INSTITUTO POLITÉCNICO NACIONAL

CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN

Confidencialidad en VoIP para el sistema operativo Android.

Tesis que presenta

Ing. Joel Martínez Ortuño

Para Obtener el Grado de

Maestría en Ciencias en Ingeniería de Cómputo con opción en sistemas digitales

Directores

Dr. Moisés Salinas Rosales.
Dr. Raúl Acosta Bermejo.

México D.F. Julio de 2014

SIP-14 bis

INSTITUTO POLITÉCNICO NACIONAL
SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de México, D.F. siendo las 11:00 horas del día 10 del mes de junio de 2014 se reunieron los miembros de la Comisión Revisora de la Tesis, designada por el Colegio de Profesores de Estudios de Posgrado e Investigación del:

Centro de Investigación en Computación

para examinar la tesis titulada:

"Confidencialidad en VoIP para el sistema operativo Android"

Presentada por el alumno:

Martínez	**Ortuño**	**Joel**
Apellido paterno	Apellido materno	Nombre(s)

Con registro: **A 1 2 0 5 8 0**

aspirante de: **MAESTRÍA EN CIENCIAS EN INGENIERÍA DE CÓMPUTO CON OPCIÓN EN SISTEMAS DIGITALES**

Después de intercambiar opiniones los miembros de la Comisión manifestaron ***APROBAR LA TESIS***, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISIÓN REVISORA

Directores de Tesis

Dr. Moisés Salinas Rosales

Dr. Raúl Acosta Bermejo

Dr. Ponciano Jorge Escamilla Ambrosio

Dr. Rolando Menchaca Méndez

Dr. Eleazar Aguirre Anaya

M. en C. Sergio Sandoval Reyes

PRESIDENTE DEL COLEGIO DE PROFESORES

Dr. Luis Alfonso Villa Vargas

INSTITUTO POLITECNICO NACIONAL
CENTRO DE INVESTIGACION
EN COMPUTACION
DIRECCION

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de México el día 10 del mes Junio del año 2014, el (la) que suscribe Joel Martínez Ortuño alumno (a) del Programa de Maestría en Ciencias en Ingeniería de Cómputo con Opción en Sistemas Digitales con número de registro A120580, adscrito al Centro de Investigación en Computación, manifiesta que es autor (a) intelectual del presente trabajo de Tesis bajo la dirección del Dr. Moisés Salinas Rosales y el Dr. Raúl Acosta Bermejo y cede los derechos del trabajo intitulado Confidencialidad en VoIP para el sistema operativo Android, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección jmartinez.ice@gmail.com. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Joel Martínez Ortuño .

Nombre y firma

Resumen

En la actualidad VoIP se ha establecido como una alternativa a las redes de telefonía tradicional, dado que la convergencia de los servicios de telecomunicación permite la reducción de costos y una mayor versatilidad comparada con la telefonía tradicional. El constante crecimiento de popularidad de los dispositivos móviles como lo son los teléfonos inteligentes, ha permitido la creación de aplicaciones de VoIP para estos dispositivos.

La necesidad de proporcionar seguridad sin afectar el rendimiento de estas aplicaciones es un deseo crucial hoy en día para aquellos sectores que utilizan este tipo de soluciones y que ademas manejan una serie de datos confidenciales. El presente trabajo propone una solución a estos problemas, proporcionando un esquema de cifrado para los flujos de datos de VoIP que se transmiten durante una llamada, este esquema de cifrado opera bajo el sistema operativo Android, el esquema de cifrado propuesto minimiza el consumo de CPU y de ancho de banda y proporciona un nivel de seguridad robusto.

En la tesis se presenta un esquema de cifrado que provee confidencialidad a los flujos de VoIP, que además otorga un rendimiento que mejora el de trabajos propuestos con anterioridad, esto se lográ haciendo uso de componentes adicionales de hardware, entre ellos, del procesador digital de señales. Se detalla la manera en que las tareas son distribuidas entre el procesador de propósito general y el procesador digital de señales. Finalmente se plantean propuestas de los trabajos que pueden derivarse de la presente investigación y que complementen la misma.

Abstract

Nowadays VoIP has been stablished as an alternative to traditional telephone networks, because the convergence of telecommunications services allows cost reduction and greater versatility compared to traditional telephony. The constantly growing of popularity of mobile devices such smarthphones, has enabled the creation of VoIP applications for these devices.

The needed to provide a better security without affecting the performance of these applications is a crucial desire today for those sector that use these solutions and also handled a big amount of confidential data. This project propose a solution by providing an encryption scheme for VoIP data streams that are transmitted during a call, the proposed encryption scheme minimizes the consumption of the CPU and bandwidth and provides a robust level of security.

The thesis describes in detail the problem and its alternatives solutions, the protocol stack used for data transport and signaling when making a call its also gived. Requirements on design and implementation are provided, the results are described and some of the future work that could be done to improve this work its gived.

Agradecimientos

Con todo cariño y amor para mis padres, Rita Ortuño y Macario Joel, co-autores de este proceso que han hecho todo en la vida para que yo pudiera lograr mis sueños, por motivarme a escalar un peldaño más en mi preparación profesional, por confiar en mi y siempre brindarme su apoyo incondicional, a ustedes por siempre mi corazón y mi más sincero agradecimiento.

A mis hermanas, Jessica Minerva y Barbara Natalia, les agradezco la confianza y la seguridad que depositaron en mi a lo largo de este camino que comenzó en el inicio de mi vida profesional.

A mi pareja, Paulina Castañeda, por su apoyo y su paciencia, quien lloró y rió en cada momento junto a mí y fue capaz de contenerme cuando todo iba mal. Gracias.

Gracias a mis directores de tesis, el Dr. Moisés Salinas Rosales y el Dr. Raúl Acosta Bermejo, por permitirme trabajar bajo su tutela, otorgándome sus conocimientos y experiencias, que hicieron posible la realización de este trabajo.

Agradezco a los amigos que adquirí durante este proceso y que fueron parte importante para lograr este trabajo. Gracias por compartir sus conocimientos y vivencias que hicieron de mi estancia en el instituto una etapa agradable y satisfactoria.

Al Instituto Politécnico Nacional, por ser la casa de estudios que ha forjado parte de mi personalidad y criterio a lo largo de 10 años de mi vida, por brindarme las mejores experiencias y la mejor preparación que ahora permite que me sienta realizado tanto profesional como personalmente.

También agradezco al Centro de Investigación en Computación por otorgarme la invitación a prepararme y obtener el grado de Maestría en Ciencias en un instituto reconocido en México por el CONACYT como competente a nivel internacional. Finalmente agradezco a esta última institución por el apoyo económico que permitió que enfocará mi concentración solo en la realización de este proyecto.

"...No existen imposibles,
siempre y cuando tengas el valor para lograrlo."
Joel Martínez.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Diseño de la Investigación

En el primer capítulo se aborda el diseño de la investigación que tiene como fundamento sentar las bases sobre las cuales se basará el desarrollo del proyecto, se da una breve información de los antecedentes del proyecto, se provee el planteamiento del problema así como la justificación y se proporcionan los objetivos general y particulares que se alcanzarán en el desarrollo de este proyecto. Finalmente se proporciona al lector la manera en la que está constituido el escrito a fin de facilitar su comprensión.

1.1. Antecedentes

La telefonía tradicional nace a partir de la necesidad de poder comunicar a dos personas separadas por una distancia considerable. La transmisión y recepción de la voz se realizan mediante dispositivos llamados teléfonos, que como se muestra en la figura ?? inicialmente estaban compuestos de un micrófono de carbón, un electro imán y un diafragma.

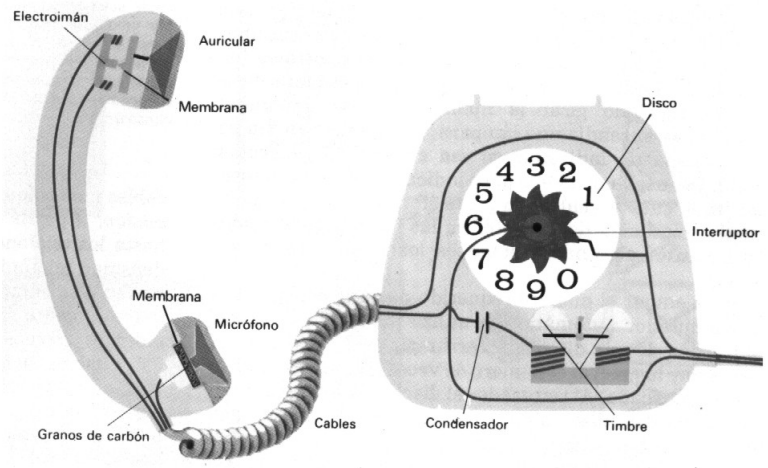


Figura 1.1: Teléfono de carbón antiguo.

En un principio la comunicación se realizaba a través de un proceso llamado conmutación de circuitos. Este proceso implica establecer un canal de comunicación dedicado entre dos estaciones que se deseen comunicar, en donde se reservan recursos de transmisión y de conmutación de la red para su uso exclusivo en el circuito durante la conexión. La conmutación de circuitos para el enlace de llamadas era realizada por personas, pero con el incremento de usuarios, las centrales no podían satisfacer la demanda en aumento. Al surgir este inconveniente se crearon los switches (dispositivos que conmutan de forma automática a cada uno de los usuarios), lo que permite que se le de mayor rapidez y fluidez al sistema de comunicación.

A este tipo de red se le conoce como PSTN *(Public Switching Telephone Network)*, y es hasta la actualidad la que permite que se puedan realizar llamadas a cualquier parte del mundo. La PSTN proporciona la conexión y desconexión de los terminales telefónicos, donde se identifican las siguientes funciones básicas:

- Conmutación. Identificación y conexión de los usuarios en una determinada trayectoria.
- Señalización. Suministro e interpretación de señales de control y supervisión que se necesitan para realizar la conmutación.
- Transmisión. Intercambio de información entre los dos usuarios que se han conectado.

Las redes de telefonía pública PSTN fueron creadas para manejar tráfico de voz y aunque también son capaces de gestionar el tráfico de datos, no lo hacen de manera eficiente, lo cual generó un cambio hacia las redes de conmutación de paquetes debido a que superan a la voz como principal generador de tráfico.

Tras el auge de Internet y la digitalización de la voz, la redes comenzaron a basarse sobre el protocolo IP *(Internet Protocol)* con la finalidad de soportar aplicaciones para las cuales la red PSTN no fue diseñada como lo es el envío de datos, voz y video, con ello surge un concepto nuevo en la telefonía que es VoIP.

VoIP (Voice over Internet Protocol) es una tecnología utilizada para la operación de servicios de transmisión de voz sobre redes de datos que operan con una serie de protocolos basados sobre IP. Esta tecnología se utiliza comúnmente para ejecutar servicios de telefonía IP. En términos generales, esta tecnología conlleva el proceso de la digitalización de la voz a través del uso de convertidores analógico-digitales (ADC), que envían la información a través de las redes de datos, para finalmente reconstruir la información utilizando un convertidor digital-análogo (DAC).[16]

El protocolo IP se encarga únicamente de dirigir los paquetes del origen al destino, a medida que van llegando a cada nodo intermedio, a través de una o varias redes distintas, sin embargo puede ocurrir que por congestión de la red, la información pueda perderse en el camino. Con la finalidad de resolver este tipo de incovenientes surgen nuevos protocolos basados en IP y uno de ellos es TCP *(Transmission Control Protocol)* el cual funciona sobre IP para proporcionar transmisión garantizada de datos: si algún paquete se pierde en el camino, este protocolo genera un aviso para que el emisor lo renvíe y así el paquete asegure su llegada al receptor.

La comunicación fiable de información que permite TCP es muy útil para aplicaciones que impliquen la transmisión de ficheros (como, por ejemplo, la propia Web). Sin embargo, para ciertas aplicaciones, impone un cierto coste en la transmisión, por la información de control que se incluye en cada paquete. En aplicaciones que requieren baja latencia como la transmisión de voz, no es aceptable el retardo que impone el procesamiento adicional de TCP, sin embargo sí se puede tolerar la pérdida de alguno que otro paquete lo que resulta en un instante de silencio. Por ello, junto a este protocolo fiable, se desarrolló otro que, como el propio IP, no ofrece ninguna garantía de recepción, pero a cambio es mucho más ligero y rápido: UDP (User Datagram Protocol). Por esto último el protocolo UDP es uno de los protocolos que en conjunto con otros forman la pila de protocolos que integran las comunicaciones de voz sobre redes de datos.

La tecnología VoIP puede describirse en dos partes principales; la primera de ellas es la señalización y la segunda el transporte de datos. La señalización se realiza utilizando protocolos como SIP *(Session Initiation Protocol)*, H.323 y MGCP; por otro lado el transporte de datos se lleva acabo utilizando el protocolo RTP *(Real time Transport Protocol)* el cual es usado para entregar los paquetes de datos de voz durante la conversación. En ambos casos el protocolo IP provee el servicio de conectividad[15]. El estudio de estos protocolos se realizará posteriormente.

Adicional a esto existen diferentes códecs que permiten codificar y decodificar los datos auditivos, es decir algoritmos que reducen la cantidad de bits que ocupa una cadena de datos dada por una señal de audio, en otros términos sirven para comprimir las señales de audio con el objetivo de ocupar el menor espacio posible con una buena calidad de audio y descomprimirlos para ser reproducidos o manipulados en diferentes formatos. Actualmente existe una gran cantidad de codecs estandarizados para su implementación en diferente tipo de aplicaciones, en el capítulo 3 se analizarán a detalle.

Con el despliegue de este tipo de tecnologías y su popularidad de uso, particularmente en dispositivos móviles, el análisis de la seguridad que proporcionan empieza a ser motivo de estudio, pues las redes basadas en IP sufren de vulnerabilidades y ataques que requieren la toma de medidas de seguridad para su prevención y corrección.

La seguridad de la información quien se encarga de dichas vulnerabilidades, es el conjunto de medidas preventivas y reactivas que emplean los sistemas tecnológicos con la finalidad de resguardar y proteger la información. Su objetivo es el proporcionar los siguientes parámetros de seguridad[20]

- Confidencialidad. Garantiza que la información solo pueda ser accedida por las partes autorizadas para ello. Su implementación constituye uno de los principales objetivos de la seguridad informática.
- Autenticación[1]. El receptor del mensaje verifica el origen del mismo, evitando que un intruso o tercer escucha no se identifique como alguien más.
- Integridad. El receptor del mensaje verifica que el mensaje no haya sido modificado en su camino, un intruso o tercer escucha no puede sustituir información falsa por legitima.
- No repudio. Proporciona protección contra la posibilidad de que alguna de las partes involucradas en la comunicación niegue haber enviado o recibido un mensaje.

La seguridad de la información hace uso de ciertas herramienta para lograr este tipo de objetivos y entre ellas esta la criptografía, la cual es el proceso de enmascarar un mensaje de tal manera que este sea oculto para quienes no estén autorizados para conocerlo[16]. Este tipo de herramienta requiere de un análisis para su compresión y su posterior implementación, el cual se proporcionará más adelante.

El mercado de la telefonía móvil sigue creciendo de manera imparable, especialmente los dispositivos con tecnologías integradas que ademas de ofrecer características básicas de un teléfono convencional, cuentan con servicios de valor agregado y se caracterizan por la

[1]También se le denomina Autenticidad que proviene del Griego verdadero o genuino del autor. Con este mismo sentido se ha creado modernamente el verbo autentificar, que se considera también válido. Ver http://www.rae.es/dpd/?key=autenticar.

integración de hardware similar a las de un ordenador personal.

Los teléfonos inteligentes o *smartphones* se distinguen por muchas características, entre las que destacan las pantallas táctiles, un sistema operativo así como la conectividad a Internet vía red celular o Wi-Fi y el acceso al correo electrónico.

Otras aplicaciones que suelen estar presentes son las cámaras integradas, la administración de contactos, el software multimedia para reproducción de música y visualización de fotos y video-clips y algunos programas de navegación así como, ocasionalmente, la habilidad de leer documentos de negocios en una gran variedad de formatos.

En lo últimos años se ha popularizado el uso de estos dispositivos en particular aquellos dispositivos con tecnologías integradas con WiFi y VoIP[11]. Además de que un factor importante que ha desatado su expansión es el sistema operativo del cual hacen uso. Existen multitud de sistemas operativos para este tipo de dipositivos, si bien las más extendidas actualmente son BlackBerry OS, Windows Mobile, iPhone OS y el sistema móvil de Google, Android.

Un aspecto importante al momento de comparar las plataformas móviles es un estudio donde se muestra la evolución del mercado de los sistemas operativos para estos dispositivos según el número de terminales vendidos. En la figura 1.2 se aprecia que en el año del 2012 Android poseía el 74 % del mercado a escala mundial, por delante de iOS que tenía el 22 %, mientras que para el año 2013 Android poseía ya el 76 % del mercado[2].

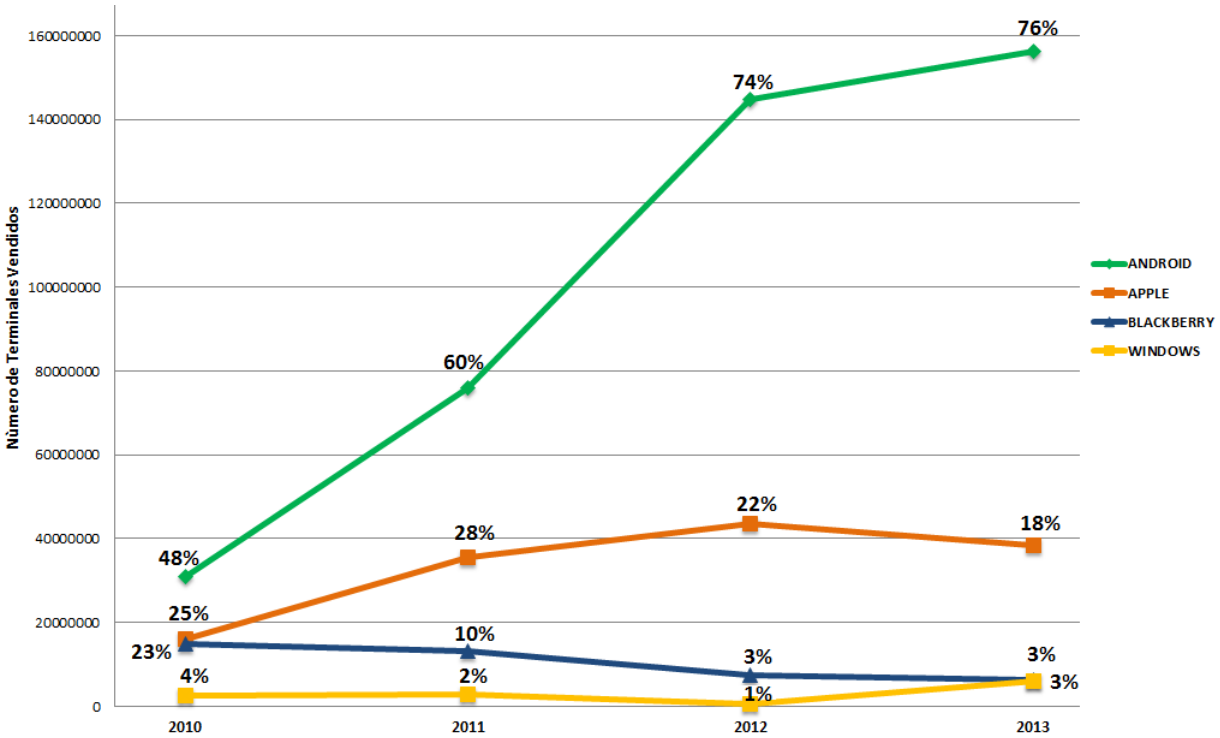


Figura 1.2: Presencia de Android en smartphones.

[2]Fuente consultada el día 3 de Enero del 2014. http://www.mobilestatistics.com/mobile-statistics/

1.2. Planteamiento del Problema

A pesar de las ventajas que ofrece la posibilidad de utilizar aplicaciones de VoIP en dispositivos móviles, existe un principal inconveniente del uso de esta tecnología y son los problemas de seguridad que la afectan, el motivo principal es que esta tecnología se conecta a través de la gran red de Internet y tanto el software como el hardware que la componen son similares a las redes utilizadas hoy en día. Por tal motivo VoIP se vuelve vulnerable a ataques intermediarios, en inglés (Man in the Middle Attack MITM).

La información que se transmite a través de la red pública de datos durante una llamada puede ser victima de software capaz de capturar, reconstruir y/o modificar este tipo de conversaciones. Las implementaciones estándar de VoIP ofrecen numerosas oportunidades para los hackers entre ellas:

- Escuchas ilegales y grabado de llamadas.
- Robo de información confidencial.
- Modificación de llamadas telefónicas.

Al ser VoIP una tecnología que se apoya en otras capas y protocolos ya existentes, hereda ciertos problemas de las capas y protocolos sobre los que está construido, siendo estos algunas de las amenazas más importantes de VoIP.

El cifrado que implementan aplicaciones existentes a fin de resolver estas problemáticas, no se apoyan en los demás componentes de hardware que componen al dispositivo, sino que basan sus esquemas en la ejecución de procesos sobre el mismo componente, el procesador de propósito general, lo que resulta en la generación de retardos en la transmisión de la voz.

1.3. Justificación

Las vulnerabilidades de los sistemas de información, en particular las heredadas por los sistemas de comunicaciones de voz IP y la necesidad de diseñar esquemas de cifrado eficientes, que permitan que el cifrado de la voz no impacte en la inteligibilidad de la llamada y que ademas permitan brindar servicios de confidencialidad sin inyectar tiempos de retardo mayores a los ya propuestos, justifican la realización de este proyecto en el área de la de computación.

En el mundo actual, la movilidad es omnipresente, sobre todo para aquellos sectores empresariales y gubernamentales que requieren que día a día se mejore la productividad de la organización. En la actualidad, aprovechar la facilidad con que se tiene acceso a un dispositivo móvil y, sobre todo, aquellos que cuentan con el S.O. Android permiten la adopción de tecnologías de la información como aplicaciones avanzadas de voz y video, sin menoscabo de la seguridad, que contribuyan al mejoramiento de la eficiencia de las organizaciones.

Actualmente Android es el sistema operativo mas utilizado en los dispositivos móviles, con un mercado del 56 % y alrededor de 400,000 aplicaciones disponibles para su descarga en Google Play [18]. La aceptación de estos dispositivos ha sido sumamente alta y al contar con capacidades de cómputo no muy lejanas a las de una computadora de escritorio permiten el despliegue de aplicaciones de VoIP en las cuales se pudiera manejar información importante.

Es por ello que el presente trabajo busca como finalidad atender las vulnerabilidades

que existen en los sistemas de información, particularmente en las comunicaciones de voz que se despliegan sobre las redes de datos y en sistemas móviles con S.O. Android, con la finalidad de optimizar los procesos de cada organización brindando comunicaciones seguras y de bajo costo computacional.

1.4. Objetivos

Objetivo General

- Diseñar un esquema de cifrado eficiente, capaz de proveer confidencialidad a comunicaciones de VoIP para el S.O. Android, utilizando algoritmos de criptografía simétrica para la protección contra posibles escuchas de terceros de información.

Objetivos Particulares

- Seleccionar una aplicación de VoIP que opere bajo el S.O. Android y posibilite la modificación de los protocolos de transmisión.
- Analizar los principales codecs utilizados en la transmisión de VoIP para seleccionar un caso de estudio.
- Diseñar un esquema de cifrado para flujos de VoIP con base en el algoritmo simétrico AES.
- Evaluar la implementación del esquema de cifrado de VoIP en términos de seguridad y rendimiento.

1.5. Alcance de la tesis

El diseño del esquema de cifrado se evaluará realizando una prueba de concepto que se construirá bajo una plataforma de desarrollo que permita obtener las mismas características con las que se cuenta en un dispositivo móvil y que además opere bajo el sistema operativo Android.

La plataforma de desarrollo con el esquema de cifrado será capaz de realizar el registro de dos o más usuarios en un servidor que proveerá los servicios de telefonía IP, a través del cual se realizarán las pruebas de comunicación. El diseño del esquema solo proveerá servicios de confidencialidad y la comunicación se llevará en primera instancia a través de la red de área local *(LAN)*.

1.6. Contribuciones

Las principales contribuciones que se tienen en el presente trabajo son las siguientes. La primera de ellas es el diseño de un esquema de cifrado para flujos de VoIP que opera en el procesador digital de señales (DSP) y que mejora el rendimiento de esquemas ya propuestos que son analizados en el siguiente capítulo.

La generación de un medio de comunicación que resuelve los problemas de comunicación entre el procesador de propósito general y el procesador digital de señales.Finalmente la última contribución radica en brindar el servicio de confidencialidad a las llamadas de VoIP sin la necesidad de inyectar paquetes adicionales que agranden el tamaño del mismo y por ende ocupen mayor ancho de banda.

1.7. Organización de la metodología desarrollada

El trabajo que se presenta se basa en el método hipotético-deductivo, se realiza un análisis detallado de la problemática a fin de presentar la mejor solución que resuelva dicho problema. Así pues, se presentan las contribuciones esperadas y se realiza un análisis de todas las variables que pudieran afectar dicho proceso, se lleva acabo un diseño en base a ciertos requerimientos y se realiza la construcción de un prototipo que permita evaluar el diseño del esquema de cifrado.

1.8. Organización del trabajo

El presente trabajo se divide en siete capítulos que se detallan a continuación:

- En el primer capítulo se detalla el diseño de la investigación, se establece el planteamiento del problema que se desea resolver, la justificación, los objetivos y los alcances del trabajo.
- El segundo capítulo realiza una recopilación de los trabajos que se han encargado de dar posibles soluciones al problema en cuestión hasta el momento, lo que servirá para realizar una comparativa entre el trabajo que se presenta y los trabajos realizados.
- El tercer capítulo se centra en el estudio de los diferentes codificadores de voz, protocolos para la transmisión de VoIP y de las herramientas criptográficas que se usan para proveer servicio de confidencialidad, en este capítulo se realiza el estudio del algoritmo de cifrado AES y el modo de operación CTR.
- El cuarto capítulo explica detalladamente el proceso que se realizó para el diseño del esquema de cifrado propuesto, se establecen los requisitos de diseño y la construcción de un prototipo a fin de evaluar el esquema de cifrado.
- El capítulo cinco establece las condiciones bajo las cuales se realizan las pruebas, se analizan los resultados y se establece una comparativa entre el esquema de cifrado propuesto y los trabajos analizados en el capítulo dos.
- En la parte final del documento se presentan las conclusiones obtenidas a partir de la investigación así como la serie de productos obtenidos y las aportaciones generadas por la presente. También se incluyen los trabajos a futuro que pudieran complementar la propuesta generada.
- Por último es reportada la bibliografía de las fuentes de información consultadas durante el desarrollo de este trabajo, así como una serie de anexos que a pesar de no ser esenciales para la investigación, fue necesario darlos a conocer y desarrollarlos para la realización de este trabajo.

Capítulo 2

Esquemas de cifrado de VoIP.

En este capítulo se realiza un estudio detallado de los trabajos de investigación que se han realizado en cuanto a las implementaciones de VoIP. Algunas indagan sobre la seguridad en VoIP y sus aplicaciones en sistemas móviles, otras realizan un exhaustivo análisis de la seguridad de los protocolos de VoIP y finalmente otras tantas analizan las vulnerabilidades en el S.O. Android y sus aplicaciones.

2.1. Implementación para asegurar VoIP habilitado en dispositivos móviles.

En [23] se diseñó un esquema de cifrado para VoIP haciendo uso del protocolo de señalización SIP y toda la pila de protocolos que éste implica. Con la finalidad de cifrar la voz se empleó el protocolo SRTP *(Secure Real Time Protocol)*.

En la figura 2.1 el llamante A o B inician la llamada, después la pila de protocolos SIP envía una petición de INVITACIÓN al otro llamante, el cual responde con un mensaje de acuse de recibo (Acknowledge ACK), una vez que el ACK es recibido la pila de protocolos SIP crea un diálogo y la sesión SIP correspondiente, es entonces cuando ambos usuarios pueden comunicarse utilizando la sesión establecida. Mientras los usuarios se encuentran en la llamada la información es cifrada utilizando el protocolo seguro SRTP y el paquete resultante es enviado utilizando el protocolo UDP.

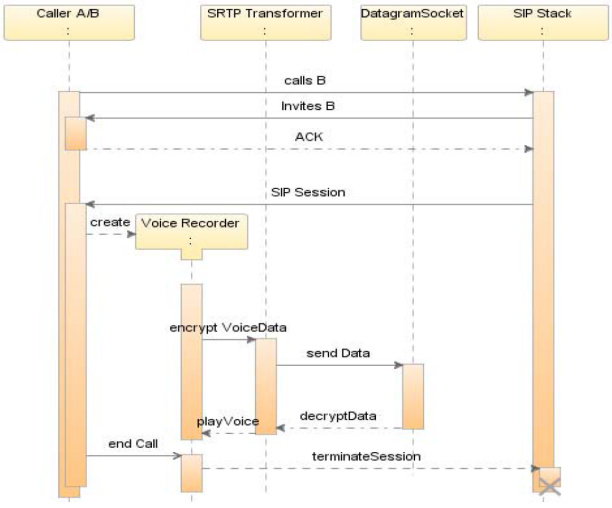


Figura 2.1: Flujo de mensajes en VoIP usando SRTP.

El diseño de la aplicación y su ejecución se realizaron sobre un emulador de Android, en la figura 2.2 se muestran las capas implementadas.

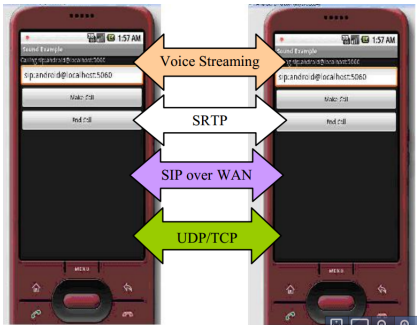


Figura 2.2: Emulador Android para VoIP.

Cada capa en el diagrama es asociada a un paquete en particular en el diseño del código. EL cifrado se lleva acabo utilizando el protocolo SRTP, algunas de las ventajas que proporciona son:

- La confidencialidad es garantizada al utilizar cargas de cifrado para RTP y RTCP.
- Dado que SRTP cifra RTP y RTCP la integridad de los mensajes se asegura.
- El marco de seguridad de SRTP puede ser extendido, lo que permite la implementación de nuevos algoritmos criptográficos.

2.2. VoIP Seguro en móviles.

En el esquema de cifrado de VoIP diseñado en [9] se empleó el protocolo *Inter-Asterisk eXchangue (IAX)*, el cual es un protocolo emergente desarrollado en conjunto con el software abierto Asterisk. Para la demostración del funcionamiento se empleó *Kiax*, un software de código abierto diseñado exclusivamente para el uso del protocolo IAX y se hizo uso de un paquete de criptografía diseñado para proveer servicios de seguridad a las aplicaciones, la librería *Cryptlib*.

En la figura 2.3 se muestran la estructura en capas de la aplicación Kiax.

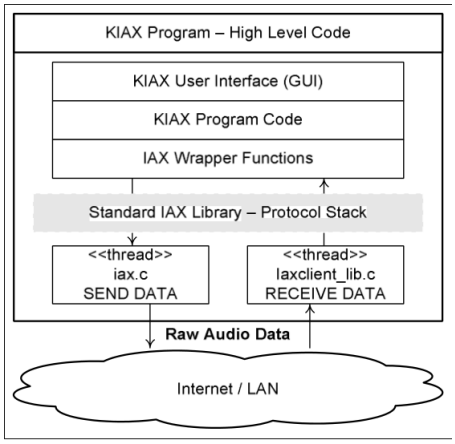


Figura 2.3: Estándar de la arquitectura Kiax.

La modificación realizada como se muestra en la figura 2.4 agrega una capa de seguridad, la cual captura las muestras de audio y las cifra antes de ser enviadas a la red de datos. Las pruebas

que se realizaron en este trabajo para la validación de la implementación fueron realizadas en un equipo de cómputo con un procesador Intel a 800 Mhz y memoria RAM de 512 MB.

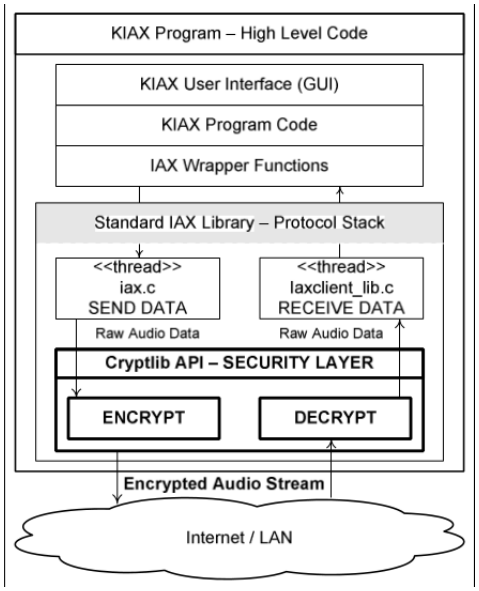


Figura 2.4: Arquitectura Kiax modificada.

En primera instancia se elaboró un análisis donde se muestra el consumo máximo, mínimo y promedio de CPU cuando la aplicación no contiene modificación alguna, se analizó el máximo consumo de ancho de banda y el retardo que el proceso de cifrar conlleva. Como se muestra en la tabla 2.1 esta comparativa muestra tanto los resultados obtenidos cuando se cifra haciendo uso del algoritmo AES en modo CBC y CFB.

Tabla 2.1: Resultados de rendimiento KIAX.

Cliente VoIP	Kiax	Kiax	Kiax
Cifrado:	Ninguno	AES/CBC	AES/CFB
Uso Min CPU:	5.812%	17.818 %	14.629%
Uso Max CPU:	10.020%	27.427%	27.22%
Uso Promedio CPU:	7.935 %	23.447%	22.592%
Max Ancho de Banda:	1.75 kB/s	2.48 kB/s	1.75 kB/s
Retardo Promedio:	-------	37 ms	37 ms

En este trabajo se realizaron análisis de rendimiento con algoritmos de cifrado diferentes a AES, entre ellos IDEA y RC4, sin embargo AES en cualquiera de los dos modos resultó ser el algoritmo con menos costo computacional al momento de realizar el cifrado, es por ello que su rendimiento no se menciona en la tabla.

2.3. Seguridad en VoIP empleando VPN y IPSec.

La solución presentada en [19], consiste en el uso de firewalls, basados en una red privada virtual, *(VPN Virtual Private Network)*, sobre el protocolo de internet seguro *(IPSec Internet Protocol Security)*.

La demostración consiste en tres fases. En la primera de ellas se demuestra las vulnerabilidades al utilizar un sistema sin protección. El sistema consta de dos teléfonos SIP, un proxy SIP que utiliza Asterisk. La configuración de los teléfonos SIP requieren localizar al proxy SIP con la finalidad de autentificarse y administrar el flujo de llamadas. Adicionalmente cada teléfono

conectado implementa el uso del codec G.711 ley μ.

Tanto el proxy SIP, como los agentes de usuario, son conectados a la red de área local a través de un switch. Además un intruso, conecta un equipo portátil a uno de los switches y monitorea el tráfico de la red con un sniffer como Ethereal. La carga de los datos puede ser capturada y guardada en un archivo de audio para después ser reproducida.

En la segunda fase, IPSec es habilitado en el borde de los routers, y de igual manera se demuestran las vulnerabilidades que existen si el intruso se encuentra dentro de la red de área local, además este proceso implica aumentar la carga de trabajo a estos dispositivos.

En la fase final, se realizó la implementación de una VPN y IPSec entre los usuarios SIP y el switch, es decir el primer punto de conexión de los telefonos SIP, el instruso captura los datos transmitidos durante una llamada entre los usuarios SIP, sin embargo, los datos capturados ahora no pueden ser interpretados pues la información va protegida.

En este trabajo se demuestra que este tipo de solución puede ser fácilmente implementada en pequeñas oficinas, cumple con la confidencialidad de los datos y la integridad de los mismos.

2.4. Protocolo SRTP.

El protocolo SRTP (Secure Realtime Tansport Protocol) define un perfil del protocolo RTP, el cual es capaz de proveer confidencialidad, autenticación del mensaje y protección contra reenvios a los datos RTP en aplicaciones unicast y multicast[2].

SRTP provee un framework para el cifrado y la autenticación de los mensajes de los paquetes RTP y RTCP, provee confidencialidad a la carga útil del paquete RTP, esto se lleva a cabo cifrando la carga útil de RTP haciendo uso del algoritmo de cifrado AES en el modo de operación contador. En el modo contador el vector de inicialización es un vector que no se repite nunca durante la transmisión de paquetes, este es tomado de un parámetro en la cabecera RTP, el parámetro SSRC (Synchronization Source) al cual se le aplica la operación XOR con el número de índice del contador y una llave de cifrado.

El principal problema de SRTP en dispositivos móviles radica en problemas NAT (Network Address Translation) transversal, esto debido a que este comportamiento no está estandarizado lo que conlleva a que las traducciones de direcciones IP con un protocolo de seguridad implementado como SRTP rompa la conectividad entre los dos puntos finales[19].

2.5. Aplicaciones en el mercado.

Hoy en día es posible descargar una serie de aplicaciones para el S.O. Android, las cuales implementan ciertas medidas de seguridad que permiten realizar llamadas seguras.

Entre las aplicaciones que implementan el protocolo *TLS (Transport Layer Security)* y *SRTP (Secure Real Time Protocol)* se encuentran:

- *Bria Android-VoIP Softphone.*
- *Acrobits Softphone.*

- *CSipSimple.*

La siguientes utilizan el protocolo ZRTP para intercambiar las llaves de seguridad e iniciar una sesión segura:

- *HushCrypt - Secure Phone Calls*
- *CryptMyCall.*
- *VoIP/SIP Dialer.*

Así mismo se encontraron aplicaciones que utilizan métodos de cifrado como el algoritmo AES de 256 y 128 bits y RSA, sin embargo éstas no se citan en el trabajo dado que no existe aún algún tipo de verificación que permita conocer que la aplicación funciona o algún usuario que ya haya trabajado con ellas. A continuación se hará una breve reseña de las dos aplicaciones más descargadas y utilizadas.

Bria Android-VoIP Softphone.

Bria Android es un softphone basado en el protocolo SIP, emplea los códecs de audio G.711a/u, G.722(HD), iLBC, GSM y SILK. Adicionalmente añade el códec G.729 el cual proporciona excelente calidad de audio dado su bajo ancho de banda el cual es ideal para conexiones con ancho de banda limitado, como 3G. En cuando a seguridad esta aplicación cifra mediante el uso de TLS y SRTP, soporta la configuración de VPN. Finalmente es una aplicación de paga.

CSIPSimple.

Es una aplicación de voz sobre IP basada en la pila de protocolos SIP. Soporta una variedad de codecs, entre ellos Speex, G.711 (u-law/a-law), GSM, iLBC, G.729, G.722, AMR, iSAC y SILK. Adicionalmente puede instalarse un plugin con la finalidad de que soporte los siguientes codecs, G.726, G.722.1 y G.729.

Cuenta con la opción de soportar la transmisión de video, con el uso del codec H264 y H263. En cuestiones de seguridad se pueden habilitar los protocolos SRTP,TLS 1.0 y ZRTP. CSipSimple es una aplicación de código abierto. Es decir esta puede ser modificada y descargada gratuitamente.

2.6. Resumen de contenido.

Los esquemas de cifrado estudiados en este capítulo proporcionan un panorama de las propuestas que han brindado una solución a los problemas de confidencialidad presentados en la transmisión de VoIP, sin embargo algunas de ellas, debido al avance tecnológico, se han convertido en obsoletas, mientras que otras como la propuesta generada en [19] es la más cercana al diseño que se presenta en el capítulo 4, sin embargo el rendimiento que otorga esta propuesta puede ser mejorado.

A fin de abordar el diseño del esquema de cifrado planteado, es necesario conocer una serie de conceptos que son proporcionados en el siguiente capítulo.

Capítulo 3

Marco de referencia.

3.1. Estándares de Codificación de Voz.

En un estudio de la tecnología VoIP, se debe de tratar el objetivo principal, que es la transmisión de la señal de voz. La naturaleza de la señal es ser analógica, es decir continua tanto en el tiempo como en la amplitud, sin embargo este tipo de señales no pueden ser transmitidas en sistemas digitales por lo que deberá ser convertida a una señal digital. Como se muestra en la figura 3.1 este cambio de formato implica el muestreo de la señal, la cuantificación de las muestras obtenidas y finalmente la codificación adecuada para su transmisión por el canal de comunicación[16].

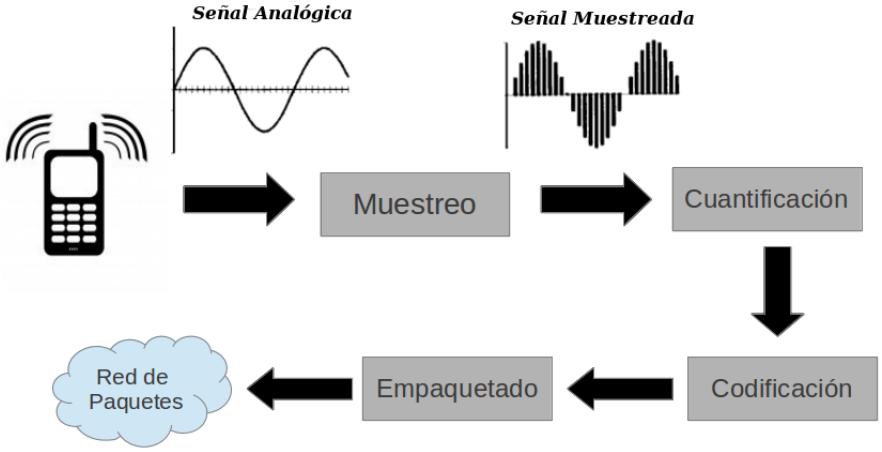


Figura 3.1: Digitalización de la señal de voz

El muestreo de la señal (figura 3.2), consiste en tomar valores discretos o muestras de la forma de onda cada determinado tiempo, una de las caracteristicas de la señal que va a marcar el proceso de muestreo es su limitación en banda, según el teorema de Nyquist[1] la frecuencia con que han de recogerse las muestras de una señal debe ser, al menos, del doble de su ancho de banda:

$$f_s \geqq 2 * Ancho\ de\ Banda\ de\ la\ Señal$$

[1]Teorema formulado en forma de conjetura por primera vez por Harry Nyquist en 1928 *(Certain topics in telegraph transmission theory)* y fue demostrado formalmente por Claude E. Shannon en 1949 *(Communication in the presence of noise)*.

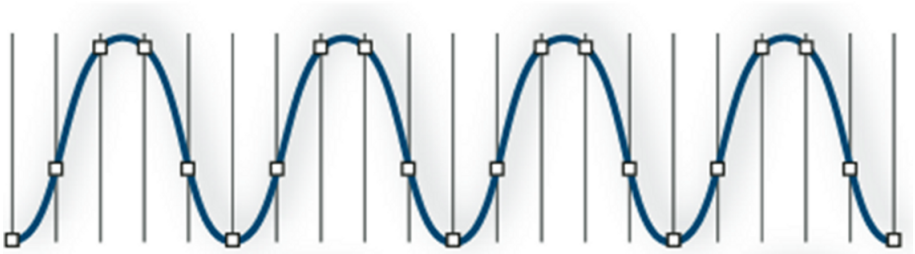


Figura 3.2: Etapa de muestreo.

Por lo tanto si el espectro de la voz se encuentra de los 300-3800 Hz, un muestreo ideal debería ser a 7600 Hz con lo cual se estaría asegurando la reconstrucción de la señal. El resultado del muestreo es un conjunto de valores de señal tomados en ciertos instantes de tiempo, en esta etapa la señal sigue siendo continua en amplitud por lo que es necesario discretizar los valores en este dominio también. Esto se lleva a cabo al asignar a cada una de las muestras un valor de uno de los M posibles y mantener ese valor hasta el siguiente instante de muestreo, a este proceso como se aprecia en la figura 3.3 se le denomina cuantificación y su objetivo es conseguir que la señal quede representada en un número finito de N bits.

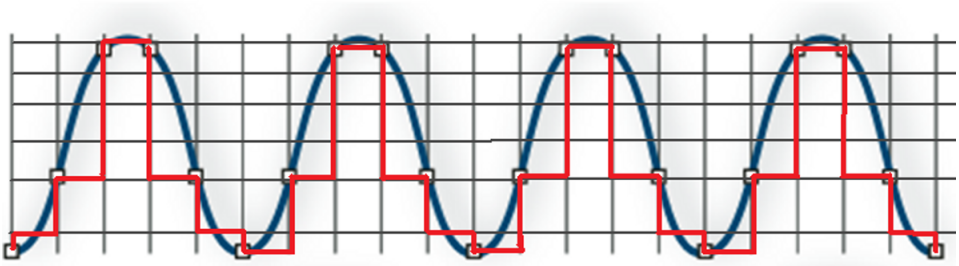


Figura 3.3: Etapa de cuantificación.

El resultado será la representación digital de la señal, existen dos modos de cuantificación, la uniforme, que es donde el paso del cuantificador es constante, es decir, supone que todas las amplitudes de la señal acontecen con la misma probabilidad, y el modo de cuantificación no uniforme o logarítmica en el que el paso del cuantificador varia, este modo funciona para el caso de la señal de voz, en la que los valores más probables son los de las amplitudes medias.

Una vez que la señal presenta ya un formato digital, el siguiente paso es codificarla, y aunque también se entiende por tal el proceso completo de la digitalización y comprensión, en este caso consiste en asignar un código binario a cada uno de los valores discretos de la señal, es decir con n bits se codifican M valores, siendo $M=2^n$

El proceso completo de digitalización de voz se realiza con el codificador/decodificador, también denominado *códec*, que además de llevar a cabo la conversión análogo/digital, comprime la secuencia de datos y proporciona la posible cancelación de ecos y la compresión de la forma de onda, lo que permite el ahorro del ancho de banda. Otra método para lograrlo es el uso de la supresión del silencio, que es el proceso de no enviar los paquetes de la voz entre silencios en conversaciones humanas. Por lo tanto es necesario realizar la correcta selección del códec a utilizar con la finalidad de obtener la mejor calidad de voz con los mínimos requerimientos de energía y ancho de banda.

Existen muchos códecs especificados por la UIT-T*(Unión Internacional de Telecomunicaciones)* para ser empleados en aplicaciones de VoIP. A continuación se realiza un análisis del

funcionamiento de estos códecs con la finalidad de proveer un panorama que permita realizar la mejor selección del códec a utilizar.

3.1.1. Códec G.711

Este es uno de los códecs mas comunes en el área de la telefonía y existen 3 razones principales de ello. La primera es que G.711 ha sido un estándar para digitalizar las señales de voz para las compañías telefónicas desde 1960, la segunda es que debido a su alta tasa de bits, proporciona una buena calidad de señal de audio y finalmente la tercera razón es que este codificador no es complejo, presenta una pequeña carga para transcodificar el audio lineal a un recurso multimedia, así facilita que el procesamiento de las señales de voz de este códec sea empleado en los dispositivos de cómputo convencionales sin la necesidad de utilizar dispositivos especializados como los DSP *(Digital Signal Processing)*[13].

G.711 es un códec que se rige bajo el esquema de *Modulación por Codificación de Pulsos (PCM)* para realizar la digitalización del audio. Las señales son muestreadas a una frecuencia de 8KHz, como se muestra en la figura 3.4 en un intervalo de 0.125ms, cada una de estas muestras de la señal es representada en un código binario.

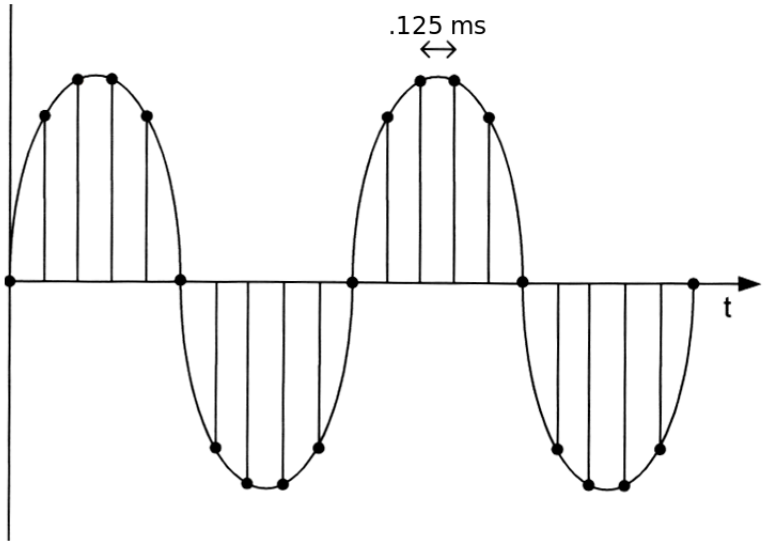


Figura 3.4: Muestreo de una señal análoga.

Así es como las frecuencias de 4 KHz, son representadas en una tasa de muestreo de 8 KHz, lo suficiente para representar la voz humana con una calidad satisfactoria. Otro factor importante que afecta la calidad de la señal, es la cuantificación. El códec G.711 emplea 8 bits de resolución para representar cada valor en un instante de muestreo, lo que resulta en 256 niveles de cuantificación (2^8). La diferencia entre el valor actual de la señal análoga y el valor cuantificado se le denomina ruido de cuantificación y es la distorsión de la señal original.

El esquema PCM utilizado por el códec G.711, no distribuye los 256 niveles linealmente a través del rango de amplitudes de la señal, sino que usa la cuantificación logarítmica. Las amplitudes bajas son cuantificadas con una alta resolución es decir, se emplean más niveles de cuantificación, mientras que las amplitudes altas se cuantifican con menor resolución. Esto se debe a que las señales de voz tienden a localizarse con mayor frecuencia en los niveles bajos de amplitud.[13] En la figura 3.5 se muestra la cuantificación no uniforme para una señal análoga.

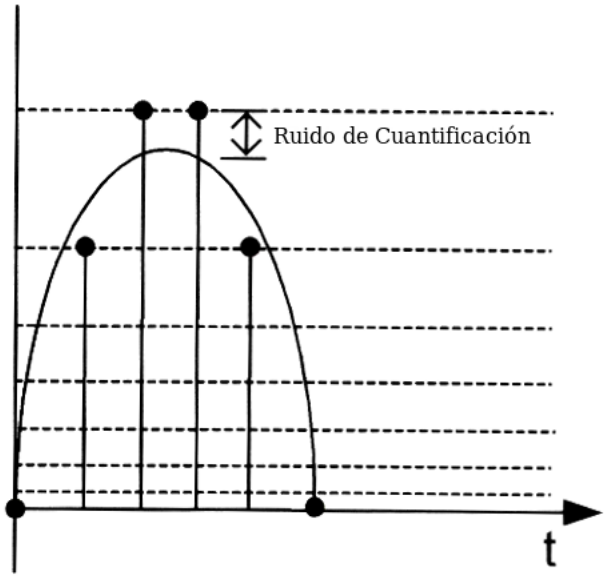


Figura 3.5: Ruido de cuantificación y cuantificación no uniforme.

El códec G.711 especifica dos esquemas de codificación conocidos como ley μ y ley A, cada uno de ellos emplea funciones logarítmicas diferentes. El método de codificación ley μ es empleado en Estados Unidos de América y Japón, mientras que la ley A es utilizada en Europa y el resto del mundo. El codificador ley μ mapea la muestra de 14 bits en una muestra de 8 bits, por su parte la ley A mapea una muestra de 13 bits en 8 bits. El generar 8 bits de datos a una frecuencia de muestreo de 8 KHz resulta en una velocidad de bits de 64 Kbit/s[5].

3.1.2. Códec G.722

Estandarizado por la UIT-T es un códec de audio de banda ancha que garantiza una alta calidad para diversas aplicaciones como VoIP, videoconferencias, teleconferencias, y las aplicaciones de streaming de Internet. G.722 es un compresor de transformación basado en que está optimizado para la voz, la música y otros tipos de audio, ya que es capaz de proporcionar una experiencia de alta calidad. Utiliza MLT (Modulated Lapped Transform) y opera en tramas de 20 ms (320 muestras) de audio. MLT es una reconstrucción crítica de muestreo y perfecta de transformación lineal.

G.722 opera con un ancho de banda de 7 kHz a 50 kHz y una tasa de bits de 16, 24 y 32 kbps. Por lo que es muy útil para aplicaciones de alta calidad en sistemas como VoIP. El intervalo en el que se envía una trama es de 20 ms.

El sistema de codificación utiliza modulación por pulsos codificados adaptativo diferencial de subbanda (SB-ADPCM) a una velocidad binaria de hasta 64 kbit/s. La banda de frecuencias se divide en dos subbandas (superior e inferior) y las señales de cada una se codifican utilizando ADPCM. El sistema tiene tres modos básicos de funcionamiento, correspondientes a las velocidades binarias utilizadas para la codificación de audio de 7 kHz: 64, 56 y 48 kbit/s. Los dos últimos modos permiten obtener, respectivamente, un canal de datos auxiliar de 8 kbit/s o de 16 kbit/s, que se proporciona dentro de los 64 kbit/s mediante el uso de bits de la sub-banda inferior [7].

3.1.3. Códec G.723.1

Estandarizado en 1988, es descrito como un codificador de doble velocidad para comunicaciones multimedia que transmite a una velocidad de 5.3 Kbit/s y 6.3 Kbit/s. Fue el resultado de una competencia que hizo el ITU para diseñar un codificador para llamadas telefónicas en modems de 28.8 y 33 kbit/s. Hubo 2 muy buenas soluciones y el ITU decidió usar las 2.

Este códec se optimizó para representar la voz con una calidad alta a las velocidades mencionadas mediante una complejidad limitada. Codifica la voz u otras señales audio en tramas mediante la codificación predictiva lineal de análisis por síntesis. La señal de excitación del códec de alta velocidad es la cuantificación multiimpulso de máxima verosimilitud (MP-MLQ) y la del códec de velocidad baja es la predicción lineal excitada por tabla de códigos algebraicos (ACELP). Este códec codifica la voz u otras señales de audio en tramas de 30 ms. Además, tiene un preanálisis de 7.5 ms, lo que resulta en un retardo algorítmico total de 37.5 ms.[3]

3.1.4. Códec G.729

El códec en cuestión está diseñado para operar con una señal digital obtenida tras efectuar, primero un filtrado con la anchura de banda telefónica de la señal analógica de entrada, seguido de su muestreo a 8000 Hz y su conversión a una modulación por pulsos codificados (PCM) lineal de 16 bits, para entrar en el codificador. La salida del decodificador deberá reconvertirse a una señal analógica siguiendo un método similar.

Requiere un ancho de banda bajo pero proporciona una buena calidad de audio (MOS=4.2) por este motivo es un códec que se toma a consideración para su implementación en aplicaciones como VoIP. Codifica el audio en tramas: cada trama es de 10ms, dada la frecuencia de muestreo de 8 kHz, una trama de 10 ms contiene 80 muestras de audio. El algoritmo de codificación codifica cada trama en 10 bytes, resultando en un bitrate de 8 kbit/s[6].

Es un codificador con licencia. Lo habitual es comprar el hardware que lo implementa (IP teléfono o enrutador) en cuyo caso la licencia ya fue pagada por el constructor del chip usado en el dispositivo.

3.1.5. Códec GSM

GSM *(Global System for Mobile communications)* es un sistema digital de radio móvil,que es extensamente empleado en Europa y otras partes del mundo. GSM ha utilizado una variedad de codecs con la finalidad de comprimir el audio de 3.1 kHz a 6.5 y 13 kbps. Originalmente dos códecs fueron diseñados, el *Half Rate* y el *Full Rate*. Ambos utilizan un sistema basado en LPC (Linear Predictive Coding). Fue el primer códec digital estandarizado por el ETSI y desarrollado cerca del año 1990, por esta razón no asegura una alta calidad de voz, está basado en los paradigmas RPE-LTP *(Regular Pulse Excitation - Long Term Prediction)*.

Trabaja con tramas de audio de 20 ms (160 muestras) y comprime cada trama a 33 bytes, resultando en un bitrate de 13 kbit/s. La trama mide exactamente 32 y 1/2 bytes, por lo que 4 bits no son usados en cada trama. Tiene un MOS de 3.7. La variante de GSM puede ser utilizado de manera gratuita por lo que se encuentra seguido en aplicaciones open-source VoIP.

3.1.6. Códec Speex

Speex es un códec de recurso libre, el proyecto Speex apunta a reducir la barrera de entrada para aplicaciones de voz, proporcionando una alternativa gratuita a los costosos códecs de voz de patente. Por otra parte, Speex está bien adaptado para las aplicaciones de Internet y proporciona características útiles que no están presentes en la mayoría de los otros codecs. Speex se basa en *CELP (Code-excited linear prediction)* y está diseñado para comprimir voz a velocidades de bits que van desde 2 hasta 44 kbps[21].

Opera en la banda estrecha de 8 kHz, banda ancha de 16 kHz, y ultra banda ancha de 32 kHz de compresión en el flujo de bits mismo, implementa detección de actividad de voz (VAD) y cancelación de ecos así como la supresión de ruido. Agrega un retardo de 34 ms y tiene un MOS de 3.67.

En la figura 3.6 se muestra la comparación entre el códec Speex y el códec G711, G.729a y GSM en base al MOS (Meaning Opinion Score) que es una medida de percepción del audio después de ser comprimida por un códec en particular. La puntuación es asignada por un grupo de escuchas que emplearon el procedimiento especificado en el estándar de la IUT-T P.800 y P.830. El rango va de 5 (calidad de audio excelente) a 1 (calidad de audio mala)[1].

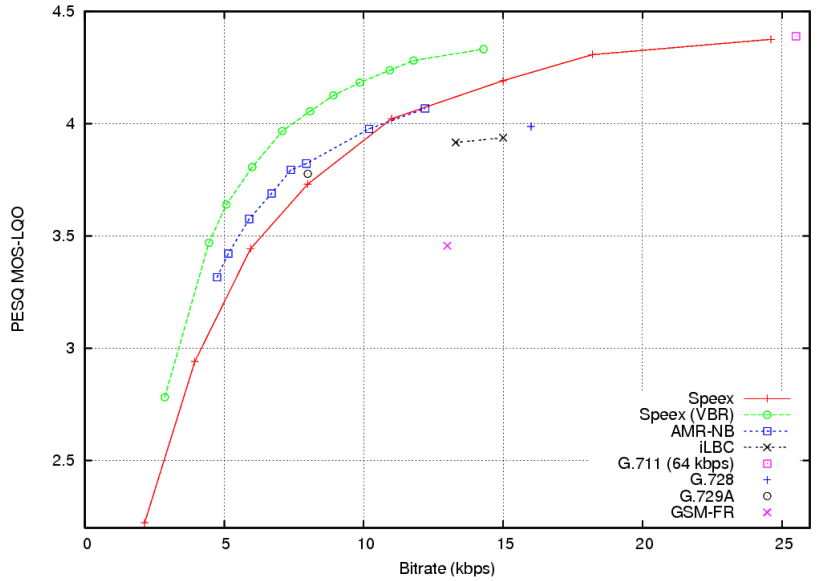


Figura 3.6: Comparación de MOS. (Speex, G.711, G.729 y GSM)

3.2. Protocolos para la transmisión de VoIP.

Al igual que en las redes de datos, las redes de voz sobre paquetes requieren de una serie de protocolos que especifiquen las funcionalidades y servicios que este tipo de tecnología debe de proveer. Los protocolos al ser aceptados internacionalmente permiten la inter-operabilidad entre las distintas plataformas que ofrecen los fabricantes.

En las redes telefónicas convencionales, una llamada consta de tres fases: establecimiento, comunicación y desconexión. Durante el establecimiento se reservan los recursos necesarios que se utilizarán en la fase de comunicación, lo cual permite que los datos puedan fluir de extremo a extremo, finalmente en la fase de desconexión, los recursos empleados son liberados y se transmite la información necesaria para que la llamada pueda ser tarificada. Este tipo de

esquema sigue siendo valido en la tecnología VoIP, aunque cuenta con algunos matices.

Tradicionalmente se distinguen dos grupos de protocolos que pueden ir sobre TCP y/o UDP y ambos sobre IP.

- **Protocolos de transporte.** Definen como debe realizarse la comunicación entre los extremos por un canal de comunicaciones previamente establecido. Los protocolos de transporte más empleados son RTP y RTCP.
- **Protocolos de señalización.** Su principal objetivo es establecer un canal de comunicación a tráves del cual fluya la información del usuario y liberar el canal cuando finalice la comunicación. Para ello debe existir un diálogo entre los componentes de la red y los terminales de usuario. Los protocolos de señalización que actualmente se usan son H.323 y SIP, este último será el objeto de estudio.

A continuación se proporciona un análisis de los protocolos de transporte y señalización que hacen posible la comunicación de voz sobre IP con el propósito de su adecuada comprensión.

3.2.1. Protocolo RTP

El protocolo RTP *(Real Time Protocol)* es un estándar de comunicación definido en la RFC[2] 3550 es un estándar que define las comunicaciones de audio y video en tiempo real sobre las redes IP, provee funciones de transporte extremo a extremo, ofrece servicios tales como la identificación del tipo de carga, número de secuencia y *timestamping*.

RTP por si mismo, no provee ningún mecanismo para asegurar la entrega de tráfico en tiempo real, ni provee algúna garantía de calidad de servicio, en cambio suministra los recursos para que los datos se entreguen de manera sincronizada, el número de secuencia incluido en este protocolo permite al receptor reconstruir el paquete enviado de manera secuencial. Aunque la norma no lo establece, RTP suele emplearse sobre el protocolo UDP ya que el servicio fiable que proporciona TCP daría lugar a retardos, lo cual resultaría en una mala calidad de transmisión y percepción al momento de realizar una llamada.

La cabecera de RTP tiene el siguiente formato:

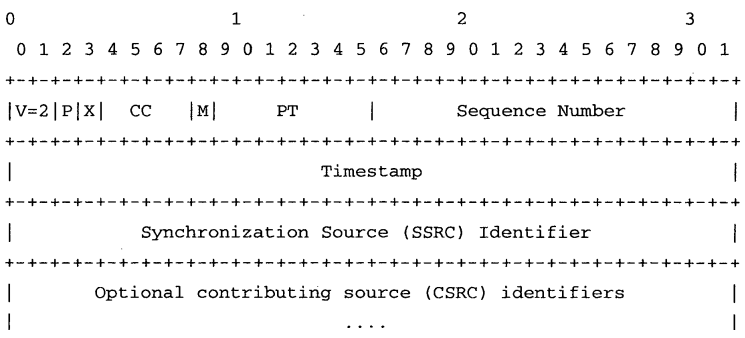


Figura 3.7: Cabecera RTP

Donde:

- ***Número de versión RTP*** *(V - versión number):* 2 bits. La versión definida por la especificación actual es 2.

[2]RFC *(Request for Comments)* son una serie de notas técnicas y organizacionales que cubren aspectos sobre las redes informaticas, incluyendo protocolos, procedimientos y conceptos, que comenzaron a publicarse en 1969.

- ***Relleno*** *(P - Padding):* 1 bit. Si el bit del relleno está activado, hay uno o más bytes al final del paquete que no es parte de la carga útil. El último byte del paquete indica el número de bytes de relleno. El relleno es usado por algunos algoritmos de cifrado.

- ***Extensión*** *(X - Extensión):* 1 bit. Si el bit de extensión está activado, entonces el encabezado fijo es seguido por una extensión del encabezado. Este mecanismo de la extensión posibilita implementaciones para añadir información al encabezado RTP.

- ***Conteo CSRC*** *(CC):* 4 bits. El número de identificadores CSRC que sigue el encabezado fijo. Si la cuenta CSRC es cero, entonces la fuente de sincronización es la fuente de la carga útil.

- ***Marcador*** *(M - Marker):* 1 bit. Un bit de marcador definido por el perfil particular de medios.

- ***Tipo de Carga útil*** *(PT - Payload type):* 7 bits. Un índice en una tabla del perfiles de medios que describe el formato de carga útil. Los mapeos de carga útil para audio y vídeo están especificados en el RFC 3551.

- ***Número de Secuencia*** *(Sequence number):* 16 bits. Un único número de paquete que identifica la posición de éste en la secuencia de paquetes. El número del paquete es incrementado en uno para cada paquete enviado.

- ***Sellado de tiempo*** *(Timestamp):* 32 bits. Refleja el instante de muestreo del primer byte en la carga útil. Varios paquetes consecutivos pueden tener el mismo sellado si son lógicamente generados en el mismo tiempo - por ejemplo, si son todo parte del mismo frame de vídeo.

- ***SSRC*** *(Synchronization source):* 32 bits. Identifica la fuente de sincronización. Si la cuenta CSRC es cero, entonces la fuente de carga útil es la fuente de sincronización. Si la cuenta CSRC es distinta a cero, entonces el SSRC identifica el mixer(mezclador).

- ***CSRC*** *(Optional contributing source):* 32 bits cada uno. Identifica las fuentes contribuyentes para la carga útil. El número de fuentes contribuyentes está indicado por el campo de la cuenta CSRC; Allí puede haber más de 16 fuentes contribuyentes. Si hay fuentes contribuyentes múltiples, entonces la carga útil son los datos mezclados de esas fuentes.

En resumen, RTP permite identificar el tipo de información transmitida, agregar marcadores temporales y números de secuencia a la información transmitida y controlar la llegada de los paquetes a destino. Además, los paquetes de difusión múltiple pueden utilizar RTP para enrutar conversaciones a múltiples destinatarios.

3.2.2. Protocolo RTCP

El protocolo RTCP (*Real Time Control Protocol*) es un protocolo que al igual que en conjunto con RTP se especifica en el RFC 3550, este protocolo de comunicación proporciona información de control asociada a un flujo de datos multimedia RTP, es decir, trabaja en conjunto a RTP y aunque es un protocolo opcional y no trasporta ningún dato por sí mismo, se recomienda su uso, pues informa acerca de la calidad de servicio porporcionada por RTP.

Recolecta estadísticas de la conexión, paquetes enviados y recibidos así como el *Jitter (variabilidad del retardo)*, con el fin de detectar situaciones en las que la calidad de la

transmisión no es suficiente, y aunque no provee mecanismos necesarios para mejorar las prestaciones de la red, si permite a las aplicaciones que lo usan utilizar esta información para mejorar la calidad del servicio, ya sea controlando el flujo de datos o utilizar otro códec de compresión más baja.

3.2.3. Protocolo de señalización SIP

El protocolo SIP (*Session Inition Protocol*) es un protocolo de señalización especificado bajo la RFC 3261, define cómo establecer, modificar o finalizar una sesión de comunicación entre dos o más extremos. SIP define los elementos que participan en un entorno de comunicación que utilice este protocolo así como el sistema de mensajes que intercambian éstos. Los mensajes están basados en HTTP y se emplean esencialmente en procedimientos de registro y para establecer entre qué direcciones IP y puertos TCP/UDP intercambiarán datos los usuarios.

En SIP se establece una arquitectura que permite ofrecer presencia y movilidad a un usuario SIP, en ella se identifican terminales de usuarios, servidor de registro, servidor proxy, servidor de localización y servidor de redirección. Básicamente cuando un usuario o terminal SIP desea establecer una comunicación con otro, envia un mensaje a su servidor proxy para conocer la dirección física que tiene en ese momento el destinatario, en esta fase el llamante indicará al servidor proxy la dirección lógica del usuario con el cual se desea comunicar, por ejemplo el nombre del usuario, con lo cual el servidor de localización resuelve la dirección física del usuario llamado.

Una de las claves para el desarrollo e implementación de SIP es la sencillez de la suite de protocolos que maneja[16]. SIP utiliza a efectos de transporte los protocolos UDP, RTP y RTCP, mientras que para la codificación y compresión de la voz utiliza los protocolos G.7xx.

Dentro de la arquitectura del protocolo SIP se pueden identificar los agentes de usuario cliente (*UAC, User Agent Client*), que son las terminales que solicitan iniciar una nueva llamada o terminar una en curso y de iniciar las sesiones SIP, y el agente de usuario servidor (*UAS, User Agent Server*), quien es el responsable de aceptar las peticiones de establecimiento de sesión recibidas. Los servidores o servicios de red actúan como intermediarios entre los agentes de usuario de los cuales se distinguen los siguientes:

- **Servidor proxy.** Es una entidad intermediaria que realiza peticiones de parte de otros clientes. Estas peticiones son servidas internamente o reenviadas a otros servidores.
- **Servidor de localización.** Es quien proporciona información acerca de un usuario. Si un usuario A desea comunicarse con B, necesita descubrir la localización de éste a fin de enviar una petición de establecimiento de sesión.
- **Servidor de redirección.** Acepta una petición SIP y mapea la dirección en cero o más direcciones nuevas devolviendolas al cliente.
- **Servidor de registro.** Acepta peticiones de los agentes de usuario clientes y actualiza la información relativa a cada uno de ellos en una base de datos de localización.

SIP es considerada como una arquitectura punto a punto descentralizada, pues gran parte de su arquitectura reside en los terminales y aunque no existan elementos centrales *(proxy, registro)* se podría operar sin ellos si se conociera la dirección física del interlocutor.

El diálogo entre los clientes y los servidores SIP se basa en el intercambio de mensajes de texto. La estructura genérica de un mensaje SIP, sea este de petición o respuesta, contiene una línea de comienzo, una o más cabeceras y una línea vacía que indica el final de las cabeceras y el cuerpo del mensaje opcionalmente. Los mensajes de petición son enviados por los agentes de usuario cliente a los servidores SIP y generalmente toda petición viene acompañada de una respuesta por parte del servidor, a excepción del ACK que no requiere respuesta, toda respuesta SIP tiene asociado un código numérico que indica el resultado del intento de servir la petición al cliente.

La manera más básica de manejar una sesión SIP es mediante los siguientes pasos:

- Registro, iniciación y localización del usuario.
- Descripción de la sesión multimedia que se pretende establecer.
- Aceptación de conexión del otro extremo.
- Establecimiento de la llamada.
- Comunicación.
- Terminación de la llamada.

Los mensajes SIP se transportan utilizando el protocolo UDP y además se emplea el protocolo SDP (*Session Description Protocol*) como encargado de la descripción de las características de la sesión entre los extremos. El principal objetivo de SDP es proporcionar información acerca de los flujos de datos a los respectivos receptores, de esta forma cada receptor dispone de una descripción de la sesión multimedia en la que participa.

3.3. Herramientas Criptográficas para la Confidencialidad

Durante el paso del tiempo una serie de elaborados protocolos y mecanismos se han creado para lidiar con los problemas que enfrenta la seguridad de la información, en este capítulo se analizarán los mecanismos de seguridad que se emplean así como las herramientas usadas para construir un sistema de seguridad robusto capaz de proveer confidencialidad al sistema propuesto.

3.3.1. Criptografía.

Criptografía es el estudio de técnicas matemáticas relacionadas a los aspectos de la seguridad de la información tales como la confidencialidad, la integridad de los datos, autentificación de la entidad y autentificación de los datos [10]. La criptografía por si sola no es en su totalidad una herramienta de la seguridad de la información si no que es una serie de técnicas o *primitivas* que conforman una sola herramienta.

En un sistema básico de comunicación como se muestra en la figura 3.8, existen dos participantes (A y B), los cuales desean comunicarse, y un tercer participante (E) quien representa un potencial intruso. Cuando A desea enviar un mensaje a B, lo cifra utilizando un método preestablecido con B. Usualmente el método de cifrado es conocido por E, lo único que mantiene seguro al mensaje enviado es la llave de cifrado k. Cuando B recibe el mensaje cifrado o *criptograma* lo regresa a su estado original a través de la llave de descifrado.

En este caso E pudiera tener algunos de los siguientes propósitos: leer el mensaje, conocer

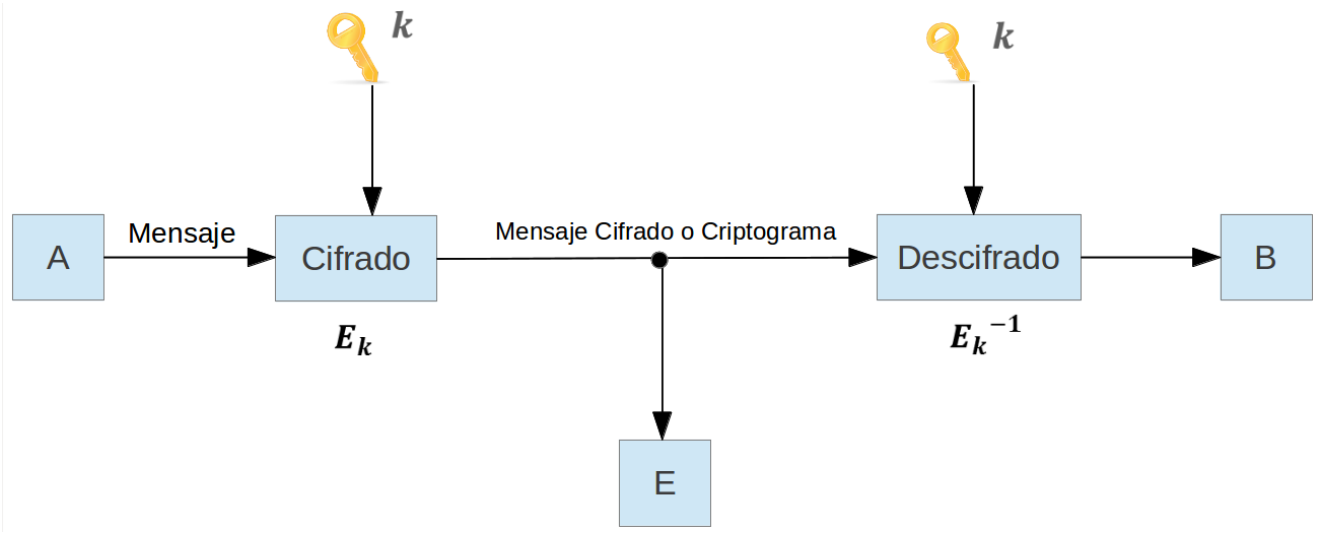


Figura 3.8: Sistema básico de comunicación en la criptografía.

la llave y por lo tanto leer todos los mensajes cifrados con esa llave, alterar el mensaje enviado de tal manera que B piense que A mando el mensaje alterado, hacerse pasar por A y comunicarse con B. En este caso se estarían corrompiendo con los objetivos de la seguridad de la información, los cuales son confidencialidad, integridad y autentificación respectivamente.

Los métodos de cifrado y descifrado caen en dos categorías: *criptografía de llave simétrica y criptografía de llave asimétrica o pública.*

- Criptografía Simétrica.

En la técnicas de criptografía simétrica las llaves de cifrado y descifrado son conocidas por A y B, es decir la llave en ambos casos es la misma. Consideremos un sistema de seguridad donde existe ambas transformaciones de cifrado y descifrado, $\{E_e : e \in K\}$ y $\{D_d : d \in K\}$ respectivamente, donde K es el espacio de la llave. Se dice entonces que el sistema criptográfico es de llave simétrica si por cada par de llaves asociadas (e,d),es computacionalmente "fácil"determinar d conociendo solo e y determinar e a partir de d.

En la figura 3.9 se ilustra un ejemplo de un sistema de cifrado simétrico, en muchos de los esquemas prácticos de la criptografía simétrica la llave es la misma es decir $e = d$ de ahí el término adecuado de simetría a dicho sistema. Uno de los mayores inconvenientes de este sistema es encontrar un método eficiente para el acuerdo de llave, es decir la distribución de la llave, ya que ésta se transmite a través de un canal de comunicación inseguro. Algunos de los criptosistemas simétricos utilizados actualmente son el *DES* y el *AES*, algoritmos de cifrado que se estudiarán más adelante.

- Criptografía Asimétrica.

Por su parte la técnica de criptografía asimétrica como se aprecia en la figura 3.10, viene a resolver el problema de la distribución de llave que se presenta en el sistema anterior, en este caso la llave de cifrado se hace pública, pero la llave de descifrado se convierte en computacionalmente "imposible"de encontrar si no se cuenta con la información que es solo conocida por el destinatario B.

Consideremos la transformación $\{E_e : e \in K\}$ como el bloque de cifrado y $\{D_d : d \in K\}$ como el bloque de descifrado, para ambos casos K es el espacio de llaves. Cada una de

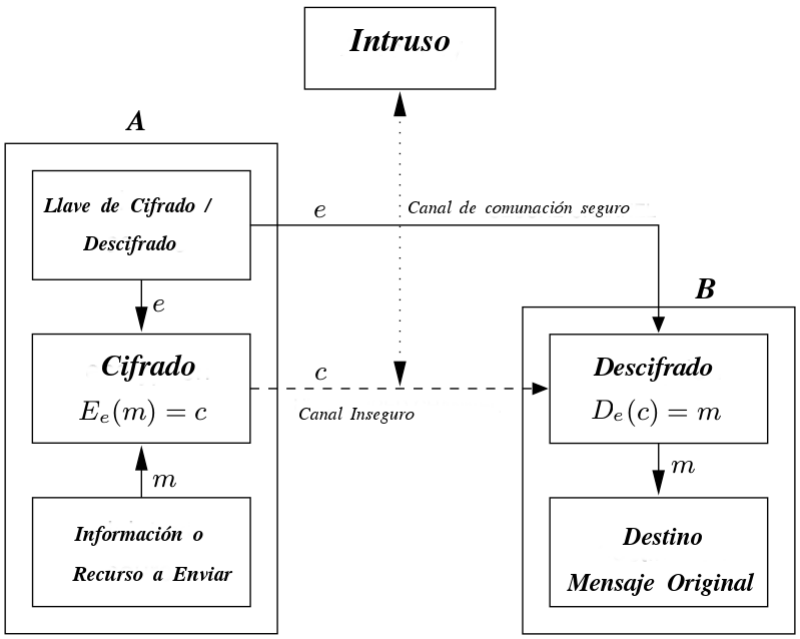


Figura 3.9: Criptografía simétrica.

las transformaciones tiene la propiedad de que si se conoce E_e, es computacionalmente "imposible"que a partir de un mensaje cifrado $c \in C$, se pueda encontrar el mensaje original $m \in M$ tal que $E_e(m) = c$. Esta propiedad implica que a partir de la llave e, es "imposible"determinar la correspondiente llave de descifrado d.

Bajo este esquema consideremos que entre A y B existe una comunicación como la que se ilustra en la figura 3.10, donde B selecciona un par de llaves (e,d), envía la llave (e) tambien denominada *llave pública* a A, a través de un canal inseguro de comunicación y mantiene la llave (d) o *llave privada* en secreto y segura. Con esto A puede entonces enviar un mensaje (m) a B, haciendo uso de la transformación de cifrado determinada por la llave pública de B. Para descifrar el mensaje B aplica la transformación inversa únicamente con la llave privada (d).

En comparación con el sistema de criptografía simétrica, este esquema es capaz de transmitir la llave de cifrado a través de un canal de comunicacion inseguro, mismo por el cual se enviará la información cifrada, además al ser pública la llave de cifrado (e), se permite que cualquier entidad pueda enviar mensajes a B, siendo B el único que puede descifrarlos.

Las técnicas de la criptografía asimétrica son robustas en cuestiones de seguridad sin embargo imponen un cierto coste computacional al momento de cifrar grandes cantidades de datos, por esta razón la criptografía de llave pública es utilizada para cifrar pequeñas cantidades de datos, por ejemplo el envió de llaves que se utilizarán en un sistema de llave simétrica[22].

3.3.2. Data Encryption Standart *DES*

En 1975 el NIST (*National Institute of Standarts and Technology*) publicó lo que era en esencia el algoritmo de cifrado DES, este algoritmo ha sido extensamente usado en el comercio electrónico, por ejemplo en la industria de los bancos. El algoritmo DES es un algoritmo de cifrado simétrico de bloques, fragmenta la información a cifrar en bloques de 64 bits y cifra cada bloque por separado[22].

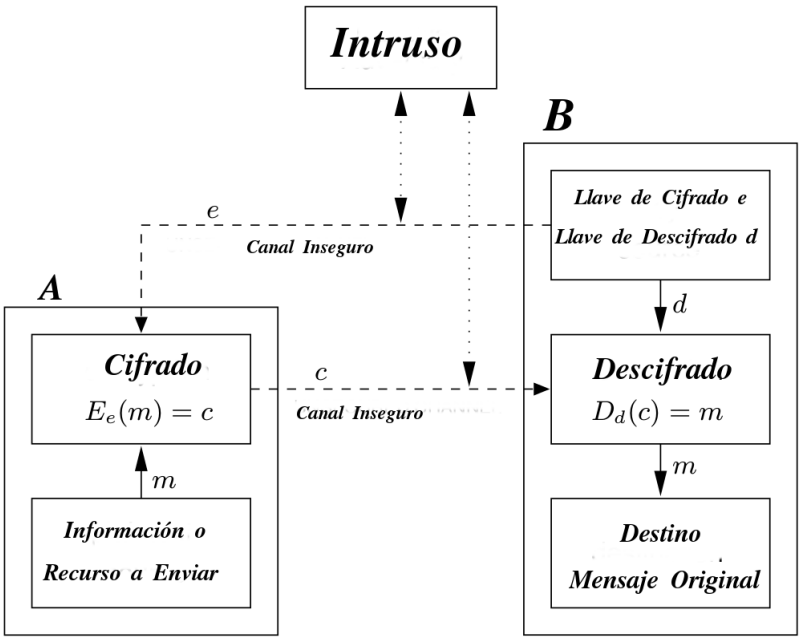


Figura 3.10: Criptografía asimétrica.

El algoritmo DES como se muestra en la figura 3.11 consta de 16 rondas de cifrado, en donde la llave de cifrado *(K)* es de 56 bits pero es expresada en 64-bits y de la cual se derivan 16 sub-llaves que se emplearán para cada ronda de cifrado.

El algoritmo DES (figura 3.11) comienza con un texto plano *(m)* de 64 bits y consta de 3 etapas:

1. Los bits del texto plano m son permutados con la finalidad de obtener m_0 que es la permutación inicial de 64 bits. Se define $m_0 = L_0R_0$, donde L_0 son los primeros 32 bits de m_0 y R_0 son los últimos 32 bits.

2. Para la siguiente etapa que son las rondas de cifrado se realiza lo siguiente, tomando en consideración 16 rondas de cifrado $1 \leq i \leq 16$ los siguientes bloques de 32 bits se definen de la siguiente manera.

$$L_i = R_{i-1} \tag{3.1}$$

$$R_i = L_{i-1} \bigoplus f(R_{i-1}, K_i) \tag{3.2}$$

   donde K_i es una cadena de 48 bits obtenida de la llave K y f es una función que se explicará a continuación.

3. Se realiza un intercambio de derecha a izquierda y viceversa con la finalidad de obtener $R_{16}L_{16}$, se aplica una permutación inversa y se obtiene el texto cifrado.

La función $f(R_{i-1}, K_i)$ se describe en la figura 3.12 de la siguiente manera:

1. Se realiza una expansión de bits, de 32 a 48 resultando $E(R)$.

2. Se realiza la operación XOR: $E(R) \bigoplus K_i$ de donde se obtienen 48 bits que son descritos como $B_1, B_2, ...B_8$ donde cada bloque B_j está compuesto por 6 bits formando asi los 48 bits obtenidos de la operación.

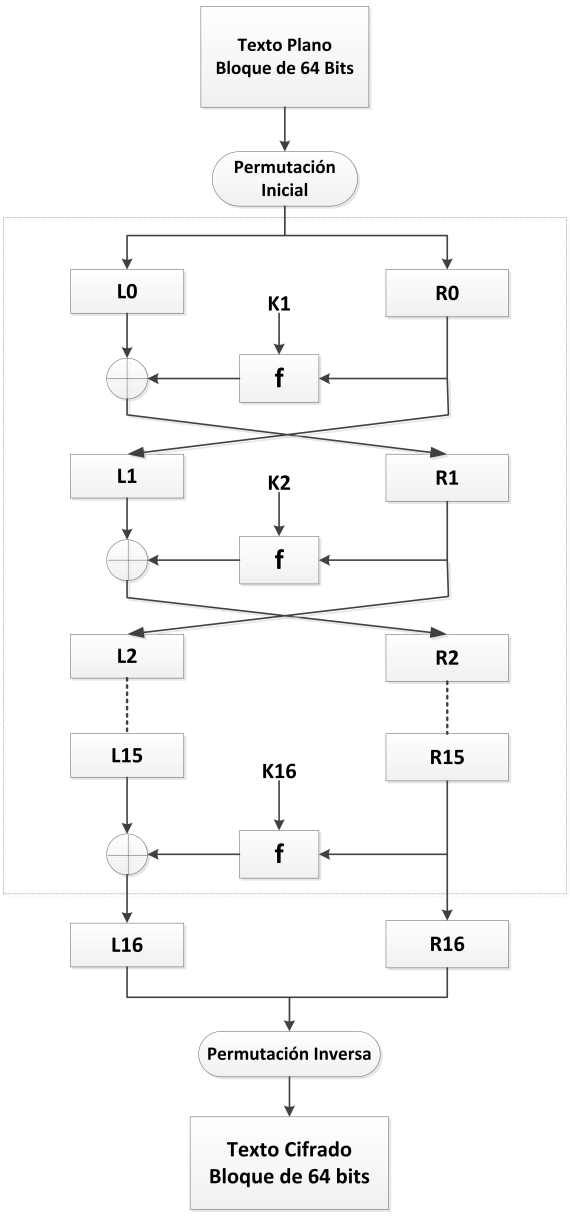


Figura 3.11: Algoritmo DES.

3. Finalmente cada bloque de 6 bits B_j es sustituido por 4 bits de acuerdo a las *S-boxes*[3] de donde se obtienen 8 bloques C_j los cuales se concatenan y se permutan entre si con el objetivo de obtener una cadena de 32 bits, estos bits resultantes son $f(R_{i-1}, K_i)$.

Para cada una de las iteraciones de $f(R_{i-1}, K_i)$ se utiliza una llave K_i diferente, ésta se obtiene de acuerdo al diagrama mostrado en la figura 3.13.

3.3.3. Advance Encryption Standart *AES*

El algoritmo AES fue creado en el año de 1998, gracias a una convocatoria por parte del NIST que tenía como finalidad reemplazar el algoritmo DES como estándar de cifrado, entre los requisitos necesarios para el desarrollo del algoritmo se destacaba el uso de llaves de 128, 192 y 256 bits, el algoritmo debería operar con bloques de entrada de 128 bits y debería ser capaz de ser ejecutado en diferentes tipos de plataformas, desde procesadores de 8 bits hasta

[3]Componente básico de los algoritmos de cifrado de clave simétrica. En general es vista como una tabla de búsqueda donde se toma un número m de bits y lo transforma en m bits de salida.

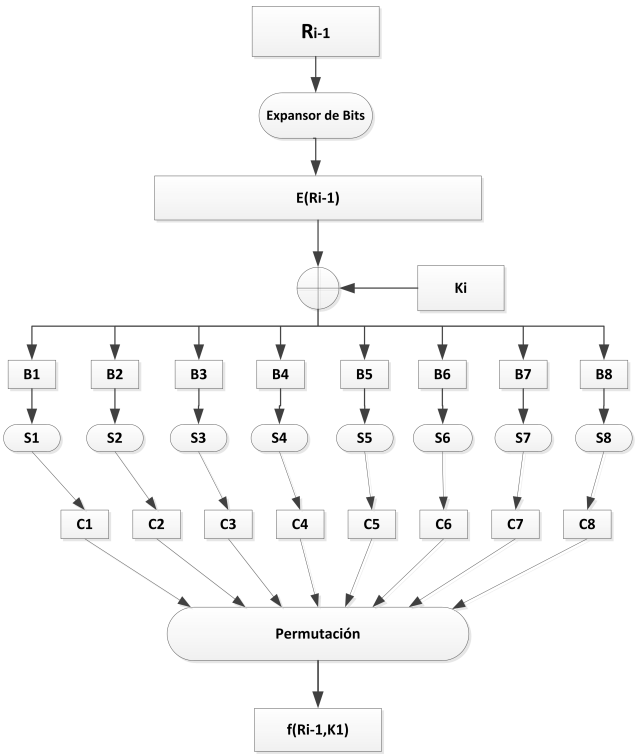


Figura 3.12: Función DES.

procesadores de 32 bits como los computadores de uso personal[22].

AES es un algoritmo de cifrado por bloques de 128 bits, la longitud de la llave (K) puede tomar la longitud de 128, 192 y 256 bits, en cada caso el algoritmo se compone de 10, 12 y 14 rondas respectivamente. Cada ronda se compone de 4 pasos llamados capas como se muestra en la figura 3.14.

Finalmente si se juntan las 10 rondas para el caso donde K es de 128 bits el algoritmo general de AES se conforma de acuerdo a la figura 3.15:

Para cada una de las rondas AES existe una llave derivada de la ronda de llave, en donde la ronda 0 pertenece a la llave original. La entrada del texto plano *(128 bits)* es agrupada en 16 bytes y se les denomina $a_{0,0}, a_{1,0}, a_{2,0}, a_{3,0}, a_{0,1}, ..., a_{3,3}$. Con ello se genera una matriz de 4x4 de la siguiente manera:

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} \tag{3.3}$$

Una vez obtenida la matriz se procederá a explicar cada una de las capas de las rondas AES:

1. **AddRound Key.**

   Consiste en generar una nueva matriz de 4x4, resultante de la operación XOR entre la matriz anterior o la matriz original del texto plano (solo en el primer caso) y la matriz de llaves correspondientes a la ronda de llaves en la que se encuentre el proceso. Es decir:

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} \bigoplus \begin{pmatrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \end{pmatrix} \tag{3.4}$$

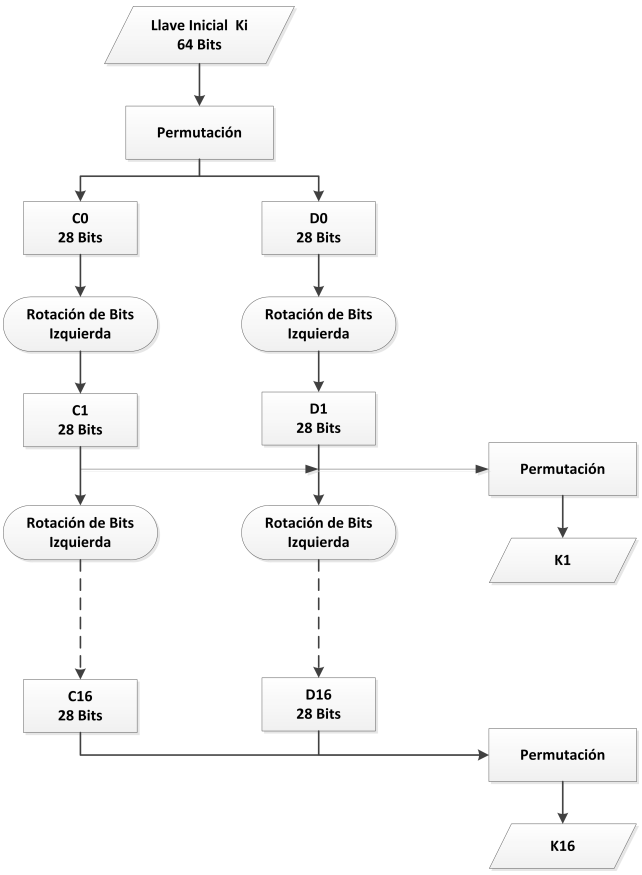


Figura 3.13: Generados de llaves DES.

Figura 3.14: Ronda AES.

$$= \begin{pmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{pmatrix} \tag{3.5}$$

2. **ByteSub.**

   En este paso cada uno de los bytes en la matriz es reemplazado por otro byte de acuerdo a una tabla a la cual se le denomina *S-Box*. La matriz resultante se obtendrá a partir del siguiente esquema: se considera un byte (8 bits) como $b_{0,0} = abcdefgh$, se busca en la tabla *S-Box* el valor correspondiente a la fila $abcd$ y columna $efgh$,(se considera la numeración de 0 a 15) el valor decimal convertido a binario será el valor resultante para el byte $b_{0,0}$.

3. **ShiftRow.**

   Se realizan corrimientos de bytes a la izquierda de la siguiente manera, la primera fila queda original, la segunda fila se recorre un byte a la izquierda, la tercera fila se recorre 2 bytes a la izquierda y la cuarta fila se recorre 3 bytes a la izquierda quedando la matriz resultante de la siguiente manera:

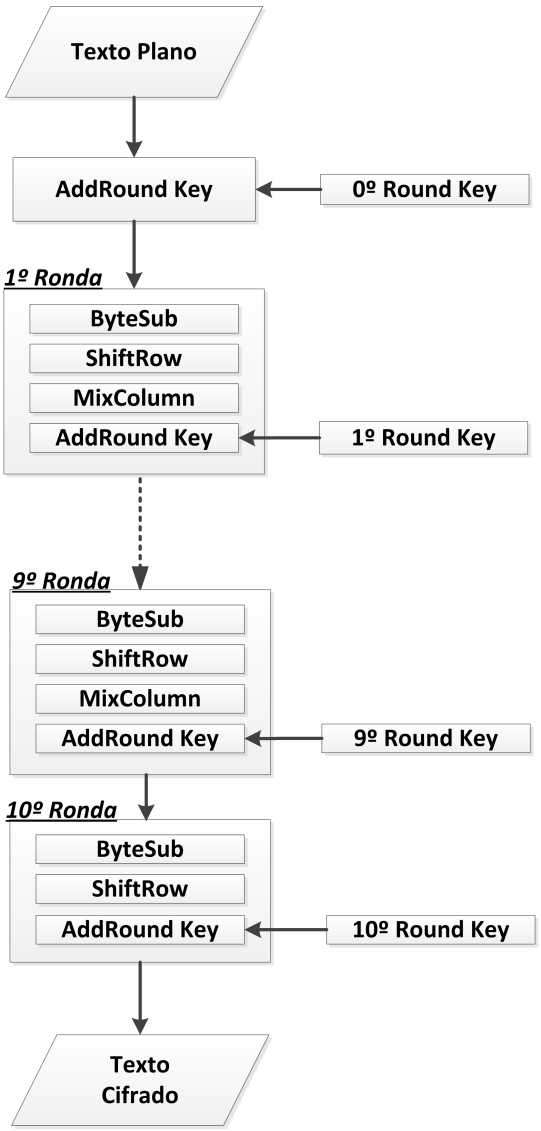


Figura 3.15: Algoritmo AES.

$$\begin{pmatrix} c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} \\ c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3} \end{pmatrix} = \begin{pmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,1} & b_{1,2} & b_{1,3} & b_{1,0} \\ b_{2,2} & b_{2,3} & b_{2,0} & b_{2,1} \\ b_{3,3} & b_{3,0} & b_{3,1} & b_{3,2} \end{pmatrix} \tag{3.6}$$

4. **MixColumn.**

   Se considera a un byte como un elemento del campo finito $GF(2^8)$, a partir de ello se realiza la operación multiplicación módulo (x^4+1) por el polinomio fijo $a(x)$ dado por la siguiente expresión:

$$a(x) = 3x^3 + x^2 + x + 2 \tag{3.7}$$

   El resultado será una nueva matriz de 4x4 como se observa en la figura 3.16:

Para la última ronda se lleva a cabo únicamente los pasos ByteSub, ShiftRow y AddRound Key de donde se obtiene el texto cifrado.

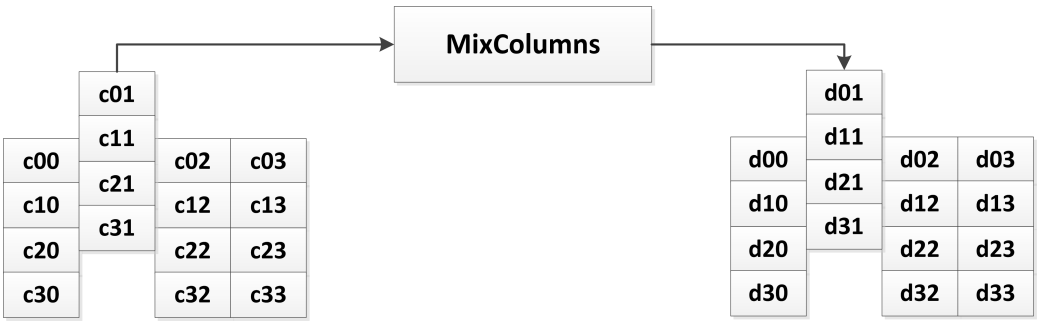


Figura 3.16: MixColumn.

Para cada una de las rondas AES como se aprecia en la figura 3.15, existe una llave derivada de una ronda de llave o planificador de llaves. La llave original que consta de 128 bits es expresada de igual manera en una matriz de 4x4 bytes, que se expande 40 columnas más de la siguiente manera: se nombra a las primeras 4 columnas $W(0), W(1), W(2), W(3)$, las nuevas columnas se generan recursivamente, es decir las columnas se definen por $W(i-1)$ y a partir de ello se establece la siguiente relación:

Si i no es múltiplo de 4:

$$W(i) = W(i-4) \bigoplus W(i-1) \tag{3.8}$$

Si i es múltiplo de 4 entonces:

$$W(i) = W(i-4) \bigoplus T(W(i-1)) \tag{3.9}$$

Donde $T(W(i-1))$ es una transformación de $W(i-1)$ que se obtiene de la siguiente manera. Se considera a los elementos de $W(i-1)$ como a, b, c, d, estos elementos se intercambian y se obtiene b, c, d, a, a continuación se reemplazan los bytes con el elemento correspondiente a la S-Box que se empleó en el paso *ByteSub* del algoritmo AES, se obtienen los bytes e, f, g, h y finalmente se calcula una constante de ronda:

$$r(i) = 00000010^{i-4/4} \tag{3.10}$$

$T(W(i-1))$ será la columna vector compuesta por 3.11.

$$(e \bigoplus r(i), f, g, h \tag{3.11}$$

3.3.4. Modo de Operación CTR

Tanto los algoritmos de cifrado AES y DES se describen como algoritmos de cifrado por bloques, para el caso de AES un bloque de texto plano de 128 bits es transformado a un bloque de texto cifrado, de esta manera los algoritmos de cifrado por bloques pueden ser ejecutados de diferentes modos de operación con la finalidad de garantizar diferentes grados de confidencialidad e integridad.

El modo de operación CTR *(Counter Mode)* por sus siglas en inglés es un modo que proporciona confidencialidad, se conforma por bloques de entrada al cifrador llamados *contadores*, los cuales producen una secuencia de bloques de salida que a continuación realizan la operación XOR con el texto plano para producir el texto cifrado y viceversa[14]. La secuencia de los contadores deberá tener la propiedad de que cada bloque de entrada es diferente a los demás bloques, bajo esta recomendación los contadores para un mensaje cualquiera se denotan

como $T_1, T_2, ..., T_n$ de esta manera el modo CTR se define como:

CTR (Cifrado):

$$O_j = CIFR_k(T_j) \qquad forj = 1, 2, ..., n; \tag{3.12}$$

$$C_j = P_j \bigoplus O_j \qquad forj = 1, 2, ..., n-1; \tag{3.13}$$

CTR (Descifrado):

$$O_j = CIFR_k(T_j) \qquad forj = 1, 2, ..., n; \tag{3.14}$$

$$P_j = C_j \bigoplus O_j \qquad forj = 1, 2, ..., n-1; \tag{3.15}$$

En el caso donde se cifra, de la ecuación 3.12 la función $CIFR_k$ es invocada por cada bloque contador y a la salida que es un bloque de datos se le realiza la operación XOR con el correspondiente bloque de datos de texto plano que finalmente producirá el texto cifrado. De igual manera para el caso de la ecuación 3.14 la función de cifrado $CIFR_k$ es invocada por cada bloque contador y a la salida se le realiza la operación XOR con el correspondiente bloque de datos de texto cifrado con la finalidad de recuperar el texto plano. El modo de operación CTR se ilustra en la figura 3.17.

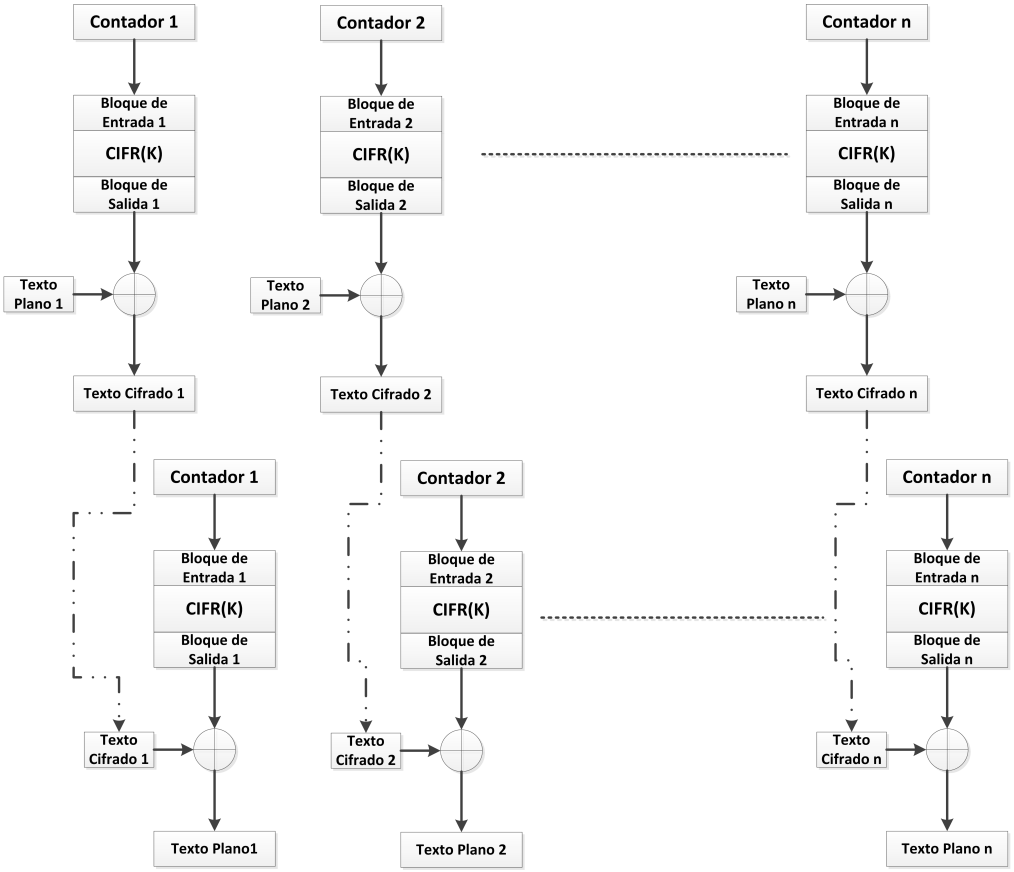


Figura 3.17: Modo de Operación CTR.

3.4. Resumen de contenido.

Los diferentes codificadores analizados en este capítulo brindan las especificaciones necesarias a fin de seleccionar el más adecuado. Entre los codificadores con mejor calidad auditiva se encuentran el códec G.711 y el códec G.729, sin embargo, este último requiere la adquisición de una licencia para su uso.

El algoritmo de cifrado AES se ha convertido en un estándar para los esquemas de cifrado que actualmente se usan, esto debido a un proceso de estandarización que duró 5 años. Es el algoritmo de criptografía simétrica más popular y existen diferentes modos de operación, de entre los cuales destaca el modo contador, pues este modo sólo hace uso de la rutina de cifrado para cifrar y descifrar los datos que se desean proteger, adicionalmente, permite que la entrada de un bloque que se desea cifrar no dependa de la salida cifrada del bloque anterior, lo cual es muy útil al momento de operar bajo redes que son propensas a la pérdida de paquetes.

Puntualizados estos conceptos, el siguiente capítulo aborda el diseño del esquema de cifrado y los detalles que llevaron a la construcción de un prototipo capaz de hacer uso del esquema de cifrado propuesto.

Capítulo 4

Diseño del esquema de cifrado.

En este capítulo se detallan las características consideradas al momento de realizar el diseño del esquema de cifrado. Se introduce retomando lo requisitos necesarios para el diseño y posteriormente se realiza el diseño del esquema, una vez que se cuenta con éste se procede a detallar la manera en que realiza la construcción de un prototipo capaz de evaluar el desempeño del diseño del esquema, además de que se mencionan los inconvenientes que se tuvieron al momento de realizar esta prueba de concepto.

4.1. Requisitos para el Diseño.

En base a los objetivos presentados al inicio de este proyecto, se listan a continuación los requisitos necesarios para alcanzar los objetivos, lo cual dará una pauta del porqué se escogieron tales elementos del diseño y la manera en que estos fueron constituidos.

- El diseño del esquema de cifrado no deberá inyectar retardos de tiempo considerables a las llamadas de VoIP.
- La inteligibilidad del audio no debe ser afectada por el esquema de cifrado.
- El rendimiento del hardware deberá ser optimizado para obtener el menos costo computacional al momento de realizar el cifrado.
- El esquema de cifrado deberá estar basado en el algoritmo de cifrado AES, pues es un algoritmo de cifrado que ha demostrado ser seguro después de un proceso de estandarización que duró 5 años por parte del Instituto Nacional de Estándares y Tecnología (NIST).
- Se requiere de una aplicación de VoIP que opere en el sistema operativo Android y que los protocolos de comunicación en los que se basa puedan ser modificados.
- Es necesario contar con una plataforma de desarrollo que cuente con un hardware similar al que se puede obtener hoy en día en un dispositivo móvil de gama media.

Básicamente se desea contar con un esquema de cifrado de VoIP con el que un usuario sea capaz de realizar llamadas con la característica de proveer confidencialidad a la información que éste transmite. Cuando el usuario llame a través de la aplicación que hace uso del esquema de cifrado propuesto, solamente el receptor autorizado podrá entender el flujo de datos de audio que se le envían, haciendo uso de igual manera el esquema de descifrado en su caso.

En las siguientes secciones se describen los puntos necesarios para cumplir con los requisitos

de diseño y los objetivos general y particulares que se establecieron al iniciar este proyecto, posteriormente se realiza la construcción de un prototipo que hace uso del esquema de cifrado propuesto y se describe cada una de las etapas que se siguieron para llegar al esquema final.

4.2. Selección de la aplicación y del codec para VoIP.

Las aplicaciones existentes para VoIP descritas en el capitulo 2 nos proporcionan un panorama de las funcionalidades que de ellas podemos obtener, sin embargo al momento de realizar un desarrollo o modificación de sus protocolos de comunicación es necesario tomar en cuenta tanto los protocolos que emplea así como el código de programación en el que fue escrito.

Tabla 4.1: Resumen de Aplicaciones empleados en VoIP.

Nombre	Características
Bria Android	Basada en el protocolo SIP, codecs g.711 a/u, g.722, gsm, iLBC y silk. Aplicación de paga. No es posible modificar los protocolos de comunicación.
CSIPSimple	Basada en el protocolo SIP, codecs g.711 a/u, gsm, iLBC, g.729, g.722, AMR, Silk y g.726. Aplicación de código abierto escrita en Java y C. Amplia documentación y soporte acerca del proyecto.
LinPhone	Basada en el protocolo SIP, codecs g.711 a/u, gsm, g.729 y g.722 .Aplicación de código abierto escrita en Java y C. Poca o nula documentación y soporte acerca del proyecto.

A partir del estudio realizado se analizó la tabla 4.1 y se decidió seleccionar la aplicación *CSIPSimple* cuyas características principales son las siguientes:

- Integración con el sistema operativo Android.
- Desarrollada en el lenguaje de programación Java.
- Protocolos de comunicación implementados en C/C++ bajo la librería de software abierto ***PJSIP***.
- Codecs Soportados: PCM U/A (g.711), speex, g.722, gsm, g.729,g.726.

Esta aplicación ademas de permitir al desarrollador la modificación tanto de la interfaz como de sus protocolos, lo provee con una gran documentación acerca de la programación y los requerimientos necesarios para poder llevar a cabo integraciones y desarrollos bajo su plataforma.

El uso de esta aplicación conlleva al análisis de la librería ***PJSIP***. Es una plataforma de comunicaciones multimedia escrita en el lenguaje C que implementa los protocolos base como SIP, SDP, RTP, STUN, TURN y ICE. PJSIP hace posible que sus aplicaciones multimedia sean implementadas en cualquier tipo de procesador y sistema operativo.

En general CSIPSimple puede ser ejecutado en cualquier dispositivo que cuente con el sistema operativo Android, mientras que la pila de protocolos que lo soporta puede ser ejecutado

bajo cualquier tipo de sistema, en este caso PJSIP soporta a CSIPSimple y al permitir que se le realice cualquier modificación en su programación automáticamente CSIPSimple se ve afectado por dicho cambio, en otras palabras solo es necesario modificar la programación realizada en PJSIP para que la aplicación en Android funcione de acuerdo a los nuevos cambios.

La elección del codificador de voz se basó en el estudio del marco conceptual, de donde se obtuvo la tabla 4.2:

Tabla 4.2: Resumen de Códecs empleados en VoIP.

Nombre	Bit Rate	Sampling Rate	Frame Size	MOS
G.711	64 Kb/s	8 KHz	20ms	4.3
G.722	64 Kb/s	16 KHz	20ms	4.5
G.723	6.3 Kb/s	8 Khz	30ms	3.65
G.729	8 Kb/s	8 KHz	10ms	4.2
GSM	13 Kb/s	8 KHz	22.5 ms	3.7
Speex	8-32 Kb/s	2-44 Kbps	30-34 ms	3.67

En ella se describen brevemente las características mas importantes de los códecs, se indican los siguientes parámetros:

- ***Bit Rate.*** Indica la cantidad de información que se manda por segundo.
- ***Sampling Rate.*** Brinda la frecuencia de muestreo de la señal,(cada cuanto se toma una muestra de la señal analógica).
- ***Frame Size.*** Proporciona el intervalo de milisegundos en el que se enviá un paquete con la información sonora.
- ***MOS.*** *"Mean Opinion Score"*. Mide la calidad de percepción del audio después de ser comprimida por un codec en particular, la puntuación asignada es dada por un grupo de escuchas que emplearon el procedimiento especificado en el estándar de la IUT-T P.800 y P.830. El rango va de 5 (calidad de audio excelente) a 1 (calidad de audio mala) [1].

A partir del análisis realizado se concluyó en emplear el codec g.711, el cual se rige bajo el esquema PCM, su algoritmo de codificación no presenta complejidad y puede ser ejecutado tanto en sistemas de cómputo convencionales como en sistemas embebidos y sistemas móviles.

Posee una calificación MOS de 4.3 y a pesar de que no es un codec que comprime los paquetes de audio ni tiene la mejor calificación a diferencia del codec g.729 y g.722 respectivamente, el codec g711 es un codec de licencia abierta que permite su implementación en cualquier sistema de voz.

4.3. Diseño del esquema de cifrado.

Con la finalidad de diseñar un esquema de cifrado primero es necesario conocer los elementos de hardware con los que se cuenta y para ello se seleccionó una plataforma de desarrollo que proporcionara los mismos elementos de hardware que hoy en día un dispositivo móvil puede proveer.

La plataforma de desarrollo que se empleó para este diseño fue una tarjeta *BeagleBoard-xm Rev C* que se aprecia en la figura 4.1 fabricada por la compañia Texas Instruments y cuyas características se describen en la tabla 4.3.

Tabla 4.3: Especificaciónes de hardware BeagleBoard-Xm

	Características
Procesador	Texas Instruments DM3730. Procesador de Propósito General *(GPP)* ARM Cortex A8 1GHz y Procesador Digital de Señales *(DSP)* 800MHz
Memoria	SDRAM (512 MB) 200MHz
TPS65950	Regulador de fuente de alimentación, Codecs de Audio y Reset.
Ethernet	10/100 Desde USB HUB
Conectores de Audio	3.5 mm Salida y Entrada Estéreo.
Fuente de Alimentación	USB o Corriente Directa 5V 1.5 Amp
Conexiones Adicionales	Conector RS232, microSD, Interfaz de video DVI-D, S-Video.

Una de las ventajas de emplear una tarjeta de desarrollo es que se puede hacer uso no solo del procesador de propósito general con el se cuenta, sino que además se pueden ejecutar procesos en el procesador digital de señales que éste provee. En esta plataforma se puede llevar a cabo una experimentación con mayor libertad de lo que se haría con un dispositivo móvil, pues la arquitectura de cómputo de estos dispositivos es propia del fabricante.

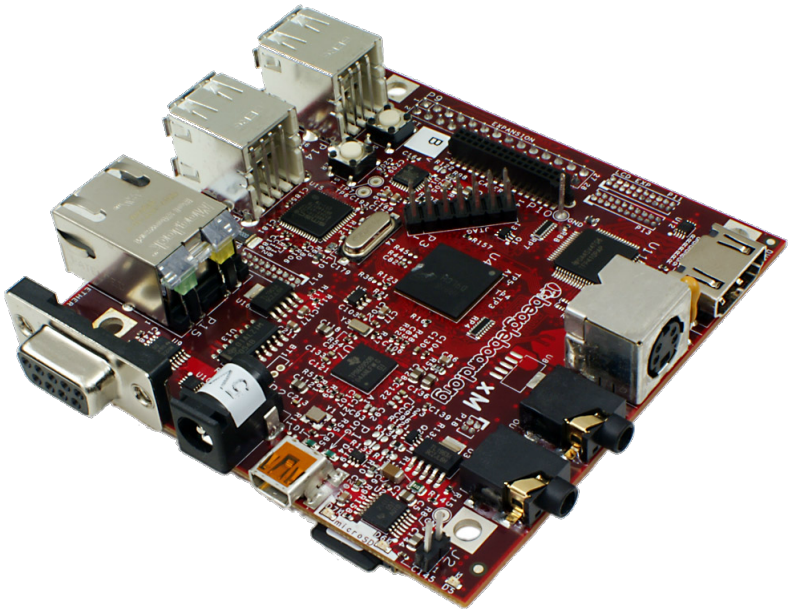


Figura 4.1: Tarjeta de Desarrollo BeagleBoard-Xm Rev C.

Los recursos que nos brinda la tarjeta de desarrollo BeagleBoard-Xm permiten que el diseño del esquema de cifrado contemple el uso del procesador digital de señales para realizar una propuesta del esquema de cifrado que permita cumplir con los objetivos establecidos.

En base a ello se realiza la siguiente propuesta de solución. En la tarjeta de desarrollo se habilitará el sistema operativo Android que a su vez será capaz de ejecutar la aplicación de VoIP CSIPSimple, la modificación de los protocolos de comunicación se llevarán a cabo en

el framework multimedia que provee PJSIP y a través del cual se cifrará el flujo de datos de VoIP deseado, para ello el cifrador será programado en el DSP con la finalidad de hacer un uso más eficiente los recursos con los que se cuenta en la tarjeta, dicho procedimiento se puede dar gracias a que el S.O. Android se basa en linux y con ello se pueden realizar llamas a sistema que permitan la ejecución de tareas en el DSP.

La figura 4.2 hace muestra del esquema propuesto.

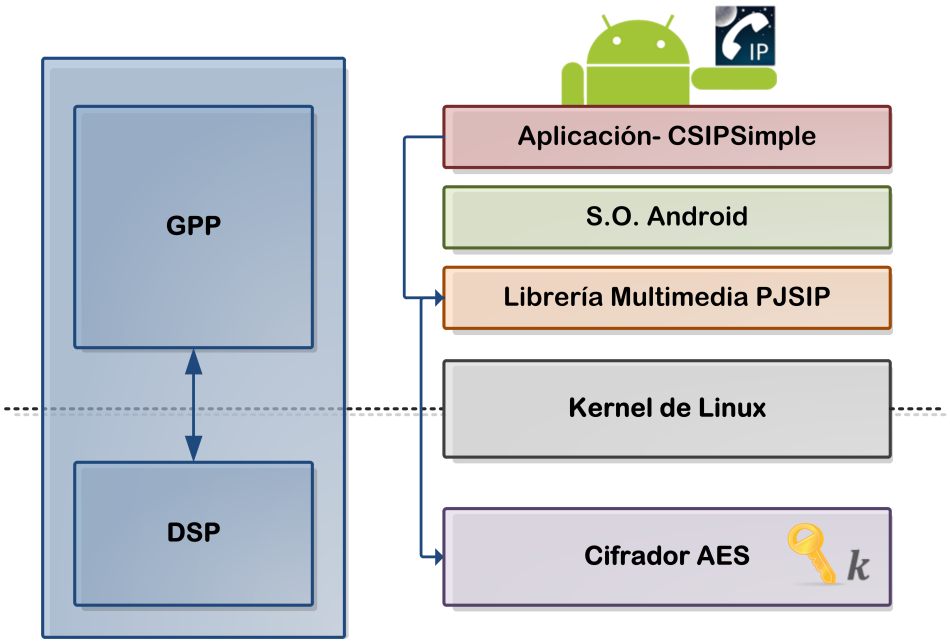


Figura 4.2: Esquema de cifrado propuesto.

Se contempla que la transmisión de los datos será a través de la interfaz ethernet de la cual nos provee la tarjeta de desarrollo, por el momento el proyecto no contempla el llevar este esquema hacia un red inalámbrica o una red de datos móviles como la que provee cualquier compañía de telefonía celular.

Para conocer el flujo de datos que se desea cifrar, es preciso realizar un análisis de la manera en que PJSIP, el framework multimedia de la aplicación realiza tanto la codificación como el transporte de lo datos. Además de una revisión a la documentación, es necesario realizar una investigación del código de programación, el diagrama de flujo presentado en la figura 4.3 nos proporciona un panorama general del flujo de datos multimedia que se sigue con dicha librería.

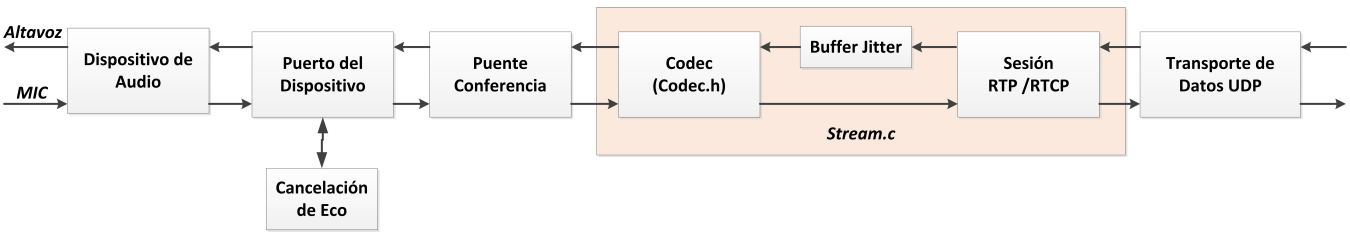


Figura 4.3: Diagrama del flujo de datos en PJSIP.

El framework multimedia hace uso de los drivers de sonido para habilitar tanto el micrófono como los altavoces, a partir de ahí los conduce hacia un puerto de sonido quien es el encargado de realizar la abstracción del audio, es decir es el medio de comunicación entre los drivers de sonido y las funciones propias que permiten que se capture o reproduzca el audio. El puente de conferencia es el encargado de proporcionar la ruta que seguirá el flujo de audio.

La siguiente etapa que se muestra en la figura 4.3 encapsulada por *"stream.c"* es la etapa de mayor interés para realizar el esquema de cifrado. En ella se identifican las siguientes etapas:

- **Codec.** En esta etapa se lleva a cabo la codificación y decodificación de la señal de audio acorde al codec previamente seleccionado por el usuario. Cada paquete o frame de datos se le proporciona a la siguiente etapa, para el caso donde el flujo de audio es enviado, el frame se proporciona a la sesión RTP/RTCP y se compone de 160 bytes. En el caso de la recepción de datos el frame es recibido de la etapa Jitter Buffer con un tamaño de frame de 80 bytes, es decir se recibe la mitad de datos que se transmite.
- **Sesión RTP/RTCP.** Esta sesión es la encargada de añadir la cabecera RTP al frame de datos recibido por el codec, posteriormente se envía a UDP a fin de que el paquete sea transmitido a la red, en la parte de la recepción quita la cabecera RTP y envía la carga útil a la etapa del códec para que las muestras de audio sean decodificadas y reproducidas. Un frame de RTP se compone 172 bytes, 12 bytes que forman la cabecera RTP y 160 bytes de carga útil.
- **Jitter Buffer.** Esta etapa es exclusiva de la recepción de datos y sirve de almacén para la recepción de paquetes con la finalidad de eliminar la variación en el tiempo de la llegada de los datos, si algún paquete no se encuentra en el buffer cuando sea necesario, este se descarta.

Con los datos proporcionados, como se ve en la figura 4.4, el diseño se enfoca hacia cifrar y descifrar el flujo de datos de audio en la sesión RTP, que es la parte donde los datos son de la misma longitud tanto en transmisión como en recepción (160 bytes).

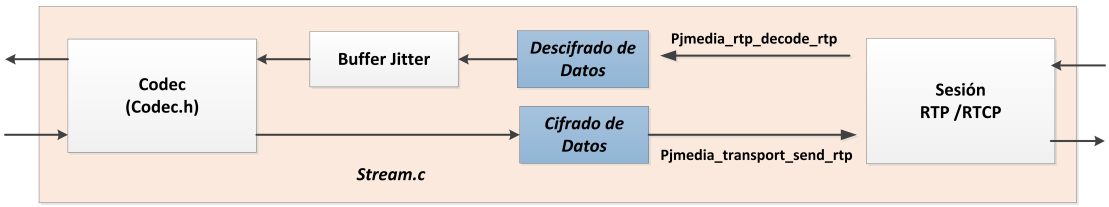


Figura 4.4: Etapa de cifrado/descifrado.

La transmisión de los datos a la red se realiza haciendo uso del protocolo UDP, pues a pesar de que UDP no ofrece integridad de los datos y no contiene confirmación de entrega ni control de flujo, el aprovechamiento del ancho de banda es mayor que con TCP. En el protocolo UDP la pérdida de un paquete resultaría en un instante de silencio en la conversación mientas que en TCP la pérdida de un paquete resultaría en la retransmisión de dicho paquete.

Bajo estas condiciones se decidió hacer uso del algoritmo de cifrado AES en modo contador *(CTR)* del cual se realizó un estudio en el capítulo 3, una de las ventajas que este modo de operación nos proporciona, es que sin un paquete de datos se pierde en la red, los siguientes paquetes aún podrán ser descifrados por el receptor, es decir la salida de un paquete de descifrado no depende del paquete anterior como lo es en los modos de operación restantes del

algoritmo AES.

Los requerimientos que son necesarios para hacer uso de este modo de operación es que se necesita de una señal de sincronía que mantenga el contador, tanto del cifrado como del descifrado en concordancia y así se puedan recobrar los mismos datos de texto plano que fueron enviados por el transmisor. Para ello se realizó un análisis de la cabecera RTP.

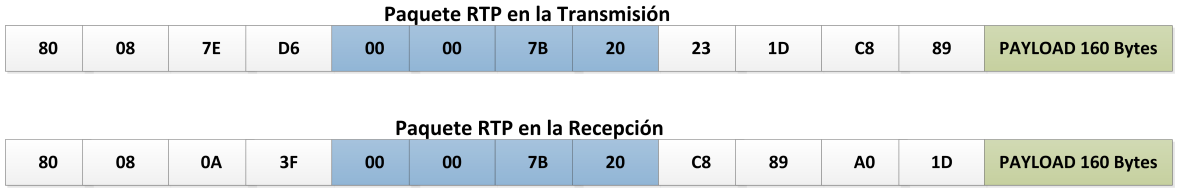


Figura 4.5: Cabecera RTP en Tx y Rx.

Como se ilustra en la figura 4.5 los valores que permanecen íntegros desde que se generan en la transmisión hasta recibirlos en el receptor son los bytes que pertenecen al timestamp. Estos bytes son los que servirán como señal de sincronía para el esquema de cifrado propuesto y que además servirán como los bytes *"nonce"* para el modo contador, quedando de la siguiente manera:

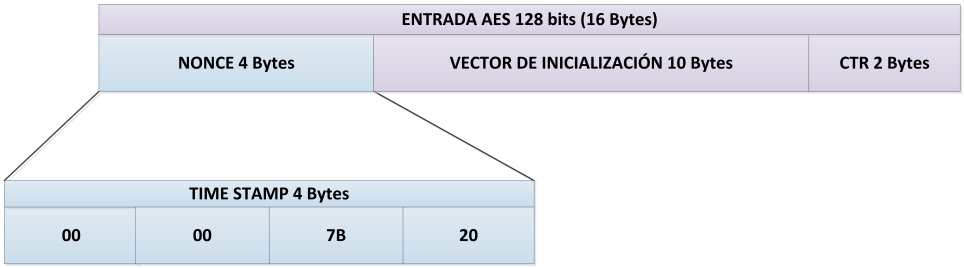


Figura 4.6: Integración del CTR.

Los 4 bytes del campo timestamp serán obtenidos de la cabecera RTP y serán colocados en los bits más significativos del vector de entrada AES(figura 4.6), como se estudió en el capítulo 5 la entrada y salida de AES es de 128 bits, es por ello que para poder conformar los 160 bytes que pertenecen al tamaño de un frame de datos de audio es necesario realizar la ejecución de AES 10 veces. Es decir, por cada frame de audio que se obtenga se realizaran 10 ejecuciones de cifrado AES lo cual nos brinda una matriz de datos proporcional al tamaño del frame de audio para finalmente realizar la operación XOR y obtener un paquete de datos cifrados como se muestra en la figura 4.7.

Un detalle a considerar al momento del diseño es conocer la cantidad de información que se puede procesar en el DSP, pues lo ideal sería realizar todo el proceso de cifrado en el DSP con la finalidad de quitar carga al GPP y aumentar el rendimiento de la arquitectura con la que se cuenta, además de reducir el tiempo de procesamiento. Sin embargo el DSP solo es capaz de procesar datos a través del controlador de cache que recibe cargas de 128 Bytes y las procesa para después recibir una nueva carga de datos del mismo tamaño[4].

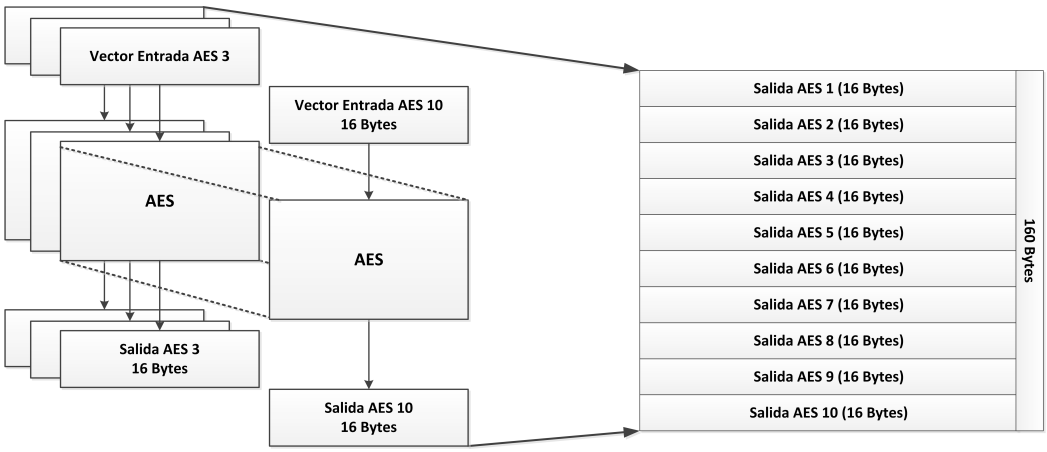


Figura 4.7: Ejecución AES.

Esto resulta en que el diseño del esquema de cifrado tenga que segmentarse, es decir los 160 bytes no pueden ser cargados en el DSP, es por ello que la operación XOR del modo de operación CTR tenga que ser realizada en el GPP, mientras que en el DSP solamente se realizaran las 10 ejecuciones AES y se obtendrá la matriz de 160 bytes.

Dado lo anterior el diseño del esquema de cifrado es el que se muestra en la figura 4.8

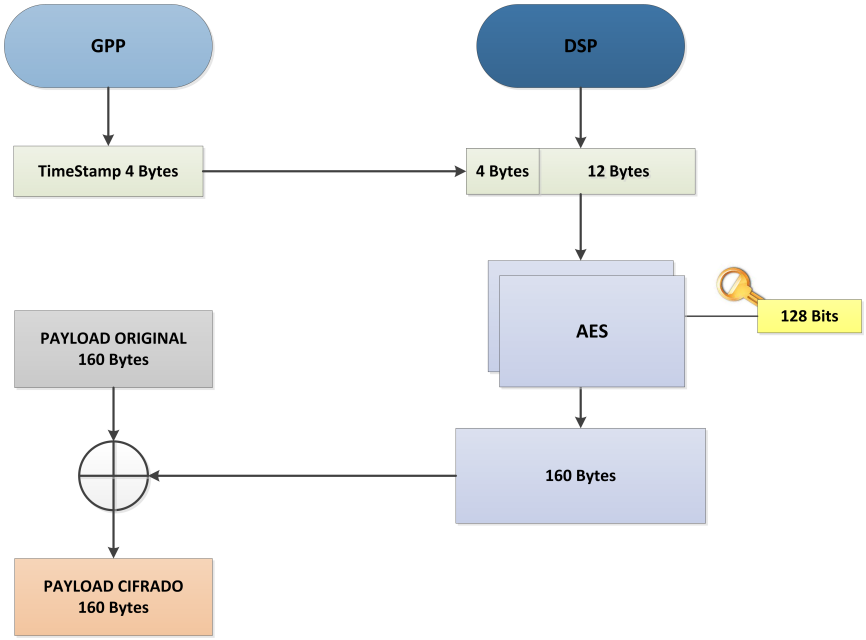


Figura 4.8: Esquema de Cifrado.

En resumen el paquete RTP se obtiene antes de ser enviado a la red de datos, se extrae la carga útil (Payload Original) y se cifra para volver a ser empaquetado por RTP. Para el caso donde se requiere descifrar la información se sigue el mismo procedimiento y se obtendrá la carga útil original.

La llave de cifrado que se ha seleccionado ha sido la de 128 bits (16 Bytes), lo cual proporciona un menor coste computacional, incluso cuando éste sea ejecutado desde el DSP el utilizar una llave de mayor tamaño resultaría en la generación de un tiempo de retardo mayor en una conversación.

4.4. Construcción del prototipo.

El primer paso para poder habilitar la plataforma de desarrollo a fin de evaluar el esquema de cifrado propuesto es cumplir con los requisitos necesarios para poder hacer uso de la aplicación CSIPSimple y poder modificar sus protocolos de comunicación. A continuación se listan las dependencias necesarias y la configuración del equipo a fin de poder realizar la prueba de concepto.

- S.O. Ubuntu 12.04 LTS
- Las siguientes dependencias instaladas: subversion, git, quilt, unzip, wget, swig2.0, python, make y yasm
- Instalar Android NDK y SDK.
- Descargar la applicación para desarrolladores CSIPSimple a tráves del siguiente comando.

```
$ svn checkout http://csipsimple.googlecode.com/svn/trunk/ CSipSimple-trunk
```

El SDK *(Software Development Kit)* es un kit de desarrollo de software con el cual se pueden desarrollar aplicaciones para el S.O Android, el NDK *(Native Development Kit)* es un conjunto de herramientas que permiten embeber código compilado en lenguaje C o C++ en una aplicación de Android escrita en el lenguaje Java.

La aplicación CSIPSimple hace uso de la *Interfaz Nativa de Java (JNI)* por sus siglas en inglés, la cual permite que la librería PJSIP pueda ser empleada por la aplicación CSIPSimple, mantenga su estructura y el lenguaje en el que fue escrito y que además permita la modificación de los protocolos de comunicación en la misma aplicación.

Adicionalmente se requiere realizar la instalación del sistema operativo Android en la tarjeta de desarrollo Beagleboard y habilitar el DSP de la misma. Para llevar a cabo lo anterior son necesarias las siguientes herramientas:

1. Tener instalado Java JDK 6 en un equipo de cómputo.
2. Obtener la imagen de Android 4.0.3 que se proporciona en la página de Texas Instruments[1].
3. Habilitar el DSP de la Beagleboard en el kernel de linux versión 2.3.67.
4. Instalar la imagen en una sdcard mayor de 4 Gb, insertarla en la tarjeta de desarrollo e iniciar el sistema.

A fin de ejecutar los pasos correctos para la instalación del sistema operativo se siguen las recomendaciones dadas por Texas Instruments en su manual de instalación[8], asi mismo se provee en la sección de anexos apartado A, una guía de instalación con la corrección de errores al momento de compilar. Para poder habilitar el DSP en el kernel de linux es necesario ingresar a la carpeta donde se encuentran los recursos del kernel y ejecutar la siguiente instrucción:

```
$ make menuconfig ARCH=arm CROSS_COMPILE=arm-eabi
```

[1]TI Android ICS 4.0.3 DevKit-3.0.0 Developers Guide.
http://processors.wiki.ti.com/index.php/TIAndroidICS-4.0.3DevKit3.0.0_DevelopersGuide

Se activa en el kernel la opción *dspbridge* localizada dentro del menú "drivers", "staging driversçomo se aprecia en la figura 4.9. Con el sistema operativo Android instalado en la tarjeta de desarrollo y el DSP habilitado, es necesario cargar una imagen base que contenga todas las primitivas del DSP/BIOS que permitirán la comunicación básica del GPP con el procesador digital de señales.

A fin ejecutar tareas en el DSP se utilizó un framework llamado *"DSP for Dummies (DFD)"* diseñado para facilitar a los desarrolladores esta tarea, la codificación de este programa se presenta en dos partes: el binario que se ejecuta en el DSP y el binario que se ejecuta en el procesador ARM (GPP). En general si se desea realizar algún trabajo en el DSP se necesita escribir los datos en un buffer de entrada que tomará el DSP y éste devolverá el resultado en un buffer de salida que entregará al GPP, dicho proceso se especifica más adelante.

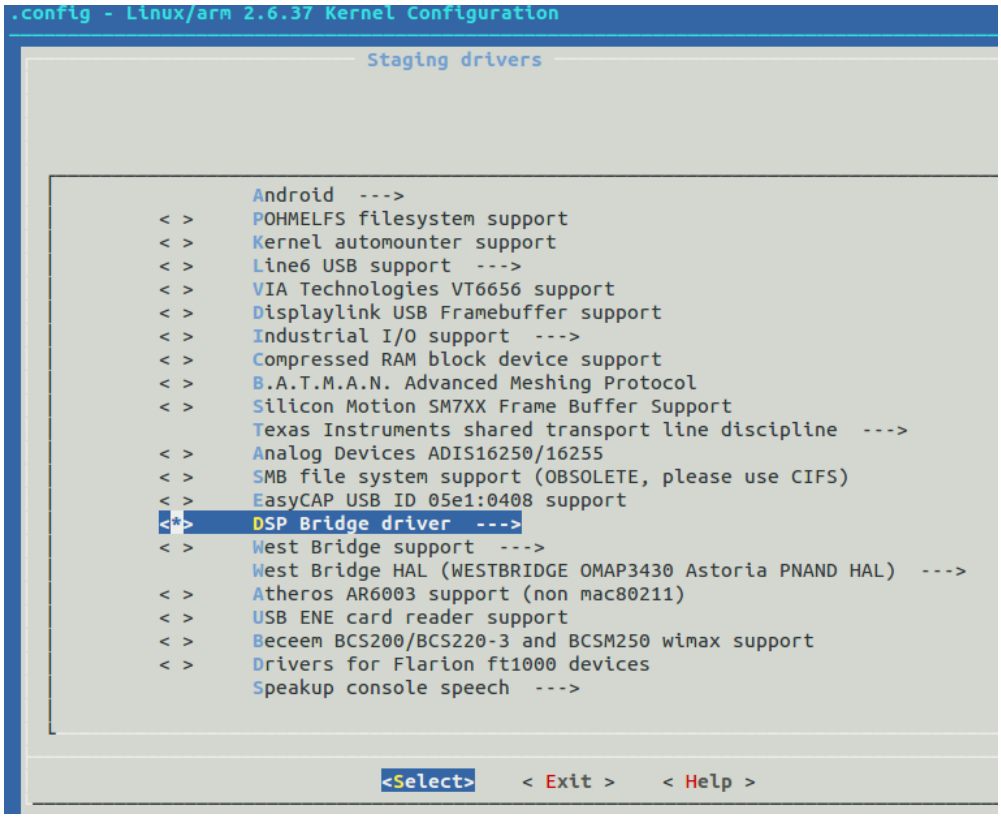


Figura 4.9: Kernel Linux 2.3.67.

Con el entorno de desarrollo en el equipo de cómputo se procede a realizar las modificaciones en los protocolos de comunicación, para ello se localiza el archivo "stream.c" que, como se muestra en la figura 4.10 es el archivo donde se ejecutan los procesos de codificación y decodificación de los paquetes RTP. En él se pueden identificar las siguientes funciones:

```
1 pjmedia_transport_send_rtp (pjmedia_transport *tp,
2                             const void        *pkt,
3                             pj_size_t         size);
4
5 pjmedia_rtp_decode_rtp  (pjmedia_rtp_session  *ses,
6                                        const void        *pkt,
7                                        int               pkt_len,
8                                        const void        **payload,
9                                        unsigned          *payloadlen);
```

La primera de ellas hace posible el envió de los paquetes hacia la red haciendo uso del protocolo UDP, en ella se especifican los parámetros *pkt* y *size*, que definen el paquete de datos a enviar y el tamaño del paquete respectivamente. En la figura 4.10 se observa que el paquete es capturado antes de ingresar a esta función donde pkt se representa por 172 bytes, de él se extrae la carga útil que es almacenada en un buffer de datos temporal de tamaño 160 bytes. Es decir se le quita la cabecera RTP (12 bytes) y de ésta solo se extraen 4 bytes (timestamp) y se asignan a una variable que será transmitida a la entrada del cifrador AES.

La segunda función permite la decodificación de un paquete RTP en la recepción. En ella se especifican los parámetros *payload* y *payloadlen*, que al igual que la función anterior definen los datos del paquete recibido y el tamaño de éste. Dado que el procedimiento para cifrar y descifrar es el mismo para ambos casos nos limitaremos a explicar solo el diseño del esquema de cifrado y se asumirá que el proceso para descifrar el paquete de datos en la recepción es el mismo, solo que éste se lleva a cabo antes de la función de decodificación del paquete RTP.

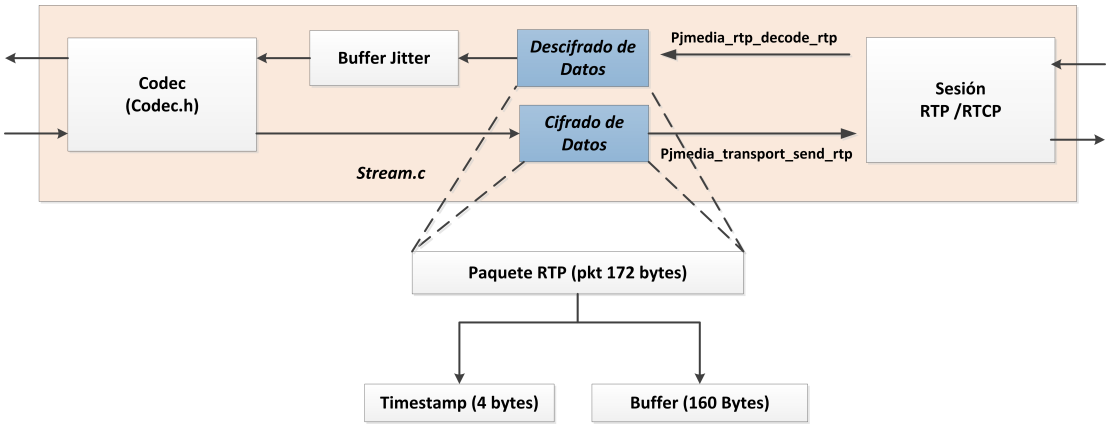


Figura 4.10: Funciones Cifrado/Descifrado.

Una vez que se cuenta con los datos de audio en un buffer y el timestamp, es necesario transmitir este último al DSP. Uno de los inconvenientes al momento de realizar este pase de datos es que no se puede realizar una integración de código del *DFD (DSP for Dummies)* con la librería PJSIP, esto debido a la incompatibilidad del set de instrucciones que se manejan para el envió de procesos al DSP, es decir este set no puede ser compilado y por ende ejecutado por el NDK de Android.

A fin de solucionar este inconveniente se diseñó un medio de comunicación entre ambos binarios, PJSIP y DFD. Para ello se hizo uso de diferentes métodos de IPC (Inter Process Communication) de los cuales se mencionan los siguientes:

- Memoria Compartida (shmget)
- Mapeo de Memoria (mmap)
- Archivo Compartido (fopen)
- Sockets (socket)

Sin embargo el único medio eficiente y funcional fue hacer uso de sockets, pues al momento de hacer uso de alguno de los anteriores, el esquema de cifrado no operaba de manera continua o el proceso de la llamada de voz se veía interrumpido debido a que el tiempo en el que se

generaban los frames de audio era menor al tiempo en el que se realizaba el pase de datos al DSP, lo que resultaba en interrupciones del sistema operativo o reseteos del mismo.

El uso de sockets requiere de un programa intermediario capaz de entregar y recibir los mensajes a ambos procesos, por lo que se diseñó un servidor a base de sockets también. Esta pequeña aplicación "servidor"tiene las siguientes características:

- Opera a base de funciones socket bajo el puerto 5001
- Los sockets son construidos para operar solo sobre el host (AF_UNIX), es decir no se permiten conexiones que no provengan del mismo host.
- Los sockets son orientados a conexión (SOCK_STREAM)
- Siempre está escuchando posibles conexiones de clientes del mismo host.
- Se compilo haciendo uso del cross compiler arm−none−linux−gnueabi−gcc

Como se muestra en la figura 4.11 básicamente se crea una conexión a tráves de (*socket()*) con las características mencionadas anteriormente y espera la primera conexión (*listen()*), que será la proveniente de la aplicación CSIPSimple, una vez que se realiza la petición de conexión el servidor la acepta (*accept()*) y verifica que ésta se haya realizado exitosamente para poder leer el buffer de datos que la aplicación le ha transmitido(*read()*), cierra el socket abierto (*close()*) y acepta una nueva conexión, esta vez proveniente del binario DFD, de la misma manera el servidor acepta la conexión y verifica que se haya realizado exitosamente para esta vez transmitirle el buffer de datos que recibió previamente (*write()*), en esta ocasión el socket permanece abierto hasta que el binario DFD transmite el buffer de datos que desea enviar a la aplicación CSIPSimple, el servidor lee el buffer de datos y en ese momento cierra la conexión.

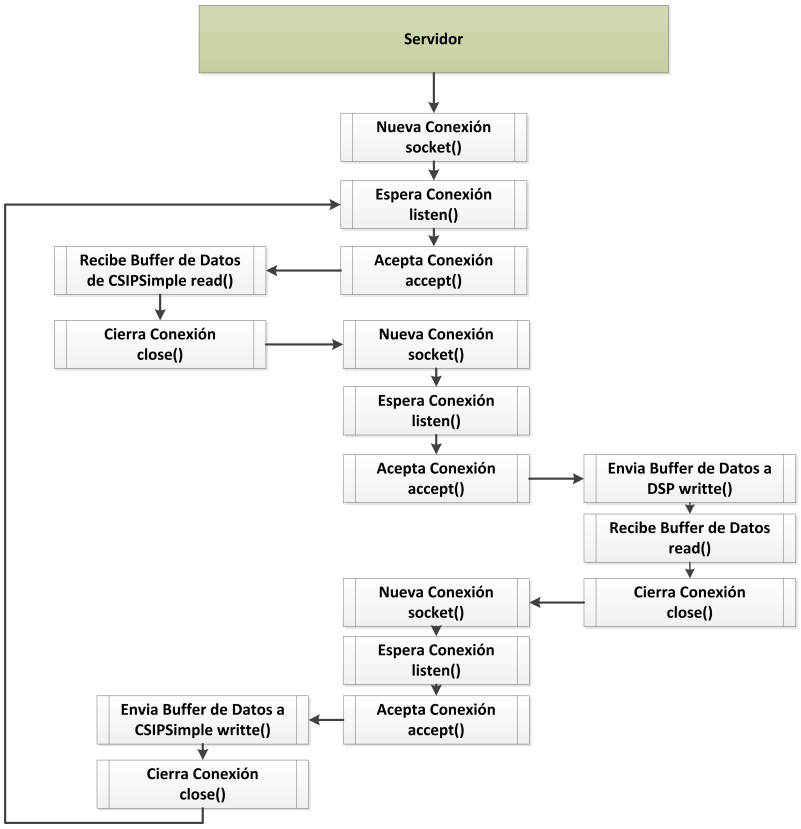


Figura 4.11: Medió de comunicación entre procesos.

Finalmente el servidor espera por una última conexión que provenga de la aplicación CSIPSimple para enviarle el buffer de datos transmitido por el binario DFD y una vez que se haya realizado cierra la conexión, y el servidor espera a que una nueva petición de conexión le sea solicitada.

Con el servidor ejecutándose en segundo plano es posible el pase de datos entre las aplicaciones binarias. Retomando el proceso que se ejecuta en la aplicación CSIPSimple y considerando que se tiene un servidor con el cual se puede realizar el envió de información se realiza lo siguiente. Como se muestra en la figura 4.12 después de haber obtenido el buffer de datos original (160 bytes) y el timestamp (4 bytes), se crea un socket con las mismas características que las del servidor a excepción de que éste no esperará por alguna conexión, sino que se conectará al servidor y enviará los 4 bytes pertenecientes al timestamp.

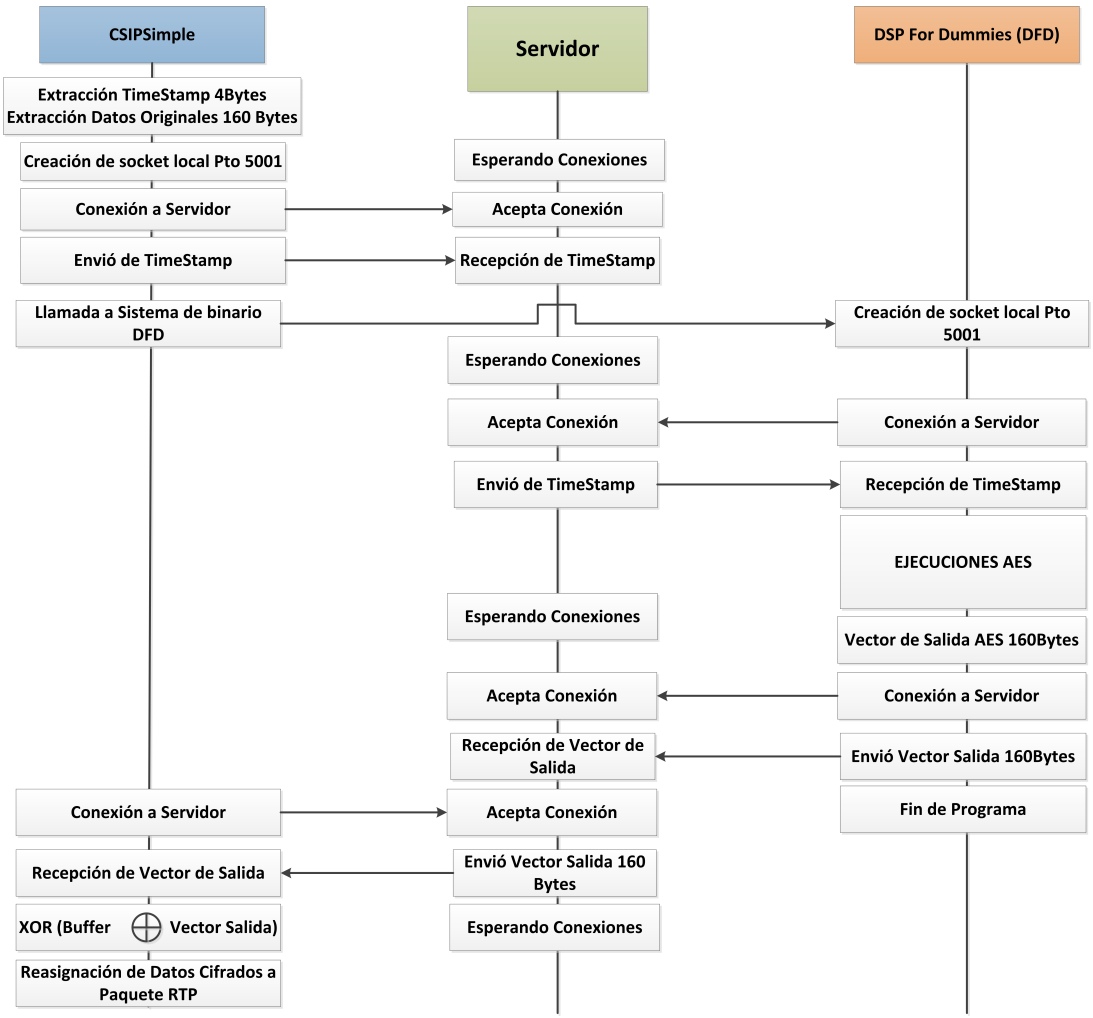


Figura 4.12: Diagrama de Secuencia de Cifrado.

A través del uso de llamadas a sistema y haciendo uso del comando (*system*) se ejecuta el binario generado por el framework DSP for Dummies (DFD). Lo que este comando nos permite es tener la facilidad de ejecutar una ventana de comandos, desde la cual se le dará la instrucción de ejecutar el binario DFD, por ejemplo se establece la ruta donde esté localizada la aplicación binaria y se ejecuta:

```
1 system ("cd /data/data/com.csipsimple/ ; ./dummy");
```

El binario DFD será quien entre sus tareas ejecutará 10 veces el algoritmo AES y producirá a través del binario del servidor el vector de salida de 160 bytes,(este procedimiento se explicará mas adelante),a continuación se crea un socket igual al anterior y éste lee la información del servidor. El último paso del modo contador es ejecutado en esta etapa, y es la operación XOR entre el buffer de 160 bytes previamente almacenado y el vector recibido a través del servidor. Finalmente se reasignan los datos al paquete RTP y se enviá a través de la red.

Como se aprecia en la figura anterior el framework DFD realiza la ejecución del algoritmo de cifrado AES. Sin embargo para poder entender más a fondo el funcionamiento de este ejecutable es necesario explicar los requerimientos necesarios para que pueda ser ejecutado por la aplicación CSIPSimple y su manera de operar que a continuación se detallan.

Como se mencionó anteriormente el framework DFD ***(DSP for Dummies)*** fue diseñado para permitir realizar una interfaz sencilla para el manejo de procesos en el DSP con el cual cuenta la tarjeta de desarrollo. DFD se encarga de la coherencia en la memoria cache, análisis y envió de mensajes y presenta una interfaz simple para el programador. El framework contiene una serie de archivos que hacen posible el funcionamiento de DFD. De ellos los principales donde se centra la atención son dummy_dsp.h y dummy_arm.c. En la primera de ellas se agregan las funciones que se desean ejecutar en el DSP, cada función se relaciona estrechamente con un código de operación (OPCODE), además cada función toma los buffers de entrada y salida a través del cual se enviará y recibirá la información procesada por el DSP.

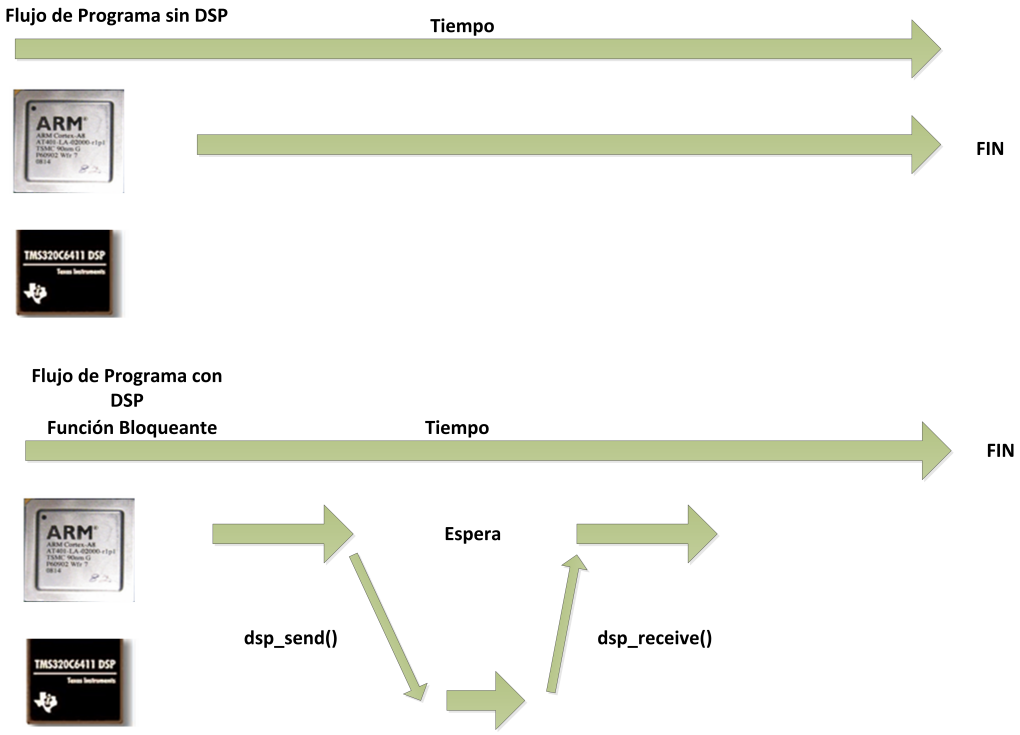


Figura 4.13: Flujo de Programa sin y con DSP.

El segundo de ellos contiene la función principal del código de programación, en ella se define la función que se desea ejecutar el DSP, y se rellena el buffer que se enviará al DSP. DFD implementa dos tipos de funciones para la realización de tareas en el DSP: bloqueantes y no bloqueantes. Una función bloqueante espera a que el DSP termine su ejecución para continuar con la ejecución del programa, mientras que la no bloqueante regresa un valor booleano indicando si el DSP ha terminado o no su tarea, en este tiempo el procesador ARM (GPP) puede realizar otra tarea.

El principal propósito de hacer uso del DSP es aumentar el rendimiento de ambos procesadores y hacer mas eficiente la realización de las tareas. En la figura 4.13 se muestra el flujo que seguiría el programa sin y con el uso del DSP. DFD realiza la implementación de la multiplicación, suma y restas de matrices como ejemplo para iniciarse en la programación del DSP.

A continuación se detallan las herramientas necesarias para realizar la programación de tareas en el DSP y la programación del algoritmo de cifrado AES que se realizó en el DSP.

- Descargar el código fuente del sitio proporcionado por Texas Instruments. [2]
- Instalar el *toolchain* de ARM y configurar el cross compiler arm−none−linux−gnueabi−gcc
- Contar con *(Code Generation Tools)* CGT v.6.0.7 y DSP/BIOS 5.21.03.
- Configurar el MakeFile del proyecto DFD especificando la librería que servirá de enlace hacia las demás librerías. En este caso se especifica –dynamic-linker=/lib/ld-2.8.so

Estos requisitos permiten que el proyecto DFD pueda ser modificado y compilado a fin de generar dos archivos necesarios para la ejecución en la tarjeta de desarrollo.

El archivo ejecutable dummy y el archivo dummy.dll64p son los generados, este último es una librería de enlace dinámico que permite la ejecución de código haciendo uso de otras librerías. Ambos archivo deberán ser exportados al sistema de archivos del sistema operativo Android para su ejecución.

En el archivo dummy_arm.c como se muestra en la figura 4.14 se define las operaciones que realizará el procesador ARM (GPP), en él se contiene la función principal y se establece la creación de un socket en el puerto 5001, con las mismas propiedades que los anteriores a fin de recibir los datos enviados por la aplicación CSIPSimple (timestamp 4 Bytes) a través del servidor. Una vez que se cuenta con este parámetro se configura un nodo de trabajo en el dsp haciendo uso de la instrucción *setup_dsp()*, en el que se especifica la función que se ejecutará en el DSP, pues en él se puede programar diferentes funciones a ejecutar, en este caso solo se configura la función de cifrado y ésta es la que se invoca.

A continuación se llena el buffer de entrada que se transmitirá al DSP. Con los parámetros listos se envía el nodo de trabajo con la instrucción *dsp_send()* y espera a que el DSP termine la tarea y envíe un mensaje de término a fin de extraer el buffer de salida. Una vez que el mensaje de término es recibido (*dsp_irecv()*), se extrae el vector de salida del DSP y se transmite al servidor a través del socket que se había abierto previamente. Se cierra la conexión con el servidor y se le indica al DSP que el nodo de trabajo puede ser finalizado haciendo uso de la instrucción *dsp_finish()*.

Por su parte el archivo dummy_dsp.h es el que contendrá la carga de procesos para ejecutarse en el DSP y está compuesto de la siguiente manera:

[2] Texas Instruments File: DSP-for-dummies.tar.gz http://processors.wiki.ti.com/images/c/c3/Dsp−for−dummies.tar.gz

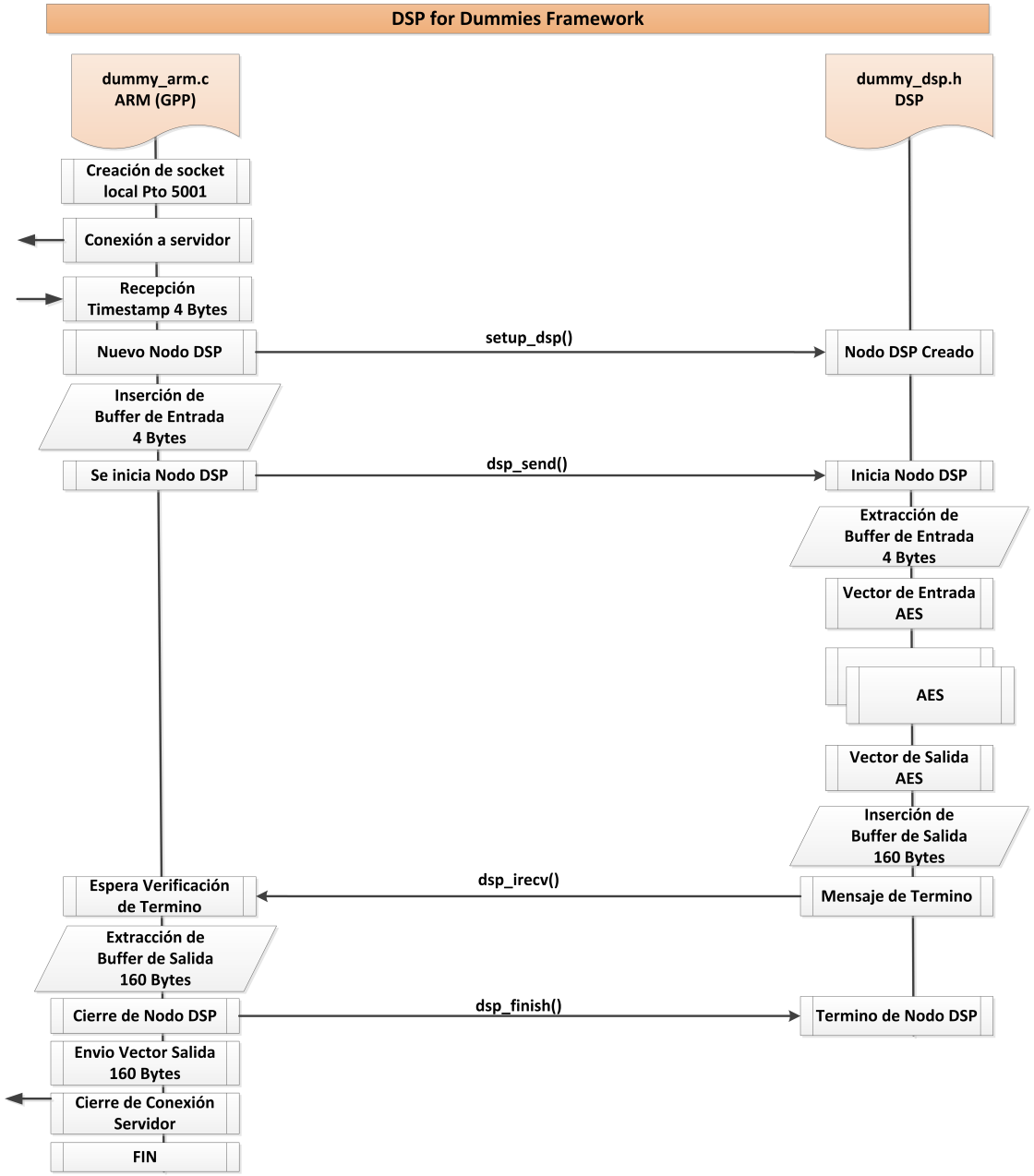


Figura 4.14: Diagrama de flujo DFD.

Primero se definen el número de funciones que se ejecutarán en el DSP y que serán invocadas por el procesador ARM (GPP), en este caso solo se parametriza la función **aes_cifrado()** que recibe un OPCODE=1, es decir este código será el que invoque el procesador ARM(GPP) cada vez que desee ejecutar este proceso en el DSP. Se define posteriormente los valores de las *S-Box* y se define la operación que realiza cada una de las funciones de las rondas AES.

Luego se extraen los datos del buffer de entrada (4 bytes) que fueron exportados por el procesador ARM (GPP), éstos son colocados en los bits más significativos del vector de entrada de lo que será el algoritmo de cifrado AES. Se asume que la llave ha sido previamente enviada o se tiene almacenada, para este caso se uso una llave de cifrado de 128 bits, y con estos parámetros se comienza la ejecución del algoritmo.

A continuación se realiza la expansión de la llave de cifrado, se manda llamar a la rutina *KeyExpansion()* que genera como resultado un esquema de 10 llaves de 128 bits derivadas de la original,como se muestra en la figura 4.15, inicialmente se introduce la llave original y se forma un arreglo de 4x4 bytes.

$$\begin{pmatrix} w_{0,0} & w_{0,1} & w_{0,2} & w_{0,3} \\ w_{1,0} & w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,0} & w_{2,1} & w_{2,2} & w_{2,3} \\ w_{3,0} & w_{3,1} & w_{3,2} & w_{3,3} \end{pmatrix} \tag{4.1}$$

De 4.1 se toma la última columna y se manda a la función *RotWord()*, lo que se realiza es básicamente una permutación donde se toma el arreglo de entrada y retorna el arreglo permutado de la siguiente manera:

$$RotWord([w_{0,3}, w_{1,3}, w_{2,3}, w_{3,3}]) = [w_{1,3}, w_{2,3}, w_{3,3}, w_{0,3}] \tag{4.2}$$

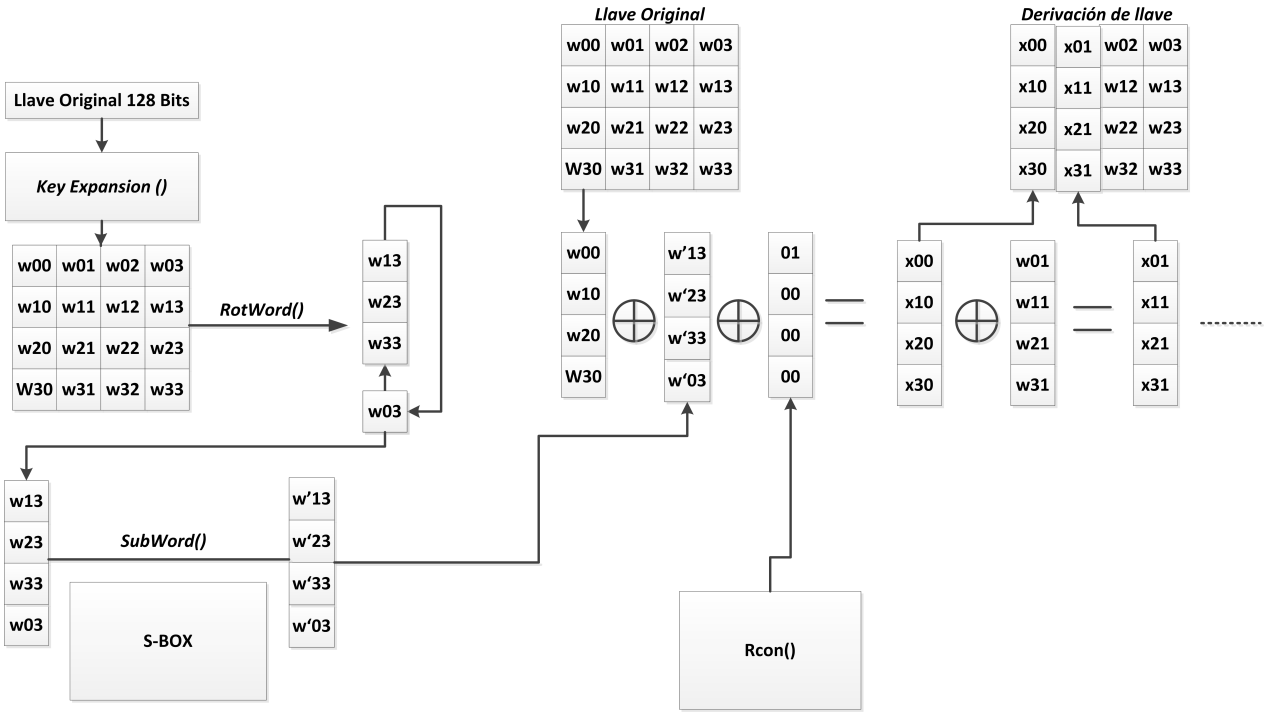


Figura 4.15: Rutina KeyExpansion.

Luego se ejecuta la función *SubWord()* que realiza el intercambio de los bytes acorde a las *S-Box* correspondientes. Al arreglo resultante se le aplicará la operación XOR con el arreglo de la primera columna de la llave original y con un arreglo denominado Rcon[] el cual es la constante de ronda que se analizó en el capitulo 5, el resultado de esta operación será la composición de la primera columna del arreglo de la llave derivada uno.

$$\begin{bmatrix} w_{0,0} \\ w_{1,0} \\ w_{2,0} \\ w_{3,0} \end{bmatrix} \oplus \begin{bmatrix} w'_{1,3} \\ w'_{2,3} \\ w'_{3,3} \\ w'_{0,3} \end{bmatrix} \oplus \begin{bmatrix} 01 \\ 00 \\ 00 \\ 00 \end{bmatrix} = \begin{bmatrix} x_{0,0} \\ x_{1,0} \\ x_{2,0} \\ x_{3,0} \end{bmatrix} \tag{4.3}$$

La siguiente columna de la llave derivada se obtienen a partir de realizar la operación XOR entre el resultado de 4.3 y la segunda columna de 4.1 es decir:

$$\begin{bmatrix} x_{0,0} \\ x_{1,0} \\ x_{2,0} \\ x_{3,0} \end{bmatrix} \bigoplus \begin{bmatrix} w_{0,1} \\ w_{1,1} \\ w_{2,1} \\ w_{3,1} \end{bmatrix} = \begin{bmatrix} x_{0,1} \\ x_{1,1} \\ x_{2,1} \\ x_{3,1} \end{bmatrix} \tag{4.4}$$

El resultado de 4.4 formará la segunda columna de la llave derivada, las siguientes dos columnas se forman de la misma manera, es decir, se toma el resultado anterior y se realiza la operación XOR con la 3ª y 4ª columna respectivamente. Este procedimiento se realiza hasta completar las llaves necesarias para cada una de las rondas AES, para este caso dado que la llave es de 128 bits el procedimiento se realiza 10 veces.

Una vez que se obtienen las llaves derivadas se puede proceder a ejecutar el algoritmo AES. En el mismo archivo dummy_dsp.h a continuación se ejecuta la función *a_ encrypt()*, quien es la encargada de ejecutar cada ronda que conforma al algoritmo de cifrado AES. A esta función se le introduce como parámetros un vector conformado por 128 bits (16 bytes) en los que los 4 bits más significativos corresponden al timestamp y se ingresa también el esquema de llaves obtenido de la función *KeyExpansion()* descrita anteriormente.

Al igual que en la función de expansión de llave se toma el vector de entrada y se realiza una matriz de 4x4 bytes, lo que permite trabajar cada una de las rondas AES. A la matriz inicial formada se le denomina "state" y esta matriz es enviada a la primera etapa del algoritmo de cifrado AES. El resultado de esta transformación es enviado a las siguientes rondas hasta terminar las 10 rondas correspondientes como se aprecia en el siguiente pseudocódigo:

```
1  void a_encrypt)(in[],out[],k[],)
2  {
3  AddRoundKey(state,k0);
4  SubByte(state);ShiftRow(state);MixColumn(state);AddRoundKey(state,k1);
5  SubByte(state);ShiftRow(state);MixColumn(state);AddRoundKey(state,k2);
6  SubByte(state);ShiftRow(state);MixColumn(state);AddRoundKey(state,k3);
7  SubByte(state);ShiftRow(state);MixColumn(state);AddRoundKey(state,k4);
8  SubByte(state);ShiftRow(state);MixColumn(state);AddRoundKey(state,k5);
9  SubByte(state);ShiftRow(state);MixColumn(state);AddRoundKey(state,k6);
10 SubByte(state);ShiftRow(state);MixColumn(state);AddRoundKey(state,k7);
11 SubByte(state);ShiftRow(state);MixColumn(state);AddRoundKey(state,k8);
12 SubByte(state);ShiftRow(state);MixColumn(state);AddRoundKey(state,k9);
13 SubByte(state);ShiftRow(state);AddRoundKey(state,k10);
14 }
```

Como se aprecia para cada una de las rondas se proporciona una llave diferente en la etapa *AddRoundKey()*, la primera etapa se ejecuta con la llave original y a partir de ahi las siguientes rondas hacen uso del esquema de llaves proporcionado. Para la última ronda se omite el paso *MixColumn()* de acuerdo a los lineamientos establecidos por el algoritmo AES.

La salida de esta función nos otorga un arreglo de 128 bits(16 bytes) y con la finalidad

de obtener un arreglo del mismo tamaño del frame de datos que se necesita (160 bytes) se procede a ejecutar el algoritmo de cifrado AES hasta completar el tamaño del arreglo. Para cada ejecución del algoritmo de cifrado AES que se realice, al vector de entrada AES se le aumenta en una unidad el contador con la finalidad de obtener una salida de datos diferente como se muestra en la figura 4.16.

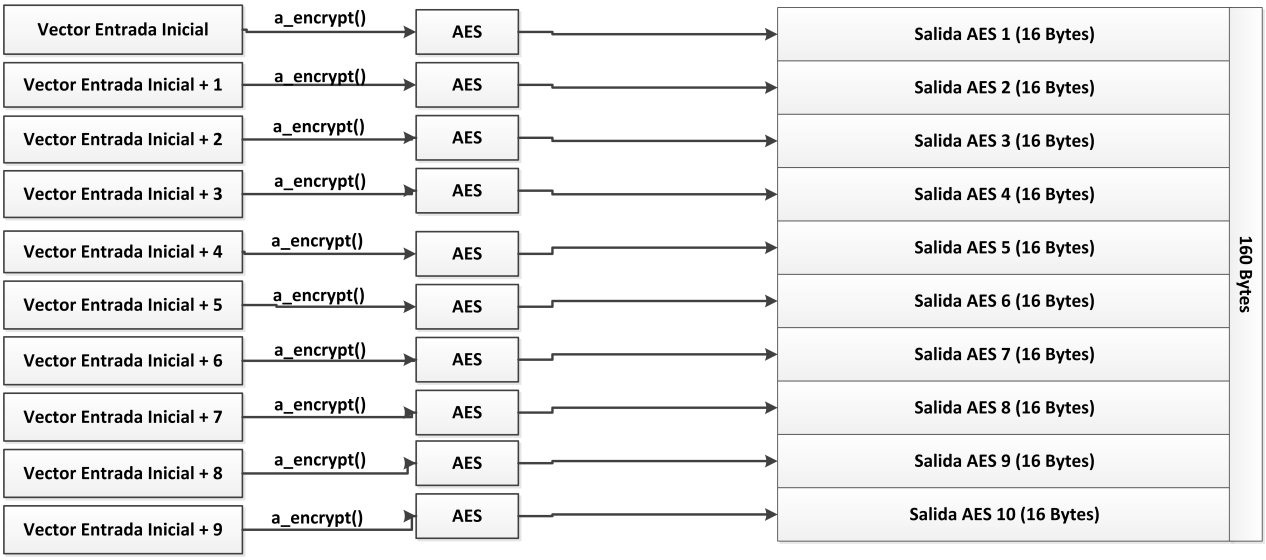


Figura 4.16: Ejecuciones AES en DSP.

Finalmente el arreglo de 160 bytes es almacenado en el buffer de salida del cual el procesador ARM (GPP) tomará los datos para transmitirlos al servidor y posteriormente a la aplicación CSIPSimple y realizar la operación XOR para finalizar el proceso de cifrado.

4.5. Resumen de contenido.

Se diseñó un esquema de cifrado que hace uso del algoritmo de criptografía simétrica AES en modo contador, a fin de poder cifrar y descifrar de manera correcta. Fue utilizada una señal de sincronía que permite generar el mismo buffer de salida de palabras cifrantes, tanto en la etapa de cifrado como en la etapa de descifrado.

El esquema de cifrado propuesto hace uso de un componente de hardware adicional, el procesador digital de señales, el cual es empleado para ejecutar el algoritmo de cifrado AES. Dado que no existe un método para comunicar los procesos generados en el procesador de propósito general y los procesos generados en el procesador digital de señales, se creó un esquema de comunicaciones que permite la interacción entre ambos procesos.

Finalmente, se construyó un prototipo con las mismas prestaciones con las que cuenta un dispositivo móvil, hace uso del sistema operativo Android y del esquema de cifrado propuesto, esto permite que se puedan realizar pruebas al diseño del esquema de cifrado, tanto de rendimiento como de seguridad, mismas que se analizarán en el siguiente capítulo.

Capítulo 5

Pruebas y Resultados.

En este capítulo se abordan los puntos necesarios para evaluar el esquema de cifrado propuesto en términos de seguridad y rendimiento. Se establecen las condiciones bajo las cuales se realizan las pruebas del esquema de cifrado y en base a éstas se obtienen ciertos resultados, mismos que sirven para realizar una comparación entre los trabajos más cercanos a éste, del cual se hizo un estudio en el estado de arte. Se presentan las comparativas y se obtienen las conclusiones que permiten evaluar el esquema de cifrado.

5.1. Arquitectura del esquema de pruebas.

A fin de poder realizar las pruebas necesarias que evaluarán el esquema de cifrado, es necesario contar con una serie de requisitos. Dado que se trabaja con VoIP es preciso habilitar un dispositivo que proporcione las funcionalidades que un proveedor de VoIP presta. Para ello se hizo uso del software codigó abierto *Asterisk*, este software tiene la característica de poder convertir un ordenador de propósito general en un sofisticado servidor de comunicaciones de VoIP.

A continuación se detallan los elementos necesarios para la instalación de Asterisk:

- Equipo de cómputo o máquina virtual con sistema operativo Ubuntu 12.04 LTS
- Descargar el software Asterisk Versiòn 1.8 de la página oficial[1]
- Interfaz ethernet en el equipo donde se realizará la instalación.

Con los recursos necesarios se procede a realizar la instalación siguiendo los lineamientos establecidos por la documentación que establece el proveedor[12]. El objetivo es contar con una arquitectura de red como la que se aprecia en la figura 5.1.

El servidor asterisk además de ser instalado debe ser corrrectamente configurado. Se definen principalmente dos ficheros a configurar *sip.conf* y *extensions.conf*. El primero de ellos permite configurar las cuentas de los usuarios que harán uso de los servicios de VoIP, en él se definen tanto los codecs permitidos como el nombre y contraseña del usuario, mientras que el segundo establece el plan de numeración que seguirá el servidor asterisk para cada usuario, es decir, establece los lineamientos a seguir cuando se realiza el marcado de la llamada y el colgado de la llamada entre otras acciones.

[1] Centro de Descarga Asterisk http://www.asterisk.org/downloads/asterisk/all-asterisk-versions

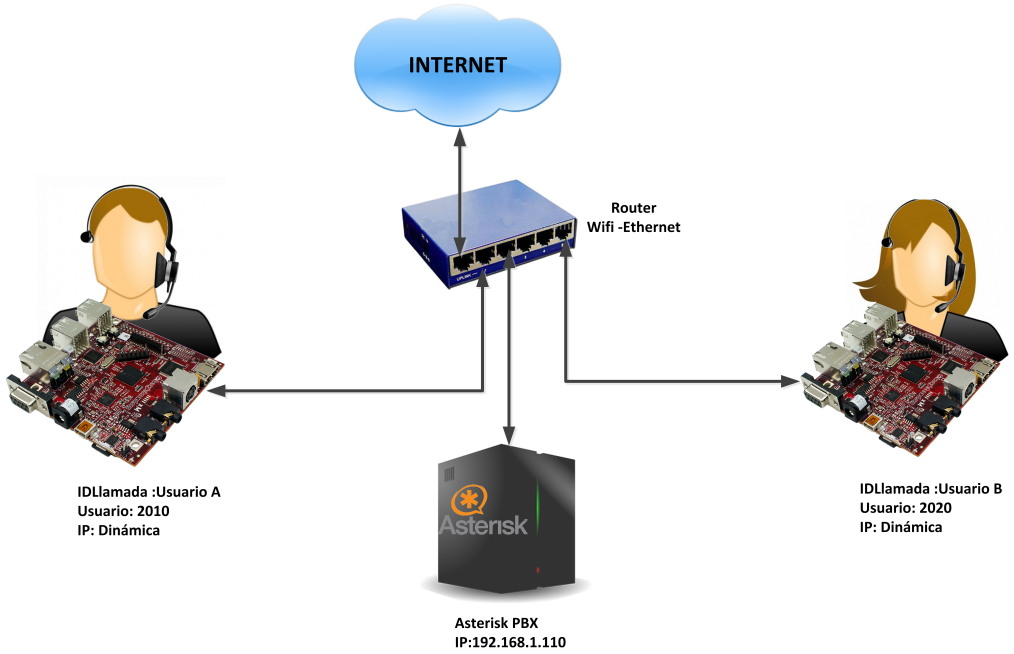


Figura 5.1: Arquitectura del esquema de pruebas.

En el primer fichero se define una configuración general, se establece como puerto por defecto el 5060 y se definen los usuarios como a continuación se muestra:

```
[2010]
type=friend
secret=****
context=SeguridadCIC
disallow=all
allow=alaw
allow=ulaw
```

El nombre del usuario se define por el parámetro entre corchetes al principio de la configuración en este caso *2010*, el parámetro *type* define el tipo de usuario, asterisk contempla 3 tipos de usuario: *peer, user y friend*, el primero de ellos solo se pueden recibir llamadas, el segundo solo puede realizar llamadas y el tercero puede realizar ambas acciones. El siguiente parámetro (*secret*) establece la contraseña del usuario, misma que deberá colocar al configurar la cuenta SIP en el dispositivo deseado. El parámetro siguiente sirve con enlace al archivo extensions.conf el cual definirá el plan de marcado para este usuario y por último se definen los codecs a emplear, es necesario deshabilitar todos los codecs primero y después habilitar solo aquellos que se desean usar, en este caso se habilitan los codecs G.711 para ambos esquemas la ley A y la ley μ

En el segundo fichero se establece el plan de marcación para cada una de las extensiones de acuerdo a lo siguiente:

```
[SeguridadCIC]
exten => 2010,1,Dial(SIP/2010,30,Ttm)
exten => 2010,2,Hangup
exten => 2010,102,Voicemail(2010)
exten => 2010,103,Hangup
```

Se define el contexto con el cual se hará referencia al plan de marcado, el primer parámetro establece el intento de marcación al usuario 2010, el segundo establece el colgado de la llamada una vez que el usuario ha terminado la llamada, el tercero y el cuarto pertenecen al buzón de voz en caso de que el usuario no conteste la llamada y una vez dejado el mensaje se procede a colgar. La configuración e instalación detallada del servidor puede ser encontrada en la sección de anexos apartado B.

Con el servidor de VoIP habilitado es necesario configurar las cuentas en la tarjeta de desarrollo para ello se accede a la aplicación CSIPSimple y en el MenúAjustesCuentasAñadir cuentaBasico se procede a configurar la cuenta que se habilitó en el servidor Asterisk como se aprecia en la figura 5.2.

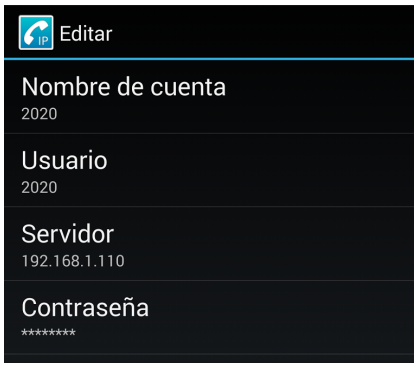


Figura 5.2: Configuración de cuenta en CSIPSimple.

En ese apartado se configura el nombre que se le asignará a la cuenta SIP, el nombre del usuario que deberá ser el mismo que se habilitó en el servidor asterisk, la dirección IP del servidor y la contraseña del usuario. Una vez que se configuren estos parámetros se guarda la configuración y acto inmediato la aplicación iniciará sesión en el servidor de VoIP como se muestra en la figura 5.3.

Figura 5.3: Inició de sesión en servidor.

5.2. Condiciones de prueba.

Una vez que se cuenta con la arquitectura del esquema de pruebas es necesario establecer las condiciones bajo las cuales se realizarán las pruebas que evaluarán el esquema de cifrado, para ello en la aplicaciòn CSIPSimple se establecen las siguientes condiciones.

Se configura el codec a usar en la aplicación, se accede a MenúAjustesMultimediaCodecs, CSIPSimple establece dos configuracionesde codec, una para banda ancha y otra para redes donde el ancho de banda es limitado, en ambos casos se trabaja con el codec PCMA 8KHz que corresponde al codec G.711 ley A como se muestra en la figura 5.4.

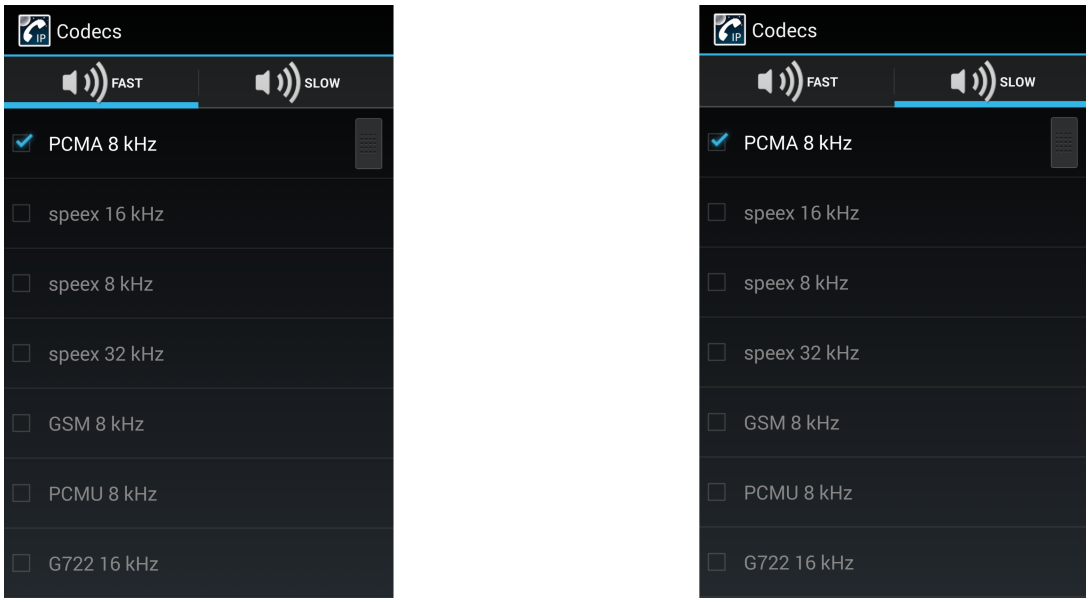


Figura 5.4: Configuración de codecs en la aplicación.

Con esto se tendría configurado tanto el servidor como la aplicación para iniciar las pruebas. Adicionalmente y como especificación para el posterior análisis de los resultados de rendimiento se establece que la única aplicación que se ejecutará al momento de realizar las pruebas es la de CSIpSimple y lo que de ella se derive, así como los procesos generados por el propio sistema operativo. Esta premisa permite realizar el análisis de rendimiento del esquema de cifrado considerando únicamente los procesos generados por la aplicación.

5.3. Pruebas de seguridad.

Con la finalidad de evaluar la seguridad presentada por el esquema de cifrado y a fin de demostrar que la manera en que se diseñó el esquema propuesto no degrada las propiedades del algoritmo de cifrado en este trabajo, se presentan a continuación un número de parámetros a evaluar como el coeficiente de correlación, entropía de la información, calidad del cifrado y el espectrograma de la señal de audio, los cuales permitirán cuantificar la calidad del esquema de cifrado diseñado.

Para llevar a cabo este análisis se realizaron 20 pruebas en donde se recolectaron los datos transmitidos durante la llamada, esto se realizó con ayuda del software de código abierto Wireshark, para todas las pruebas se capturaron los paquetes RTP antes de ser cifrados y enviados a la red y después se capturaron los datos cifrados en el medio con Wireshark. Es importante mencionar que las 20 pruebas realizadas pertenecen a 20 llamadas con 20 voces de

diferentes personas. La tabla 5.1 muestra las 20 pruebas realizadas asi como las personas que ayudarón en la realización de estas.

Tabla 5.1: Pruebas Realizadas.

Sujeto	Características	Tiempo de llamada
1	Hombre. 24 Años	60 seg
2	Mujer. 24 Años	64 seg
3	Hombre. 24 Años	60 seg
4	Hombre. 27 Años	58 seg
5	Hombre. 26 Años	61 seg
6	Mujer. 23 Años	60 seg
7	Hombre. 24 Años	58 seg
8	Hombre. 28 Años	67 seg
9	Mujer. 24 Años	60 seg
10	Hombre. 24 Años	60 seg
11	Mujer. 24 Años	60 seg
12	Mujer. 26 Años	56 seg
13	Hombre. 25 Años	60 seg
14	Hombre. 25 Años	60 seg
15	Hombre. 24 Años	60 seg
16	Hombre. 24 Años	60 seg
17	Mujer. 24 Años	63 seg
18	Hombre. 25 Años	58 seg
19	Hombre. 24 Años	60 seg
20	Hombre. 26 Años	60 seg

Las pruebas constaron en un tiempo promedio de 60 segundos por llamada, el genero de la persona y su edad solo sirven como referencia para comprobar que sin importar la fuente de información el resultado a la salida del cifrador seguirá siendo el mismo.

El cálculo de los parámetros de seguridad a evaluar se llevó a cabo tomando en cuenta solo la carga útil del paquete RTP la cual fue exportada a un archivo binario y se analizó con ayuda de la herramienta de software Matlab.

5.3.1. Coeficiente de correlación.

El coeficiente de correlación mide el grado de similitud entre dos variables, este parámetro es útil al momento de juzgar la calidad de un cripto-sistema. Se dice que un cripto-sistema es "bueno"si el algoritmo de cifrado oculta todos los atributos del texto plano mientras que los valores del texto cifrado son totalmente aleatorios y sin correlación alguna[17]. Si el texto cifrado y el texto plano son totalmente diferentes entonces su coeficiente de correlación deberá ser muy bajo o cercano a cero. Si el coeficiente de correlación es igual a uno entonces existe una perfecta correlación lo que significa que el proceso de cifrado ha fallado dado que el texto plano y el texto cifrado son idénticos.

Entonces matemáticamente el coeficiente de correlación esta dado por la siguiente ecuación:

$$C.C = \frac{Cov(x,y)}{\sigma_x * \sigma_y} \tag{5.1}$$

$$\sigma_x = \sqrt{VAR(x)} \tag{5.2}$$

$$\sigma_y = \sqrt{VAR(y)} \tag{5.3}$$

Donde en 5.1 C.C corresponde al coeficiente de correlación y Cov es la covarianza entre dos valores, uno que pertenece al texto plano y otro que pertenece al texto cifrado, en 5.2 y 5.3 VAR() corresponde a la varianza del valor correspondiente.

La tabla 5.2 muestra los valores obtenidos cuando se calcula el coeficiente de correlación en los datos obtenidos antes y después de ser cifrados. En ella se aprecia que los valores pertenecientes al texto plano en las 20 pruebas realizadas es de 1, lo que indica una perfecta correlación, mientras que los valores obtenidos en el texto cifrado mantienen un valor cercado a cero, lo cual indica que no existe correlación entre los datos del texto plano y del texto cifrado.

Tabla 5.2: Coeficiente de correlación.

No.	Texto Plano	Texto Cifrado
1	1	.0000979
2	1	.0000779
3	1	.0000734
4	1	.0000975
5	1	.0000902
6	1	.0000252
7	1	.0000010
8	1	.0000958
9	1	.0000215
10	1	.0000764
11	1	.0000689
12	1	.0000435
13	1	.0000482
14	1	.0000317
15	1	.0000940
16	1	.0000024
17	1	.0000067
18	1	.0000486
19	1	.0000373
20	1	.0000618

5.3.2. Entropía de la información.

Otro parámetro interesante que se analizó es la entropía de la información pues éste provee el grado de incertidumbre en cualquier sistema de comunicación, esta teoría fue propuesta por el matemático Claude Elwood Shannon en 1949, la entropía de cualquier mensaje puede ser calculada de acuerdo a 5.4

$$H(m) = \sum_{i=0}^{2^{N-1}} p(m_i) * log_2 \frac{1}{p(m_i)} \tag{5.4}$$

De donde $p(m_i)$ representa la probabilidad de ocurrencia del byte m_i, por ende el valor de la entropía de un mensaje es la sumatoria de la entropía de cada uno de los valores con los que se codifican los datos, es decir 2^8 posibles valores. Si la entropía es evaluada para este caso y

se considera que cada byte tiene la misma probabilidad de ocurrencia, el resultado ideal sería obtener 8, dado que cada byte es representado por 8 bits, el valor de la entropía nos indica que la fuente de información es una fuente aleatoria uniforme. En general el valor de la entropía para una fuente de información real es mucho menor debido al hecho de que raramente se transmite un mensaje aleatorio, si la entropía no es 8 o aproximada a 8 existiría cierto grado de predecir el mensaje que se transmite. La tabla 5.3 muestra los valores obtenidos a partir de la ecuación 5.4 para cada una de las pruebas realizadas.

Tabla 5.3: Entropía de la información.

No.	Texto Cifrado
1	7.9994
2	7.9996
3	7.9994
4	7.9994
5	7.9994
6	7.9994
7	7.9994
8	7.9994
9	7.9995
10	7.9994
11	7.9994
12	7.9994
13	7.9995
14	7.9994
15	7.9994
16	7.9994
17	7.9995
18	7.9996
19	7.9994
20	7.9994

Los resultados obtenidos muestran que el valor de la entropía es cercado al valor ideal 8, lo cual implica que las fugas de información en el esquema de cifrado son despreciables y el esquema de cifrado hace un uso adecuado del algoritmo de cifrado AES, la figura 5.5 muestra gráficamente como la fuente de información para el caso del texto cifrado se comporta como una fuente uniformemente aleatoria, mientras que para el texto plano existe cierto grado de predicción del mensaje que se transmite debido a que no todos los valores tienen la misma probabilidad de ocurrencia, las gráficas de las pruebas restantes se pueden ver en la sección de anexos apartado C.

5.3.3. Desviación estándar.

Se evalúa la calidad del esquema de cifrado mediante el histograma de datos que provee el texto plano y el texto cifrado, un algoritmo de cifrado ideal cifra los datos de tal manera que su distribución en el histograma sea uniforme, la figura 5.6 muestra el histograma de datos cuando la información permanece original y cuando ésta es cifrada. Se puede observar que la frecuencia de aparición de los datos para el segundo caso es homogénea y uniforme, sin embargo una percepción visual del cambio que induce el esquema de cifrado en los datos no basta para concluir que el esquema posee buena calidad.

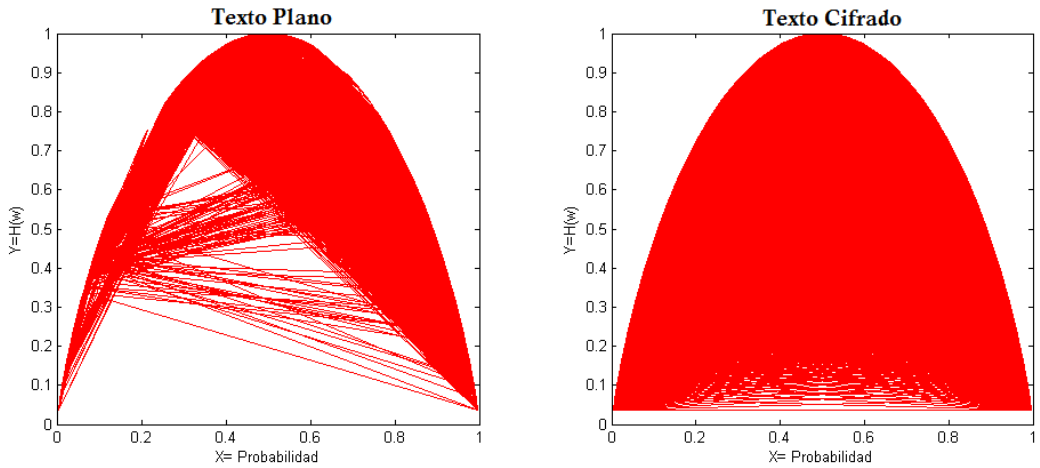


Figura 5.5: Entropía de los datos.

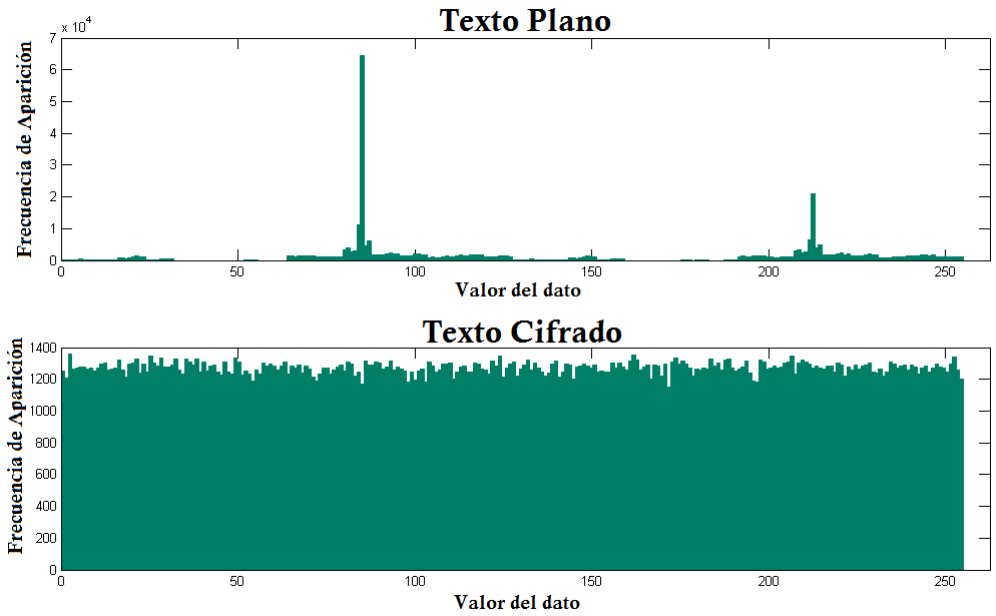


Figura 5.6: Histograma de datos.

A fin de determinar un valor cuantitativo para evaluar la calidad del esquema de cifrado se hace uso de la desviación estándar que matemáticamente está dada por la expresión 5.5, donde S es la desviación estándar, x_i es el dato entre los 256 posibles valores, $\bar{X}$ es el promedio de los datos y n es el número de datos en total. Si la diferencia de la desviación estándar es máxima entre el texto plano y el texto cifrado, se dice que el esquema de cifrado posee una calidad buena, la tabla 5.4 muestra los valores obtenidos en las 20 pruebas realizadas.

$$S = \sqrt{\frac{\sum_{i=1}^{n}(x_{i-\bar{X}})^2}{n-1}} \tag{5.5}$$

Tabla 5.4: Desviación estandar de la frecuencia de aparición de los datos.

No.	Texto Plano	Texto Cifrado
1	4332.81	35.6641
2	5860.10	34.7768
3	5132.05	37.8921
4	7112.94	36.6461
5	4759.10	35.2813
6	5778.94	36.9524
7	7587.12	37.7815
8	6958.02	36.1620
9	7953.34	34.2412
10	6982.64	35.2379
11	7059.36	34.5610
12	7861.54	35.1086
13	7684.35	34.0586
14	7948.32	34.7512
15	7512.56	35.0024
16	7428.12	34.6822
17	7945.32	34.1578
18	7941.56	34.6351
19	7546.27	34.2510
20	7864.35	34.6293

Los resultados mostrados en la tabla 5.4 confirman la calidad del esquema de cifrado, se puede observar que la diferencia entre los valores obtenidos del texto plano y el texto cifrado son máximos, el tener una desviación estándar menor en el texto cifrado confirma la uniformidad de la distribución de datos del histograma presentado en la figura 5.6. Los histogramas de las pruebas restantes se pueden ver en la sección de anexos apartado C.

5.3.4. Espectrograma de la señal.

Finalmente un análisis que permita visualizar el contenido auditivo de la señal de audio que se transmite es el realizar una gráfica del espectrograma de la señal, en él se pueden visualizar las frecuencias contenidas en los datos de la señal de audio, si consideramos que la frecuencia de la voz humana opera entre los 300 y 3800 Hz, se espera que el espectrograma del texto plano muestre la variedad de frecuencias de la voz del locutor que transmite, mientras que en el texto cifrado se espera que no exista un patrón de frecuencias que puedan corresponder a las frecuencias comprendidas por la voz de una persona.

La figura 5.7 muestra el análisis de las frecuencias encontradas cuando la voz no es cifrada y cuando es cifrada, para el primer caso, el texto plano, la frecuencia de la voz predomina en las zonas rojas que corresponden a las frecuencias comprendidas entre los 300 y 2500 Hz, esta voz corresponde a la de una mujer de 26 años y como se puede observar el patrón de sus frecuencias de voz se encuentra perfectamente definido por el espectrograma realizado. Los espectrogramas de las pruebas restantes se pueden ver en la sección de anexos apartado C.

Para el caso del texto cifrado las frecuencias perceptibles no corresponden en ningún caso a las frecuencias de la voz humana, es por ello que el espectrograma muestra toda la gama de frecuencias desde los 0 hasta los 4000 Hz, con esto se demuestra el contenido auditivo que se transmite en el medio y el contenido original que el remitente en realidad desea que se

reciba.

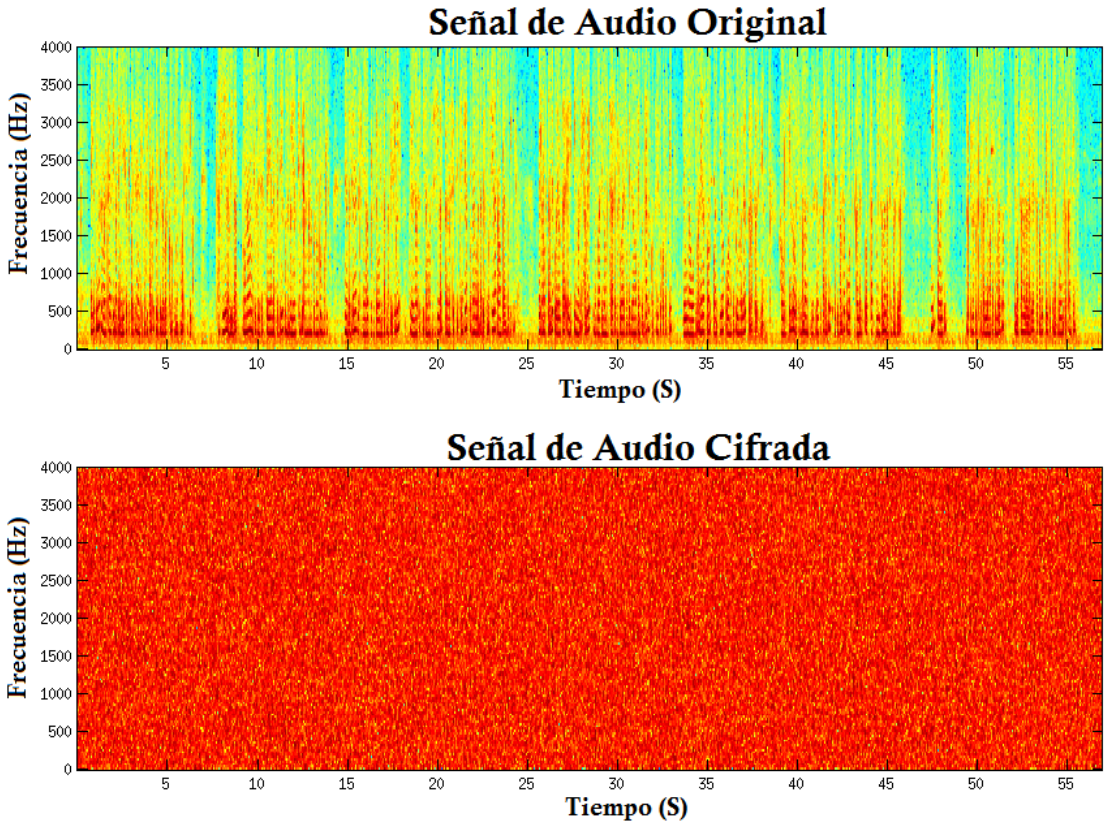


Figura 5.7: Espectograma de la señal de audio.

5.4. Pruebas de rendimiento.

A fin de evaluar el rendimiento del esquema de cifrado, se realizó un análisis del consumo de CPU de los procesos generados por la aplicación CSIPSimple, con ayuda del software wireshark se midió el ancho de banda máximo generado por la llamada de VoIP y finalmente se tomó el retardo de tiempo promedio que genera el esquema de cifrado. Esta última prueba se realizo tomando en cuenta la hora inicial y final en que una llamada era realizada. En primera instancia se recolectaron los datos arrojados por la aplicación cuando no se hace uso del esquema de cifrado y más tarde se realizó el análisis cuando se hace uso del esquema de cifrado, éste último contempla dos casos: el primero de ellos cuando el algoritmo de cifrado es programado en el GPP y el segundo contempla el uso del esquema de cifrado propuesto, es decir cuando se hace uso del DSP.

La figura 5.8 muestra el consumo de CPU cuando se hace uso de la aplicación sin utilizar del esquema de cifrado, en este caso la aplicación permanece sin ningún tipo de modificación lo que nos permite tener una base para comparar los datos cuando el esquema de cifrado propuesto se habilite.

Como se aprecia en la gráfica cuando se inicia la llamada desde la aplicación se derivan dos procesos, uno definido como *CSIPSimple* y otro como *SIPSTACK* el primero de ellos hace referencia al proceso que refleja la apertura de la aplicación en Android, mientras que el

segundo de ellos muestra el proceso generado por la pila de protocolos SIP, este proceso es al que se le deberá prestar más atención a fin de realizar el análisis de resultados.

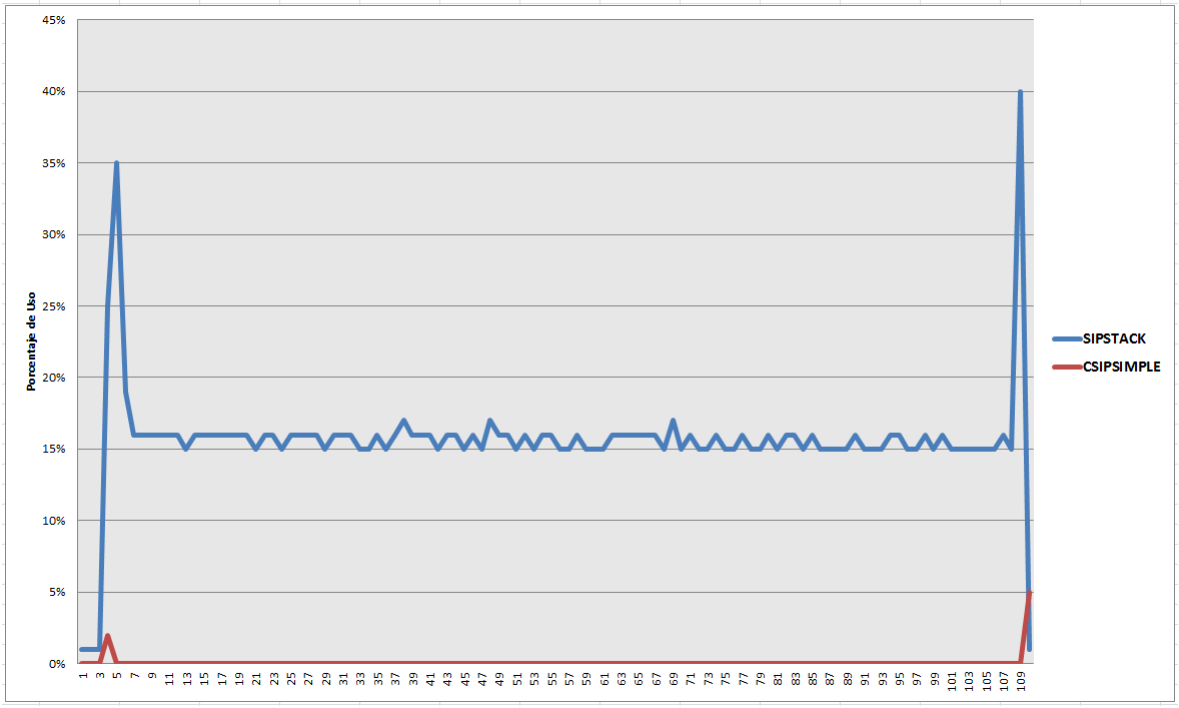


Figura 5.8: Consumo de CPU sin cifrar.

Tabla 5.5: Consumo de CPU sin cifrar.

	CSIPSIMPLE	SIPSTACK
Cifrado:	----------	**Ninguno**
Uso Min CPU:	**0%**	**1%**
Uso Max CPU:	**5%**	**40%**
Uso Promedio CPU:	**0%**	**15%**
Max Ancho de Banda:	---------	**9.9 kB/s**
Retardo Promedio:	---------	---------

La tabla 5.5 nos muestra el consumo máximo, mínimo y promedio de CPU al momento de realizar una llamada, nos muestra el máximo ancho de banda que emplee para realizar la misma y no existe un retardo de tiempo dado que no se hace uso del esquema de cifrado.

En la figura 5.9 y la tabla 5.6 se muestra el consumo de CPU durante una llamada cuando el cifrado es programado en el procesador de propósito general, se aprecia inmediatamente que el uso del CPU aumenta en un 13 % al momento de cifrar y que éste implica un retardo de 21.72 ms.

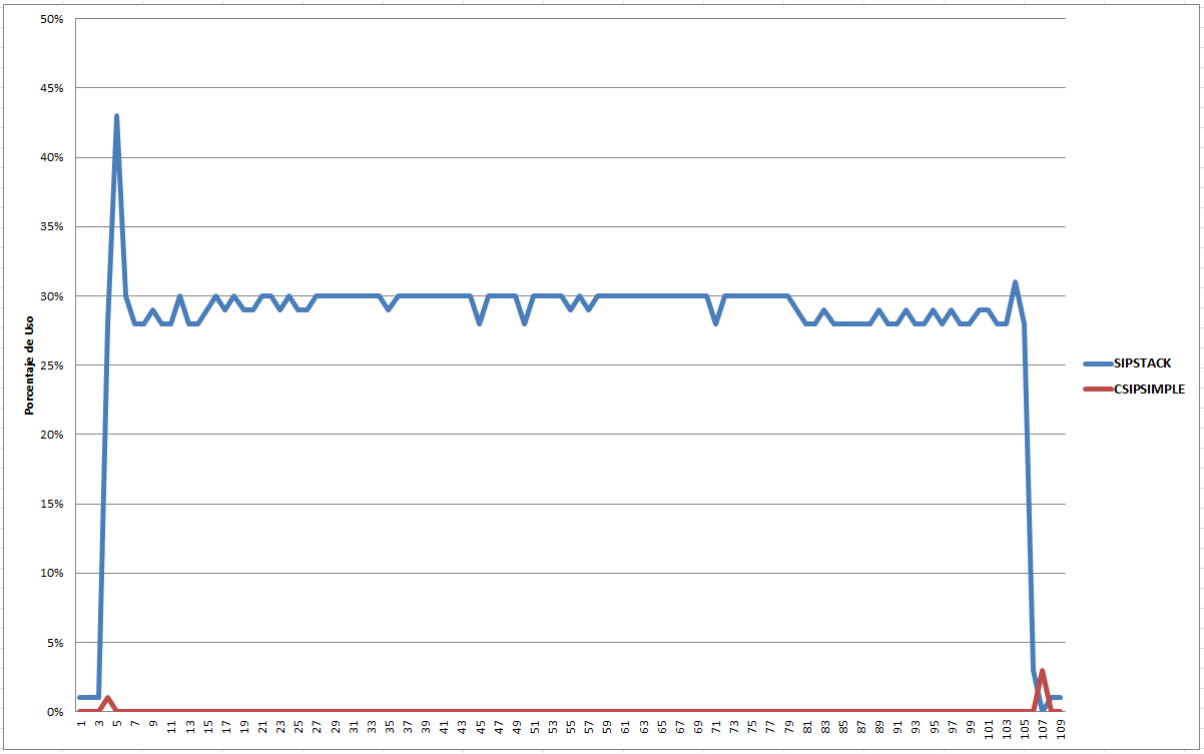


Figura 5.9: Consumo de CPU sin cifrar.

Tabla 5.6: Consumo de CPU con cifrador en GPP.

	CSIPSIMPLE	SIPSTACK
Cifrado:	--------	AES/CTR en GPP
Uso Min CPU:	0%	1%
Uso Max CPU:	3%	43%
Uso Promedio CPU:	0%	28%
Max Ancho de Banda:	-------	9.9 kB/s
Retardo Promedio:	--------	21.72 ms

Finalmente la figura 5.10 muestra el rendimiento obtenido cuando el esquema de cifrado es programado en el DSP como se propuso al inicio del diseño. En la gráfica se puede apreciar un tercer proceso, este hace referencia al binario del servidor diseñado con fines de comunicación entre ambas tareas de procesamiento, sin embargo la carga que este proceso presenta al CPU es mínima, haciendo un uso promedio de CPU del 2 % y uso máximo del 3 %.

Tabla 5.7: Consumo de CPU con cifrador en DSP.

	CSIPSIMPLE	SIPSTACK
Cifrado:	--------	AES/CTR en DSP
Uso Min CPU:	0%	1%
Uso Max CPU:	9%	27%
Uso Promedio CPU:	0%	20%
Max Ancho de Banda:	-------	9.9 kB/s
Retardo Promedio:	-------	20.08 ms

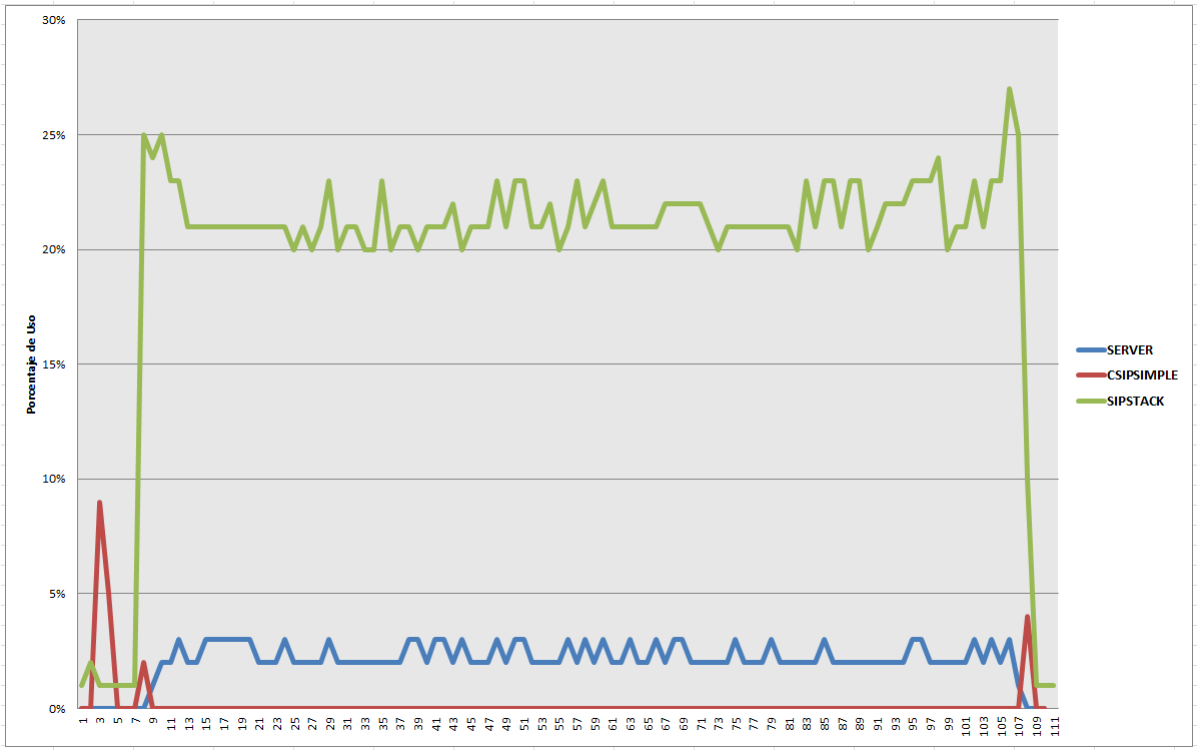


Figura 5.10: Consumo de CPU con cifrado en DSP.

La tabla 5.7 muestra el porcentaje de uso del procesador, en este caso el consumo promedio de CPU se mantuvo en el 20 % lo que demuestra que el procedimiento del cifrado se realiza en el DSP y con ello mejora el rendimiento del CPU. Se aprecia también que el retardo promedio al hacer uso del esquema de cifrado en el DSP conlleva un tiempo de 20.08 ms aproximadamente, ligeramente menor al que se genera en el GPP, sin embargo con esto se optimiza el uso de los recursos con los que se cuenta en el hardware y se hace un uso más eficiente del CPU.

Como se aprecia el ancho de banda en los 3 casos siempre permanece igual dado que el esquema de cifrado no genera paquetes adicionales ni agrega bytes a la cabecera RTP, lo cual permite que el esquema de cifrado no tenga complicaciones para operar bajo las circunstancias sobre las cuales ya opera la propia aplicación de VoIP.

A fin de comparar el trabajo realizado con otros trabajos, la tabla 5.8 muestra los datos analizados del esquema de cifrado propuesto, y el esquema de cifrado propuesto en [9].

Tabla 5.8: Tabla comparativa de esquemas de cifrado.

Aplicación VoIP	Kiax	CSIPSimple	Kiax	Kiax	CSIPSimple	CSIPSimple
Cifrado:	Ninguno	Ninguno	AES/CBC	AES/CFB	AES/CTR en GPP	**AES/CTR en DSP**
Uso Min CPU:	5.812%	1%	17.818 %	14.629%	1%	**1%**
Uso Max CPU:	10.020%	40%	27.427%	27.22%	43%	**27%**
Uso Promedio CPU:	7.935 %	15%	23.447%	22.592%	28%	**20%**
Max Ancho de Banda:	1.75 kB/s	9.9 kB/s	2.48 kB/s	1.75 kB/s	9.9 kB/s	**9.9 kB/s**
Rango de Retardo:	---------	---------	37 ms	37 ms	21.72ms	**20.08 ms**

De la tabla anterior observamos que la aplicación Kiax sin hacer uso del esquema de cifrado hace menor uso del CPU que la aplicación CSIPSimple, sin embargo al momento de comparar

los resultados se observa que el incremento del consumo de CPU en el mejor de los casos es de 17.02 %, es decir este consumo es el incremento cuando se cifra en Kiax con AES en modo CFB, en cambio la aplicaciòn CSIPSimple a pesar de usar un 7 % más de CPU sin emplear el esquema de cifrado, el incremento que muestra al cifrar en el GPP es de 13 % y en el DSP del 5 % lo cual muestra una mejora significativa en el esquema de cifrado propuesto en el presente trabajo.

5.5. Resumen de contenido.

Las pruebas realizadas y analizadas en este capítulo permitieron evaluar el esquema de cifrado propuesto, se observó que el rendimiento obtenido mejora el de trabajos anteriores. El consumo de CPU de 20 % y el retardo inyectado de 20.08 ms, se coloca por debajo del rendimiento otorgado por el trabajo presentado en [19].

Las pruebas de seguridad analizaron diferentes parámetros, mismos que corroboraron que el diseño del esquema de cifrado no presenta algún tipo de vulnerabilidad para los datos de voz que se transmiten en el medio de comunicación, ni tampoco para el propio algoritmo de cifrado.

Uno de los inconvenientes detectados en la prueba de concepto fue que la etapa de descifrado llevada a cabo en la tarjeta de desarrollo presenta la problemática detallada a continuación: después de ejecutar el esquema de cifrado durante un tiempo mayor a un minuto, la aplicación se interrumpe y el sistema operativo se pasma. Esta problemática no se resolvió de momento, sin embargo se cree que debido a que el flujo de datos en la etapa de descifrado contiene un bloque adicional que es el *jitter buffer*, éste genera un retardo más, que unido al retardo generado por el esquema de cifrado, provocan que el DSP no sea capaz de descifrar la totalidad de los datos entrantes y por ende, la aplicación CSIPSimple falle.

Conclusiones y Aportaciones.

En el presente trabajo fue realizado un esquema de cifrado capaz de proveer confidencialidad a los flujos de datos que se transmiten durante una llamada de VoIP. El esquema fue diseñado para operar sobre una aplicación que funciona bajo el sistema operativo Android haciendo uso del codificador de voz G.711. El esquema de cifrado opera en la plataforma de desarrollo Beagleboard-XM Rev C, y a fin de hacer que el uso de los recursos con los que se cuenta sea más eficiente, el algoritmo de cifrado simétrico AES se ejecuta en el procesador digital de señales.

Las pruebas de seguridad realizadas para analizar el comportamiento del esquema de cifrado y el nivel de confidencialidad que éste provee constataron que la manera en que se diseñó el esquema de cifrado no genera vulnerabilidades en el sistema ni fugas de información, dado que la fuente de información cuando se cifra, se comporta como una fuente de información aleatoria.

Fue obtenido un esquema de cifrado eficiente, que minimiza el retardo de tiempo inyectado aproximadamente en 2 ms menos que si se cifrará en el procesador de propósito general. El esquema de cifrado mejora los rendimientos de consumo de CPU en comparación a esquemas de cifrado propuestos por las investigaciones realizadas con anterioridad, indicadas en el capítulo dos.

Las principales aportaciones que otorga el presente trabajo son las siguientes:

- Un esquema de cifrado seguro y eficiente que hace uso del DSP con el que se cuenta en la plataforma de desarrollo, esto demuestra que el uso de los recursos con los que se cuenta mejora el esquema de cifrado considerablemente sin afectar la seguridad del mismo.
- Un esquema de comunicaciones que provee una solución a los problemas de memoria compartida que existen entre dos programas donde uno de ellos hace uso del DSP, esta solución permite que dos programas independientes puedan comunicarse de manera transparente.

Productos Obtenidos y Trabajo Futuro

Productos Obtenidos.

El desarrollo del presente trabajo genera una serie de productos, para los cuales actualmente se tiene el registro de software en trámite ante el Instituto Nacional de Derechos de Autor. Los siguientes componentes de software son los productos obtenidos en esta investigación.

- Generador de palabras cifrantes AES.
- Esquema de comunicación entre DSP y GPP.
- Rutina de cifrado para flujos de datos RTP en VoIP.

Los componentes de software generados conforman el esquema de cifrado propuesto.

Trabajo a Futuro.

Este proyecto tiene muchas posibilidades de mejora y desarrollo en el área de cómputo, entre ellas existe la posibilidad de desarrollar el esquema de cifrado para aplicaciones directamente sobre dispositivos móviles que empleen el sistema operativo Android. Esto si fuera posible desarrollar un framework capaz de explotar los recursos como el procesador digital de señales con el que se cuenta en un dispositivo móvil comercial, ya que hasta la fecha la arquitectura y el desarrollo de software que haga uso de este hardware es propio de los fabricantes.

El esquema de cifrado opera bajo el codec G.711 lo cual permite que se pueda realizar un cifrado de toda la carga útil del paquete RTP. Se propone realizar un cifrado selectivo en el cual solo se cifren los bytes de información necesarios para proveer confidencialidad, de esta manera el esquema de cifrado sería aún más eficiente.

Finalmente se propone la generación de una nueva señal de sincronía que proveea un contador con un tiempo de vida mayor al que se tiene a fin de evitar el ciclado del mismo en el tiempo que se mencionó. Esto generaría un vector de inicialización sin repetición y por lo tanto un esquema de cifrado más robusto.

Glosario

ACK Acknowledge.

ADC Convertidor Análogo-Digital.

AES Advanced Encryption Standart.

DAC Convertidor Digital-Análogo.

DSP Procesador Digital de Señales.

GPP Procesador de Propósito General.

GPRS General Packet Radio Service.

GSM Global System for Mobile Communications.

IAX Inter-Asterisk Exchange.

IPSec Secure Internet Protocol.

IP Internet Protocol.

MITM Main In The Middle Attack.

NIST National Institute of Standarts and Technology.

VoIP Voice over Internet Protocol.

PCM Modulación por pulsos codificados.

PSTN Public Switching Telephone Network.

SIP Session Initiation Protocol.

S.O. Sistema Operativo.

SRTP Secure Real Time Protocol.

UIT-T Unión Internacional de Telecomunicaciones.

VPN Virtual Private Network.

WLAN Wireless Local Area Network.

Bibliografía

[1] Calidad de transmisión telefónica métodos de evaluación objetiva y subjetiva de la calidad. *Telecommunication Standartization Sector of ITU P.830* (1996).

[2] The secure real-time transport protocol (srtp). *Request for Comments 3711* (2004).

[3] Códec de voz de doble velocidad para la transmisión en comunicaciones multimedia a 5,3 y 6,3 kbit/s. *Telecommunication Standartization Sector of ITU G.723* (2006).

[4] Cache coherency issue for dmm between arm and dsp. *Texas Instruments Digital Media Processors DM3730, DM3725* (2011).

[5] Modulación por impulsos codificados (mic) de frecuencias vocales. *Telecommunication Standartization Sector of ITU G.711* (2011).

[6] Coding of speech at 8 kbit/s using conjugate-structure algebraic-code-excited linear prediction. *Telecommunication Standartization Sector of ITU G.729* (2012).

[7] Coding of voice and audio signals. 7khz audio-coding within 64 kbits/s. *Telecommunication Standartization Sector of ITU G.722* (2012).

[8] Ti-android-ics-4.0.3-devkit-3.0.0 developers guide. *Texas Instruments 3.0.0* (2013).

[9] Alex Talevski, E. C., and Dillon, T. Secure mobile voip. *International Conference on Convergence Information Technology* (2007).

[10] Alfred J. Menezes, P. v. O. *Handbook of applied criptography.* CRC Press, 2001.

[11] Arturo Baz Alonso, I. F. Dispositivos móviles. *Universidad de Oviedo* (2012).

[12] Bryant, R. Asterisk 1.8 documentation. *Wiki Asterisk 1.8* (2010).

[13] Burke, D. *Speech Processing for IP Networks.* John Wiley, Ltd, 2007.

[14] Dworkin, M. Recommendation for block cipher modes of operation. *National Institute of Standards and Technology 800-38A* (2001).

[15] Guo Fang Mao, Alex Talevski, E. C. Voice over internet protocol on mobile devices. *International Conference on Computer and Information Science* (2007).

[16] José Manuel Huidobro Moya, D. R. M. Tencnología voip y telefonía ip. *Editorial Grupo Alfaomega* (2004), 11–20.

[17] S. Kamali, R. S. A new modified version of advanced encryption standart based algorithm for image encryption. *Electronics and Information Engineering International Conference* (2010).

[18] Sascha Fahl, Marian Harbach, M. S. Why eve and mallory love android: An analysis of android ssl security. *Philipps University of Marburg* (2012).

[19] Sergio Chacon, Driss Benhaddou, D. G. Secure voice over internet protocol (voip) using virtual provate networks (vpn) and internet protocol security (ipsec). *IEE* (2006).

[20] Stamp, M. *Information Security principles and practice.* John Wiley Sons, Inc., 2007.

[21] Valin, J.-M. The speex codec manual. *Xiph.org Foundation Versión 1.2* (2007).

[22] Washington, L. C. *Introduction to Cryptography with Coding Theory.* Pearson Prentice Hall, 2007.

[23] Yeun, C. Y., and Al-Marzouqi, S. M. Practical implementations for securing voip enabled mobile devices. *Third International Conference on Network and System Security* (2010).

ANEXOS

Anexo A

Instalacion Android 4.0.3 ICS

A fin de realizar una instalación limpia del sistema operativo android, mas allá del manual que provee Texas Instruments a continuación se detallan los pasos seguidos para su instalación y la corrección de errores durante la compilación del sistema.

Es necesario actualizar el sistema e instalar las siguientes dependencias:

```
$ sudo add-apt-repository "deb http://archive.canonical.com/ lucid partner"
$ sudo apt-get update
$ sudo apt-get install git-core gnupg flex bison gperf libsdl-dev
  libesd0-dev libwxgtk2.6-dev build-essential zip curl libncurses5-dev
  zlib1g-dev minicom tftpd uboot-mkimage expect
```

Es necesario contar con la versión de java 6 a fin de poder compilar la imagen del sistema operativo. Una vez que se cuenta con estos requerimientos se procede a obtener los recursos de la imagen del sistema operativo. Para ello se ejecuta lo siguiente:

```
$mkdir ~/bin
$PATH=~/bin:$PATH
$curl https://dl-ssl.google.com/dl/googlesource/git-repo/repo > ~/bin/repo
$chmod a+x ~/bin/repo
$mkdir $HOME/rowboat-android
$cd $HOME/rowboat-android
$repo init -u git://gitorious.org/rowboat/manifest.git -m TI-Android-ICS-4.0.3-DevKit-3.0.0.xml
$repo sync
```

Estos comandos crearán una carpeta en el directorio raiz donde se depositarán los recursos necesarios de la imagen del sistema operativo, entre ellos se encuentra el sistema de archivos, el kernel de linux a emplear y el cross compiler que es necesario configurar en la terminal para poder realizar la compilación para la arquitectura ARM. El siguiente comando configura el cross compiler.

```
$ export PATH=$HOME/rowboat-android/prebuilt/linux-x86/toolchain/arm-eabi-4.4.3/bin:$PATH
```

A continuación se construyen los módulos necesarios para el sistema operativo, se contruye el x-loader, el cual generará un archivo binario llamado x-load.bin.

```
$ cd rowboat-android/x-loader
$ make CROSS_COMPILE=arm-eabi- distclean
$ make CROSS_COMPILE=arm-eabi- omap3beagle_config
$ make CROSS_COMPILE=arm-eabi-
```

Es necesario crear un archivo llamado MLO que permite iniciar el sistema operativo. Para ello es necesario contar con el archivo binario signGP, se copia el ejecutable a la carpeta rowboat-android/x-loader y se ejecuta:

```
$ ./signGP ./x-load.bin
$ mv x-load.bin.ift MLO
```

A continuación se construye el Bootloader el cual creará el archivo binario u-boot.bin

```
$ cd rowboat-android/u-boot
$ make CROSS_COMPILE=arm-eabi- distclean
$ make ARCH=arm CROSS_COMPILE=arm-eabi-  omap3_beagle_config
$ make ARCH=arm CROSS_COMPILE=arm-eabi-
```

Se construye el kernel de linux que empleará el sistema operativo Android, en él se habilitará el DSP Bridge a fin de poder hacer uso del procesador digital de señales.

```
$ cd rowboat-android/kernel
$ make ARCH=arm CROSS_COMPILE=arm-eabi- distclean
$ make ARCH=arm CROSS_COMPILE=arm-eabi- omap3_beagle_android_defconfig
$ make ARCH=arm CROSS_COMPILE=arm-eabi- uImage
```

A fin de habilitar el DSP ejecutar o siguiente:

```
$ make menuconfig ARCH=arm CROSS_COMPILE=arm-eabi-
```

En la sección Menuconfig->Drivers->StagingDrivers->DspBridge[*] activar la casilla como se muestra y salvar la configuración. Ejecutar lo siguiente:

```
$ make ARCH=arm CROSS_COMPILE=arm-eabi-
$ make ARCH=arm CROSS_COMPILE=arm-eabi- uImage
```

Estos comandos generarán la uImage, la imagen del kernel en kernel/arch/arm/boot con el dsp habilitado. A continuación se construirá el sistema de archivos de Android. Ejecutar lo siguiente:

```
$ cd rowboat-android/
$ make TARGET_PRODUCT=beagleboard OMAPES=5.x -j2
```

Al momento de realizar la compilación se generarán una serie de errores que se corrigen de la siguiente manera. Instalar las siguientes dependencias:

```
$ sudo apt-get install git gnupg flex bison gperf build-essential zip curl libc6-dev
libncurses5-dev:i386 x11proto-core-dev libx11-dev:i386 libreadline6 dev:i386
libgl1-mesa-glx:i386 libgl1-mesa-dev g++-multilib mingw32 tofrodos python-markdown
libxml2-utils xsltproc zlib1g-dev:i386
```

Corregir los siguientes errores al momento de compilar.

Error:
*** [out/host/linux-x86/obj/EXECUTABLES/obbtool_intermediates/Main.o] Error 1
Solución:

```
$ gedit build/core/combo/HOST_linux-x86.mk
```

```
1 Reemplazar        HOST_GLOBAL_CFLAGS += -D_FORTIFY_SOURCE=0
2 Por: HOST_GLOBAL_CFLAGS += -U_FORTIFY_SOURCE -D_FORTIFY_SOURCE=0
```

ERROR:
***[out/host/linux-x86/obj/STATIC_LIBRARIES/liboprofile_pp_intermediates/arrange_profiles.o]

Solución:

```
$ gedit external/oprofile/libpp/format_output.h
```

```
Reemplazar      mutable counts_t & counts;
Por             counts_t & counts;
```

Error: ***[.....external/gtest/src/gtest_main.o] Error 1
Solución:

```
$ gedit external/gtest/src/../include/gtest/internal/gtest-param-util.h
```

Agregar la siguiente línea al inicio de las declaraciones del programa.

```
#include <cstddef>
```

Error:
*** [out/host/linux-x86/obj/EXECUTABLES/test-librsloader_intermediates/test-librsloader]
Solución:

```
$ gedit external/llvm/llvm-host-build.mk
```

Agregar la siguiente línea al principio del archivo.

```
LOCAL_LDLIBS := -lpthread -ldl
```

Error:
***[out/host/linux-x86/obj/EXECUTABLES/llvm-rs-cc_intermediates/slang_rs_export_foreach.o]

Solución:

```
$ gedit frameworks/compile/slang/Android.mk
```

```
Reemplazar:
local_cflags_for_slang:= -Wno-sign-promo -Wall -Wno-unused-parameter -Werror

Por:
local_cflags_for_slang:= -Wno-sign-promo -Wall -Wno-unused-parameter
```

Una vez realizado el proceso de compilación exitosamente realizar lo siguiente:

```
$ cd out/target/product/beagleboard
$ mkdir android_rootfs
$ cp -r root/<*> android_rootfs
$ cp -r system android_rootfs
$ ../../../../build/tools/mktarball.sh ../../../host/linux-x86/bin/fs_get_stats \\
  android_rootfs . rootfs rootfs_.tar.bz2
```

El archivo rootfs.tar.bz2 será el sistema de archivos del sistema operativo. A continuación se copiarán los archivos generados a una carpeta en común y se procederá a realizar la instalación del sistema operativo en la tarjeta sd. Ejecutar lo iguiente:

```
$ mkdir ~/image_folder
```

```
$ cp <android sorce path>/kernel/arch/arm/boot/uImage ~/image_folder
$ cp <android souorce path>/u-boot/u-boot.bin ~/image_folder
$ cp <android source path>/x-loader/MLO ~/image_folder
$ cp ~/TI-DevKit-Android/mk-bootscr/boot.scr ~/image_folder
$ cp <android source path>/out/target/product/beagleboard/rootfs.tar.bz2 ~/image_folder
$ cp ~/TI-DevKit-Android/mk-mmc/mkmmc-android.sh ~/image_folder
```

Modificar las siguiente línea a fin de ejecutar el comando que instale el sistema operativo en la tarjeta sd. Abrir el archivo mkmmc-android.sh en un gedit y editar lo siguiente:

```
1 Linea 66: SIZE=‘fdisk -l $DRIVE | grep disk | awk ’{print $5}’‘
2 Por   66: SIZE=‘fdisk -l $DRIVE | grep $DRIVE | awk ’{print $5}’‘
```

Se inserta la tarjeta sd y se ejecuta el siguiente comando:

```
$ cd ~/image_folder
$ sudo ./mkmmc-android.sh <your SD card device e.g.:/dev/sdc> MLO u-boot.bin uImage boot.scr\\
  rootfs.tar.bz2
```

El sistema operativo ahora está en la tarjeta sd y puede ser ejecutado.

Anexo B

Instalación Asterisk

A continuación se describe el procedimiento para contar con un servidor de VoIP, en este caso la centralita de VoIP Asterisk. Los requisitos necesarios para la instalación es contar con algún equipo que cuente con el sistema operativo Ubuntu 12.04 LTS, que se considera una de las versiones más estables, descargar el software asterisk versiòn 1.8 del centro de descargas en el siguiente link:

- $http://www.asterisk.org/downloads/asterisk/all-asterisk-versions$

Es necesario contar con todas las actualizaciones del sistema operativo para ello se ejecuta lo siguiente:

```
$ sudo apt-get update
```

Con el sistema operativo actualizado se instalan las siguientes dependencias:

```
$ sudo apt-get install bison ncurses-dev libssl-dev libnewt-dev cvs procps
  debhelper dpkg-dev gettext html2text po-debconf build-essential automake
  flex libtool libncurses5-dev libssl-dev
$ sudo apt-get install -qy zlib1g-dev libiksemel-dev
$ sudo apt-get install libxml2-dev
$ sudo apt-get install sqlite3 libsqlite3-dev
$ sudo apt-get install rubygems 1.9.1
$ sudo apt-get install ruby-full build-essential
$ sudo aptitude install ruby build-essential libopenssl-ruby ruby1.8-dev
$ sudo gem install sqlite3-ruby
```

Si en algún momento de instalar alguna dependencia algún error se produce, basta con observar el mensaje de error e instalar la dependencia que pide, este caso varía de acuerdo a las actualizaciones que cada una de las dependencias vaya liberando al momento de realizar la instalación.

Con el archivo del software asterisk descargado, se procede a extraer su contenido e ingresar al directorio generado a fin de realizar lo siguiente en una terminal de comandos:

```
$ ./configure
$ sudo make
$ sudo make install
```

Una vez que se cuenta con el sistema instalado es necesario configurar los parámetros que indicarán la manera en que el servidor asterisk permitirá el acceso a los usuarios, para ello se editan dos archivos principalmente, *sip.conf* y *extensions.conf*. Es necesario abrir los archivos como usuario administrador para que el sistema permita la modificación de éstos. El primer archivo a modificar será el que contenga la configuración general *sip.conf*.

Se agregan las siguientes líneas al final del archivo.

```
udpbindaddr= 192.168.1.110:5060 // Establece la direccion IP
                                // y puerto del servidor.

[2010]                          // Primer Usuario
type=friend
secret=****
context=SeguridadCIC
disallow=all
allow=alaw
allow=ulaw

[2020]                          // Segundo Usuario
type=friend
secret=****
context=SeguridadCIC
disallow=all
allow=alaw
allow=ulaw
```

Se procede a configurar el segundo archivo *extensions.conf*, éste establecerá las reglas de marcación que se seguirán para cada una de las extensiones. Se añaden los siguientes parámetros al final del archivo.

```
[SeguridadCIC]
exten => 2010,1,Dial(SIP/2010,30,Ttm)
exten => 2010,2,Hangup
exten => 2010,102,Voicemail(2010)
exten => 2010,103,Hangup
```

Una vez que se tienen configurados los parámetros necesarios se puede iniciar el servidor. Para ello se ejecuta el siguiente comando:

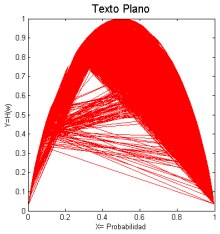
```
$ sudo asterisk -vvc
```

Una vez ejecutado se observará que el servidor está corriendo, cada vez que un usuario se conecte puede ser monitoreado desde la terminal.

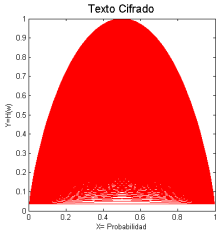
Anexo C

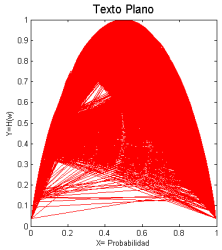
Gráficas de pruebas realizadas

C.1. Entropia

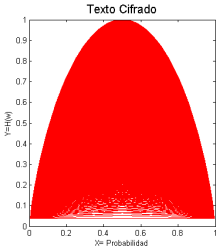


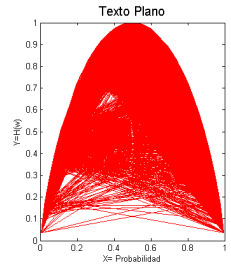
(a) Sujeto 1



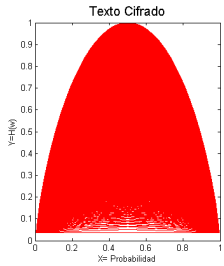


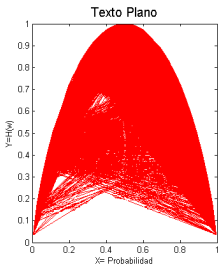
(b) Sujeto 2



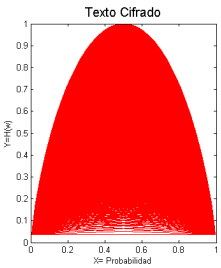


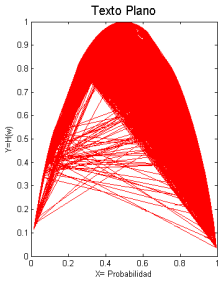
(c) Sujeto 3



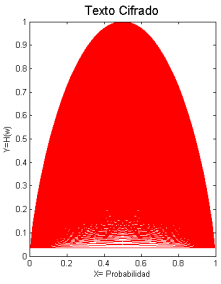


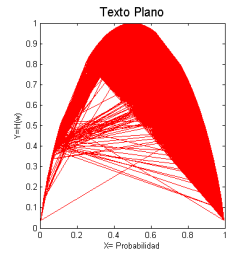
(d) Sujeto 4



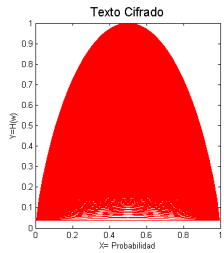


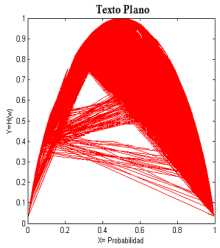
(e) Sujeto 5



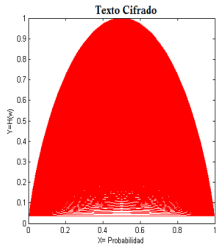


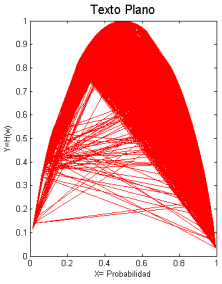
(f) Sujeto 6



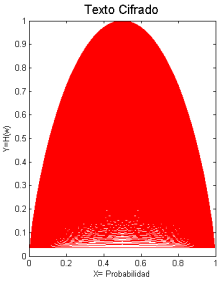


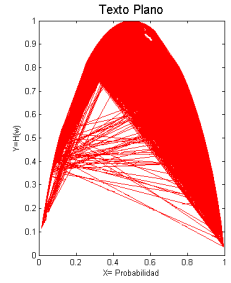
(g) Sujeto 7



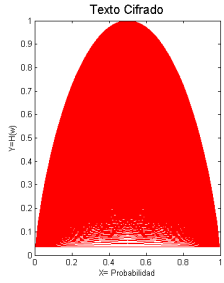


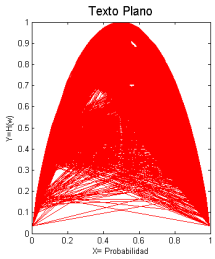
(h) Sujeto 8

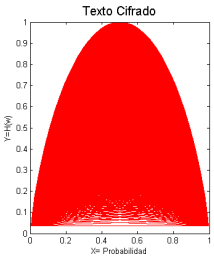




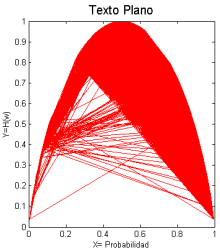
(i) Sujeto 9

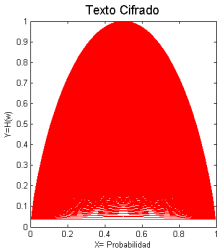




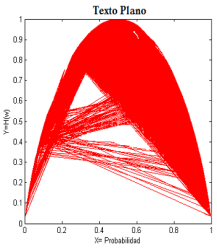


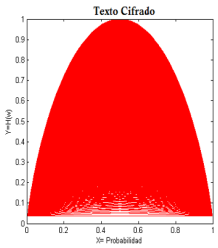
(j) Sujeto 10



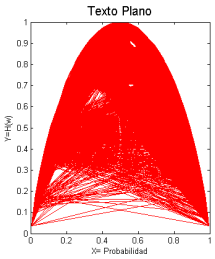


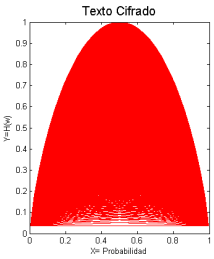
(k) Sujeto 11



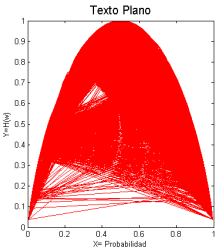


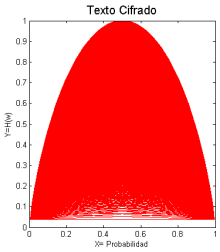
(l) Sujeto 12



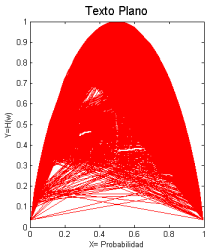


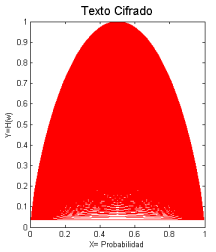
(m) Sujeto 13



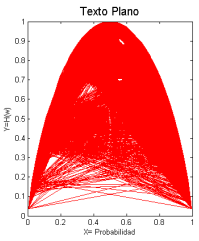


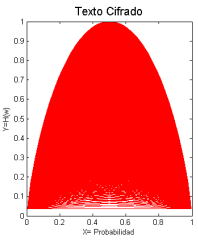
(n) Sujeto 14



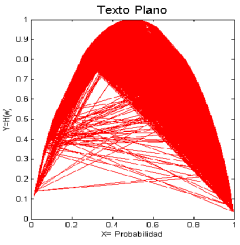


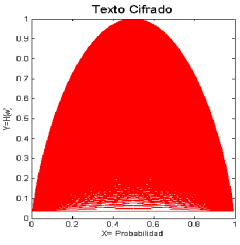
(ñ) Sujeto 15



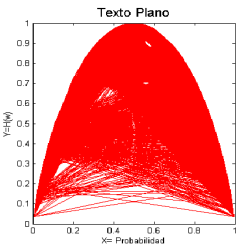


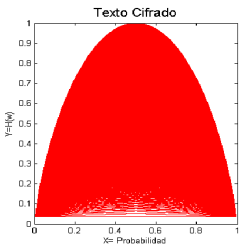
(o) Sujeto 16



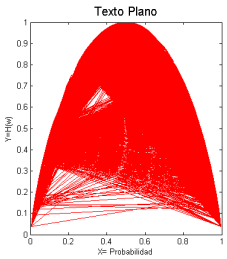


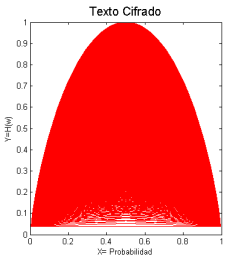
(p) Sujeto 17



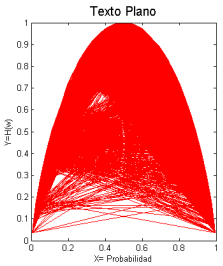


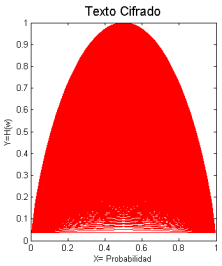
(q) Sujeto 18





(r) Sujeto 19

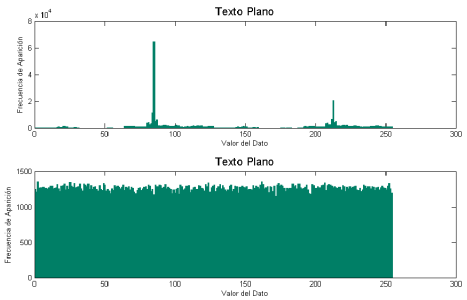




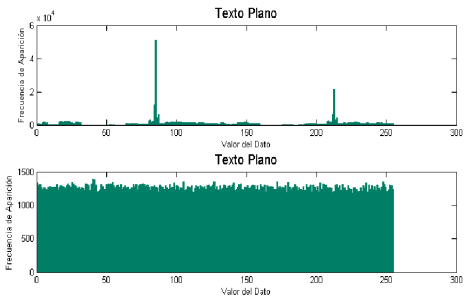
(s) Sujeto 20

Figura C.1: Entropía de las 20 Pruebas

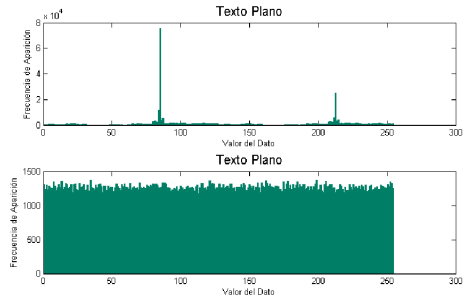
C.2. Histograma



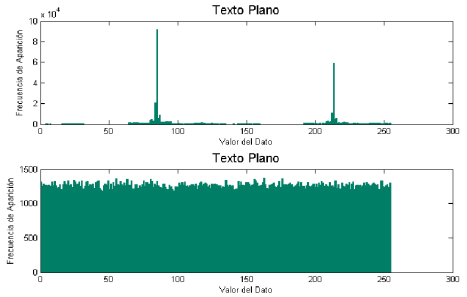
(a) Sujeto 1



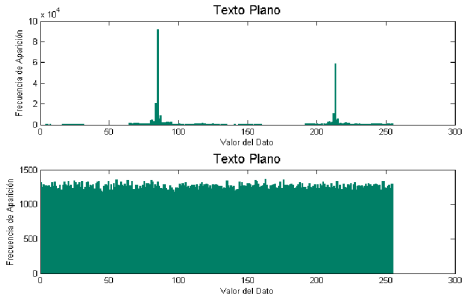
(b) Sujeto 2



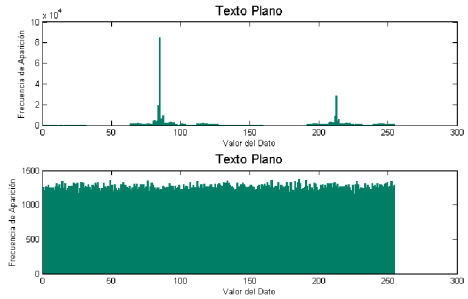
(c) Sujeto 3



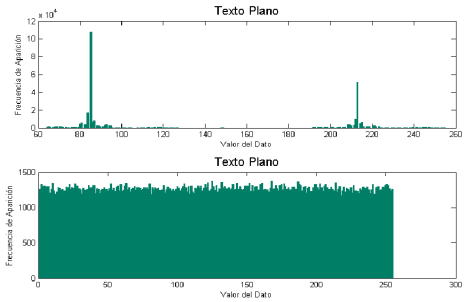
(d) Sujeto 4



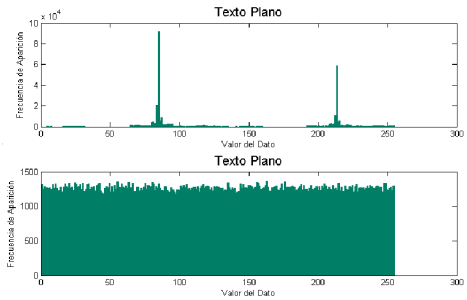
(e) Sujeto 5



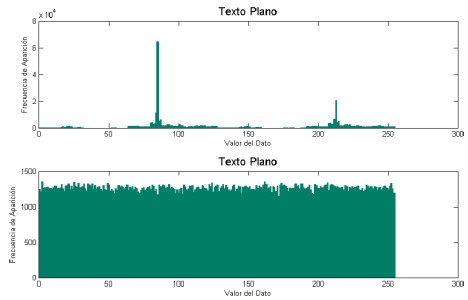
(f) Sujeto 6



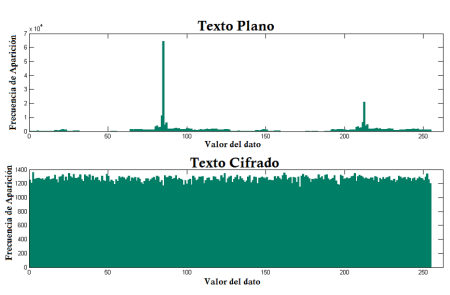
(g) Sujeto 7



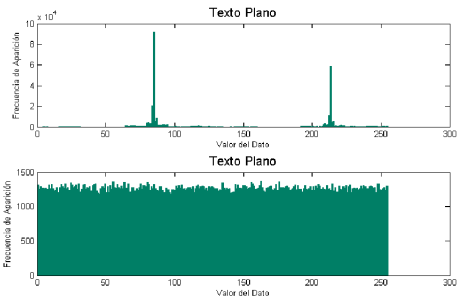
(h) Sujeto 8



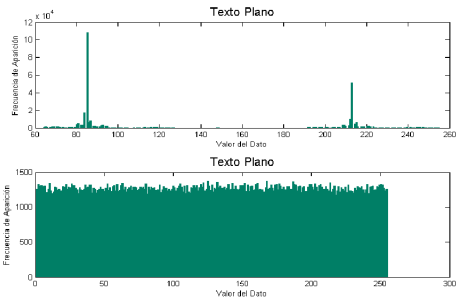
(i) Sujeto 9



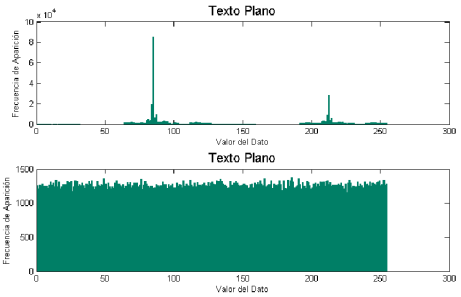
(j) Sujeto 10



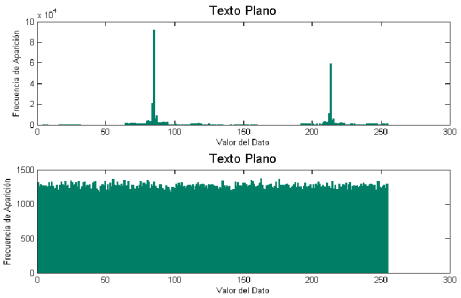
(k) Sujeto 11



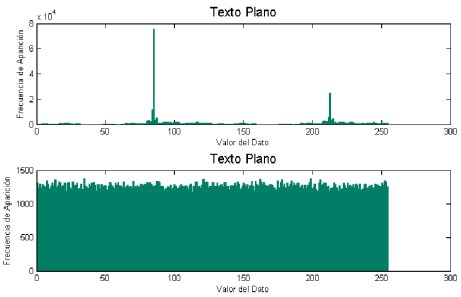
(l) Sujeto 12



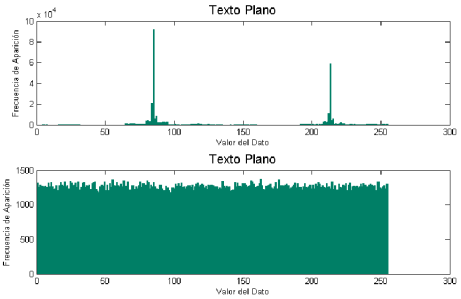
(m) Sujeto 13



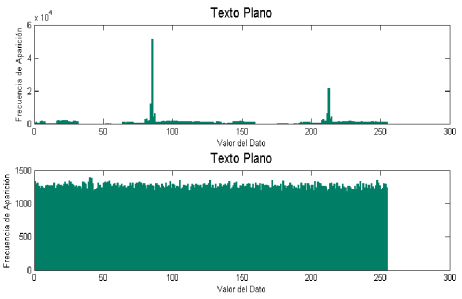
(n) Sujeto 14



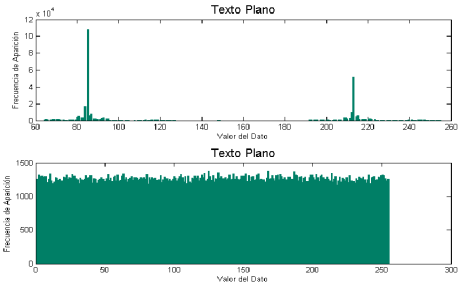
(ñ) Sujeto 15



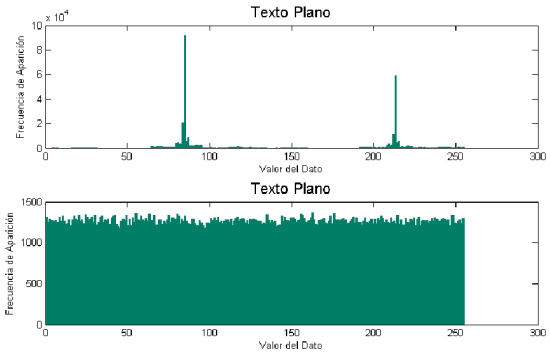
(o) Sujeto 16



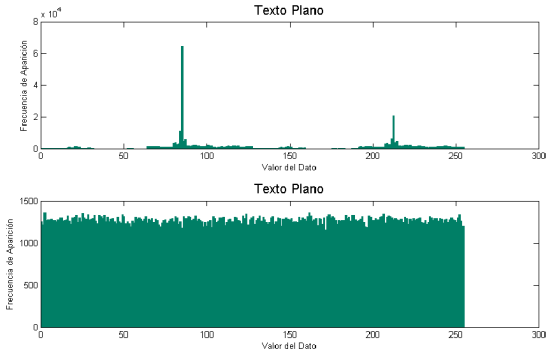
(p) Sujeto 17



(q) Sujeto 18



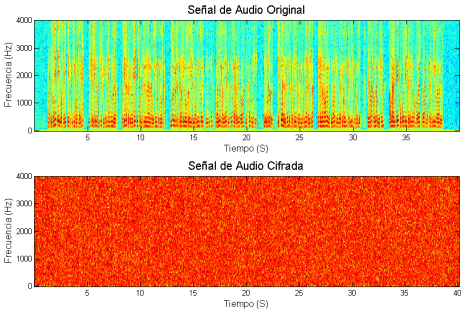
(r) Sujeto 19



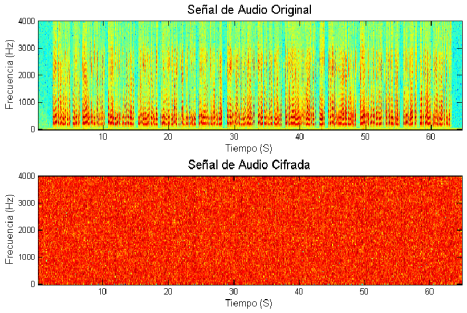
(s) Sujeto 20

Figura C.2: Histograma de las 20 Pruebas

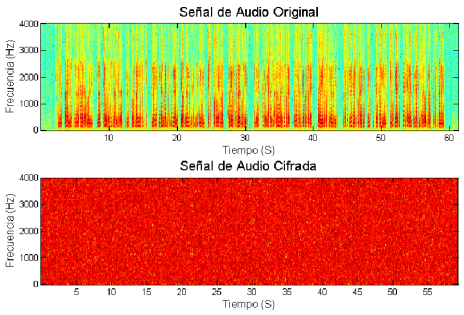
C.3. Espectrograma



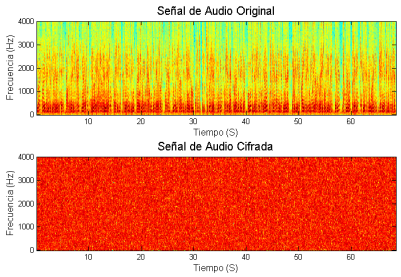
(a) Sujeto 1



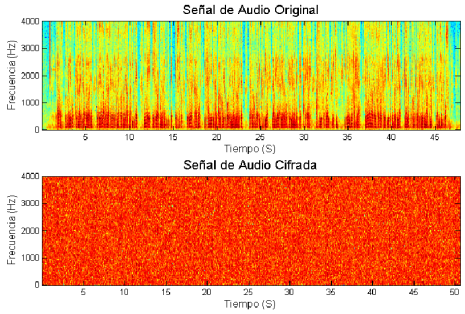
(b) Sujeto 2



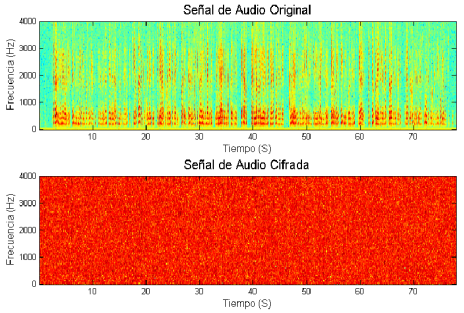
(c) Sujeto 3



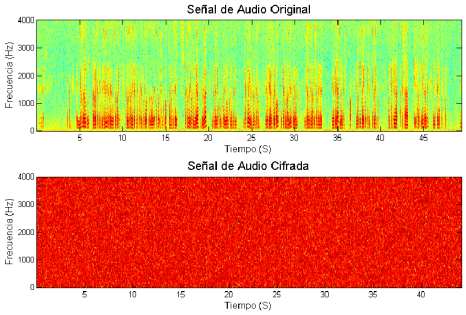
(d) Sujeto 4



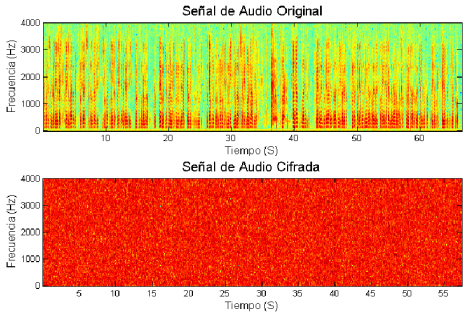
(e) Sujeto 5



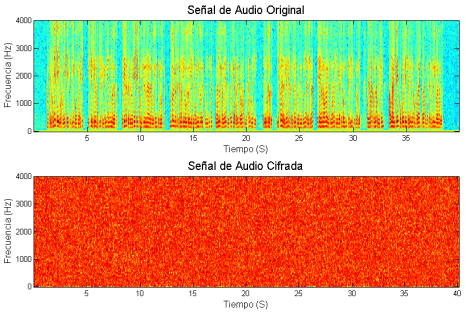
(f) Sujeto 6



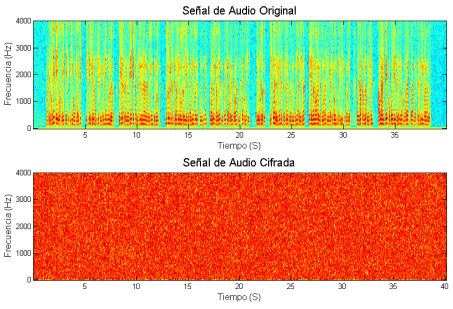
(g) Sujeto 7



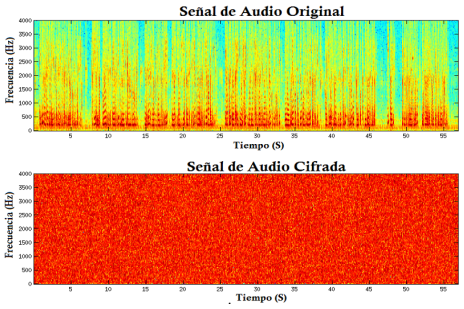
(h) Sujeto 8



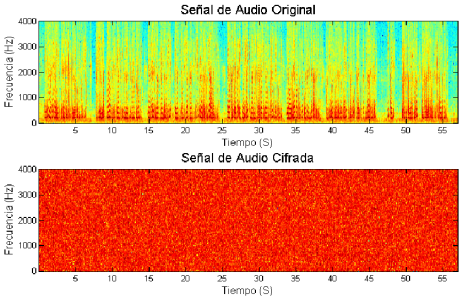
(i) Sujeto 9



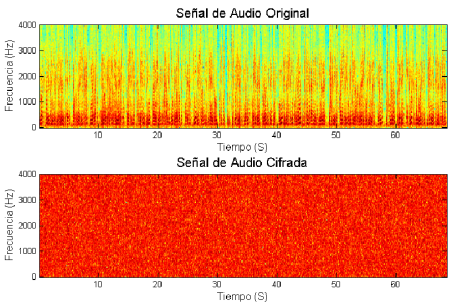
(j) Sujeto 10



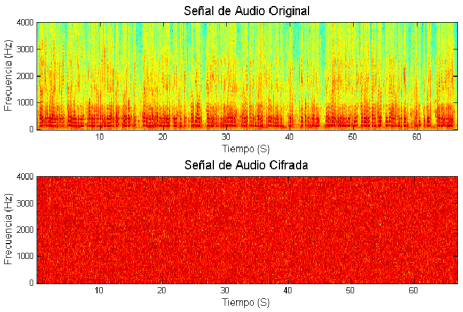
(k) Sujeto 11



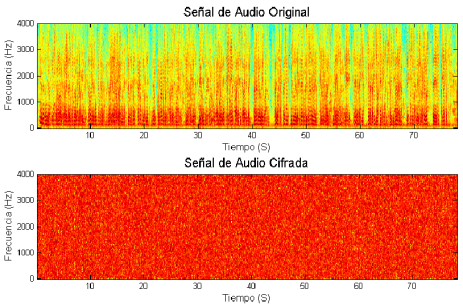
(l) Sujeto 12



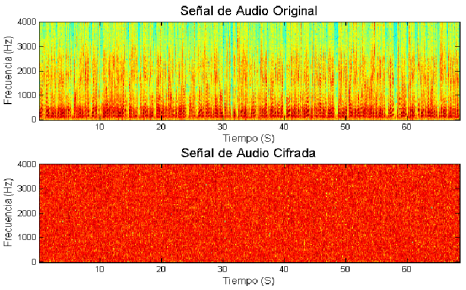
(m) Sujeto 13



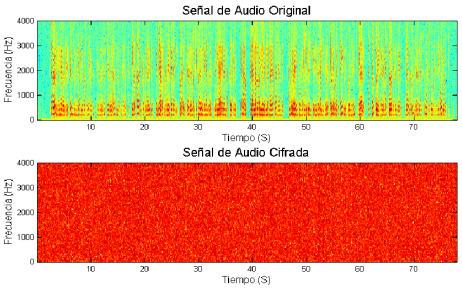
(n) Sujeto 14



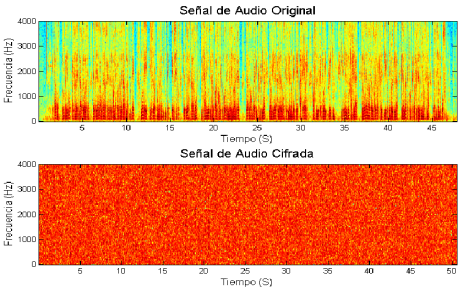
(ñ) Sujeto 15



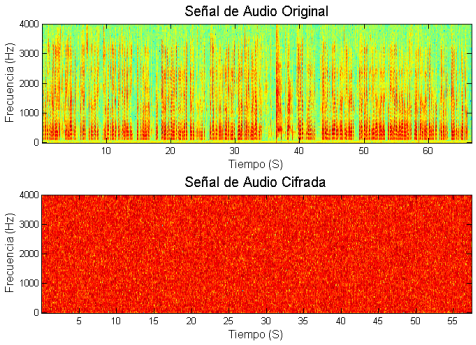
(o) Sujeto 16



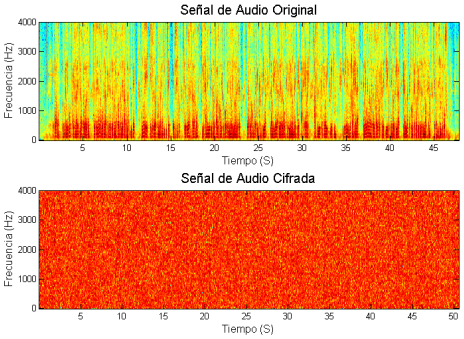
(p) Sujeto 17



(q) Sujeto 18



(r) Sujeto 19



(s) Sujeto 20

Figura C.3: Espectrograma de las 20 Señales