* | *-0.1875* | *0.2743* | *-0.4539* | *0* | *-0.3402* | *-0.4913* | *0* |
| *6741* | *6812* | *-0.1560* | *-0.1360* | *-0.1875* | *-0.1215* | *-0.4487* | *0.2768* | *1* | *-0.5138* | *-0.2979* | *1* |

**Tabla 3.33** Muestras recuperadas del archivo tres cambiando al formato CDA

El vector de referencia para el archivo cuatro se localiza en la posición 5785 para Pinnacle y 7178 para Roxio la información recuperada es la siguiente.

*Bits con error Pinnacle=[]* *// No hay bits con error*
*Mensaje='Zacatenco'*

*Bits con error Roxio=[76 90]*
*Mensaje='Zacatenco'*

| ***Ind1*** | ***Ind2*** | ***M1*** | ***M2*** | ***Mod1*** | ***Mod2*** | ***Pin1*** | ***Pin2*** | ***Bit*** | ***Rox1*** | ***Rox2*** | ***Bit*** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *5803* | *6000* | *0.1897* | *0.1309* | *0.1215* | *0.1875* | *0.3444* | *0.5344* | *1* | *0.2691* | *0.3188* | *0* |
| *6228* | *6431* | *0.1604* | *-0.1453* | *0.1875* | *-0.1215* | *0.5248* | *0.3412* | *1* | *0.3256* | *-0.2559* | *1* |
| *6618* | *6855* | *0.1138* | *-0.1263* | *0.1215* | *-0.1875* | *0.3407* | *0.5398* | *0* | *0.1511* | *-0.3663* | *0* |
| *7019* | *7239* | *-0.1905* | *0.1053* | *-0.1875* | *0.1215* | *-0.5365* | *-0.3510* | *1* | *-0.3309* | *0.1857* | *1* |
| *7420* | *7645* | *-0.1120* | *0.1593* | *-0.1215* | *0.1875* | *0.3287* | *-0.5404* | *0* | *-0.1845* | *0.3071* | *0* |
| *7856* | *8132* | *-0.1199* | *0.1710* | *-0.1875* | *0.1215* | *-0.5383* | *-0.3506* | *1* | *-0.3462* | *0.2558* | *0* |
| *8306* | *8499* | *-0.1618* | *0.1437* | *-0.1215* | *0.1875* | *-0.3477* | *-0.5497* | *0* | *-0.1799* | *0.2650* | *1* |
| *8677* | *8911* | *-0.1339* | *0.1395* | *-0.1215* | *0.1875* | *0.3475* | *0.5350* | *0* | *-0.2357* | *0.2962* | *0* |
| *9001* | *9158* | *0.1481* | *-0.1242* | *0.1875* | *-0.1215* | *-0.5346* | *0.3368* | *1* | *0.3215* | *-0.1801* | *0* |

**Tabla 3.34** Muestras recuperadas del archivo cuatro cambiando al formato CDA

Para los dos convertidores el método propuesto es eficiente. Es interesante observar de la tabla 3.43 que los archivos 1,2 y 3 recuperados tienen mas muestras que los originales mientras que el archivo 4 tiene una cantidad menor que el original

| *Archivos* | *Muestras* | *Rec-Pin* | *Rec-Rox* | *Per-Pin* | *Per-Rox* | *E -Pin* | *E-Rox* | *C-Pin* | *C-Rox* |
|---|---|---|---|---|---|---|---|---|---|
| *Archivo 1* | *314304* | *318900* | *316520* | *+4596* | *+2216* | *2082* | *1842* | *0.9634* | *0.9523* |
| *Archivo 2* | *550000* | *551900* | *550612* | *+1900* | *+612* | *2194* | *2086* | *0.9991* | *0.9860* |
| *Archivo3* | *140000* | *142200* | *141310* | *+2200* | *+1310* | *3198* | *3204* | *0.9468* | *0.9417* |
| *Archivo 4* | *561000* | *560952* | *559873* | *-48* | *-1127* | *12243* | *10943* | *0.9995* | *0.9990* |

**Tabla 3.35** Comparación de los archivos recuperados y su correlación

.

En las figuras 3.5, 3.6, 3.7 y 3.8 se muestran los retrasos de los archivos recuperados después de aplicar el convertidor Pinnacle para el formato mp3 aquí se muestra que es bastante claro este retraso.

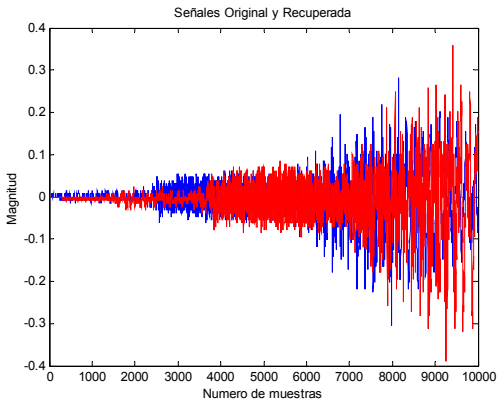



**Figura 3.5** Retraso del archivo 1

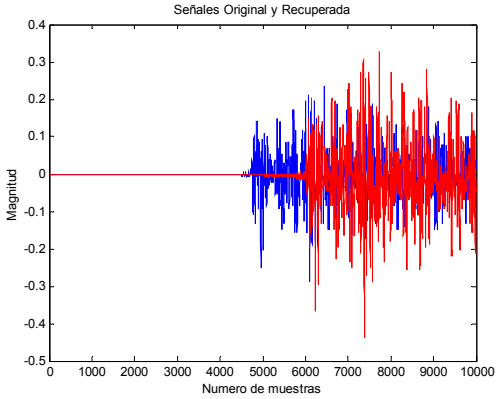



**Figura 3.6** Retraso del archivo 2

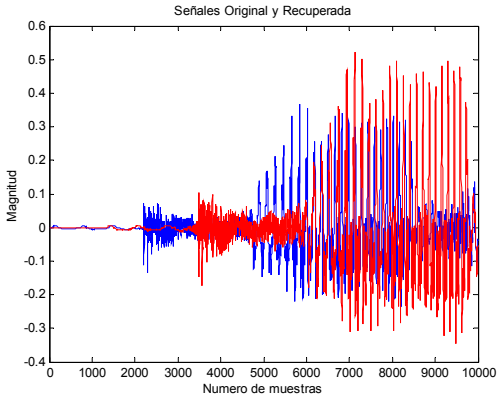



**Figura 3.7** Retraso del archivo 3

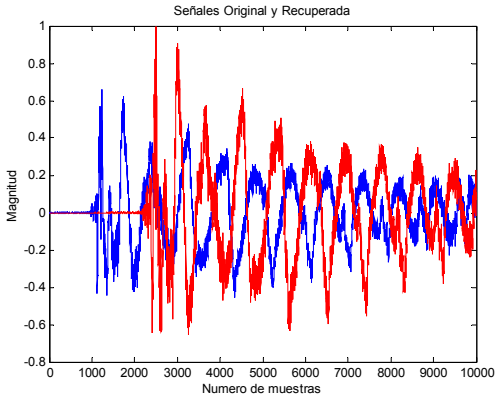



**Figura 3.8** Retraso del archivo 4

# Capítulo 4

## 4.1 CONCLUSIONES

A través de las pruebas realizadas se demuestra que el audio puede ser alterado de varios formas ya sea escalando la amplitud de las muestras, modificando su frecuencia, se puede perder un buen número de estas muestras y sufrir varios tipos de manipulaciones y aun así la calidad de sonido sigue siendo aceptable.

También hubo casos en que el índice de correlación entre el archivo original y el recuperado era bueno y visualmente ambos archivos se veían idénticos pero al escuchar el archivo la calidad de sonido no resultaba muy buena. La realización de esta tesis tuvo como objetivo presentar la investigación de dos métodos para colocar la marca de agua en un archivo de audio usando wavelet 'Daubechies' especialmente la 'Db1' y 'Db3' y la utilización del código BCH(15,7,2) y también demostrar que al menos uno de estos algoritmos es capaz de resistir estos cambios cada uno de los dos algoritmos presenta sus ventajas y desventajas.

Debemos tener en cuenta que en el caso de las pruebas realizadas con el convertidor comercial se desconoce que tipo de codificación emplean si es simétrica o asimétrica, a que se debe la modificación en la cantidad de muestras en los diferentes formatos WAV, MP3 y CDA y el gran aumento en la amplitud de las muestras así como también el cambio de signos que se presentan en algunas muestras de los archivos y para cada convertidor comercial que se utilizo cada uno de estos problemas es distinto.

El primer algoritmo estudiado fue el Patchwork resulta eficiente siempre y cuando no se someta a cambios de formato especialmente usando el convertidor comercial o se realicen pruebas de filtrado en el archivo como se demuestra en los resultados de las pruebas aunque esto parece ser una desventaja debemos tener en cuenta que el método no requiere almacenar las posiciones donde se localiza la información lo único que se requiere es conocer la clave y partir de esto se generan las posiciones, incluso la separación entre las muestras queda a nuestro criterio. Por lo tanto se puede decir que el algoritmo Patchwork coloca una marca de agua débil ya que no resiste el procesamiento, también se comento que estas marcas débiles son buenas cuando el interés es detectar si un trabajo ha sido manipulado de alguna forma sin autorización y no necesariamente la de recuperar la información, si no se recupera la información se llega a la conclusión que este trabajo ha sido alterado.

El método propuesto es robusto contra varios tipos de procesamiento como cambios de formato usando el convertidor comercial, filtrado y otras manipulaciones como se demostró en las pruebas realizadas pero esta limitado al intervalo de la codificación de 8 bits del archivo que sé este utilizando además las posiciones deben ser conocidas en el momento de recuperar la información.

Los resultados obtenidos se pueden mejorar aumentando la diferencia entre el par de muestras de la codificación empleada.

Las conclusiones finales son las siguientes:

➢Las mejores wavelets para introducir la marca son la Daubechies 1,2 y 3. Y colocar cuatro caracteres por cada diez segundos para lograr un mejor resultado.

➢El método propuesto es robusto contra cambios de formato MP3, WAV, CDA con los distintos convertidres.

➢El algoritmo Patchwork no es robusto en cambios de formato y frecuencia con todos los convertidores usados durante las pruebas**.**

## 4.2 TRABAJO A FUTURO

Un trabajo a futuro es la creación de un algoritmo en el cual sin almacenar las posiciones de la información oculta, se puede recuperar la marca y que sea robusto contra cambios de formato incluso con la utilización de algún convertidor comercial, aunque esto no seria fácil ya que tendríamos que conocer sus características como las siguientes:

1) Saber si la codificación es simétrica o asimétrica tanto en archivos de 8 y 16 bits.

2) Los filtros empleados para conocer el retardo y como modifican la señal

3) La forma en que utilizan los estándares de los formatos Wav, Mp3 Y CDA.

Con el método propuesto la marca sobrevive aun sin conocer estas características del convertidor de formato pero es necesario almacenar las posiciones además de trabajar en un intervalo determinado lo cual limita la cantidad de muestras disponibles, por estas razones la creación de un algoritmo sin estas limitantes seria conveniente especialmente en audio ya que existen muy pocos algoritmos para esta aplicación de la marca de agua.

**ANEXO A**

**PRUEBAS ADICIONALES**

## Intercambiando muestras

En este anexo se explican algunos resultados que conciernen a la incrustación de una marca de agua en un archivo wav, empezaremos comentando que el método de cuantización tiene el problema que nosotros debemos conocer las posiciones de antemano para poder recuperar la información, una pregunta lógica seria ¿Porque no solo intercambiar muestras para ocultar la información?. Esto seria más sencillo que localizar un intervalo de valores y utilizar los valores de cuantización de 8 bits del formato wav, además de almacenar las posiciones como lo hace el método de cuantización.

A continuación se explicaran las razones por lo cual esto resulta poco efectivo especialmente cuando se usa el convertidor comercial, primero se empezara mostrando la señal que se utilizara, este archivo corresponde a la señal del capitulo 3 de la figura 3.1 y se dan sus características.

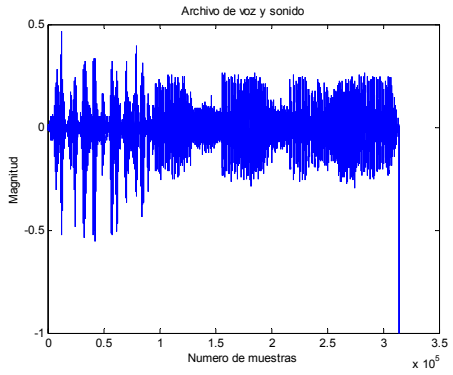



**Figura A.1**

| Archivo | Frecuencia | Codificación | Muestras | Energía |
|---|---|---|---|---|
| Archivo 2 | 44100 | 8 | 314304 | 1632.1 |

**Tabla** A.1

Este archivo se eligió por razones que se explicaran mas adelante, lo primero que se va a realizar es aplicar una wavelet ‘Daubechies 3’ a nuestro archivo y sé grafica la subseñal de promedio.

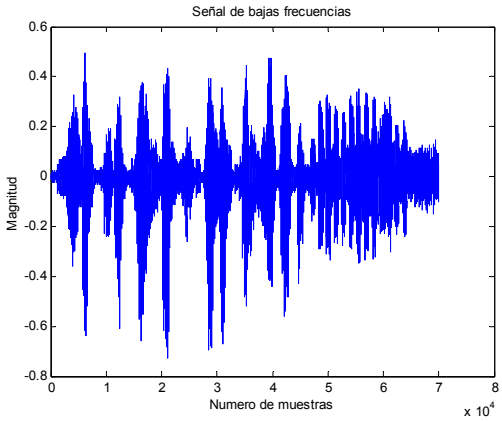



**Figura A.2** Señal de bajas frecuencias *u*

Para ocultar la marca lo que se va a realizar es intercambiar muestras, por lo tanto este intercambio se tiene que realizar sobre un intervalo que sea mas o menos constante para no intercambiar muestras de un gran valor con otras de un valor pequeño, teniendo en cuenta esto se decide utilizar el intervalo de valores de la señal de promedio que están comprendidos entre 65000 y 75000 para ocultar la marca, la palabra que se va a ocultar es 'Esime'. Se convierte la palabra a binario y la codificamos quedando de la siguiente forma:

mcontraco =

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |

Para el intercambio de las muestras para ocultar el bit comparamos un par de muestras de la señal de frecuencias bajas a partir de la posición uno, que este caso es 65000 y nos se va desplazando de 4 en 4 para localizar las muestras, es decir, 65000-65004, 65008-65012,...., hasta ocultar el ultimo bit. Para ocultar un bit con valor de uno la primera muestra debe ser la mayor de las dos y en caso de ser un bit cero la muestra uno será la menor.

*Sí bit=1*

*M1>M2*

*Sí bit=0*

*M1<M2*

Ocultando la información de esta forma dentro de Matlab y aplicando la wavelet inversa y además de escribir el archivo a su formato original para posteriormente recuperar la información el resultado es el siguiente:

*Bits con error= [ 10  65  73]*

*Mensaje =Esime*

De los resultados se tiene que existen 3 bits con error, pero como se utiliza un código de corrección estos errores se corrigen y la información se recupera correctamente, ahora repetiremos la misma prueba pero usando el convertidor comercial para modificar su formato a mp3, a una frecuencia de 128 Kbps se debe recordar que para el formato mp3 se usa un vector de referencia en el dominio wavelet para localizar la información oculta y para este archivo corresponde de las posiciones 5800-5999 y a continuación se muestran los resultados obtenidos:

*Bits con error =[ 8  17  21  24  31  39  42  47  50  55  64  69  71  75]*

*Mensaje = □pi]#*

$Vector_{pos}$ *= 6440*

De los resultados obtenidos se tiene, que de los 75 bits ocultos en el archivo 14 bits son recuperados con error y el código de corrección no puede solucionar este problema debido al gran número de errores, solo el carácter 'i' es recuperado correctamente de la palabra 'Esime' y el vector de posición esta localizado en la posición 6440.

Existe una forma para que esta técnica de intercambiar muestras funcione correctamente y se debe utilizar preferentemente en el dominio wavelet para que los cambios realizados se conserven mejor y la forma de realizarla es la, primero se deben tomar nuestras dos muestras para ocultar el bit, entonces se determina cual de las dos muestras es la mayor y se multiplica por dos, forzosamente se tiene que multiplicar por dos ya que un factor menor no crea la suficiente diferencia entre ambas muestras para poder recuperar la información oculta correctamente en el archivo, hecho esto se procede a realizar el intercambio de las muestras.

*Si M1>M2*

*M1=2*M1*

*Si M2>M1*

*M2=2*M2*

Usando esta forma de intercambiar las muestras y repitiendo la prueba del formato mp3 a 128 Kbps los resultados son los siguientes:

*Bits con error =[ 8  10  27  35  73  75]*

*Mensaje =Esime*

Es claro que el intercambio de muestras si funciona al duplicar el valor de la muestra de mayor valor pero se debe tener cuidado de seleccionar un grupo de valores que al duplicar su valor no sobresalgan demasiado o dañen al archivo considerablemente por esta razón se eligió el grupo de valores comprendidos entre 65000-75000 ya que tienen una amplitud relativamente pequeña y se pueden modificar, además las otras señales no tienen un grupo de valores que sean mas o menos constantes en un intervalo y seria mas complicado aplicar esta técnica.

También debe tener cuidado de no elegir un grupo de muestras que tengan una amplitud demasiado pequeña ya que al aplicar mas de un nivel de wavelet y utilizar un criterio de umbral de energía se corre el riesgo de que algunas muestras sean eliminadas pero tampoco debemos elegir un grupo de muestras con amplitud considerable, la cantidad de bits que se ocultan debe ser pequeña para no dañar al archivo y usarla sobre un intervalo constante de muestras, esta forma de intercambiar muestras se puede utilizar para archivos de 8 bits y 16 bits y se recupera la información en ambos casos, la separación entre muestras es de acuerdo a nuestro criterio aquí propusimos una separación de cuatro pero puede ser mayor sin problema alguno, también depende de la cantidad de bits a ocultar y el número de muestras disponibles. De acuerdo a los resultados las mejores'Daubechies' resultaron ser las Daubechies 3 y la Daubechies 5 para esta técnica.

## ANEXO B

## ELIMINACIÓN DE CIFRAS SIGNIFICATIVAS

A continuación se dan resultados para demostrar por que las formulas 2.2 y 2.3 no se deben emplear directamente aun cuando se oculta en la segunda cifra y en su lugar utilizar las modificadas 2.8 y 2.9 las cuales se muestran a continuación.

Fórmulas originales del Patchwork

$$C_{11} = P + 1 \qquad (2.2)$$
$$C_{22} = P - 1$$

$$C_{11} = P - 1 \qquad (2.3)$$
$$C_{22} = P + 1$$

Fórmulas modificadas para lograr un mejor resultado

$$C_{11} = P + 2$$
$$C_{22} = P - 1 \qquad (2.8)$$

$$C_{11} = P - 1$$
$$C_{22} = P + 2 \qquad (2.9)$$

Cuando se aplica el algoritmo Patchwork se le puede aplicar otra transformada distinta a la wavelet como la transformada coseno y eliminar las cifras significativas a partir de la tercera posición este cambio no altera la calidad de sonido del archivo pero si es un ataque fuerte para el algoritmo incluso cuando se oculta en la segunda cifra, primero se muestran los valores de la transformada coseno de las posiciones 4800:4809 del archivo 2 de prueba con sus cifras originales y posteriormente cuando solo tiene dos cifras al aplicar la transformada coseno al archivo original por bloques de 64 elementos.

*Valores en dominio coseno con todas sus cifras*

*-0.0009 0.0059 -0.3650 0.1956 0.3322 0.1817 -0.1169 0.0946 -0.0433 0.0734*

*Valores en dominio coseno con solo dos cifras*

*0 0 -0.3600 0.1900 0.3300 0.1800 -0.1100 0.0900 -0.0400 0.0700*

Para todos los valores en dominio coseno se realiza lo mismo posteriormente al aplicar la transformada coseno se escribe el archivo a su formato original.

Primero se muestra lo recuperado al aplicar el algoritmo Patchwork con las fórmulas 2.2 y 2.3 en dominio wavelet y después aplicar la eliminación de cifras en dominio coseno. La clave es 'Esime' y la información 'Zacatenco'.

*Bits con error = [17 21 28 39 43 50 56 60 64 69 86 102 106 119 122 124 127 132]*
*Mensaje=□akaN%Fco*

Caracteres con error=[1 3 5 6 7]

| ***Ind1*** | ***Ind2*** | ***M1*** | ***M2*** | ***Mod1*** | ***Mod2*** | ***Rec1*** | ***Rec2*** | ***Bit*** |
|---|---|---|---|---|---|---|---|---|
| *6798* | *7469* | *-0.0353* | *0.0308* | *-0.0253* | *0.0408* | *-0.0230* | *0.0357* | *0* |
| *6803* | *7464* | *-0.0590* | *-0.0701* | *-0.0790* | *-0.0501* | *-0.0627* | *-0.0550* | *1* |
| *6808* | *7459* | *-0.0014* | *-0.0750* | *-0.0314* | *-0.0550* | *-0.0184* | *-0.0426* | *0* |
| *6813* | *7454* | *-0.0227* | *-0.0349* | *-0.0427* | *-0.0249* | *-0.0393* | *-0.0313* | *1* |
| *6818* | *7449* | *-0.1093* | *-0.0439* | *-0.1193* | *-0.0339* | *-0.1093* | *-0.0307* | *0* |
| *6823* | *7444* | *-0.1257* | *0.0333* | *-0.1457* | *0.0233* | *-0.1321* | *0.0225* | *1* |
| *6828* | *7439* | *-0.0271* | *-0.0002* | *-0.0071* | *-0.0202* | *-0.0050* | *-0.0190* | *0* |
| *6833* | *7434* | *0.0957* | *-0.0343* | *0.0557* | *-0.0743* | *0.0574* | *-0.0614* | *0* |
| *6838* | *7426* | *0.1121* | *0.0340* | *0.1321* | *0.0140* | *0.1200* | *0.0128* | *1* |

**Tabla B.1** Valores recuperados

Es bastante claro que la información no se recupera al aplicar la eliminación de las terceras cifras en adelante, la correlación entre el archivo original y el recuperado es de 0.9964. Al aplicar el algoritmo Patchwork con las fórmulas 2.8 y 2.9 lo recuperado es lo siguiente.

*Bits con error = [21 39 56 99]*
*Mensaje = Zacatenco*

Caracteres con error = [] // No hay errores

| ***Ind1*** | ***Ind2*** | ***M1*** | ***M2*** | ***Mod1*** | ***Mod2*** | ***Rec1*** | ***Rec2*** | ***Bit*** |
|---|---|---|---|---|---|---|---|---|
| *6798* | *7469* | *-0.0353* | *0.0308* | *-0.0253* | *0.0508* | *-0.0226* | *0.0433* | *0* |
| *6803* | *7464* | *-0.0590* | *-0.0701* | *-0.0890* | *-0.0501* | *-0.0653* | *-0.0556* | *1* |
| *6808* | *7459* | *-0.0014* | *-0.0750* | *-0.0314* | *-0.0650* | *-0.0283* | *-0.0577* | *0* |
| *6813* | *7454* | *-0.0227* | *-0.0349* | *-0.0527* | *-0.0249* | *-0.0443* | *-0.0257* | *1* |
| *6818* | *7449* | *-0.1093* | *-0.0439* | *-0.1193* | *-0.0439* | *-0.1167* | *-0.0350* | *0* |
| *6823* | *7444* | *-0.1257* | *0.0333* | *-0.1557* | *0.0233* | *-0.1426* | *0.0208* | *1* |
| *6828* | *7439* | *-0.0271* | *-0.0002* | *-0.0071* | *-0.0302* | *-0.0064* | *-0.0304* | *0* |
| *6833* | *7434* | *0.0957* | *-0.0343* | *0.0557* | *-0.0843* | *0.0574* | *-0.0729* | *0* |
| *6838* | *7426* | *0.1121* | *0.0340* | *0.1421* | *0.0140* | *0.1299* | *0.0187* | *1* |

**Tabla B.2** Valores recuperados

Al aplicar las fórmulas 2.8 y 2.9 la información se recupera correctamente, esta prueba se realizo con diferentes bloques de 8, 16, 32, 64 y 128 y los resultados son similares. Este ataque se puede realizar también en dominio wavelet pero no le afecta ya que la información se oculta la información en la segunda cifra y eliminar las terceras cifras en adelante no le causa ningún daño como se muestran en los siguientes resultados.

Aplicando el algoritmo Patchwork con las fórmulas 2.2 y 2.3 lo obtenido es lo siguiente con una correlación de 0.9961.

| ***Ind1*** | ***Ind2*** | ***M1*** | ***M2*** | ***Mod1*** | ***Mod2*** | ***Rec1*** | ***Rec2*** | ***Bit*** |
|---|---|---|---|---|---|---|---|---|
| *6798* | *7469* | *-0.0353* | *0.0308* | *-0.0253* | *0.0408* | *-0.0181* | *0.0440* | *0* |
| *6803* | *7464* | *-0.0590* | *-0.0701* | *-0.0790* | *-0.0501* | *-0.0600* | *-0.0487* | *1* |
| *6808* | *7459* | *-0.0014* | *-0.0750* | *-0.0314* | *-0.0550* | *-0.0267* | *-0.0517* | *0* |
| *6813* | *7454* | *-0.0227* | *-0.0349* | *-0.0427* | *-0.0249* | *-0.0328* | *-0.0191* | *1* |
| *6818* | *7449* | *-0.1093* | *-0.0439* | *-0.1193* | *-0.0339* | *-0.1067* | *-0.0193* | *0* |
| *6823* | *7444* | *-0.1257* | *0.0333* | *-0.1457* | *0.0233* | *-0.1392* | *0.0182* | *1* |
| *6828* | *7439* | *-0.0271* | *-0.0002* | *-0.0071* | *-0.0202* | *0.0030* | *-0.0193* | *0* |
| *6833* | *7434* | *0.0957* | *-0.0343* | *0.0557* | *-0.0743* | *0.0443* | *-0.0614* | *0* |
| *6838* | *7426* | *0.1121* | *0.0340* | *0.1321* | *0.0140* | *0.1224* | *-0.0030* | *1* |

**Tabla B.3** Valores recuperados

*Bits con error = [29]*
*Mensaje = Zacatenco*
*Caracteres con error = []* *// No hay errores*

Lo recuperado con el algoritmo Patchwork y las fórmulas 2.8 y 2.9 con una correlación de 0.9959 se tiene lo siguiente.

*Bits con error = []*
*Mensaje = Zacatenco*
*Caracteres con error = []* *// No hay errores*

| ***Ind1*** | ***Ind2*** | ***M1*** | ***M2*** | ***Mod1*** | ***Mod2*** | ***Rec1*** | ***Rec2*** | ***Bit*** |
|---|---|---|---|---|---|---|---|---|
| *6798* | *7469* | *-0.0353* | *0.0308* | *-0.0253* | *0.0508* | *-0.0181* | *0.0466* | *0* |
| *6803* | *7464* | *-0.0590* | *-0.0701* | *-0.0890* | *-0.0501* | *-0.0752* | *-0.0513* | *1* |
| *6808* | *7459* | *-0.0014* | *-0.0750* | *-0.0314* | *-0.0650* | *-0.0267* | *-0.0489* | *0* |
| *6813* | *7454* | *-0.0227* | *-0.0349* | *-0.0527* | *-0.0249* | *-0.0427* | *-0.0217* | *1* |
| *6818* | *7449* | *-0.1093* | *-0.0439* | *-0.1193* | *-0.0439* | *-0.1067* | *-0.0380* | *0* |
| *6823* | *7444* | *-0.1257* | *0.0333* | *-0.1557* | *0.0233* | *-0.1493* | *0.0109* | *1* |
| *6828* | *7439* | *-0.0271* | *-0.0002* | *-0.0071* | *-0.0302* | *0.0030* | *-0.0291* | *0* |
| *6833* | *7434* | *0.0957* | *-0.0343* | *0.0557* | *-0.0843* | *0.0443* | *-0.0828* | *0* |
| *6838* | *7426* | *0.1121* | *0.0340* | *0.1421* | *0.0140* | *0.1412* | *-0.0002* | *1* |

**Tabla B.4** Valores recuperados

Es bastante claro de los resultados que la s fórmulas 2.8 y 2.9 mejoran los resultados de la información recuperada y que ocultar la información en la tercera cifra es inútil ya que todas las terceras cifras se pueden eliminar y al aplicar la inversa de la transformada y escribir el archivo este sé cuantiza nuevamente y se le asigna un nuevo valor.

[1] A tutorial on wavelets from an Electrical Engineering Perspective, IEEE Antennas and Propagation Magazine, T.K.Sarkar, C.Su, M.Salazar Palma L. Garcia Castillo,Vol. 40, No. 5, October 1998

[2] A Primer on wavelets and their Scientific Applications, James S. Walker, Chapman & Hall/CRC, 1998

[3] WAVELETS AND SUBBAND CODING, Martín Vetterli, Jelena Kovacevic, Pentice Hall, 1995.

[4] CODING TECHNIQUES: An Introduction to compression and Error Control, Graham Wade, PALGRAVE, 2000

[5] Error Control Coding Second Edition,Shu Lin, Daniel J. Costello Jr., Pearson Prentice Hall, 2004.

[6] Modified Patchwork Algorithm: A novel Audio Watermarking Scheme, In-Know Hyoung Joung Kim, Department of control and Instrumentation Engineering, Kangwon National University, Korea, 2001 IEEE.

[7] Information Hiding: Tecniques for Steganography and digital watermarking, Stefan Katzenbeisser, Fabien A.P. Petitcolas,Artech House, 2000

[8] Digital Watermarking: Principles and Practice, Ingemar Cox, Mattew Miller, Jeffrey Bloom, Mathew Miller, Edit. Morgan Kaufmann,2002

[9] Disappearing Cryptography, Information Hiding,Second Edition, Peter Wayner, Morgan Kaufmann,2002

[10] Wavelet Based Audio Watermarking Techniques: Robustness and Fast Synchronization, Hong Oh Kim, Bae Keun Lee and Nam Yong Lee, KAIST, November 2001.