UNIVERSIDAD AUTÓNOMA METROPOLITANA
UNIDAD AZCAPOTZALCO

DIVISIÓN DE CIENCIAS BÁSICAS E INGENIERÍA

LICENCIATURA EN INGENIERÍA EN COMPUTACIÓN

MANUAL DE INSTALACIÓN DEL PROYECTO TERMINAL

TRANSMISIÓN DE MENSAJES DE TEXTO EN AUDIO OGG USANDO ESTEGANOGRAFÍA

Realizado por:

KARLA NAYELI CHÁVEZ SANABRIA
Matrícula 206306325

Bajo la supervisión de:

DR. FRANCISCO JAVIER ZARAGOZA MARTÍNEZ

Proyecto Terminal realizado durante los trimestre 11-I y 11-P

Septiembre de 2011

INTRODUCCIÓN

El presente manual describe los requisitos de y pasos a seguir para la instalación del programa obtenido en el Proyecto Terminal “Transmisión de mensajes de texto en audio ogg usando esteganografía”.

REQUISITOS DEL SISTEMA

El programa funciona bajo el sistema operativo Linux, además requiere de lo siguiente:

- Compilador gcc.
- Un editor de texto para poder crear y visualizar los mensajes a insertar o extraer.
- Un reproductor de audio que reproduzca los formatos wav y ogg.

Además, requiere la instalación de los siguientes repositorios:

- libfftw3
- libfftw-devel
- libsndfile1
- libsndfile-devel

Nota: Los nombres de los repositorios varían de acuerdo a la distribución de Linux que el usuario use y conforme a su arquitectura.

INSTALACIÓN

- Colocar el código fuente de este en algún directorio de su sistema de archivos
- Compilar el código fuente (Leer manual de usuario).
- Poner en marcha el ejecutable con los parámetros necesarios (Leer manual de usuario).

UNIVERSIDAD AUTÓNOMA METROPOLITANA
UNIDAD AZCAPOTZALCO

DIVISIÓN DE CIENCIAS BÁSICAS E INGENIERÍA

LICENCIATURA EN INGENIERÍA EN COMPUTACIÓN

MANUAL DE USUARIO DEL PROYECTO TERMINAL

TRANSMISIÓN DE MENSAJES DE TEXTO EN AUDIO OGG USANDO ESTEGANOGRAFÍA

Realizado por:

KARLA NAYELI CHÁVEZ SANABRIA
Matrícula 206306325

Bajo la supervisión de:

DR. FRANCISCO JAVIER ZARAGOZA MARTÍNEZ

Proyecto Terminal realizado durante los trimestre 11-I y 11-P

Septiembre de 2011

INTRODUCCIÓN

El presente manual de usuario describe el uso correcto del programa realizado en el Proyecto Terminal “Transmisión de mensajes de texto en audio ogg usando esteganografía”.

Se describirá con ejemplos sencillos el funcionamiento del programa que implementa la esteganografía en audio utilizando la técnica de codificación de fase, con el cual se puede insertar mensajes de texto en archivos de audio wav y ogg, para posteriormente extraerlos cuando el usuario lo desee.

Antes de leer este manual, asegúrese de haber seguido las instrucciones especificadas en el manual de instalación incluido en el disco de Entregables del presente Proyecto Terminal.

INSERTAR UN MENSAJE DE TEXTO EN AUDIO WAV

Para poder insertar un mensaje de texto en un archivo de audio wav es necesario contar con el código fuente ***codificacion_fase_wav.c,*** además de los siguientes archivos de entrada:

- Archivo de audio wav
- Archivo de texto con el mensaje a insertar

Es necesario que estos archivos se encuentren en la misma ubicación.

El archivo de texto que usaremos en el ejemplo se llama ***mensaje_original.txt*** y se muestra en la Figura 1.

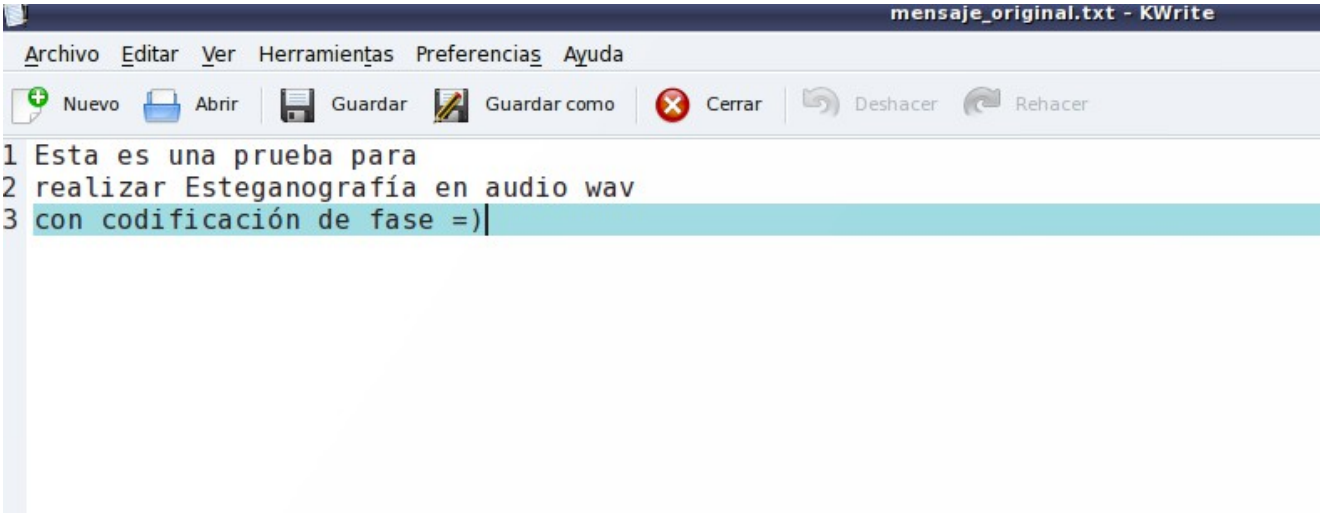


Figura 1. Archivo* mensaje_original.txt*, que contiene el mensaje a insertar

Ahora es necesario compilar el código fuente, siguiendo la sintaxis mostrada en la Figura 2:

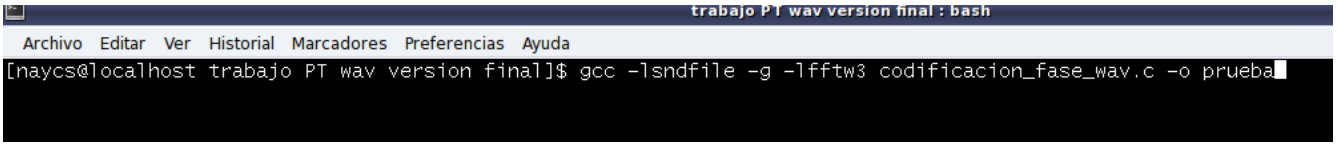


***Figura 1. Sintaxis para la compilación del código fuente codificacion_fase_*wav.c**

donde:

- **-lsndfile** = bandera que necesaria para compilar la biblioteca Libsndfile
- **-g** = incluye en el ejecutable generado la información necesaria para poder rastrear los errores usando un depurador.
- **-lfftw3** = bandera necesaria para compilar la API FFTW.
- **codificacion_fase_wav.c** = nombre del código fuente.
- **-o prueba** = genera un archivo ejecutable de nombre “prueba”. El usuario es libre de elegir el nombre de dicho archivo

Ahora ponemos en marcha el ejecutable de la siguiente manera:

./prueba archivo_wav_portador mensaje_a_insertar 1

La opción 1 indica que lo que deseamos es insertar un mensaje

Importante: Incluir la extensión de los archivos de entrada al momento de la ejecución.

Una vez hecho esto, obtenemos en pantalla algunos datos del mensaje que se desea insertar y del archivo wav donde se insertará, como lo muestra la Figura 3:

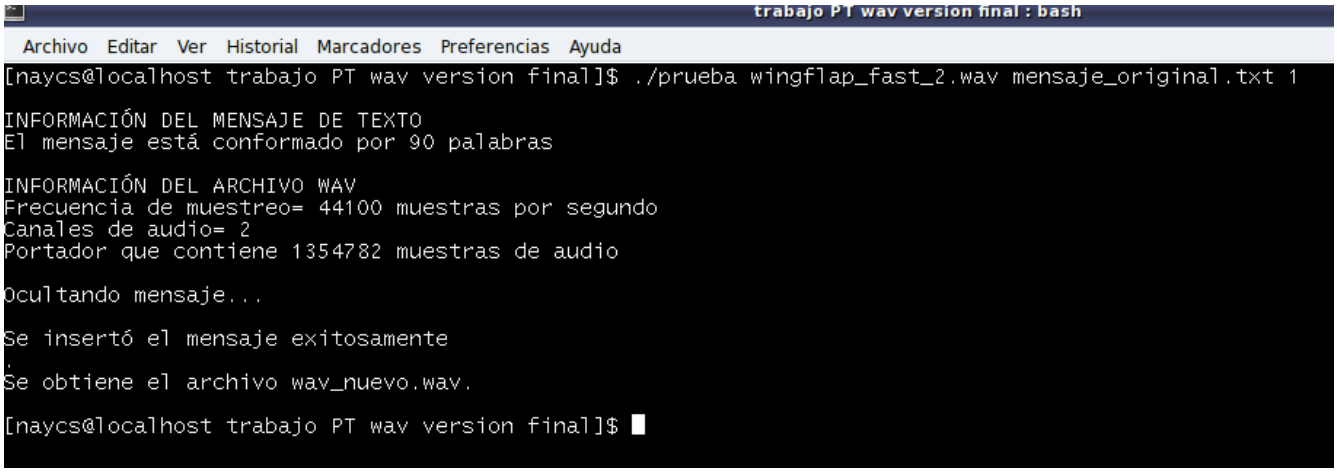


Figura 3. Resultado de la ejecución del programa para insertar mensaje

Además obtenemos las siguientes archivos de salida:

- **muestras.txt** = Archivo que contiene las muestras de audio del archivo wav escritas en números enteros
- **mensaje_binario.txt** = Este archivo contiene el mensaje a insertar convertido a código binario.
- **m.txt** = Archivo que contiene la longitud en bits del mensaje a insertar.
- **wav_nuevo.wav** = Archivo wav que contiene el mensaje oculto. Su nombre puede ser cambiado si el usuario lo desea.

El usuario puede prescindir de los 2 primeros archivos de salida si lo desea, pero el archivo ***m.txt*** debe conservarlo ya que es indispensable para la recuperación del mensaje posteriormente.

EXTRAER UN MENSAJE DE TEXTO DEL AUDIO WAV

Ahora procedemos a extraer el mensaje escondido en el archivo ***wav_nuevo.wav***, siguiendo el siguiente formato:

./prueba archivo_wav archivo_texto 2

La opción 2 indica que lo que deseamos es extraer el mensaje.

Importante: Incluir la extensión de los archivos de entrada al momento de la ejecución.

Una vez hecho esto, de nuevo obtenemos en pantalla algunos datos de los archivos de entrada, como lo muestra la Figura 4:

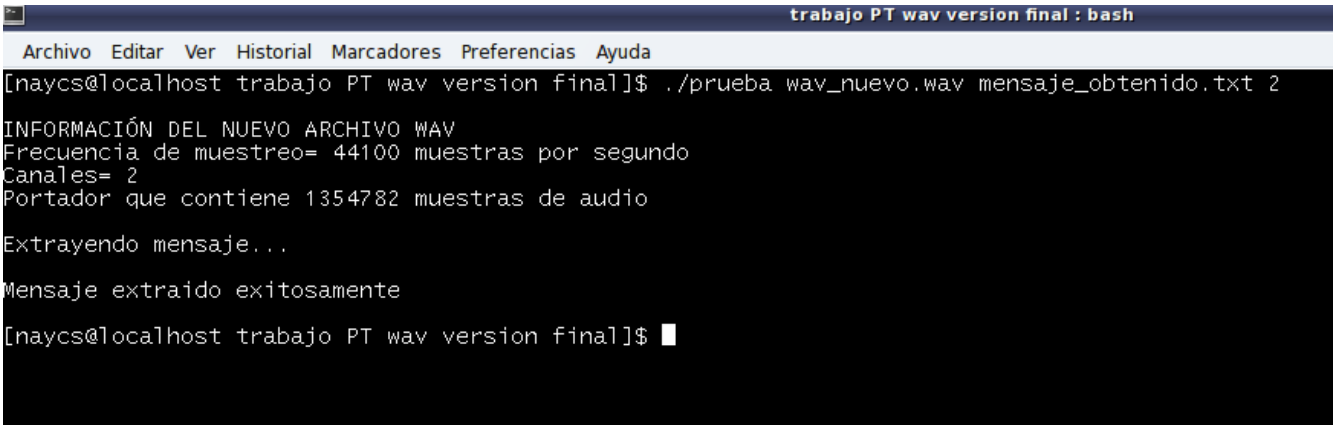


Figura 4. Resultado de la ejecución del programa para extraer mensaje

Además de que obtenemos las siguientes archivos de salida:

- **muestras_audio_msj.txt** = Archivo que contiene las muestras de audio del archivo wav de entrada escritas en números enteros
- **mensaje_binario2.txt** = Este archivo contiene el mensaje en código binario extraído del archivo de audio wav. Dicho mensaje se almacena en muestras de 8 bits
- **mensaje_obtenido.txt** = Este archivo puede tomar el nombre que el usuario decida desde el momento de la ejecución del programa. Contiene el mensaje extraído decodificado a código ASCII.

El usuario puede prescindir de los 2 primeros archivos si lo desea. El archivo importante es ***mensaje _obtenido.txt*** (o como el usuario decida nombrarlo) ya que es el que contiene el mensaje extraído. La salida obtneida en el ejemplo se muestra en la Figura 5.

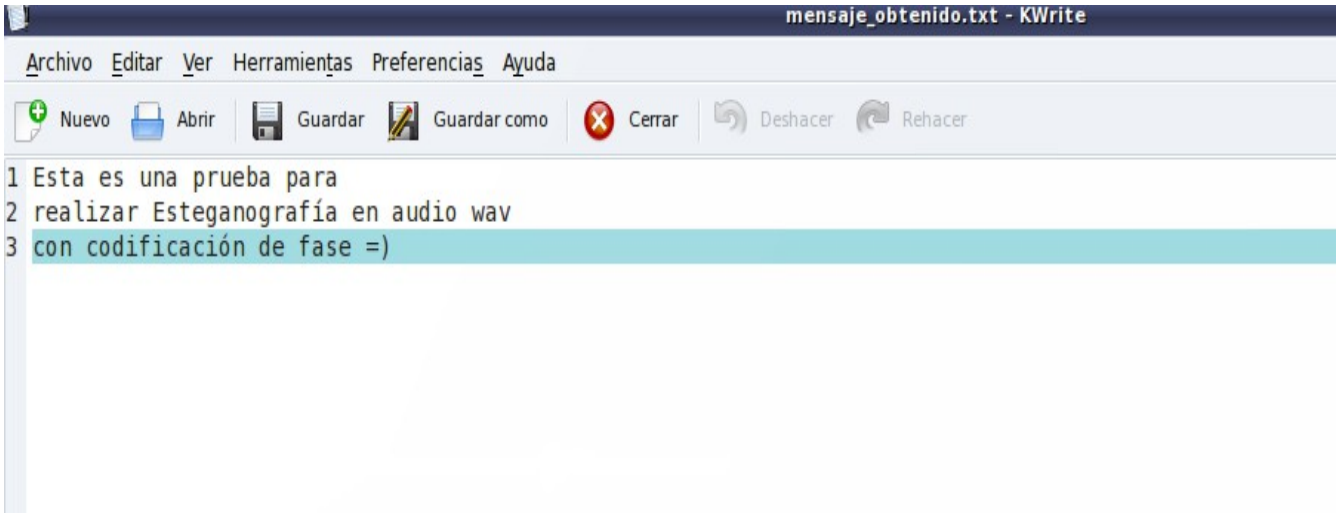


***Figura 5. Archivo* mensaje_obtenido.txt*, que contiene el mensaje extraído del archivo* wav_nuevo.wav**

INSERTAR UN MENSAJE DE TEXTO EN AUDIO OGG

Para poder insertar un mensaje de texto en un archivo de audio ogg es necesario contar con el código fuente ***codificacion_fase_ogg.c,*** además de los siguientes archivos de entrada:

- Archivo de audio ogg
- Archivo de texto con el mensaje a insertar

Es necesario que estos archivos se encuentren en la misma ubicación.

El archivo de texto que usaremos en el ejemplo se llama ***mensaje_original.txt*** y se muestra en la Figura 6.

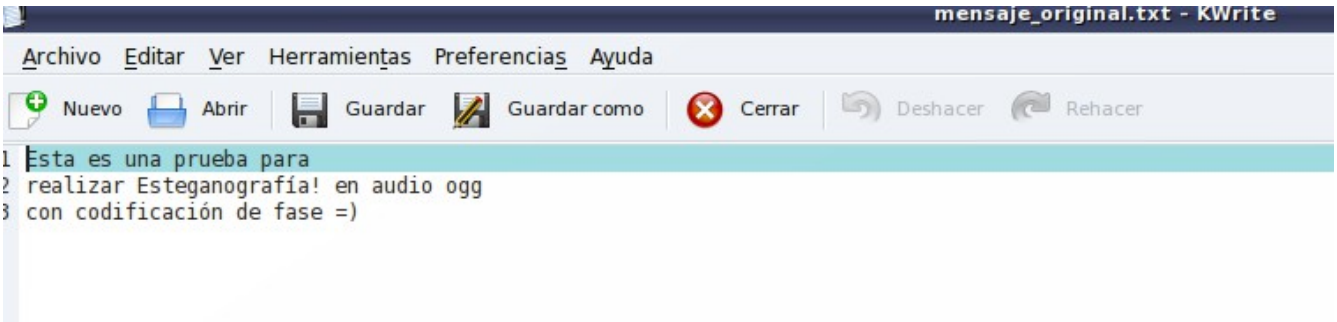


***Figura 6.* Mensaje que se desea insertar.**

Ahora ponemos en marcha el ejecutable de la siguiente manera:

./prueba archivo_ogg_portador mensaje_a_insertar 1

De nuevo, la opción 1 indica que lo que deseamos es insertar un mensaje

Importante: Incluir la extensión de los archivos de entrada al momento de la ejecución.

Una vez hecho esto, obtenemos en pantalla algunos datos del mensaje que se desea insertar y del archivo ogg donde se insertará, como lo muestra la Figura 7.

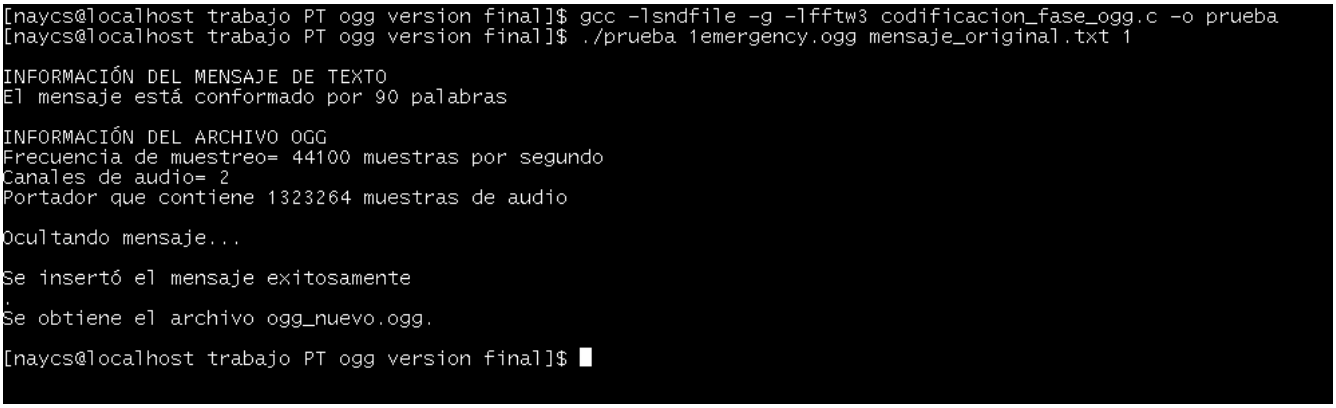


Figura 7. Resultado de la ejecución del programa para insertar mensaje

Además obtenemos las siguientes archivos de salida:

- **muestras.txt** = Archivo que contiene las muestras de audio del archivo wav escritas en números enteros.
- **mensaje_binario.txt** = Este archivo contiene el mensaje a insertar convertido a código binario.
- **m.txt** = Archivo que contiene la longitud en bits del mensaje a insertar.
- **ogg_nuevo.ogg** = Archivo ogg que contiene el mensaje oculto. Su nombre puede ser cambiado si el usuario lo desea.

El usuario puede prescindir de los 2 primeros archivos de salida si lo desea, pero el archivo ***m.txt*** debe conservarlo ya que es indispensable para la recuperación del mensaje posteriormente.

EXTRAER UN MENSAJE DE TEXTO DEL AUDIO OGG

Ahora procedemos a extraer el mensaje escondido en el archivo ogg_nuevo.ogg, siguiendo el siguiente formato:

./prueba archivo_ogg archivo_texto 2

De nuevo, la opción 2 indica que lo que deseamos es extraer el mensaje.

Importante: Incluir la extensión de los archivos de entrada al momento de la ejecución.

Una vez hecho esto, de nuevo obtenemos en pantalla algunos datos de los archivos de entrada, como lo muestra la Figura 8:

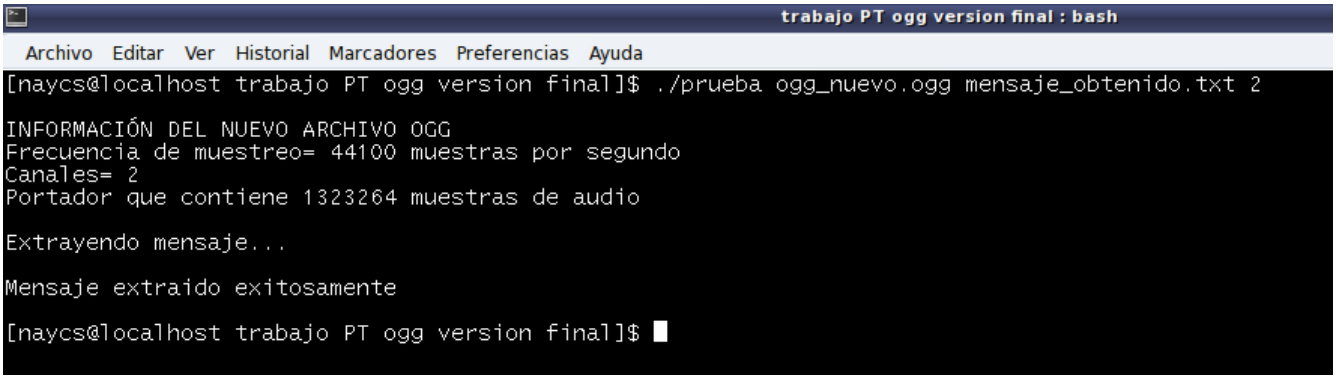


Figura 8. Resultado de la ejecución del programa para extraer mensaje

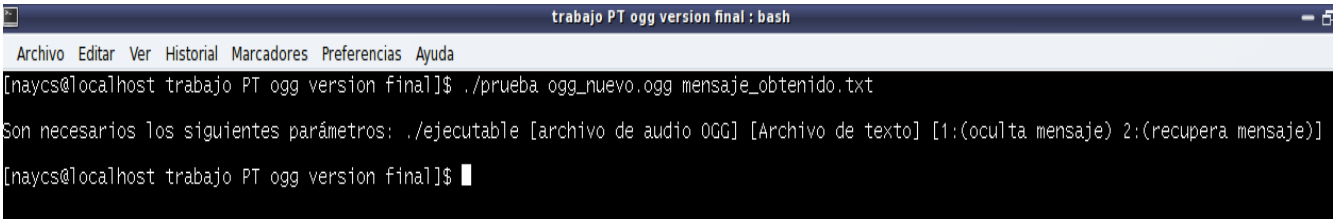
Además de que obtenemos las siguientes archivos de salida:

- **muestras_audio_msj.txt** = Archivo que contiene las muestras de audio del archivo ogg de entrada escritas en números enteros
- **mensaje_binario2.txt** = Este archivo contiene el mensaje en código binario extraído del archivo de audio ogg. Dicho mensaje se almacena en muestras de 8 bits
- **mensaje_obtenido.txt** = Este archivo puede tomar el nombre que el usuario decida desde el momento de la ejecución del programa. Contiene el mensaje extraído decodificado a código ASCII.

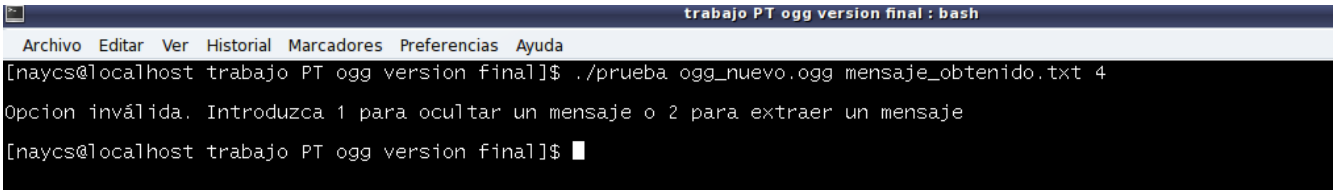
El usuario puede prescindir de los 2 primeros archivos si lo desea. El archivo importante es ***mensaje _obtenido.txt*** (o como el usuario decida nombrarlo) ya que es el que contiene el mensaje extraído.

ERRORES EN EL USO DEL SISTEMA

- Si se omite alguno de los parámetros necesarios para el insertar o extraer un mensaje de texto, aparecerá el siguiente mensaje recordando la sintaxis correcta:

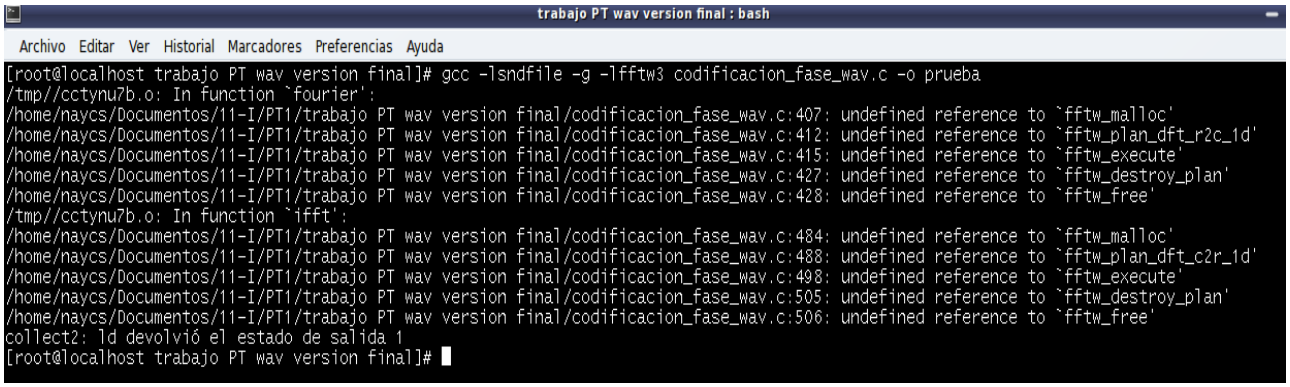


- Para insertar un mensaje se requiere del número de opción 1, para extraer un mensaje se requiere del número de opción 2. Si se ingresa un número de opción diferente aparecerá el siguiente mensaje de error:



- Si se presenta un error distinto a los anteriores, verifique que al momento de dar marcha al ejecutable, ya sea para insertar o extraer un mensaje, indique el nombre de los archivos de entrada necesarios con todo y su extensión (*.wav, *.ogg, *.txt).

- Si los mensajes de error que aparecen en su pantalla son los siguientes, asegúrese de haber instalado los repositorios necesarios para poder compilar el código fuente (Lea con atención el manual de instalación incluido en la sección de Entregables).



Los pasos anteriores se explican más detalladamente en la Figura 1

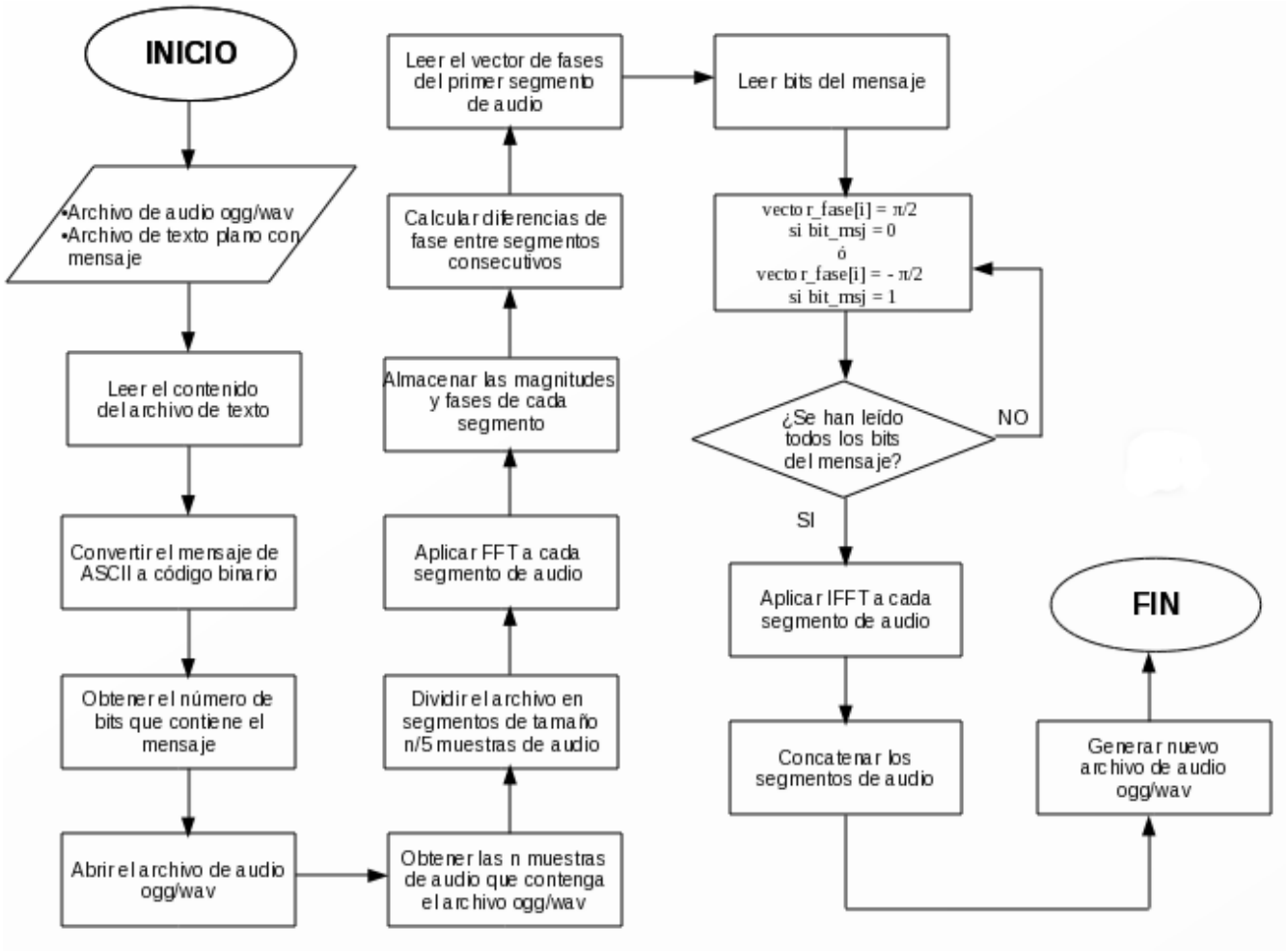


Figura 1. Diagrama de flujo para la inserción de un mensaje

Para el manejo de los archivos de audio wav y ogg, se usó la biblioteca Libsndfile [4], una API con licencia GNU escrita en lenguaje C que se usa para lectura y escritura de archivos de audio y que acepta múltiples formatos. Dicha biblioteca fue escrita para compilarse bajo el ambiente Linux pero también funciona bajo otros sistemas operativos como Windows y MacOS. Esta herramienta de trabajo resultó bastante útil ya que entre los formatos que soporta se encuentra el formato wav y recientemente fue agregado el formato ogg y resultó muy conveniente para leer los archivos (ogg/wav) de entrada y para la gereración del nuevo archivo de audio con el mensaje escondido en él.

Otra herramienta que fue de utilidad fue la biblioteca FFTW [5], escrita en C que nos permite calcular la Transformada Discreta de Fourier en una o más dimensiones, con una entrada de tamaño arbitrario y con la capacidad de hacer cálculos con números reales y complejos. También es una biblioteca escrita bajo la licencia GNU que funciona bajo la plataforma Linux y fue usada para el cálculo de las Transformada Rápida e Inversa de Fourier.

Extracción del mensaje oculto

La extracción del mensaje (objeto) del archivo de audio (portador) resultó ser un proceso más corto y simple que su inserción, para poder llevarlo a cabo son necesarias las siguientes entradas:

- Archivo de audio ogg/wav con el mensaje oculto.
- Longitud en bits del mensaje a extraer.
- Destino del mensaje extraido.

El algoritmo se resume en los siguientes pasos:

1. Leer el archivo de audio ogg/wav generado y obtener sus n muestras de audio.
2. Obtener las n/5 muestras del primer segmento de audio.
3. Aplicar la FFT al primer segmento de audio para obtener el vector de fase de éste (donde se encuentra almacenado el mensaje).
4. Decodificamos los valores del vector de fase, es decir, si el valor en el vector es π/2 se escribirá un 0 en el archivo de texto donde se guardará el mensaje, y escribirá un 1 si el valor en el vector es -π/2.
5. Repetir el paso anterior hasta que la cantidad de elementos leídos del vector de fase sea igual al número de bits del mensaje a recuperar.
6. Almacenar los unos y ceros obtenidos en muestras de 8 bits.
7. Decodificar el mensaje obtenido en binario, a su equivalente en código ASCII.
8. Almacenar el mensaje obtenido en un archivo de texto.

Los pasos anteriores se explican más detalladamente en la Figura 2

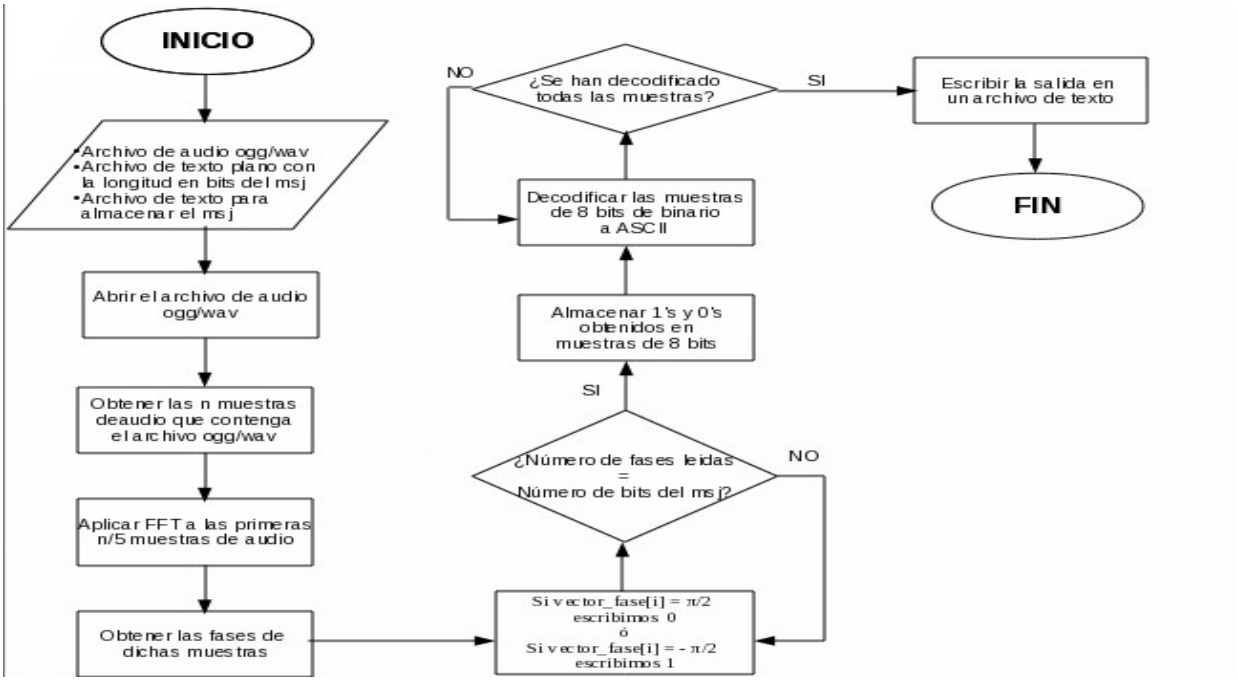


Figura 2. Diagrama de flujo para la extracción de un mensaje

En la siguiente sección se mencionarán algunas modificaciones y pruebas hechas a los algoritmos para mejorar su rendimiento y posteriormente analizaremos los resultados obtenidos.

ANÁLISIS DE RESULTADOS

Una vez integrados las funcionalidades de insertar y extraer mensajes de texto en audio ogg y wav en un sólo código escrito en C, se procedió a hacer las pruebas correspondientes.

Se escribieron 2 códigos diferentes: uno que implementara la esteganografía en audio con codificación de fase en audio ogg y otro que lo hiciera en audio wav. Ambos códigos sólo difieren en la función que formaba el nuevo archivo de audio con el mensaje oculto .

Resultados obtenidos utilizando audio wav

Para las pruebas utilizamos el mensaje de texto que muestra la Figura 3.

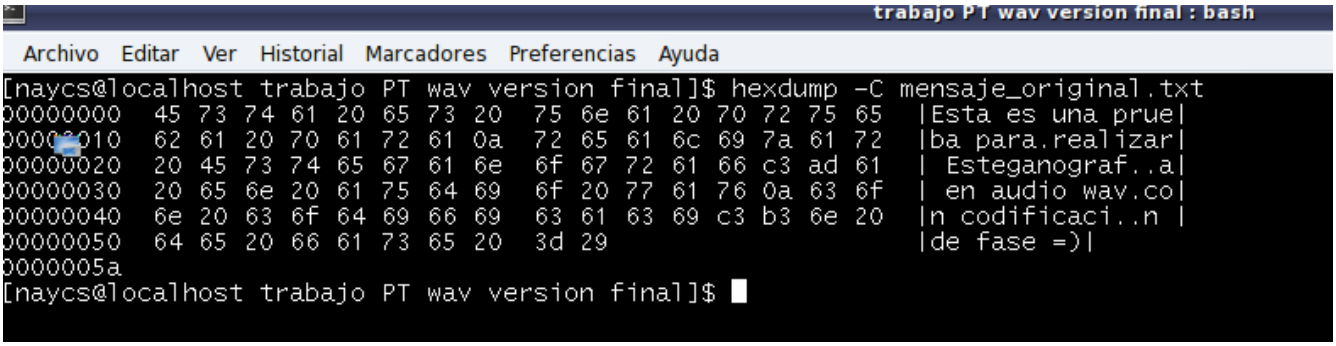


Figura 3. **Mensaje que se desea insertar.**

Estas pruebas fueron realizadas usando el código fuente ***codificacion_fase_wav.c***, compilándolo como se indica en la Figura 4:

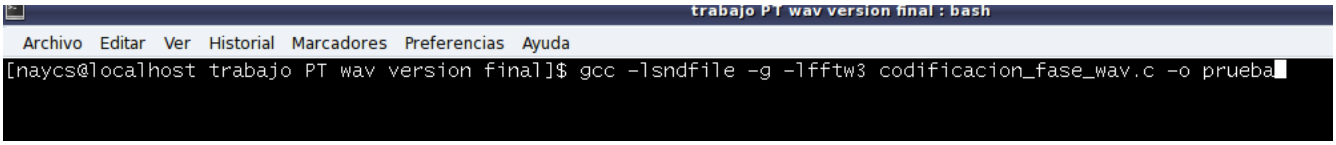


Figura 4. Sintaxis para la compilación del código fuente codificacion_fase_**wav.c**

donde:

- **-lsndfile** = bandera que necesaria para compilar la biblioteca Libsndfile
- **-g** = incluye en el ejecutable generado la información necesaria para poder rastrear los errores usando un depurador.
- **-lfftw3** = bandera necesaria para compilar la API FFTW.
- **codificacion_fase_wav.c** = nombre del código fuente.
- **-o prueba** = genera un ejecutable llamado “prueba”

Ahora ponemos en marcha el ejecutable de la siguiente manera:

./prueba archivo_wav_portador mensaje_a_insertar 1

La opción 1 indica que lo que deseamos es insertar un mensaje

Una vez hecho esto, obtenemos en pantalla algunos datos del mensaje que se desea insertar y del archivo wav donde se insertará, como lo muestra la Figura 5:

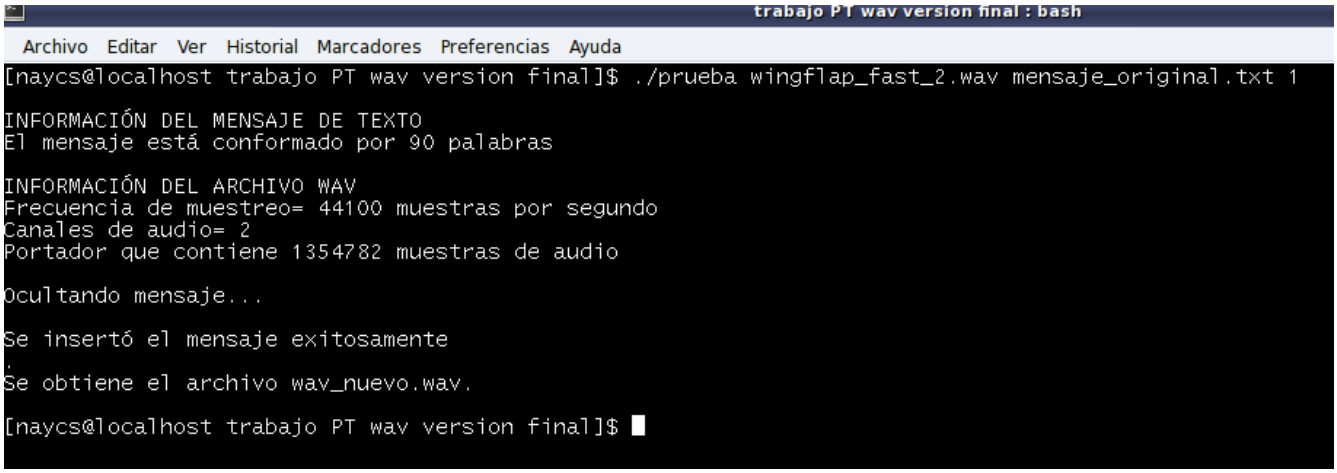


Figura 5. Resultado de la ejecución del programa para insertar mensaje

Además obtenemos las siguientes archivos de salida:

- **muestras.txt** = Archivo que contiene las muestras de audio del archivo wav escritas en números enteros
- **mensaje_binario.txt** = Este archivo contiene el mensaje a insertar convertido a código binario.
- **m.txt** = Archivo que contiene la longitud en bits del mensaje a insertar.
- **wav_nuevo.wav** = Archivo wav que contiene el mensaje oculto. Su nombre puede ser cambiado si el usuario lo desea.

El usuario puede prescindir de los dos primeros archivos de salida si lo desea, pero el archivo m.txt debe conservarlo ya que es indispensable para la recuperación del mensaje posteriormente.

Se escuchó la calidad del archivo wav resultante, que al principio no fue la deseada, ya que difería mucho del archivo wav original.

Analizando las posibles causas y la idea original del algoritmo, se dedujo que el calcular las diferencias de fase entre segmentos de audio consecutivos y después usarlas junto con las magnitudes originales para reconstruir la señal de audio original, en lugar de usar las fases originales (excepto la fase original del 1er segmento), sólo afectaba la calidad de audio y no era indispensable para el funcionamiento del algoritmo, ya que para inserción del mensaje sólo se realiza en la fase del primer segmento de audio y el resto del archivo no se modifica.

Así que se decidió no calcular las diferencias de fase y usar las 4 fases originales (excepto la primera) para reconstruir la señal de audio. Realizando esto se volvió a generar el archivo de audio wav, obteniendo una mejor fidelidad de audio con respecto al archivo wav original.

También se pudo constatar que insertar un mensaje corto (en este caso de 90 palabras) no produce diferencias perceptibles en el audio con respecto al wav original.

Ahora procedemos a extraer el mensaje escondido en el archivo wav_nuevo.wav, siguiendo el siguiente formato:

./prueba archivo_wav archivo_texto 2

La opción 2 indica que lo que deseamos es extraer el mensaje.

Una vez hecho esto, de nuevo obtenemos en pantalla algunos datos de los archivos de entrada, como lo muestra la Figura 6:

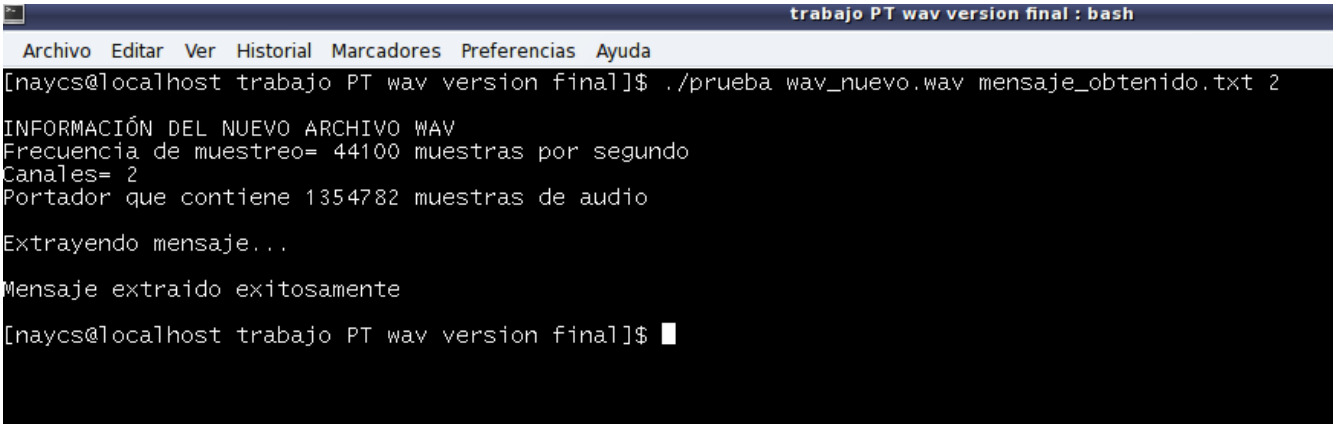


Figura 6. Resultado de la ejecución del programa para extraer mensaje

Además de que obtenemos las siguientes archivos de salida:

- **muestras_audio_msj.txt** = Archivo que contiene las muestras de audio del archivo wav de entrada escritas en números enteros
- **mensaje_binario2.txt** = Este archivo contiene el mensaje en código binario extraído del archivo de audio wav. Dicho mensaje se almacena en muestras de 8 bits
- **mensaje_obtenido.txt** = Este archivo puede tomar el nombre que el usuario decida desde el momento de la ejecución del programa. Contiene el mensaje extraído decodificado a código ASCII.

El resultado se muestra en la Figura 7:

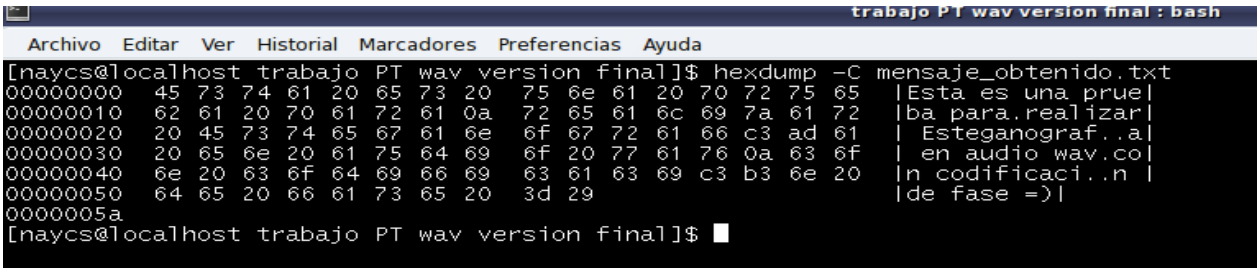


Figura 7. Mensaje de texto extraído del archivo de audio wav_nuevo.wav

Comparando el mensaje a insertar con el mensaje obtenido, podemos ver que el resultado fue el esperado.

Se realizará una prueba más, pero en esta ocasión se probará insertar un mensaje más largo, como lo muestra la Figura 8.

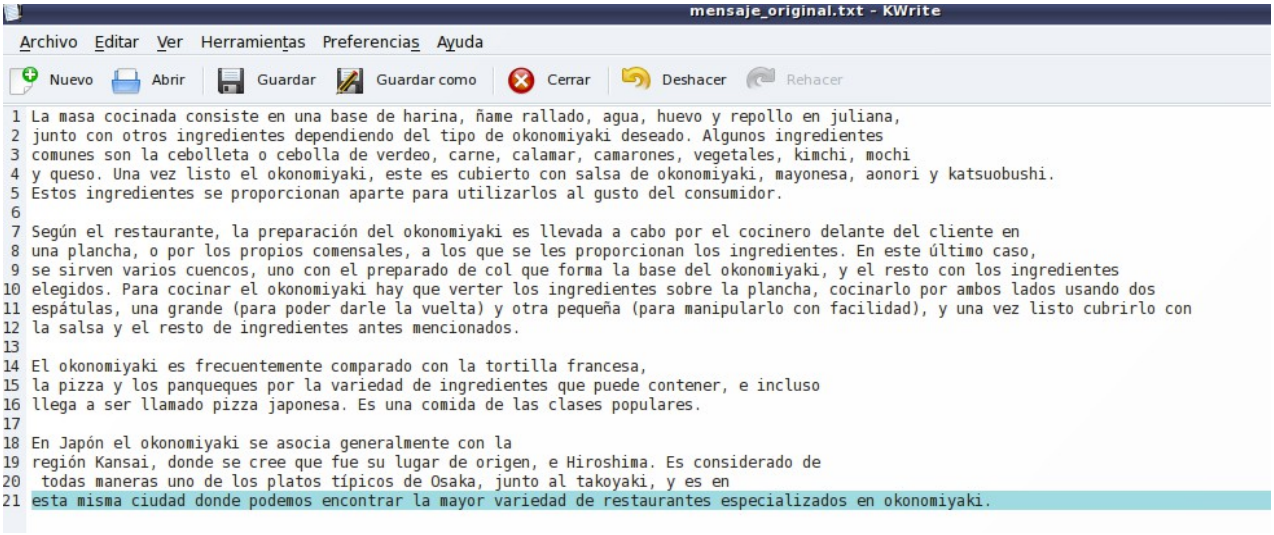


***Figura 8.* Mensaje que se desea insertar.**

Ejecutamos de nuevo el programa, como lo muestra la Figura 9.

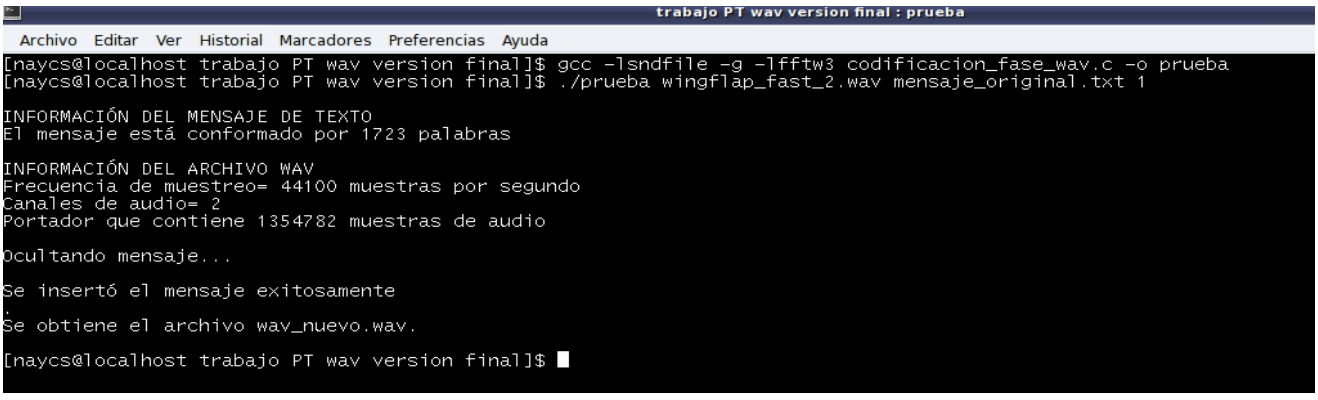


Figura 9. Resultado de la ejecución del programa para insertar mensaje

Una vez insertado el mensaje, se procedió a escuchar el archivo resultante ***wav_nuevo.wav,*** donde se pudo notar que esta vez había algunas diferencias perceptibles en la calidad del audio (específicamente en los primeros segundos del audio) con respecto al archivo wav original. Con esto se puede decir que para mensajes grandes (en este caso, de 1723 palabras), la calidad de audio del archivo resultante será menor con respecto al audio original.

Procedemos ahora a extraer el mensaje como lo indica la Figura 10.

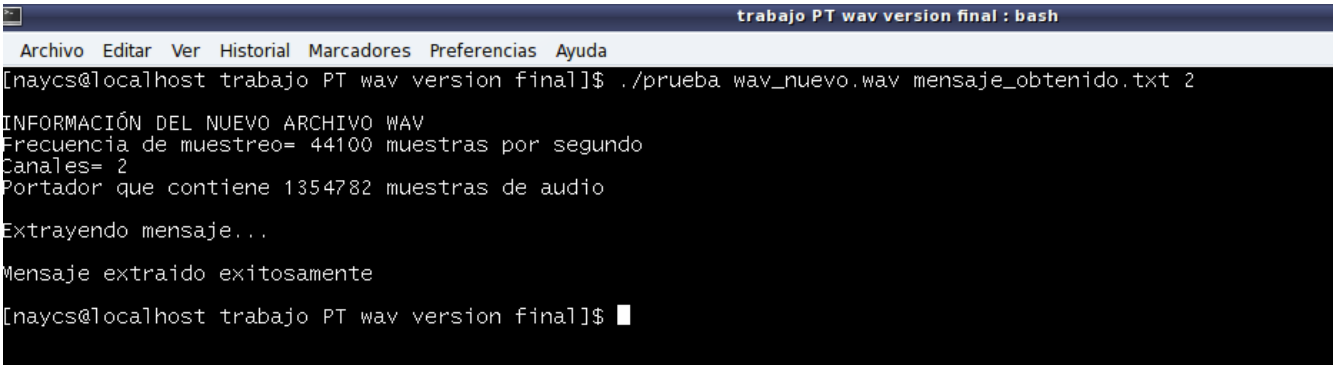


Figura 10. Resultado de la ejecución del programa para extraer mensaje

Como se puede observar en la Figura 11, el resultado de la extracción del mensaje fue el esperado.

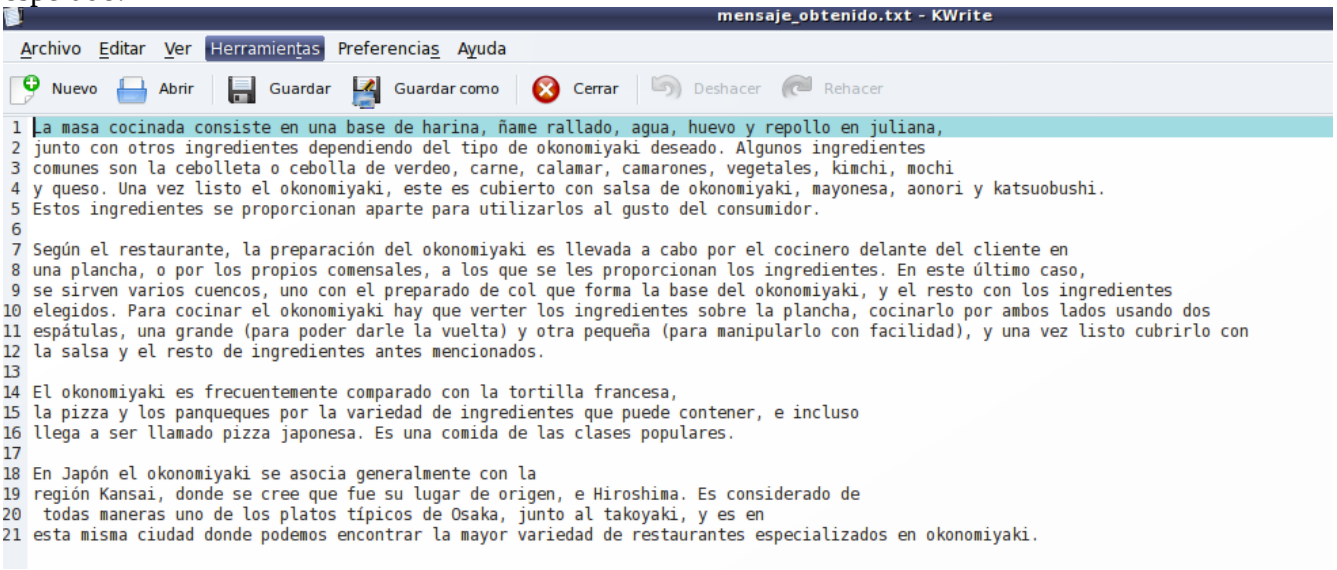


Figura 11. Mensaje de texto extraído del archivo de audio wav_nuevo.wav

Observando estos resultados, se puede decir con seguridad que la codificación de fase aplicada en archivos de audio wav ha resultado ser exitosa.

Resultados obtenidos utilizando audio ogg

Para las pruebas se utilizó el mensaje de texto que se muestra en la Figura 12

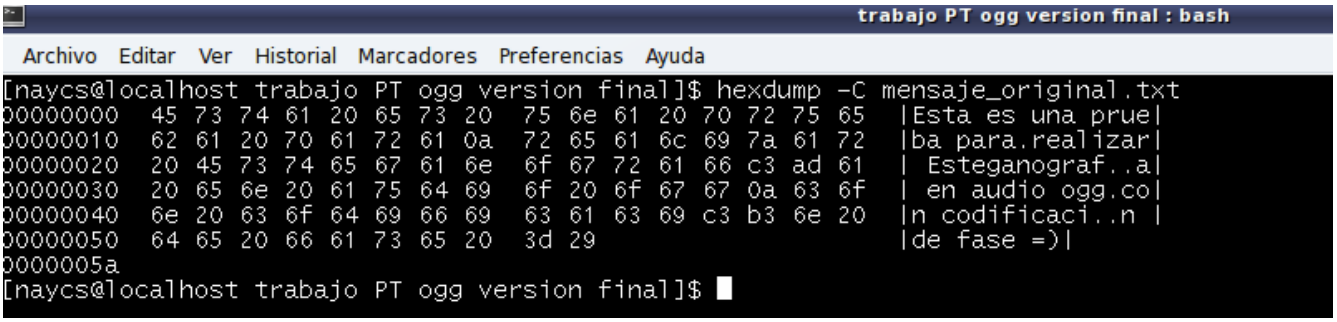


Figura 12. **Mensaje que se desea insertar.**

Estas pruebas fueron realizadas usando el código fuente ***codificacion_fase_ogg.c***, compilándolo como se hizo en las pruebas con audio wav, cambiando solamente el nombre del código fuente.

Ahora ponemos en marcha el ejecutable de la siguiente manera:

./prueba archivo_ogg_portador mensaje_a_insertar 1

De nuevo, la opción 1 indica que lo que deseamos es insertar un mensaje.

Una vez hecho esto, obtenemos en pantalla algunos datos del mensaje que se desea insertar y del archivo ogg donde se insertará, como lo muestra la Figura 13.

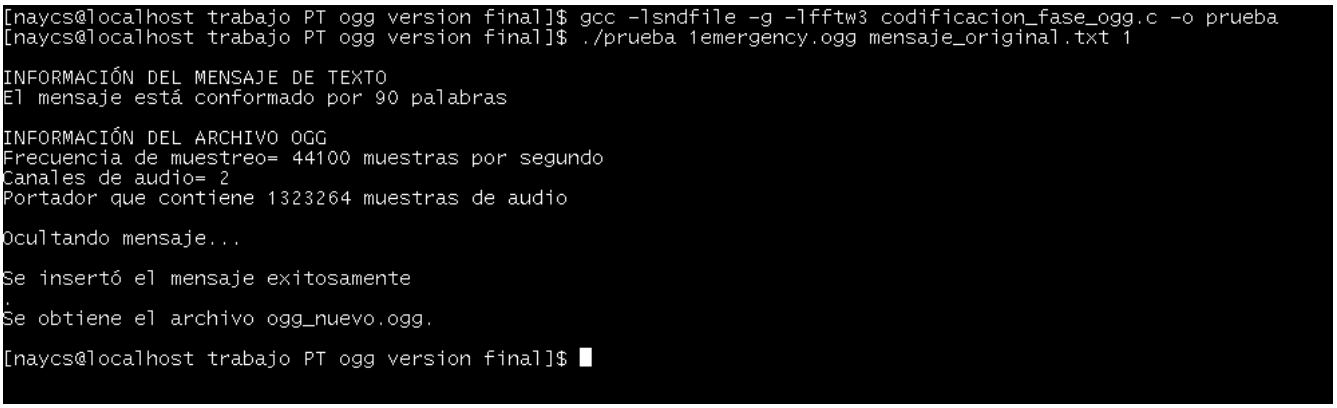


Figura 13. Resultado de la ejecución del programa para insertar mensaje

Además obtenemos las siguientes archivos de salida:

- **muestras.txt** = Archivo que contiene las muestras de audio del archivo wav escritas en números enteros.
- **mensaje_binario.txt** = Este archivo contiene el mensaje a insertar convertido a código binario.
- **m.txt** = Archivo que contiene la longitud en bits del mensaje a insertar.
- **ogg_nuevo.ogg** = Archivo ogg que contiene el mensaje oculto. Su nombre puede ser cambiado si el usuario lo desea.

El usuario puede prescindir de los dos primeros archivos de salida si lo desea, pero el archivo m.txt debe conservarlo ya que es indispensable para la recuperación del mensaje posteriormente.

Se escuchó la calidad del archivo ogg resultante. Ya que en este código también se prescindió de usar las diferencias de fase y se usaron las diferencias originales, obtuvimos una buena calidad de audio con respecto al ogg original.

También se puedo constatar que insertar un mensaje corto (en este caso de 90 palabras) no produce diferencias perceptibles en el audio con respecto al wav original.

Ahora procedemos a extraer el mensaje escondido en el archivo wav_nuevo.wav, siguiendo el siguiente formato:

./prueba archivo_ogg archivo_texto 2

De nuevo, la opción 2 indica que lo que deseamos es extraer el mensaje.

Una vez hecho esto, de nuevo obtenemos en pantalla algunos datos de los archivos de entrada, como lo muestra la Figura 14:

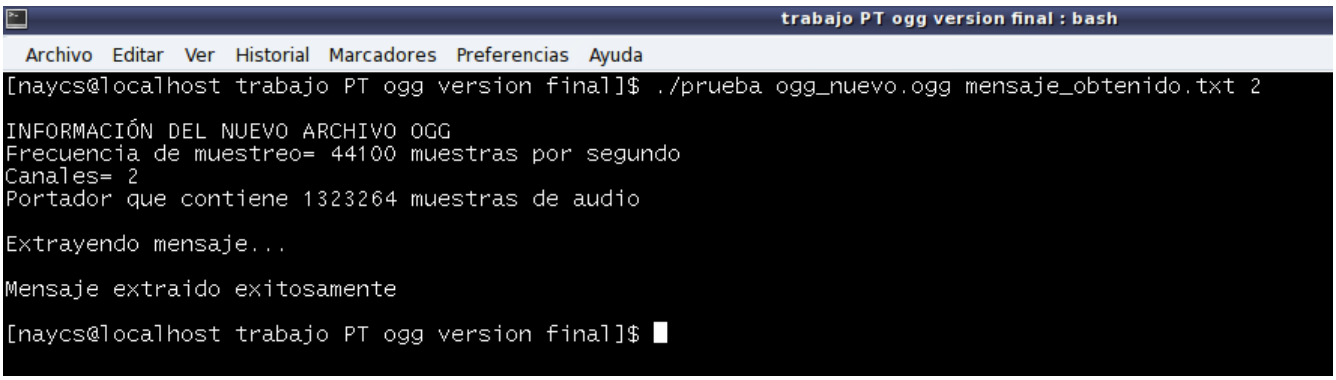


Figura 14. Resultado de la ejecución del programa para extraer mensaje

También se obtienen los mismos archivos de salida que en las pruebas hechas con audio wav.

El resultado se muestra en la Figura 15:

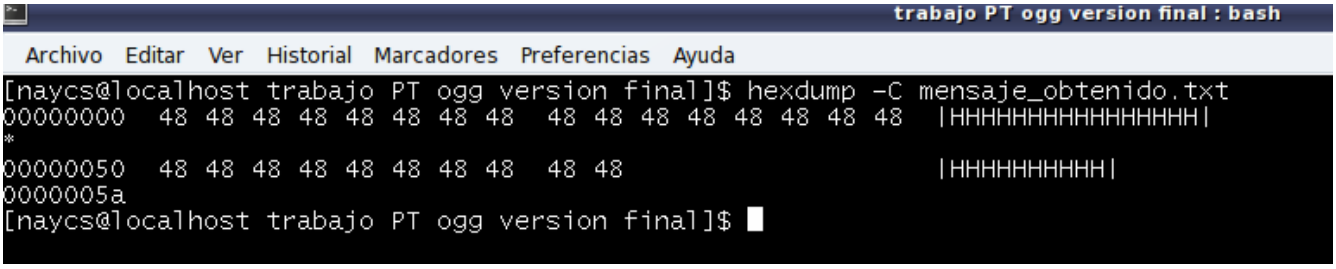


Figura 15. Mensaje de texto extraído del archivo de audio ogg_nuevo.ogg

Comparando el mensaje a insertar con el mensaje obtenido, podemos ver que el resultado no fue el esperado.

Analizando los posibles factores que provoquen que el mensaje no pueda ser recuperado tal como se insertó, se decidió inspeccionar el vector de fase donde se almacenaría el mensaje de texto, tal como lo muestra la Figura 16.

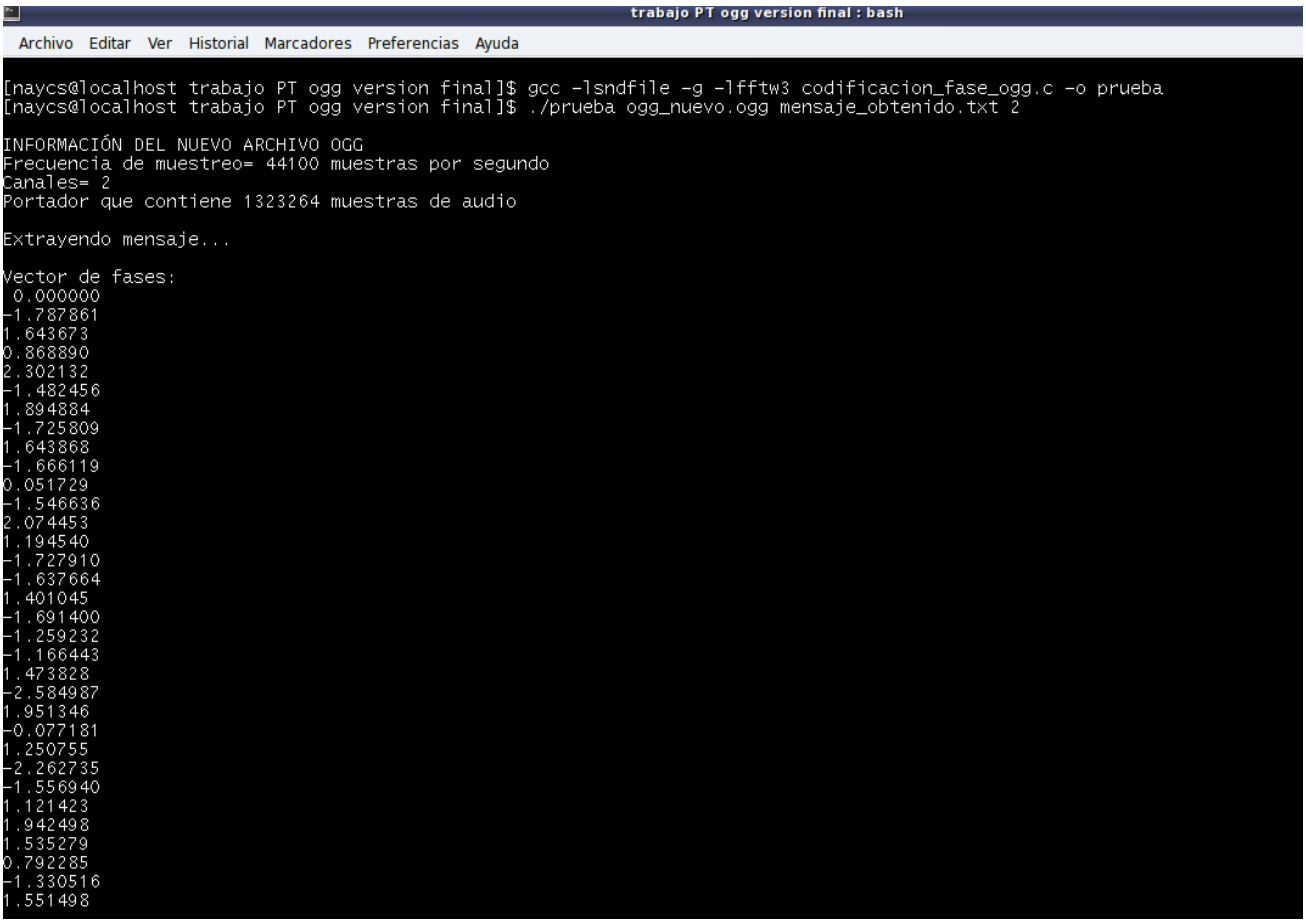


Figura 16. Vector de fases del archivo ogg_nuevo.ogg

Observando los valores de las fases de dicho vector, se pudo observar que ninguno de ellos almacena el valor de π/2 (1.570797 aproximadamente) o de -π/2 (-1.570797 aproximadamente) que se insertó anteriormente, por lo tanto, después de decodificar el valor de las fases a código binario jamás obtendremos un 0 o un 1.

Una causa muy probable por la cual no se almacenaron los valores de π/2 o -π/2 insertados en el vector de fase es que al generar el nuevo archivo ogg, la compresión de su formato originó que las fases cambiaran significativamente su valor.

Entonces se optó que el programa **codificacion_fase_ogg.c** ampliara el rango de valores para obtener el mensaje en binario. Es decir, el programa recuperará un 0 si el valor de la fase se encuentra entre 1.0 y 2 y recuperará un 1 si el valor del vector de fase se encuentra entre -1.0 y -2.0.

La Figura 17 muestra los resultados obtenidos:

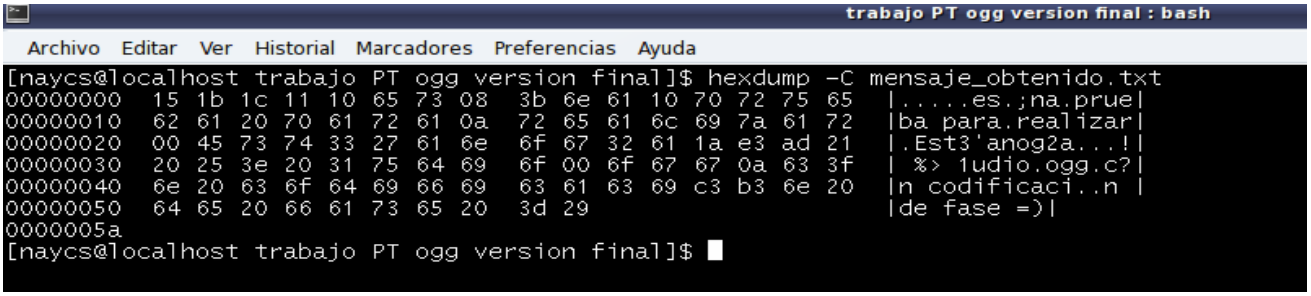


Figura 17. Mensaje recuperado después de ampliar el rango de valores aceptables en el vector de fases

Como se puede apreciar, el porcentaje de caracteres recuperados del archivo ogg aumenta considerablemente, aunque aún no se recupera el mensaje al 100%.

Se optó por ampliar de nuevo el rango de valores para obtener el mensaje en binario. En esta ocasión, el programa recuperará un 0 si el valor de la fase es un número mayor que 0 y recuperará un 1 si el valor del vector de fase es menor que 0.

La Figura 18 muestra los resultados obtenidos:

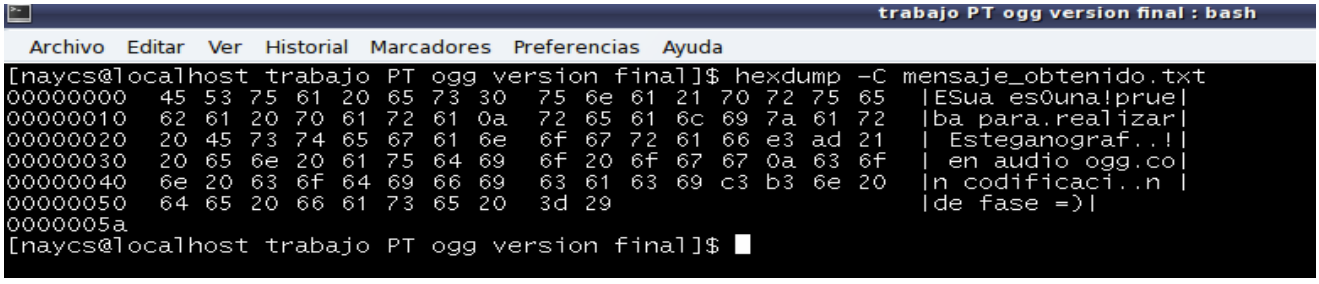


Figura 18. Mensaje recuperado después de ampliar de nuevo el rango de valores aceptables en el vector de fases

Como se puede apreciar, el número de caracteres del mensaje recuperados aumentó acercándose casi al 100%, pero sigue sin recuperarlo por completo.

Se hizo una prueba más, esta ocasión se probó insertar el mensaje largo que se insertó en las pruebas con audio wav, pero conservando los últimos cambios realizados al código fuente ***codificacion_fase_ogg.c***.

Se ejecutó de nuevo el programa, como lo muestra la Figura 19.

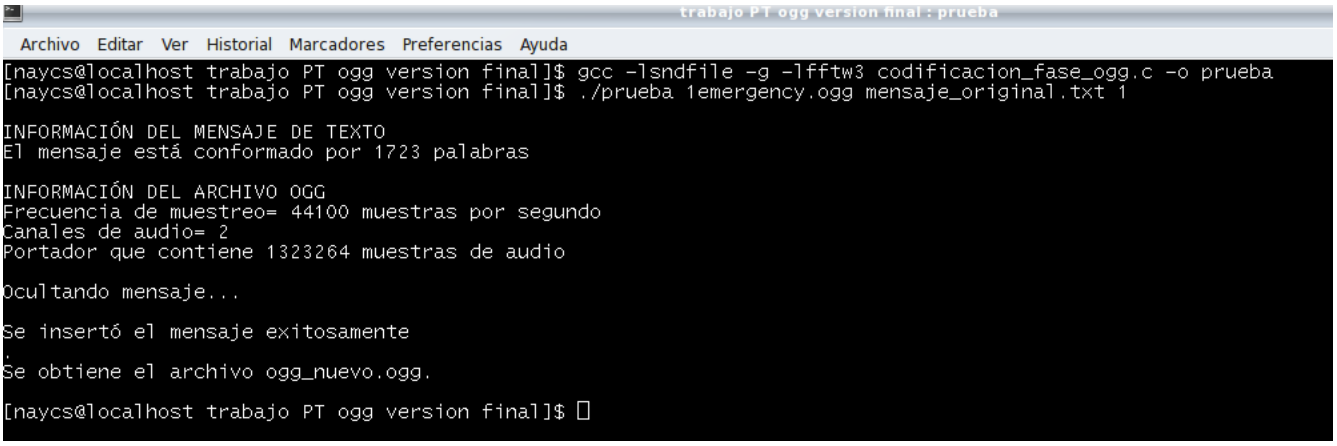


Figura 19. Resultado de la ejecución del programa para insertar mensaje

Una vez insertado el mensaje, se procedió a escuchar el archivo resultante ***ogg_nuevo.ogg,*** donde se pudo notar que esta vez había algunas diferencias perceptibles en la calidad del audio (específicamente en los primeros segundos del audio) con respecto al archivo ogg original. Con esto se puede decir que para mensajes grandes (en este caso, de 1723 palabras), la calidad de audio del archivo resultante será menor con respecto al audio original.

Se procedió a recuperar el mensaje del archivo ***ogg_nuevo.ogg***, obteniendo el resultado mostrado en la Figura 20.

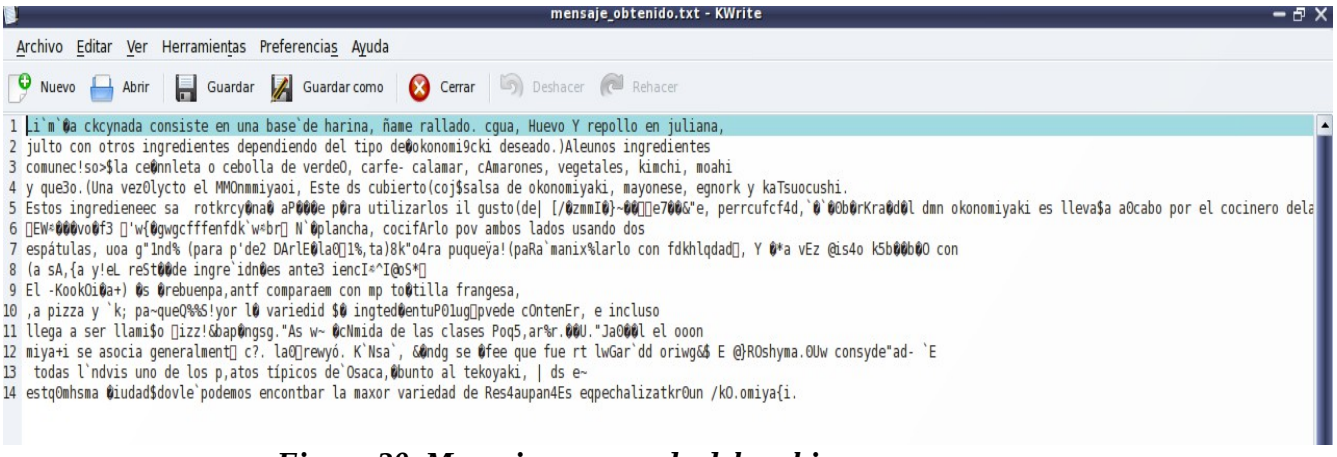


Figura 20. Mensaje recuperado del archivo ogg_nuevo.ogg

Como se puede apreciar el mensaje largo no se recuperó en una proporción similar al mensaje corto con los cambios efectuados al código, resultado que de alguna manera se esperaba, ya que el mensaje fue recuperado de un archivo de formato de compresión con pérdida.

Una vez realizadas estos resultados, se puede decir que la técnica de codificación de fase no es la mejor opción para aplicarse a un archivo ogg de compresión con pérdida.

PRUEBAS

Una vez concluido el análisis de los resultados obtenidos, se procedió a realizar pruebas para comprobar la efectividad de la esteganografía en audio utilizando codificación de fase.

Se utilizaron los siguientes archivos de audio wav [7]:

1. 3269__Jovica__Dronetail_102
2. ATTACK_LOOP_3_hihats_mastered_16_bit
3. Dronetail_01
4. Dronetail_02
5. Dronetail_03
6. Dronetail_63
7. wingflap_fast_2

También los siguientes archivos de audio ogg [8]:

1. Emergency
2. Sahara
3. Recursion
4. As Yet Untitled

Todos estos archivos de audio cuentan con la licencia Creative Commons [9].

El experimento se realizó de la siguiente manera:

- Se realizaron pruebas a 10 personas.
- Cada una de ellas escuchó 7 muestras de audio wav y 4 muestras de audio ogg
- A las 10 personas se les pidió escuchar 2 versiones de las muestras de audio: la primera muestra sin modificar y la segunda con un mensaje oculto (dado que los mensajes cortos mostraron pocas diferencias auditivas, se decidió insertar el mensaje largo utilizado en la sección de Análisis de resultados).
- Las personas calificaron las diferencias entre cada par de muestras de audio.

Los formatos utilizados para ello se muestran en las Figuras 21 y 22:

TRANSMISIÓN DE MENSAJES DE TEXTO EN AUDIO OGG UTILIZANDO ESTEGANOGRAFIA

FORMATO DE PRUEBAS PARA ARCHIVOS WAV

INSTRUCCIONES: A continuación escucharás 7 pares de muestras de audio. Califica las diferencias que encuentres en ellas basándote en la siguiente escala:

1: diferencia imperceptible
2: diferencia ligeramente imperceptible
3: diferencia un poco molesta
4: diferencia molesta
5: diferencia muy molesta

Muestra de audio	Calificación
3269__Jovica__Dronetail_102	
ATTACK_LOOP_3_hihats_mastered_16_bit	
Dronetail_01	
Dronetail_02	
Dronetail_03	
Dronetail_63	
wingflap_fast_2	

Figura 21. Formato de pruebas para archivos de audio wav.

TRANSMISIÓN DE MENSAJES DE TEXTO EN AUDIO OGG UTILIZANDO ESTEGANOGRAFIA

FORMATO DE PRUEBAS PARA ARCHIVOS OGG

INSTRUCCIONES: A continuación escucharás 4 pares de muestras de audio. Califica las diferencias que encuentres en ellas basándote en la siguiente escala:

1: diferencia imperceptible
2: diferencia ligeramente imperceptible
3: diferencia un poco molesta
4: diferencia molesta
5: diferencia muy molesta

Muestra de audio	Calificación
Emergency	
Sahara	
Recursion	
As yet Untitled	

Figura 22. Formato de pruebas para archivos de audio ogg.

De las pruebas realizadas, se obtuvieron las siguientes estadísticas:

- De las 10 personas que participaron, 3 pertenecían al género masculino, con un rango de edad de los 14 a los 38 años.

- 7 personas pertenecían al género femenino, con un rango de edad de los 16 a los 48 años.

- Con respecto a los archivos wav, se obtuvieron los siguientes resultados:
    - El 48% de las calificaciones indicaron que las diferencias entre los pares de archivos wav fueron imperceptibles.
    - El 41% de las calificaciones indicaron que las diferencias entre los pares de archivos wav fueron casi imperceptibles.
    - El 11% de las calificaciones indicaron que las diferencias entre los pares de archivos wav fueron un poco molestas.

- Con respecto a los archivos ogg, se obtuvieron los siguientes resultados:
    - El 53.5% de las calificaciones indicaron que las diferencias entre los pares de archivos ogg fueron imperceptibles.
    - El 35% de las calificaciones indicaron que las diferencias entre los pares de archivos ogg fueron casi imperceptibles.
    - El 10% de las calificaciones indicaron que las diferencias entre los pares de archivos ogg fueron un poco molestas.
    - El 2.5% de las calificaciones indicaron que las diferencias entre los pares de archivos ogg fueron molestas.

Analizando estos resultados podemos notar que, aunque más personas dijeron que las diferencias entre los pares de audio ogg fueron imperceptibles, en general los resultados de las pruebas con archivos wav tuvieron mejores calificaciones que los pares de audio ogg.

PROPUESTAS PARA MEJORAR EL PROYECTO A FUTURO

Existen diversas modificaciones que pueden hacerse al presente Proyecto Terminal. Una de ellas es encontrar la manera en la que la longitud en bits del mensaje no tenga que almacenarse en un archivo de texto, de esta manera el usuario no se verá forzado a conservar el archivo “m.txt”, ya que sin él le sería imposible recuperar el mensaje. Podría modificarse el código fuente de tal manera que se reserven unos cuantos bits en la cabecera del archivo de audio generado para almacenar la longitud en bits del mensaje a recuperar.

Se propone también que los valores dentro del vector de fase adopten valores de tal manera que, al insertarse el mensaje de texto la fase no se modifique mucho y ésto no se vea reflejado en la calidad de audio del archivo portador resultante.

La técnica de codificación de fase implementado en este Proyecto Terminal fue el tradicional, en el cual se indica que el mensaje sólo se esconde en el primer vector de fase del archivo de audio. Se propone modificar el algoritmo de manera que si se desea insertar un mensaje de texto, ésto pueda hacerse en todos los vectores de fase, de manera que se aproveche todo el espacio disponible en el archivo y se incremente la robustez de la técnica.

También se propone emplear la codificación de fase con otros formatos de audio, tanto con compresión como sin compresión, para saber con cuales formatos es más conveniente utilizar esta técnica.

Finalmente, se invita a los estudiantes de Ingeniería en Computación interesados en la esteganografía en audio que implementen otras técnicas existentes, como lo son la técnica de expansión del espectro, codificación según la paridad u ocultación en el eco. ¿O por qué no? Que propongan una nueva técnica de estaganografía en audio.

REFERENCIAS

[1] Chávez Sanabria, Karla Nayeli, "*Transmisión de mensajes de texto en audio ogg usando esteganografía*", Propuesta de Terminal de Ingeniería en Computación, Universidad Autónoma Metropolitana Unidad Azcapotzalco, México, 2010.

[2] Formatos de audio, AMI-Asociación de Música en Internet, Copyleft, Defensa consumidor Música Online, disponible: http://asociacionmusica.com/t_audio.asp, [Septiembre 6, 2011].

[3] WAV Descripción del formato de audio, disponible: http://www.coolutils.com/es/Formats/WAV, [Septiembre 6, 2011].

[4] libsndfile, disponible: http://www.mega-nerd.com/libsndfile/, [Junio 14, 2011].

[5] FFTW Home Page, disponible: http://www.fftw.org/index.html/, [Julio 11, 2011].

[6] Okonomiyaki – Wikipedia, la enciclopedia libre, disponible: http://es.wikipedia.org/wiki/Okonomiyaki, [Septiembre 4, 2011].

[7] Freesound, disponible: http://www.freesound.org/, [Agosto 5, 2011].

[8] The Music of Michael Crawford in MP3 and Ogg Vorbis - Free Sheet Music, disponible: http://www.geometricvisions.com/music/, [Agosto 5, 2011].

[9] Creative Commons, disponible: http://creativecommons.org/, [Agosto 5, 2011].