CIFRADO DE GRABACIONES DE VOZ UTILIZANDO TRIPLE-DES 96, PARA DISPOSITIVOS MÓVILES CON ANDROID

Gabriel Eduardo García Rojas
gabrielgrojas@hotmail.com
Manuel Ramírez Monroy
mramirezm1006@alumno.ipn.mx
José Antonio González Torres
jgonzalezt1503@alumno.ipn.mx
Marlon David González Ramírez
dgonzalezr@ipn.mx
Maestría en Tecnología en Cómputo Instituto Politécnico Nacional, CIDETEC

Abstract

This article proposes use DES encryption standard data, a variant Triple-DES 96 applied to the transmitted voice security on mobile devices with the Android Operating System and MediaRecorder framework. It includes in the Triple-DES 96 algorithm, the JV variable permutation theorem, which strengthens and streamlines the process of permutation from the third round of the first cycle and the thirteenth round of the third cycle of this standard; thus, the initial permutation DES standard base disappears, making the process fast and secure, protecting the left or right side and block data. Triple- DES 96 is a little known and potentially safer than its predecessor adequacy, currently used only in image and plain text encryption. Keywords: DES, Triple-DES 96, encryption, variable permutation, Android.

Introducción

El sistema operativo Android permite que se desarrollen gran variedad de aplicaciones, entre ellas se encuentran algunas que aseguran las llamadas telefónicas, como son Redphone[1] que proporciona un sistema de cifrado end-to-end con un protocolo ZRTP sobre redes 3G y Wifi; otra aplicación llamada MySecureVoice[1] utiliza los protocolos SRTP, SSL y la infraestructura de clave pública (PKI), para garantizar la integridad y seguridad de las conversaciones en todas las etapas de la transmisión de datos (end-to-end). Las llamadas están protegidas a través de AES (AdvancedEncryption Standard, AES), RSA (Rivest, Shamir y Adleman) y SHA-1 (Secure Hash Algorithm), sobre la red 3G. Finalmente Kryptos[1] es una aplicación de voz sobre protocolo de Internet (VoIP) segura, y utiliza un estándar de cifrado AES de 256 bits para cifrar comunicaciones de voz entre usuarios y cifrado RSA de 1024 bits durante el intercambio de claves simétricas. Los estándares de cifrado utilizados en las aplicaciones mencionadas manejan algoritmos convencionales, y si bien son de costo computacional bajo o elevado, Triple-DES 96 es un algoritmo no convencional, que mejora significativamente la capacidad de Triple-DES y DES[2]. Es por eso que en este artículo propone trabajar con un estándar de cifrado Triple-DES 96 que reduce el tiempo de cifrado de la voz con respecto a sus antecesores, y en comparación con otras normas, es poco conocido. Se propone también trabajar con una aplicación básica desarrollada en Android que grabe un mensaje de voz, el cual se cifrará aplicando el estándar antes mencionado y posteriormente se realizará la operación inversa

(descifrado). El artículo parte con la descripción del algoritmo Triple-DES 96, continuando con la descripción de las bibliotecas y clases del sistema operativo Android que graban mensajes de voz, y finalmente el desarrollo del enlace entre el estándar de cifrado y la aplicación de grabación de voz.

Investigación A. Algoritmo de Cifrado Simétrico Triple-DES 96. El sistema criptográfico Triple-DES 96 [2] es una variante del sistema criptográfico Triple-DES definido en el estándar internacional FIPS 146-3 [3] con un bloque de entrada de 64 bits. Sin embargo Triple-DES 96 puede expandirse a 96 bits sin perder la complejidad computacional que un algoritmo de cifrado debe tener. Una de las principales características de Triple-DES 96 es que es eliminada la permutación P de cada ronda y el tamaño del bloque de entrada es mayor, proporcionando así una importante reducción en los tiempos de cifrado. La Figura 1 muestra el diagrama a bloques del proceso del algoritmo de cifrado para una ronda inicial tipo Triple-DES 96.

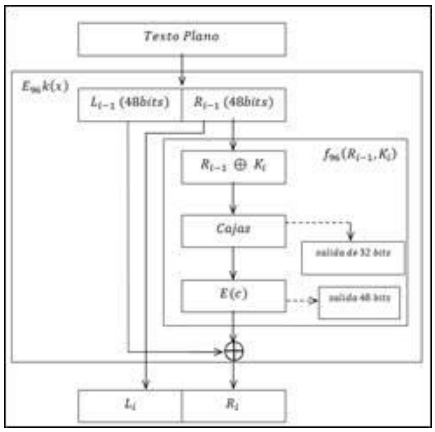

Figura 1: Diagrama a bloques primera ronda Triple-DES 96 (Elaboración propia).

Con base en esto Triple-DES 96 utiliza una permutación variable en la tercera ronda de cifrado del primer ciclo y la treceava ronda del tercer ciclo, siendo esta permutación creada por medio del algoritmo JV y el teorema factorial [4, 5]. Esta permutación es generada con los decimales contenidos en el número irracional π y a su vez se multiplica por un número entero l que incrementar la complejidad del algoritmo de cifrado. Así mismo, esta permutación se realiza después de las primeras dos rondas ya que tanto la parte izquierda L_i como la parte derecha R_i del bloque son sometidas a una operación XOR que es necesaria como en el algoritmo de la familia "Substitution-Permutation-Network" (SPN) [7], para proteger la permutación del bloque de 96 bits, es decir, que la permutación protege ambos lados del texto siendo ésta no solo una permutación simple como la que se realiza en Triple-DES. El algoritmo Triple-DES 96 se desarrolla en tres ciclos, en el primer y el tercer ciclo es aplicado un proceso de cifrado con una llave k^1 y k^3 respectivamente con $k^1 \neq k^3$ o k^1 para ambos y el segundo ciclo tiene un proceso de descifrado con una llave k^2. Cada uno de los ciclos es sometido a 15 rondas donde cada ronda i está conformada por un proceso de cifrado E_{96} k(x) donde x es el texto plano de 96 bits y k la llave de su ciclo correspondiente, el proceso de cifrado E_{96} k(x) comienza con la separación de la cadena de 96 bits en dos de 48 bits conocidas como L_{i-1} y R_{i-1}, posteriormente a la parte L_{i-1} es aplicado una función f_{96} (R_{i-1},K_i) de la ronda i que sigue los siguientes pasos:

a. 1. Se aplica a la cadena $R_{(i-1)}$ la operación XOR con la llave k_i, la llave $k_i$es obtenida del proceso del algoritmo Triple-DES [3],es decir, $R_{(i-1)} \oplus K_i$ La salida de esta operación será una cadena de 48 bits.
b. 2. Posteriormente la nueva cadena es sometida a un proceso de sustitución aplicada como en el estándar DES [6], quien proporciona como salida una nueva cadena de 32 bits nombrada como salida C.
c. 3. A la salida de las cajas C, es necesario aplicar una permutación E utilizando el estándar DES [6], así la cadena de salida para $f_{96}(R_{(i-1)},K_i) = E(C)$, será de 48 bits y se podrá aplicar XOR con la parte izquierda $L_{(i-1)}$.

Este proceso es sometido para las 15 rondas de cada uno de los ciclos del estándar Triple-DES 96, recordando la implementación de la permutación en las rondas 3 y 13 de de los ciclos 1 y 3 respectivamente antes del proceso de cifrado.

B.MediaRecorder

Android es un sistema operativo basado en un Kernel de Linux diseñado principalmente para dispositivos móviles con pantalla táctil. Android ofrece una gran variedad de API (AplicationProgramming Interface) en un ambiente de lenguaje de programación de Java, que permiten desarrollar diferentes aplicaciones. Android incluye un marco de trabajo (framework) multimedia para captura y codificación de varios formatos de audio, que son integrados a una aplicación. Es posible usar la API MediaRecorder para la obtención de audio si es soportada por el dispositivo. La Figura 2 muestra el diagrama de estados general del uso del framework.

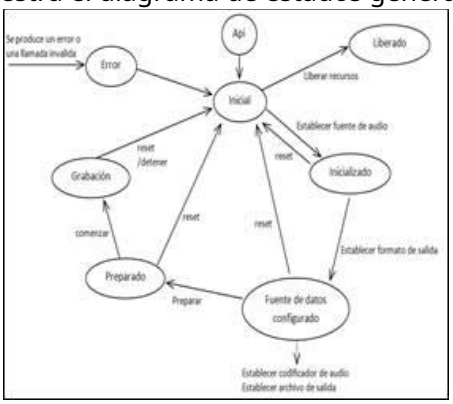

Figura 2: Diagrama General del framework MediaRocorder[9].

Los pasos para obtener audio son un poco más complicados que la reproducción, pero el sitio Androiddeveloper[8] proporciona los pasos base para desarrollarlo:

1. Crear una instancia de MediaRecorder.
2. Establecer:
• La fuente de audio con MediaRecorder.setAudioSource().
• El formato de salida de audio en MediaRecorder.setOutputFormat().
• El nombre del archivo de salida a utilizar MediaRecorder.setOutputFile().
• El codificador a utilizar a través de MediaRecorder.setAudioEncoder().
3. Hace la llamada a la instancia en MediaRecorder con MediaRecorder.Prepare().
4. Iniciar la captura de Audio utilizando MediaRecorder.start().
5. Detener la captura de Audio por medio de MediaRecorder.stop().
6. Finalizar liberando los recursos utilizados con MediaRecorder.release().

A continuación se describen las clases anidadas, métodos y variables principales que fueron utilizados para la grabación de un archivo de audio.

• MediaRecorder. Clase pública principal utilizada para grabar audio y video, su constructor por defecto es MediaRecorder(). Los métodos públicos contenidos en esta clase son los siguientes:

1. prepare(). Prepara el grabador para iniciar los datos de captura y codificación.
2. release(). Libera los recursos asociados con MediaRecorder.
3. reset(). Restaura MediaRecorder a su estado de reposo.
4. start(). Inicia la captura y la codificación de datos al archivo especificado con setOutputFile().
5. stop(). Detiene la grabación.

• MediaRecorder.AudioSource.Clase pública anidada que establece la fuente de audio, contiene el método setAudioSource(int) y puede contener las constantes:

1. CAMCORDER. Fuente de audio del micrófono con la misma orientación que la cámara si está disponible.
2. MIC. Fuente de audio del micrófono.
3. REMOTE_SUBMIX. Fuente de audio para una submezcla de los flujos de audio que se presentará de forma remota.
4. VOICE_CALL. Fuente de audio de una llamada de voz de enlace ascendente más enlace descendente.
5. VOICE_COMMUNICATION. Fuente de audio del micrófono en sintonía para las comunicaciones de voz, tales como VoIP.
6. VOICE_DOWNLINK. Fuente de audio de una llamada de voz de enlace descendente (Rx).
7. VOICE_RECOGNITION. Fuente de audio del micrófono en sintonía para el reconocimiento de voz, si está disponible
8. VOICE_UPLINK. Fuente de audio de una llamada de voz de enlace ascendente (Tx).

• MediaRecorder.OutputFormat. Clase publica anidada que define el formato de salida de audio, el método contenido es setOutputFormat(int) y sus constantes definidas son:

1. Archivo de formato AAC_ADTS
2. Archivo de formato AMR_NB
3. Archivo de formato AMR_WB
4. Archivo de formato RAW_AMR
5. Archivo de formato THREE_GPP
6. Archivo de formato MPEG_4
7. Archivo de formato AAC_ADTS

• MediaRecorder.AudioEncoder. Clase pública anidada que define la codificación de audio, esta clase tiene el método setAudioEncoder(int) y los tipos de constantes que se pueden utilizar son:

1. AAC. Códec de audio de baja complejidad (AAC-LC)
2. AAC_ELD. Códec de audio de bajo retardo AAC.
3. AMR_NB. AMR (Banda estrecha) códec de audio.
4. AMR_WB. AMR (Banda ancha) códec de audio.
5. HE_ACC Códec de audio de alta eficiencia AAC.

C. Aplicación

Listado de aplicaciones_desarrollar

Con la información recopilada se desarrolló una aplicación que realiza grabaciones de audio en un formato 3gpp, donde éstas son sometidas posteriormente a un proceso de cifrado y descifrados utilizando el estándar Triple-DES 96.

La aplicación cuenta con una interfaz con objetos de tipo botón, que establece el control de los pasos para el cifrado y descifrado de las grabaciones de voz. La Figura 3 muestra el diagrama general de funcionamiento de la aplicación.

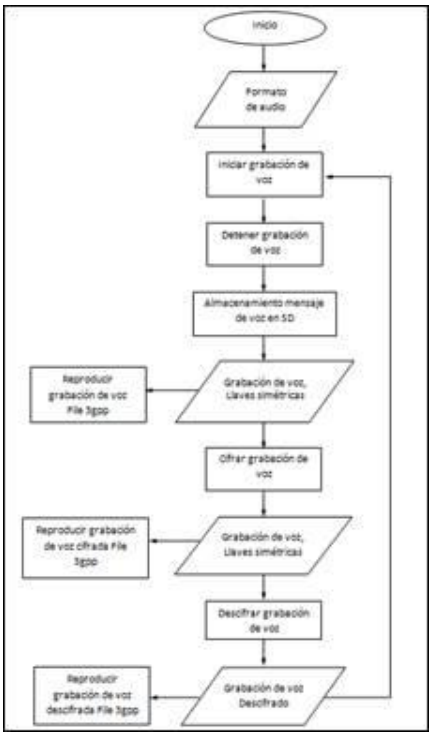

Figura 3: Diagrama general de la aplicación para cifrado de grabaciones de voz (Elaboración propia).

La idea principal de la aplicación es familiarizarse con el Framework MediaRecorder de manera que se obtenga información auditiva que posteriormente ésta sea manipulada a nivel byte, modificando completamente la información con base en el algoritmo Triple-DES 96 y obtener nuevos archivos de audio totalmente incomprensibles para aquellos terceros a los que se desea ocultar la información.

Para el algoritmo Triple-DES 96 se generaron clases y métodos propios de seguridad para el cifrado, sin la utilización de clases de la biblioteca Security contenidas en el ambiente de desarrollo de Android y Java.

La programación para el algoritmo de cifrado y descifrado está implementada en Java, utilizando la herramienta de desarrollo IDE Eclipse Indigo. La clase principal es CifradoDES3gpp, encargada de crear objetos, su principal objetivo es escribir archivos tipo 3gpp para el formato de audio. De esta tenemos tres clases derivadas: LlavesTDES96 encargada de crear las 16 llaves correspondientes para cada una de las rondas del sistema de cifrado. CifradoTDES96 encargada de cifrar el archivo 3gpp creado en la clase principal y finalmente la clase DescifradoTDES96 encomendada de descifrar el archivo 3gpp creado en la clase principal a partir de la clase CifradoTDES96.

Para la obtención de las grabaciones de voz se importaron las bibliotecas siguientes dentro del proyecto de Audio:

```
importandroid.media.MediaRecorder;
importandroid.media.MediaPlayer;
importandroid.media.MediaPlayer.OnCompletionListener;
importjava.io.File;
```

MediaPlayer reproduce las grabaciones obtenidas por MediaRecorder y los archivos cifrados y descifrados generados por las clases correspondientes. Así como File crea los nuevos archivos 3gpp.Para la grabación del archivo se instancia la clase MediaRecorder y se establecen sus atributos correspondientes.

```
MediaRecorderrecorder =newMediaRecorder()
recorder.setAudioSource(MediaRecorder
AudioSource.MIC) recorder.setOutputFormat(MediaRecorder
OutputFormat.THREE_GPP)
recorder.setAudioEncoder(MediaRecorder
AudioEncoder.AMR_NB)
```

Se crea posteriormente una ubicación en la memoria SD a través de File

```
File path = newFile(Environment
.getExternalStorageDirectory().getPath());
El cual finalmente crea un archivo temporal de audio
File archivo = File.createTempFile("temporal", ".3gpp", path);
```

A continuación se realiza la última configuración sobre el objeto recorder para comenzar la grabación, esto a través del método setOutputFilerecorder .setOutputFile(archivo.getAbsolutePath());
Una vez creados los archivos correspondientes se hace uso de los métodos prepare(), start() y stop() sobre el objeto Recorder para preparar, iniciar y detener la grabación.
Para obtener una grabación protegida o cifrada se presiona el botón cifrar sobre la aplicación, este botón recibe como parámetro una llave pública para cifrar en valores hexadecimales de 8 bytes (esta llave es ingresada por el usuario desde otra actividad de la aplicación), y un objeto de la clase principal. Las llaves cargadas en las pruebas de aplicación son las siguientes:

```
String key1 = "49354B34364C4D4E";
String key2 = "4142433132464748";
```

La clase principal encargada de crear los objetos cifrados es CifradoDES963gpp.
CifradoDES3gpp**txtcifrar** = newCifradoDES3gpp(); Un objeto creado a partir de la clase principal CifradoDES3gpp utiliza los métodos de las clases secundarias las cuales son encargados de la ejecución de los algoritmos de cifrado y descifrado.
txtcifrar.cifrarTDES96(archivo, key1, key2);

Al finalizar el cifrado la aplicación da acceso a la reproducción del archivo obtenido. El archivo generado dentro de la clase cifrarTDES96 es recuperado y almacenado en otro archivo temporal ya que es utilizado como parámetro para la clase descifraTDES96.
txtcifrar.descifrarTDES96(fileCifrado, key1, key2);
Al finalizar el descifrado la aplicación tiene acceso a la reproducción del archivo obtenido.
La Figura 4 muestra la pantalla principal de la aplicación cargada en un dispositivo Motorola Moto G primera generación.

Figura 4: Aplicación Audio Moto G.

Resultados

Las pruebas de cifrado, se realizaron grabando 10 muestras de audio con duración de 30 segundos, estas grabaciones son obtenidas directo de la aplicación, donde posteriormente se aplica el Cifrado y Descifrado, generando la aplicación 3 archivos de audio: temporal archivo original, CifradoTemp archivo cifrado, DescifradoTemp archivo descifrado.Estos archivos fueron extraídos del teléfono para comparar sus espectros y verificar la modificación correspondiente en el archivo cifrado y la recuperación en el archivo de descifrado.

La Figura 5 se representa el audio original de la grabación a una frecuencia de 48MHz, mostrando en el eje X la duración del audio y sobre el eje Y el voltaje recibido, se realiza un acercamiento en los primeros 20 segundos de grabación.

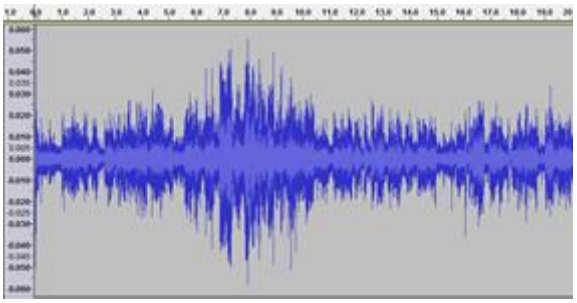

Figura 5: Espectro de grabación de audio.

Al realizar el cifrado y obtener el archivo, en la comparación de la señal se observa un cambio drástico con respecto al audio original como se observa en la figura 6, notando además que la duración de la señal se incrementa en tiempo y en potencia.

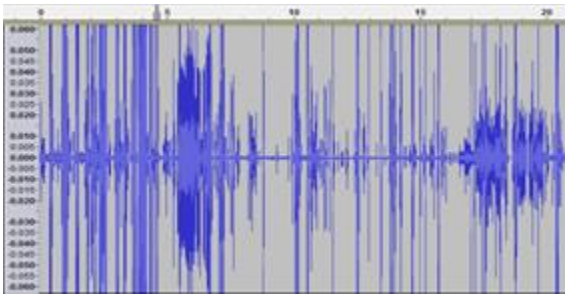

Figura 6: Espectro de grabación de audio cifrada.

En cuanto al audio obtenido en la grabación cifrada, es notorio el cambio del sonido, provocando una transformación total en la señal de salida y generando una distorsión que es incomprensible para algún tercero ajeno al dispositivo, esto debido a que el algoritmo de cifrado realiza un intercambio de bits de información alterando el espectro original y por su complejidad dificulta la recuperación del archivo base sin cambios sin la llave correcta. Finalmente en la Figura 7 se muestra el archivo descifrado con respecto al archivo original donde se observa la recuperación total de la señal.

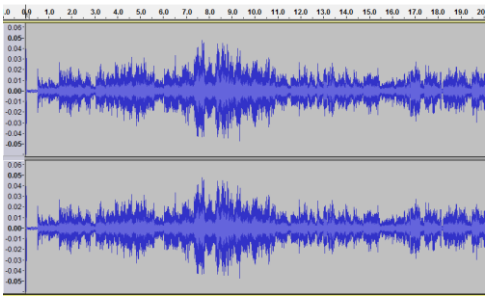

Figura 7: Espectros de grabaciones de audio original (señal superior) y descifrada (señal inferior).

Las pruebas fueron realizadas en dos dispositivos con características diferentes, tanto en software como en hardware, el primero de ellos es un modelo Sony Ericsson Xperia Neo V MT11, con las siguientes características que se muestran en la tabla 1:

Tabla 1. Características Sony Xperia Mt11.

Sistema Operativo	Android 4.0.4 Ice Cream Sandwich ICS
CPU / Procesador	QualcommSnapdragon MSM8255 / Scorpion
Clock Procesador / Núcleos	1000MHz32 bits / Single-Core
Memoria RAM	512 MB LPDDR2
Memoria Interna	1 GB
Memoria Expansible	Hasta 32Gb microSD

El segundo dispositivo es un Motorola Moto G primera generación, sus características se muestran en la tabla 2.

Tabla 2. Características Moto G 1ra Generación.

Sistema Operativo	Android 4.4.4 KitKat
Procesador	SoC Qualcomm Snapdragon 400
Clock Procesador / Núcleos	1.2GHz / Quad-Core ARM Cortex-A7
Memoria RAM	1 GB
Memoria Interna	8 GB
Memoria Expansible	N/A

En la tabla 3 se enlistan los resultados con respecto al tiempo de las pruebas para cada uno de los dispositivos, para el modelo MT11 el tamaño del archivo es de 23 KB y para el Moto G es de 60 KB, es de importancia notar que cada dispositivo genera un tamaño de archivo diferente, con lo que los tiempos se incrementan.

Tabla 3. Resultados de Cifrado y Descifrado

Sony MT11		Moto G	
Cifrado (seg)	Descifrado (seg)	Cifrado (seg)	Descifrado (seg)
7.55	6.98	12.8	12.75
7.03	7.03	12.46	12.14
7.05	7.31	12.3	12.53
7.07	7.08	12.7	12.32
7.08	6.64	11.85	11.96
7.08	7.1	12.27	12.09
6.99	7.22	11.82	12.11
6.55	7.13	12.01	12.63
7.08	7.22	12.06	12.38
7.05	7.31	12.31	13.07

El dispositivo Xperia MT11, genera un archivo de 29KB para una duración de 30 segundos con un promedio de cifrado de 7.053seg. lo que equivale a:

$$Cifrado \Rightarrow \frac{29KB}{7.053\ seg} = 4.11\frac{KB}{seg}$$

y un promedio de descifrado de 7.102seg. lo que equivale a:

$$Descifrado \Rightarrow \frac{29KB}{7.102\ seg} = 4.08\frac{KB}{seg}$$

En cuanto al dispositivo Moto G, generó archivos de 60 KB poco más del doble del modelo MT11 con un promedio de cifrado de:

$$Cifrado \Rightarrow \frac{60KB}{12.258\ seg} = 4.89\frac{KB}{seg}$$

y un promedio de descifrado de 7.102seg. lo que equivale a:

$$Descifrado \Rightarrow \frac{60KB}{12.398\ seg} = 4.83\frac{KB}{seg}$$

o que muestra que con una diferencia de tamaño el tiempo de proceso es similar de un modelo a otro.

Conclusiones

Con la información recabada, se trabajó con el framework de Audio del sistema operativo Android (disponible en todas las versiones), para realizar las pruebas de obtención de audio y la manipulación de éste. El sitio oficial de Android Developer ofrece todo lo necesario para conocer las bibliotecas contenidas en cada una de las versiones, siendo una referencia importante para el desarrollo de software en la plataforma Android, uno de los sistemas operativos para plataformas móviles más usados en el mundo. El cifrado Triple-DES 96 ofrece una gran alternativa de cifrado, ya que su complejidad es elevada gracias al tipo de permutación utilizada con respecto a sus antecesores Triple-DES y DES donde su velocidad de cifrado se ve incrementado al manejar bloques de 96 bits. Se observó gran diferencia al realizar las pruebas entre diferentes dispositivos ya que cada uno tiene diferente compresión de archivo en cuanto al formato 3gpp. Además el proceso de cifrado entre ambos, varía respecto al tamaño final del archivo. Por último es importante mencionar que este artículo es la base para realizar un proyecto que permita cifrar llamadas telefónicas en tiempo de ejecución utilizando el sistema operativo Android y los recursos de hardware, creando un sistema que pueda competir con las aplicaciones existentes en el mercado así como incitar a los alumnos de Licenciatura y Maestría a trabajar con aplicaciones enfocadas a la seguridad de la información.

Referencias

1. (2014) Google play, website [Online]. Available: https://play.google.com/store
2. V.M. Silva-García, R. Flores-Carapia, C. Rentaría-Márquez, B. Luna-Benoso "The Triple-DES-96 CryptographicSystem", Int. J. Contemp. Math. Sciences, Vol. 8, No. 19, pp.925 - 934, 2013.
3. ANSI X9.52-1998 "Triple Data Encryption Algorithm Modes of Operation".
4. V.M. Silva-García.,(2010), "Algorithm for Strengthening Some Cryptography Systems", Journal of applied Mathematics and Decision Sciences Hikari ltd. Vol. 4, No. 20, pp. 967-976, .
5. V.M. Silva;, (2010), "Reducing Computational Complexity for 192 bits SPN Encryption Algorithms with Variable Permutation", JP Journal of Algebra, Number Theory and Applications, Pushpa Publishing House. Vol. 16 No. 2, pp. 161-172, 2010.
6. Autor (**año**) *Título del artículo/libro/revista/web* texto restante
7. D. R. Stinson, CRYPTOGRAPHY: Theory and practice, Chapman & Hall/ CRC Press, 2002.
8. (2014) Android Developers, website. [Online]. Available: http://developer.android.com

9. (2014) MediaRecorder [Online] Available: http://developer.android.com/samples/MediaRecorder/index.html

10. J. T. Gironés, El gran libro de android, 2nd ed., Ed. Marcombo S.A. 2010.