Criptografía y Seguridad *(72.44)*

TRABAJO PRÁCTICO 1: CIFRADO POR BLOQUES

1. Objetivos

Experimentar con los algoritmos de cifrado por bloques, en particular AES y DES.

Analizar los resultados obtenidos al aplicar distintos modos de operación para algoritmos de cifrado por bloques.

2. Consigna

Realizar un programa en lenguaje C que, dado un archivo de sonido en formato WAV:

1) Encripte los datos de sonido mediante algún algoritmo de cifrado por bloques, obteniendo un nuevo archivo de sonido .WAV reproducible.
2) Desencripte un archivo de sonido, conocida la clave y el vector de inicialización (IV) o conocido el password, mediante algún algoritmo de cifrado por bloques, obteniendo un archivo de sonido .WAV reproducible.

3. Detalles del sistema

3.1. Encripcion – Desencripcion

El programa debe recibir como parámetros:

- **-in** audiofile (en formato .wav)
- **-out** audiofile (en formato .wav)
- **-e (e**ncripcion) o bien **–d** (desencripcion)
- **-pass password** (password de encripcion). O bien –K key –iv vectorInicializacion
- **-a <aes128 | aes192 | aes256 | des>**
- **-m <ecb | cfb | ofb | cbc>**

Ejemplo:

Encriptar el archivo de sonido “pum.wav” con aes de clave 192 bits en modo ecb, con password “hello” obteniendo como salida “pumEnc.wav”

```
$criptoWavs –in “pum.wav” –out “pumEnc.wav” –e –pass “hello” –a aes192 –m ecb
```

3.2. Archivos .WAV

El formato de archivo wav (Waveform Audio File Format) permite almacenar audio digital de una manera bastante simple, al ser un subconjunto de lo que se conoce como RIFF.

RIFF (Resource Interchange File Format) es una estructura de archivo que propone Microsoft en el documento “Multimedia Programming Interface and Data Specifications 1.0” para manejar los distintos formatos de archivo multimedia de una manera homogénea y extensible.

El bloque básico de un archivo RIFF se conoce como chunk. Un chunk, en sitaxis de C puede definirse:

```
typedef unsigned long DWORD;
typedef unsigned char BYTE;

typedef DWORD FOURCC;  //Four-character code

typedef FOURCC CKID;   //Four-character-code chunk identifier
typedef DWORD CKSIZE;  //32-bit unsigned size value

typedef struct{              //Chunk structure
      CKID  ckID;            //Chunk type identifier
      CKZIE ckSize;          //Chunk size field (size of ckData)
      BYTE  ckData[ckSize];  //Chunk data
};
```

Todo archivo RIFF comienza con un encabezado de archivo (*RIFF file header*) seguido de una secuencia de bloques de datos (*chunks*). Un archivo WAVE, por ejemplo, es un archivo en cuyo encabezado RIFF figura una identificación como archivo "WAVE" y luego consta de chunks que definen las características del archivo de audio (como por ejemplo número de canales, si está comprimido, etc) y luego los datos del sonido propiamente dichos.

Todo archivo wav tiene, obligatoriamente:

- un chunk que describe el tipo de archivo RIFF, en este caso indica que se trata de un "WAVE" file.
- Un chunk obligatorio de formato de archivo de sonido.
- Un chunk obligatorio de datos de sonido.

Ejemplo de estructura wav:

```
struct wavStr
{
    RIFF_CK riff_desc; // MANDATORY
    FMT_CK  fmt;       // Format Chunk MANDATORY
    //FACT_CK fact;    // Fact Chunk OPTIONAL
    //CUE_CK  cue;     // Cue points Chunk OPTIONAL
    //PLIST_CK   plist;// Playlist Chunk OPTIONAL
    //LIST_CK list;    // Associated data list Chunk OPTIONAL
    // more optional data…
    DATA_CK data;      // Wave Data Chunk MANDATORY
};
```

Así, el chunk de RIFF sería:

```
typedef struct{
    CKID    chunkID;    // 'RIFF'
    CKSIZE  chunkSize;  // File Size
    CKID    format;     // Format: 'WAVE'
}RIFF_CK;
```

El chunk básico de fmt sería:

```
typedef struct{
    CKID         chunkID;    //'fmt '
    CKSIZE       chunkSize;  // 16 para PCM.Size of rest of subchunk.
    /* Common fields */
    WORD         wFormatTag; // Format category,i.e.:PCM = 1 (no compres.)
    WORD         wChannels;  // Number of channels:1, mono; 2, stereo
    DWORD        dwSamplesPerSec;    // Sampling rate: Mhz
    DWORD        dwAvgBytesPerSec;
    WORD         wBlockAlign
    WORD         wBitsPerSample; //8, 16, etc.
    WORD         extraParamSize;// If PCM, doesn't exist
    BYTE         *extraParams;//space for extra params
}FMT_CK;
```

El chunk básico de data sería:

```
typedef struct{
    CKID            chunkID;    // 'data'
    CKSIZE          chunkSize;  // Bytes of data
    BYTE            *soundData; // Sound data.
}DATA_CK;
```

Para este trabajo se considerarán archivos wav sencillos, esto es sin chunks del tipo wavl (wavelist), slnt (silent), post (playlists), sin comprimir (sólo PCM).

IMPORTANTE: Leer con cuidado los datos de los campos **obligatorios** de los archivos wav. Los campos opcionales copiarlos sin procesarlos específicamente, esto es no se vuelcan a ninguna subestructura particular, sólo se copian byte a byte.

Lo que interesa para este trabajo práctico es encriptar y desencriptar los datos del sonido propiamente dicho (campo `soundData`). Las características que están almacenadas en fmt no deberían

modificarse. Por eso los archivos encriptados también deberían poder reproducirse como cualquier .wav (aunque se escuche algo irreconocible). Deben poder abrirse con editores de audio (por ejemplo Audacity).

IMPORTANTE: Considerar archivos wav sólo en formato PCM

3.3. Consideraciones generales.

- En el caso de que los datos de audio no sean múltiplo de bloque, habrá que **descartar** el archivo de audio, ya que si no deberían modificarse los campos de size.
- El resultado de la encriptación **también es un archivo de audio .wav**, que conserva las características de audio del original.
- Para la generación de clave a partir de una password, asumir que la función de hash usada es md5, y que no se usa SALT.
- Para modos de feedback considerar una cantidad de bits por default (por ejemplo, 8 bits)

4. Cuestiones a analizar.

Una vez obtenido el programa, deberán analizarse las siguientes cuestiones:

1. La cátedra proveerá de dos archivos que se componen de dos notas musicales cada uno. Comparar el resultado de encriptar con DES uno de esos archivos en modo ECB, con el resultado de encriptar el mismo archivo en modo CBC, en modo CFB y en modo OFB.
2. Comparar el resultado de encriptar con AES uno de los archivos entregados por la cátedra en modo ECB, con el resultado de encriptar el mismo archivo en modo CBC, en modo CFB y en modo OFB.
3. Comparar el resultado de encriptar con DES en modo ECB un archivo de los entregados por la cátedra, con el resultado de encriptar el mismo archivo con AES en modo ECB.
4. Tomar un archivo encriptado con DES ECB, cambiarle un bit de cifrado y desencriptar. ¿A cuántos bloques afecta el error de un bit en el cifrado? ¿Por qué sucede esto?
5. Tomar un archivo encriptado con DES CBC, cambiarle un bit de cifrado y desencriptar. ¿A cuántos bloques afecta el error de un bit en el cifrado? ¿Por qué sucede esto?
6. Tomar un archivo encriptado con DES CFB, cambiarle un bit de cifrado y desencriptar. ¿A cuántos bloques afecta el error de un bit en el cifrado? ¿Por qué sucede esto?
7. Tomar un archivo encriptado con AES CFB, cambiarle un bit de cifrado y desencriptar. ¿A cuántos bloques afecta el error de un bit en el cifrado? ¿Por qué sucede esto? (Comparar coon el caso de DES CFB)
8. Tomar un archivo encriptado con DES OFB , cambiarle un bit de cifrado y desencriptar. ¿A cuántos bloques afecta el error de un bit en el cifrado?
9. Utilice el método ECB para encriptar los archivos dados por la cátedra. Utilice el programa audacity para comparar su salida con los archivos encriptados, también provistos por la cátedra incogDES.wav y incogAES.wav. Responda sin desencriptar los archivos, qué notas posee cada archivo? Justifique. Luego, desencripte los archivos incogDES.wav y incogAES.wav y verifique empíricamente lo supuesto.

IMPORTANTE:

Se recomienda, para este análisis, usar el software Audacity (o similar) y algún editor de archivos binarios (hex editor neo o similar) que permita hacer comparaciones de los archivos, como archivos de audio y como secuencia de bytes.

Asimismo, se recomienda una lectura profunda de la bibliografía de la materia para hacer un análisis apropiado.

5. Organización de los grupos

El trabajo será realizado en grupos de 3 integrantes como máximo**. Para el 18 de abril** deberán dar a conocer los nombres de los integrantes de cada grupo y una dirección de email a la cátedra (ariasroigana@gmail.com o bien a santiagovazq@gmail.com)

Cada grupo recibirá, en la semana del 18 de abril, archivos de sonido de prueba.

6. Entrega

La fecha de entrega es el día 29 de abril.

Cada grupo entregará el ejecutable y el código en C, junto con la documentación correspondiente al uso del programa.

Además presentarán un informe con la solución correspondiente al descifrado de los archivos que se le entregarán oportunamente al grupo y con el análisis de las cuestiones planteadas en el punto 4.

7. Material de lectura recomendado

- **Sobre cifrado de Bloque:**

- **Capítulo 9 de Computer Security – Art and Science, Matt Bishop, Addison-Wesley, 2004**
- **Capítulo 7 de Handbook of Applied Cryptography, Alfred J. Menezes, Paul C. Van Oorschot, Scott A. Vanstone, CRC Press, 1997**
- **Sobre WAVS:**

- http://www-mmsp.ece.mcgill.ca/Documents/AudioFormats/WAVE/WAVE.html
- **http://www-mmsp.ece.mcgill.ca/documents/audioformats/wave/Docs/riffmci.pdf**
- https://ccrma.stanford.edu/courses/422/projects/WaveFormat/