Esquema de Cifrado y Compresión sin Pérdida para datos de ECG en Telemedicina

por

Gershom de Jesús Trinidad Blas

Tesis de Maestría sometida como requisito parcial para obtener el grado de Maestro en Ciencias en la especialidad de Ciencias Computacionales en el Instituto Nacional de Astrofísica, Óptica y Electrónica

2007

Tonantzintla, Puebla

Revisores:

Dra. Claudia Feregrino Uribe, INAOE

Dr. René A. Cumplido Parra, INAOE

Abstract

Telemedicine is a development and growth area that is based on the use of communication technologies. In this area, the patient's information is sent through computer networks, where the most commonly used today, is the wireless network.

The generated information, in many of the applications of telemedicine, tends to increase and thus, there is a need for availability of adequate spaces and storage media. Moreover, with the growth of information and the use of wireless systems the saturation of the media transmission in less time turns more evident. Also, the information security is affected by the use of these kinds of technologies, because the air is used as medium and is more vulnerable to attacks that affect the confidentiality and integrity of the data.

On the other hand, the compression and encryption are technologies that offer solutions that are used in network applications. The first one allows a reduction of information, achieving a better performance of the storage and transmission technology. The second technology provides the confidentiality and integrity of the information.

Mentioned the previous thing, this research presents a compression and encryption scheme that can be applied in telemedicine, mainly in wireless and mobile environments. In this dissertation work, an analysis of compression and encryption of algorithm is carried out to determine the best performance and adaptation to the telemedicine area. For the aforementioned, aspects as computational complexity, proccess speed, resources required to operate, among others, are taken into account.

Likewise, this paper presents a series of procedures to increase compression ratio of ECG data type. The algorithms, both compression and encryption, are implemented in software for validating the proposed scheme.

Resumen

La Telemedicina es un área con gran desarrollo y crecimiento que está basada en la utilización de tecnologías de comunicación. En dicha área, se maneja información del paciente principalmente por redes de computadoras, donde la más utilizada actualmente es la red inalámbrica.

La información generada, en muchas de las aplicaciones de telemedicina, tiende a incrementar y con ello, es necesaria la disponibilidad del espacio suficiente y medios de almacenamiento. Además, con el crecimiento de la información y con el uso de sistemas inalámbricos, resulta más evidente la saturación del medio de transmisión en un menor tiempo. También, la seguridad de tales datos se ve afectada al utilizar este tipo de medios de transmisión, ya que se utiliza al aire como medio y resulta más vulnerable a ataques que afecten la confidencialidad e integridad de los mismos.

Por otro lado, la compresión y cifrado de datos son tecnologías que ofrecen soluciones y que son utilizadas en aplicaciones de red. La primera permite la reducción de información, logrando así un mejor aprovechamiento del medio de almacenamiento y de transmisión. La segunda tecnología ofrece la confidencialidad e integridad de la información.

En este trabajo de investigación se crea un esquema de compresión y cifrado de datos que puede ser aplicado en telemedicina con datos de electrocardiograma (ECG), factible para ambientes inalámbricos y móviles. Se realiza un análisis de los algoritmos de compresión y cifrado para así determinar el mejor desempeño y adaptación al área de telemedicina. Para lo anterior se toman en cuenta aspectos de complejidad computacional, velocidad de procesamiento, recursos necesarios para operar, entre otros.

De igual forma, se presenta una serie de procedimientos que permiten aumentar la razón de compresión para datos tipo ECG y así, conseguir tanto un mejor aprovechamiento del medio de transmisión como de los medios de almacenamiento.

Los algoritmos, tanto de compresión y cifrado, son implementados en software para la validación del esquema propuesto.

Agradecimientos

A mis asesores, la Dra. Claudia Feregrino Uribe, por su orientación, apoyo, paciencia y tiempo; y el Dr. René Armando Cumplido Parra por sus consejos, críticas y respaldo.

A mis revisores de tesis, Dr. Luis Villaseñor Pineda, Dr. Gustavo Rodríguez Gómez y Dr. Saúl Eduardo Pomares Hernández, por sus observaciones.

Al Instituto Nacional de Astrofísica. Óptica y Electrónica (INAOE) por la formación académica y por los variados servicios.

A los patrocinadores anónimos, que por medio de CONACyT, permitieron la realización de esta investigación.

A mi familia, que ha sido fuente de inspiración y motivación para seguir adelante. A mí madre, la Enf. Blanca Estela Blas Chiñas, por su apoyo y amor incondicional. A mi padre, el señor Humberto Trinidad, por estar en los momentos de alegría y adversidades. A mis segundos padres, mis abuelos, los señores Leonides Blas Orozco y su distinguida esposa la señora Cecilia Chiñas de Blas, por su digno ejemplo de amor y de que las cosas sí se pueden cuando se desean. A mis hermanos, tíos, tías, primos, primas, sobrinas y sobrinos por su confianza.

A mi novia, la Lic. Martha E. Ordaz Azamar, por su gran amor, consejos, apoyo, respaldo, críticas y paciencia durante todo este tiempo.

A mis amigos por su apoyo, motivación y confianza. Particularmente, al MC. Elías Ruiz Hernández, por su apoyo e inspiración.

A las personas, que de forma indirecta, ayudaron a la realización de este trabajo.

…Gracias.

A mi madre: Enf. Blanca Estela Blas Chiñas

una madre, deja todo por su hijo;
una madre, cambia todo por su hijo;
una madre, arriesga todo por su hijo;
una madre ama con todo a su hijo;
sí, eso hace mi madre.

Índice General

CAPÍTULO 1 – Introducción

1. Introducción ……...….1

1.1. Motivación ……...1
1.2. Objetivo ……...4
1.3. Método ……5
1.4. Organización de la tesis ……..6

CAPÍTULO 2 - Antecedentes

2. Introducción ……9

2.1. Telemedicina ……...9
2.1.1. Estándares ……9
2.1.2. Tendencia ……10
2.2. Redes de Computadora ……..10
2.3. Compresión de datos ……12
2.3.1. Compresión de datos con pérdida ……...14
2.3.2. Compresión de datos sin pérdida ……14
2.3.2.1. RLE ……16
2.3.2.2. Huffman estático ……17
2.3.2.3. Huffman dinámico ……...19
2.3.2.4. Codificador aritmético ……20
2.3.2.5. LZSS ……22
2.3.2.6. LZW ……23
2.4. Cifrado de datos ……24
2.4.1. Cifradores simétricos ……..25
2.4.2. Cifradores de flujo ……...26
2.4.2.1. RC4 ……..26
2.4.3. Cifradores de bloque ……...28
2.4.3.1. DES ……29
2.4.3.2. AES ……32
2.4.4. Cifradores asimétricos ……34
2.5. Resumen de capítulo ……..34

CAPÍTULO 3 – Estado del Arte

3. Introducción ……..36

3.1. Telemedicina ……36
3.2. Compresión en telemedicina ……38
3.3. Seguridad en telemedicina ……42
3.4. Aplicaciones para telemedicina ……46
3.5. Resumen de capítulo ……..48

CAPÍTULO 4 – Compresión y Cifrado: Análisis

4. Introducción ……..50

4.1. Consideraciones ……50
4.2. Comparación entre algoritmos de compresión ……..52
4.2.1. Huffman ……..52
4.2.2. Codificador aritmético ……57
4.2.3. LZW ……62
4.2.4. LZSS ……...65
4.2.5. RLE ……66
4.3. Comparación entre algoritmos de cifrado ……68
4.3.1. AES ……71
4.3.2. DES ……74
4.3.3. RC4 ……76
4.4. Selección de algoritmos de compresión y cifrado ……78
4.5. Compresión-cifrado de datos ……..……...85
4.6. Resumen de capítulo ……..88

CAPÍTULO 5 – Módulos y Resultados

5. Introducción ……..90

5.1. Byte-Byte ……...91
5.2. 12-Bits ……...92
5.3. 12-Bits-Swap ……95
5.4. Byte-Byte-Swap ……96
5.5. División ……98
5.6. División-LZW-RLE ……102
5.7. Different ……..104

5.8. Different-LZW-RLE ……108
5.9. Divide2Different-LZW ……114
5.10. Divide2Different-LZW-RLE ……116
5.11. Comparación en compresión ……..118
5.12. Comparación en cifrado ……..124
5.13. Resumen de capítulo……129

CAPÍTULO 6 - Conclusiones

6. Introducción ……131

6.1. Objetivos de tesis ……131
6.2. Objetivos alcanzados ……...132
6.3. Conclusiones ……134
6.4. Trabajo futuro ……..135

Índice de Figuras, Tablas y Ecuaciones

CAPÍTULO 1

Figura 1.1-Esquema de red en un hospital ……....3
Figura 1.2-Diagrama de método a seguir ……....7

CAPÍTULO 2

Figura 2.1-Algoritmo para la construcción del árbol de Huffman ……....18
Figura 2.2-Compresor LZ77 ……...22
Figura 2.3-Esquema de Cifradores Simétricos ……...26
Figura 2.4-Esquema de cifrado RC4 ……..28
Figura 2.5-Esquema de cifrado DES ……..31
Figura 2.6-Esquema de cifrado AES ……..33

Tabla 2.1- División de tecnologías inalámbricas…….12

Ecuación 2.1 ……...13
Ecuación 2.2 ……...18
Ecuación 2.3 ……...18
Ecuación 2.4 ……...19
Ecuación 2.5 ……...19
Ecuación 2.6 ……...21
Ecuación 2.7 ……...21
Ecuación 2.8 ……...21
Ecuación 2.9 ……...21
Ecuación 2.10 …….21
Ecuación 2.11 …….33
Ecuación 2.12 …….33

CAPÍTULO 4

Figura 4.1-Consumo de energía en cifrado simétrico y asimétrico ……69
Figura 4.2-Consumo de energía en AES ……72
Figura 4.3-Consumo de energía en cifradores ……73
Figura 4.4-Consumo de energía en DES ……75
Figura 4.5-Comsumo de energía en RC4 y DES ……77
Figura 4.6-Consumo de energía en AES y RC4 con diferentes tamaños de llave …..84
Figura 4.7-Velocidad de cifrado en AES y RC4 ……85

Tabla 4.1-Resultados de Huffman Estático ……56
Tabla 4.2-Resultados de Huffman Dinámico ……56
Tabla 4.3-Resultados de Codificador Aritmético Estático ……61
Tabla 4.4-Resultados de Codificador Aritmético Dinámico ……..62
Tabla 4.5-Resultados de LZW ……64
Tabla 4.6-Resultados de LZSS ……...66
Tabla 4.7-Resultados de RLE ……67
Tabla 4.8-Tamaño de llave, bloque y número de rondas en AES ……..72
Tabla 4.9-Resultados de AES ……73
Tabla 4.10-Resultados de DES ……...76
Tabla 4.11-Resultados de RC4 ……...77
Tabla 4.12-Resultados de algoritmos de compresión ……79
Tabla 4.13-Resultados de LZSS, LZW y Huffman Estático ……..80
Tabla 4.13d-Resultados de LZSS ……...81
Tabla 4.13e-Resultados de LZW ……81
Tabla 4.13f-Complejidad computacional y espacial de LZSS y LZW ……...81
Tabla 4.14-Resultados de RC4, AES y DES ……..83
Tabla 4.15-Resultados de LZW-RC4 ……86
Tabla 4.16-Resultados de tiempo en LZW-RC4 ……86
Tabla 4.17-Resultados de LZW-RC4 ……87
Tabla 4.18-Resultados de tiempo en LZW-RC4 ……87

Ecuación 4.1 ……...59
Ecuación 4.2 ……...60
Ecuación 4.3 ……...60
Ecuación 4.4 ……...60

CAPÍTULO 5

Figura 5.1-Procesamiento de 12 bits ……..93
Figura 5.2-Procesamiento 12 bits con swap ……...95
Figura 5.3-Proceso de división en Bytes-Pares y Bytes-Nones ……..99
Figura 5.4-Comparando y sustituyendo valores ……...105
Figura 5.5-Algoritmo de comparación entre elementos ……...106
Figura 5.6-Ejemplo de sustitución de valores ……...109
Figura 5.7-Modificación de Bytes-Pares (A) y Bytes-Nones (B) ……112
Figura 5.8-Proceso de juntar dos bytes en uno ……115
Figura 5.9-Comparación entre tres métodos de pre procesamiento
(razón de compresión) ……..121
Figura 5.10- Comparación entre tres métodos de pre procesamiento
(tiempo de compresión) ……119
Figura 5.11-Resultados de los módulos de pre procesamiento
(razón de compresión) ……..122

Figura 5.12-Resultados de los módulos de pre procesamiento
(tiempo de compresión) 123

Tabla 5.1-Resultados de byte –byte ……92
Tabla 5.2-Resultados de 12-bits ……94
Tabla 5.3-Resultados de 12-Bits-Swap ……...96
Tabla 5.4-Resultados de Byte-Byte-Swap ……..98
Tabla 5.5-Resultados de División ……101
Tabla 5.6-Detalle en resultados de División ……101
Tabla 5.7-Resultados de División-LZW-RLE ……..103
Tabla 5.8-Detalle en resultados de División-LZW-RLE ……..104
Tabla 5.9-Resultados de Different ……107
Tabla 5.10-Algoritmos y razón de compresión de algoritmos lossy ……....107
Tabla 5.11-Detalle en resultados de Different ……..108
Tabla 5.12-Resultados de Different-LZW-RLE ……...110
Tabla 5.13-Detalle en resultados de Different-LZW-RLE ……...111
Tabla 5.14-Resultados de Divide2Different-LZW ……...115
Tabla 5.15-Detalle en resultados de Divide2Different-LZW ……...116
Tabla 5.16-Resultados de Divide2Different-LZW-RLE ……..117
Tabla 5.17-Detalle en resultados de Divide2Different-LZW-RLE ……..118
Tabla 5.18-Resultados de los módulos de pre procesamiento ……..119
Tabla 5.19-Resultados de DES en Telemedicina ……125
Tabla 5.20-Resultados de Divide2Different-LZW y DES ……...126
Tabla 5.21-Resultados de RC4 en Telemedicina ……..127
Tabla 5.22- Resultados de Divide2Different-LZW y RC4 ……...127
Tabla 5.23- Resultados de AES en Telemedicina ……128
Tabla 5.24- Resultados de Divide2Different-LZW y AES ……..129

Acrónimos

3G – Tercera Generación (Third Generation).

AES – Estándar de Cifrado Avanzado (Advanced Encryption Standard).

DCT - Transformada Discreta del Coseno (Discrete Cosine Transform).

DES – Estándar de Cifrado de Datos (Data Encryption Standard).

DPCM – Modulación por Codificación de Impulsos Diferencial (Differential Pulse Code Modulation).

ECG - Electrocardiograma.

GB – Giga Bytes.

GPRS – Servicio General de Radio Paquetes (General Packet Radio Service).

GSM – Sistema Global para comunicaciones Móvil (Global System for Mobile communications).

HT – Transformada de Hermite (Hermite Transform).

LT – Transformada de Legende (Legendre Transform).

Mbps – Mega Bits por Segundo (Mega Bits Per Second).

PDA – Asistente Personal Digital (Personal Digital Assistant).

RC – Razón de Compresión.

RLE – Codificador de Longitud (Run Length Encoding).

SSL – Capa Segura de Socket (Secure Socket Layer).

TCP – Protocolo de Control de Transferencia (Transfer Control Protocol).

UMTS – Sistema Universal de Telecomunicaciones Móvil (Universal Mobile Telecommunications System).

WLAN – Red inalámbrica de área local (Wireless Local Area Network).

WPAN –Red inalámbrica de área personal (Wireless Personal Area Network).

WWAN –Red inalámbrica de área extensa (Wireless Wide Area Network).

Capítulo 1

Introducción

En este capítulo se hace un bosquejo de este trabajo de investigación. Se abarca la motivación por la cual se incursiona en las áreas de telemedicina, redes inalámbricas, compresión y cifrado de datos. Se mencionan los objetivos a conseguir y el método para poder lograrlos. Por último, la organización de la tesis se describe al final del capítulo.

1.1 Motivación

Actualmente, el área de Telemedicina ha adquirido gran importancia debido a las ventajas que se pueden obtener en el cuidado de pacientes. La telemedicina hace posible la asistencia domiciliaria, monitoreo de signos vitales, detección de situaciones de riesgo en la salud del paciente e independencia paciente-médico, entre otras aplicaciones.

Las aplicaciones en el área de telemedicina, en su mayoría, presentan la peculiaridad de generar gran de cantidad de información en corto tiempo, ejemplo: aplicaciones donde manejen datos de electrocardiogramas. Paralelamente, muchas de esas aplicaciones tienden a ocupar medios inalámbricos o dispositivos móviles. Lo anterior presenta el problema de confidencialidad de los datos, ya que, gran cantidad de estos sistemas no utilizan alguna técnica de seguridad para la transmisión de éstos. También, se ven limitados por la capacidad máxima del medio.

La telemedicina presenta varios aspectos que hay que tomar en cuenta para el buen funcionamiento de un sistema, tales como: tipo y cantidad de datos a transmitir, medio de transporte de información, velocidad en la entrega de los datos, consumo de energía, cantidad de procesamiento, entre otros.

Cabe mencionar que en telemedicina existen diferentes aplicaciones que utilizan redes inalámbricas, las cuales demandan cierta cantidad de información y velocidades de transferencia específicas. La comunicación de datos se realiza entre dispositivos inalámbricos en donde además de los aspectos mencionados anteriormente, la confidencialidad de los datos es un factor vulnerable más difícil de cubrir que en las redes alámbricas, debido a que se utiliza el aire como medio para la transferencia de los datos, con lo cual, no es necesario estar físicamente conectado por un cable al medio para realizar un ataque. Entonces, con la utilización de medios inalámbricos, los datos son más susceptibles a manipulación por parte de entes no autorizados, lo cual puede afectar directamente la integridad de los datos y privacidad del paciente.

La cantidad de datos y velocidad de transferencia están limitadas tanto por los estándares actuales, como por las tecnologías de comunicación inalámbrica, lo que conlleva a que este tipo de redes tenga un ancho de banda más bajo con respecto a las redes alámbricas, las cuales, pueden transmitir datos en el orden de los gigabits por segundo.

Ahora bien, aplicando compresión de datos, se puede lograr un mejor aprovechamiento del medio inalámbrico, entonces, las redes inalámbricas podrán soportar mayor cantidad de datos ya que estos últimos se encuentran comprimidos. Por ejemplo, si se considera que la compresión puede reducir la información original a la mitad, entonces, por el medio inalámbrico se podrá transmitir el doble de información. Por otro lado, utilizando algoritmos de cifrado se garantiza que los datos transmitidos sean ininteligibles para así, hacer más factible la transmisión utilizando este tipo de medios inalámbricos.

Los datos médicos son utilizados para el diagnóstico y monitoreo de los pacientes, y en su mayoría dichos datos no deben sufrir alteraciones cuando éstos son procesados. Por ello, hay que considerar que tipo de algoritmos de compresión y cifrado resultan viables para su aplicación.

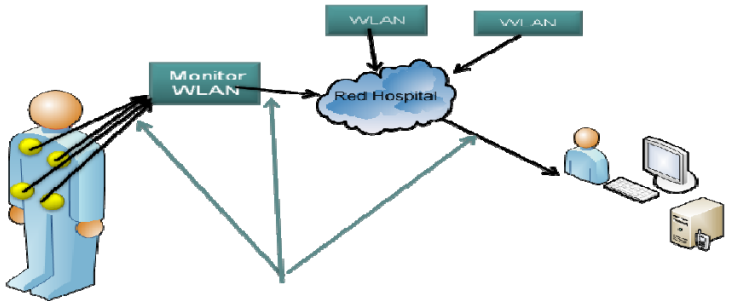

Figura 1.1-Esquema de red en un hospital.

En la figura 1.1 se muestra un esquema de una aplicación en telemedicina. En este caso, se cuenta con un determinado número de sensores, los cuales se encargan del procesamiento y transmisión de datos generados por un paciente. Dichos sensores envían la información por un medio inalámbrico (WLAN) para posteriormente ser distribuidos a la red general de un hospital y así, otros nodos inalámbricos puedan tener acceso a la misma información.

Particularmente, en este caso, no se cuenta con políticas de seguridad que impidan el acceso no autorizado de alguna entidad, por ende, tampoco se puede garantizar la confidencialidad de los datos, siendo así, vulnerable a la intercepción, modificación y/o alteración de información. Este ejemplo tampoco presenta una alternativa para la reducción de información a transmitir, en consecuencia, el medio inalámbrico tiende a congestionarse rápidamente.

En este trabajo de tesis se realiza un enfoque hacia la compresión y cifrado de los datos que serán transmitidos para así, maximizar la eficiencia a nivel de cantidad, integridad y confidencialidad de datos a transmitir.

1.2 Objetivo

El objetivo de este trabajo de tesis es crear un esquema de compresión y cifrado de datos en aplicaciones de telemedicina utilizando datos tipo ECG. Dicho esquema buscará reducir la información procesada y agregar seguridad a los mismos. Aparte, deberá de ser factible incluso para entidades con recursos de procesamiento limitados. Lo anterior con la finalidad de transmitir la mayor cantidad de datos ininteligibles para un ente no autorizado por el medio y además, que este proceso sea totalmente transparente para el usuario final.

Se propondrá que algoritmos de compresión y cifrado, trabajen en forma conjunta pero independientes entre sí. Entonces, con la utilización de un algoritmo de compresión de datos se logrará que el canal de transmisión sea capaz de enviar mayor cantidad de datos y que los dispositivos de almacenamiento puedan contener más información. Por otro lado, usando un algoritmo de cifrado, se obtendrá que la información a transmitir no sea vulnerable, ya que los datos son codificados de tal forma que solamente una persona autorizada es capaz de entender con facilidad.

Como se puede notar, lo que se pretende hacer es aprovechar más la capacidad del canal para la transmisión de datos y de los dispositivos de almacenamiento, por tal motivo, es necesaria la aplicación del o los algoritmos de compresión apropiados. De igual forma, se proporcionará una manera factible en la propagación de los datos utilizando tecnología inalámbrica que será logrado aplicando cifrado en los datos a transmitir, teniendo así, la viabilidad para poder ser aplicado en diferentes tecnologías inalámbricas.

El esquema a crear funcionará de forma transparente para el usuario y se estará contemplando su funcionamiento para que éste sea factible de utilizar en diferentes plataformas. Es decir, que pueda funcionar desde en un equipo con altas prestaciones hasta en uno con recursos limitados sin afectar el desempeño de los resultados obtenidos.

En resumen, se analizarán las aplicaciones en telemedicina, principalmente las concernientes a datos ECG para crear un esquema de cifrado y compresión de datos, que sea capaz de procesar los datos cumpliendo con las normativas de cada aplicación tales como cantidad de información y velocidad a la que deben procesarse dichos datos.

1.3 Método

El método a emplear consiste en el estudio de las aplicaciones con sus respectivas características en telemedicina y así crear el mejor esquema compresión-cifrado. En tales aplicaciones se tomarán en cuenta los tipos de datos manejados, así como los requerimientos de procesamiento para cada uno de ellos. También se hará énfasis en el ambiente de desarrollo, es decir, si trabajan de forma inalámbrica con entidades fijas o móviles, ya que las primeras cuentan con mejores recursos que los segundos.

Posteriormente, se realizará el análisis y la implementación de los algoritmos de compresión de datos y cifrado. En este punto se visualizarán aspectos tales como la complejidad de dichos algoritmos, entornos en los que presentan mejores resultados, ambientes en los que se utilizaron tales algoritmos, tiempo de procesamiento, cantidad de recursos necesarios para realizar el proceso, análisis de resultados obtenidos, entre otras características con la finalidad de determinar los algoritmos con mejor desempeño, tanto de compresión como de cifrado de datos.

También será necesario el análisis del comportamiento que tienen dichos algoritmos cuando se realiza la compresión de los datos y posteriormente, el cifrado de los mismos. Lo anterior con un par de algoritmos seleccionados previamente. Se tomará en cuenta la razón de compresión obtenida y la velocidad de procesamiento tanto para la compresión como para el cifrado. Es importante mencionar que es necesario determinar un compromiso entre razón de compresión y velocidad de procesamiento.

Continuando, y dado los resultados obtenidos de acuerdo al funcionamiento en pareja

de los algoritmos de compresión y cifrado seleccionados, se determinará cuál de las combinaciones de los algoritmos compresión-cifrado tuvo un mejor comportamiento con respecto a la razón de compresión obtenida y el tiempo de comprimir y cifrar los datos.

Finalmente, se buscará obtener una mejor razón de compresión que la ofrecida originalmente por el algoritmo de compresión, con la finalidad de aprovechar más eficientemente el medio de transmisión y de almacenamiento. Lo anterior se logrará aplicando un pre procesamiento a los datos originales para que éstos sean procesados más eficientemente por el compresor. Entonces, será necesario la creación de módulos en los que se realicen diferentes formas de pre procesar la información original, de tal suerte que, se obtenga una razón de compresión más alta.

El método empleado puede ser representado por la figura 1.2 donde el símbolo *A* representa el trabajo en conjunto de los algoritmos de compresión y cifrado respectivamente, y *B* significa las aplicaciones que no necesitan ningún esquema de compresión y/o seguridad de datos.

1.4 Organización de la tesis

Esta tesis está organizada y distribuida en seis capítulos. En el capítulo dos, se describe el marco teórico para el entendimiento y dirección que sigue esta investigación. En ese capítulo se mencionan aspectos descriptivos de cada uno de los algoritmos a utilizar, de compresión y cifrado.

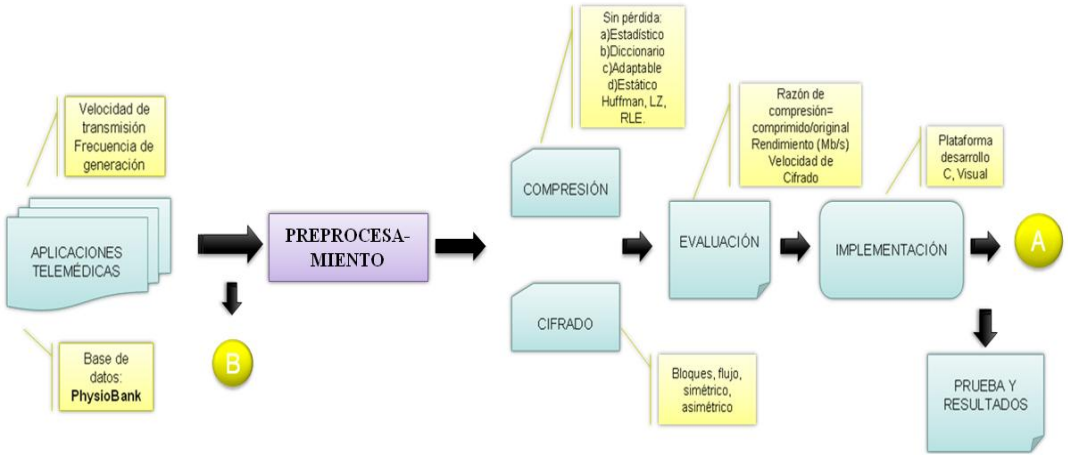

Figura.1.2-Diagrama de método a seguir.

En el capítulo tres, se hace la revisión del estado del arte en el área de telemedicina. Se mencionan trabajos que tienen contemplados esquemas de compresión y/o seguridad de datos, se menciona una gran diversidad de trabajos, los cuales, no cuentan con ninguno de los dos aspectos antes mencionados. En ese capítulo se obtiene una idea general de la situación en la que se encuentra la investigación relacionada con la compresión y cifrado de datos en telemedicina tanto para redes inalámbricas como alámbricas.

En el capítulo cuatro, se realiza un análisis más profundo de los algoritmos de compresión y cifrado. Se mencionan aspectos tales como complejidad computacional, recursos necesarios para la realización de una tarea, ambientes factibles para cada uno de los algoritmos, etc. En ese capítulo se menciona el porqué se debe elegir cierto algoritmo dependiendo del ambiente al que vaya dedicado. Consecuentemente, es en el capítulo cuatro donde, a partir de la premisa de que el esquema de compresión-cifrado pueda funcionar en la mayoría de ambientes posibles, se determina qué algoritmos se utilizarán en dicho esquema. También, en el capítulo cuatro se mostrarán resultados obtenidos por los algoritmos de compresión y cifrado. Entonces, es aquí donde se determinan los algoritmos para el esquema compresión-cifrado propuesto para telemedicina.

En el capítulo cinco, se describe una serie de procedimientos en los cuales se busca obtener una razón de compresión que mejora a la obtenida en el capítulo cuatro. Con la aplicación de tales procedimientos, en el capítulo cinco también se muestran los diferentes comportamientos obtenidos para cada uno de ellos. Como consecuencia de proponer diferentes procedimientos, el capítulo cinco se caracteriza por determinar un proceso en el cual se logra reducir aún más la cantidad de información a transmitir.

Finalmente, en el capítulo seis, se presentan las conclusiones a las que se llega en este trabajo de investigación. También, se menciona el trabajo futuro a realizar a partir de lo obtenido en esta tesis.

Capítulo 2

Introducción

En este capítulo se darán antecedentes de lo que comprende este trabajo de tesis. Se tocan aspectos relacionados con el área de telemedicina, compresión de datos, cifrado de datos y redes inalámbricas de computadoras.

2.1 Telemedicina

La telemedicina fue propuesta en los años 70's y es definida como el intercambio de información médica usando tecnología de telecomunicaciones tales como el uso de dispositivos de cómputo y redes de computadoras. En [1] definen a la telemedicina como el intercambio de conocimiento y entrega de datos médicos a distancia, usando telecomunicaciones.

Actualmente, la telemedicina ha evolucionado y crecido de tal forma que algunas variables tales como el medio de transporte a utilizar, el tipo y cantidad de datos generados para transmitir, la velocidad de procesamiento, rango de cobertura, confidencialidad e integridad de los datos, son tomadas en cuenta en todo sistema de telemedicina. Entre algunas de las aplicaciones se tiene al monitor de signos vitales, que abarca el monitoreo de presión sanguínea, temperatura y saturación de oxígeno. Existen, por otro lado, aplicaciones más complejas, las cuales requieren el procesamiento y/o transmisión de grandes volúmenes de información tales como una videoconferencia, operación a distancia, control remoto de dispositivos médicos, etc. Para más aplicaciones ver [2, 3, 4, 5].

2.1.1 Estándares

Debido a que esta área es considerada seriamente a formar parte de la práctica moderna de la medicina, es crucial que cualquier barrera que pueda interferir en el

desempeño de la misma sea identificada y remediada. Por tal motivo, se han creado una serie de estándares que regulan ciertos aspectos tales como la terminología a ocupar, procesos, requerimientos de infraestructura, etc.

El Instituto de Estándar Nacional Americano (ANSI) es una organización acreditada para desarrollar estándares en el área de los cuidados de la salud. En [6] se especifican otras organizaciones y estándares importantes.

Cualquier investigación que se haga sobre telemedicina, deberá abocarse a dichos estándares para así lograr una homogeneidad e interoperabilidad entre los diferentes elementos.

2.1.2 Tendencia

Dada la gran demanda que presenta hoy en día esta área con respecto a la telecomunicación, se ha optado por la utilización de diferentes tecnologías que permitan la transmisión de los datos. La telemedicina ocupa como medio de transmisión a las redes de computadoras, principalmente, las redes alámbricas. No obstante, actualmente existe una gran necesidad por la adopción de los medios inalámbricos para utilizarse en muchas aplicaciones telemédicas, brindando así, un mejor servicio. Hay que recalcar que las aplicaciones que utilizan redes inalámbricas se encuentran limitadas por el ancho de banda máximo que proporcione dicha tecnología.

En la siguiente sección se revisa de manera general las redes de computadoras, dando un enfoque principal hacia la tecnología inalámbrica por ser esta última la de mayor demanda en el área de telemedicina actualmente.

2.2 Redes de Computadora

Las redes utilizadas en los sistemas de telemedicina están constituidas por diferentes

medios de transmisión tales como cables, fibra óptica, ondas electromagnéticas; dispositivos hardware, por mencionar algunos; componentes de software tales como protocolos de comunicación, drivers, etc.

En [7] se define a las redes como una colección de computadoras autónomas interconectadas por una tecnología. En telemedicina se puede trabajar con un conjunto de tecnologías, con lo cual, es posible realizar una interacción entre diferentes tecnologías al mismo tiempo, es decir, puede haber un conjunto de subredes.

Las redes de computadoras se dividen en redes alámbricas y redes inalámbricas. Las primeras tienen su funcionamiento basado en la interconexión de elementos mediante un cable o medio alámbrico, mientras que las segundas no necesitan de los cables para realizar el intercambio de datos ya que transmiten ondas electromagnéticas utilizando al aire como medio de transmisión.

Una de las principales características de las redes alámbricas es su alta velocidad de transferencia, además, para poder obtener datos de esta red, es necesario estar físicamente conectado a la misma. Sin embargo, también presenta algunas desventajas ya que tiende a ser susceptible a las señales emitidas por otros aparatos electrónicos.

Por otra parte, las redes inalámbricas brindan menor ancho de banda con respecto a las alámbricas, pero con la facilidad de que la infraestructura, al momento de instalarla, es más sencilla. No obstante, el uso de este tipo de redes presenta gran vulnerabilidad en el momento de la transmisión debido al medio de transmisión utilizado.

La tecnología inalámbrica se divide en red de área local (LANs), red de área personal (PANs) y red de área amplia (WANs). En la tabla 2.1 se muestra algunas de las características de estos tipos de redes.

TIPO	DISTANCIA	ANCHO DE BANDA	STANDARS
WPAN	<10 m	0.1- 4 Mbps	Bluetooth 802.15
WLAN	100 m	1-54 Mbps	802.11a, b, g, Hiperlan/2
WWAN	>100 m	8 Kbps-2 Mbps	GSM, TDMA, CDMA, GBRS, EDGE, WCDMA

Tabla 2.1-División de tecnologías inalámbricas.

Debido a que las redes inalámbricas poseen un ancho de banda más limitado que las alámbricas, es necesario pensar en mecanismos de reducción de datos tales como los algoritmos de compresión, para evitar que la red se sature lo antes posible. De igual forma y a causa del medio de transmisión utilizado, es necesario adquirir un esquema de seguridad para garantizar la confidencialidad de los datos, lo cual puede cubrirse con algoritmos de cifrado.

En los puntos siguientes se darán detalles acerca de los algoritmos de compresión y de cifrado, posteriormente se analizará la vinculación con el área de telemedicina.

2.3 Compresión de Datos

La compresión de datos juega hoy en día un rol importante, debido a que surgen nuevas necesidades y expectativas para el procesamiento, transferencia y almacenamiento de la información. La compresión de información consiste en reducir la información original, es decir, eliminar la redundancia. Salomon [8] define a la compresión de datos como el proceso de convertir un flujo de datos de entrada (la fuente o datos originales) en otro flujo de datos (la salida o el flujo comprimido) con un tamaño menor.

Entonces, la compresión de datos es la codificación de la información de tal forma que elimina la mayor cantidad de redundancia posible dependiendo del tipo de información que se esté manejando y del algoritmo utilizado para realizar dicho proceso. Lo anterior se puede resumir en la representación de los datos originales con menor número de bits.

El proceso de codificar datos para lograr un menor tamaño de los mismos, ayuda a que el uso de estos tipos de algoritmos sea viable cuando se requiere almacenar gran cantidad de información y/o se cuenta con un espacio limitado para hacerlo. En la transmisión de datos por redes de computadoras, se puede hacer un mejor aprovechamiento del medio, evitando que se sature con mayor facilidad y más, si se trata de un medio inalámbrico. Entonces, con el proceso de compresión se puede hacer un manejo más eficiente de los recursos.

El tiempo necesario para realizar la compresión de los datos, mandarlos por un medio y descomprimirlos, es relativamente el mismo que si se mandaran los datos tal cual sin comprimir. Ahora bien, un buen algoritmo de compresión de datos debe obtener una razón de compresión menor o igual a 0.5 [9], pudiendo expresar a la razón compresión de la siguiente manera:

$$Razón\ Compresión = \frac{Datos\ comprimidos}{Datos\ originales} \tag{2.1}$$

La razón de compresión varía dependiendo del tipo de información que se esté manejando, además, es directamente afectada por el tipo de algoritmo de compresión utilizado. Los resultados obtenidos al comprimir texto plano no serán de ninguna manera igual a los obtenidos al comprimir un video o una imagen. Lo anterior se debe a que se utilizan diferentes tipos de algoritmos para diferentes tipos de datos.

Por otra parte, cuando se realiza el proceso de compresión, es posible que se pueda recuperar la información original exactamente igual a como fue generada, o bien, solamente una aproximación de ellos. El primer caso tiene una razón de compresión más baja, pero con la ventaja de que no hay pérdida de información, a diferencia del segundo, donde sí la hay. Consecuentemente, el área de compresión se divide principalmente en algoritmos de compresión de datos sin pérdida y en algoritmos de compresión de datos con pérdida. Los primeros son utilizados cuando es necesario recuperar toda la información exactamente, mientras que los segundos se utilizan

cuando es preferible una alta razón de compresión y la recuperación fidedigna de los datos no es tan importante.

2.3.1 Compresión de datos con pérdida

Este tipo de algoritmos se caracteriza por obtener razones de compresión bastante altas, comúnmente por arriba del 50%. Esto se debe a que el proceso de compresión descarta ciertos datos.

Tales algoritmos son utilizados principalmente en imágenes, video y sonido ya que este tipo de datos puede sufrir degradaciones y aún así, seguir presentando una buena calidad con respecto al original. Como ejemplo, en las imágenes se tiene que un archivo común tipo *.bmp* se puede reducir en un 95% cuando se comprime con el algoritmo *jpeg*. De igual forma, un audio de tipo *.wav* se puede reducir en un 90% utilizando un compresor *mp3*.

Aplicando el proceso de descompresión de los datos comprimidos, existe una diferencia en comparación con la representación original por la pérdida de datos. No obstante, los datos representan de manera aceptable dicha información.

Ejemplos de algoritmos que utilizan la compresión con pérdida así como su descripción, se pueden encontrar en [8, 10, 11].

2.3.2 Compresión de datos sin pérdida

Smith [12] menciona que estos tipos de algoritmos son usados cuando el archivo descomprimido es idéntico al original. Este tipo de compresión es utilizada cuando es necesaria la total recuperación de la información tal cual fue emitida. Entre los principales casos de uso se tiene la compresión de texto, código ejecutable, números, entre otros.

Este clase de compresores pueden obtener razones de compresión en el rango de 2:1 a

4:1 y para mayor detalle y ejemplos de este tipo de algoritmos ver [8, 10, 11].

Los compresores sin pérdida pueden clasificarse en estadísticos y de diccionario. Los de tipo estadístico utilizan códigos de tamaño variable de tal forma que a los elementos con mayor frecuencia de aparición en el flujo original, se les asignará el código más pequeño. Además, los compresores estadísticos pueden tener variantes con respecto al modelo utilizado para realizar la compresión tales como:

- Estático. Las probabilidades de cada símbolo se mantienen constantes. Se realiza sólo una lectura de los datos. Rápidos pero con una razón de compresión pobre.
- Semi-adaptable. Se realizan dos pasadas sobre los datos originales. En la primera se determinan las probabilidades y en la segunda se hace la codificación.
- Adaptable. Ofrece mejor razón de compresión en un tiempo mayor. Las probabilidades de los símbolos varían conforme el proceso de compresión se realiza.

Por otro lado, los basados en diccionario codifican cadenas de símbolos de acuerdo a un diccionario que bien puede ser estático o dinámico.

- Estático. Comúnmente utilizado cuando se conoce de antemano la estructura repetitiva de los datos. El diccionario permanece sin cambios.
- Dinámico. El diccionario varía conforme el proceso de compresión avanza, en consecuencia, ofrece mejor razón de compresión.

Una diferencia entre los estadísticos y los basados en diccionario, es que los primeros pueden llegar a ofrecer mejor razón de compresión que los segundos pero comúnmente son más complejos computacionalmente. Además, la razón de compresión en los primeros dependerá de qué tan bueno sea el modelo estadístico

ocupado. No obstante, la gran aceptación que tienen los algoritmos de diccionario se debe a que pueden ser aplicados a una gran cantidad de datos y obtener una compresión bastante aceptable. Otra diferencia es que la descompresión de los basados en diccionario suele ser más simple que los estadísticos, por ende, más rápidos.

Para efectos de este trabajo de tesis se analizarán algoritmos de compresión de datos *sin pérdida*, para así, poder abarcar la mayor cantidad de datos posibles. Con lo anterior, se propondrá un esquema flexible y escalable para diferentes aplicaciones en telemedicina.

En los puntos siguientes, se describirán los algoritmos de compresión de datos *sin pérdida* seleccionados.

2.3.2.1 RLE

El *Run Lenght Encoding* [8], es uno de los compresores más simples, intuitivo y rápido. Su razón de compresión se encuentra estrechamente ligada con el tipo de dato que se desee comprimir.

La idea fundamental de este algoritmo consiste en sustituir las secuencias repetidas de un símbolo por un par ordenado, en el cual se especifica tanto el símbolo como el número de veces que aparece en la fuente de entrada.

Una desventaja radical de este método es que si no hay las suficientes repeticiones en los datos originales, la compresión resultará totalmente inadecuada ya que los datos de salida superarán en tamaño al original.

RLE se puede representar de la siguiente forma:

Sean $X_1, X_2, X_3, \ldots, X_n$ los símbolos a codificar y sean $i_1,i_2,i_3,\ldots i_n$ las repeticiones que tienen los elementos X respectivamente, entonces, $(X_1,i_1), (X_2,i_2), (X_3,i_3),\ldots, (X_n,i_n)$

representan la compresión de los datos.

El proceso de descompresión de un flujo RLE se limita a la lectura de la cadena comprimida y a trasladar la información de los pares ordenados.

2.3.2.2 Huffman Estático

Este algoritmo fue presentado en el año de 1952 por el Dr. David A. Huffman y se basa en la utilización de un modelo estadístico estático y de un codificador Huffman [115]. Por basarse en un modelo estadístico, utiliza códigos de longitud variable donde los más cortos son asignados a los símbolos más frecuentes y los más largos, a los menos frecuentes.

Realiza la construcción de una estructura de datos en forma de árbol binario a partir de una lista de todos los símbolos que serán tratados de acuerdo a su probabilidad de aparición y los ordena en forma descendente. A continuación se describirá la construcción del árbol de forma general:

1. Cada uno de los símbolos, con su respectiva probabilidad de aparición, representarán nodos del árbol, es decir, habrá tantos nodos como símbolos.
2. Seleccionar dos símbolos con probabilidad mínima para formar un nuevo nodo, el cual contendrá la suma de ambas probabilidades.
3. Repetir el paso 2 hasta obtener un nodo raíz que tendrá la probabilidad igual a 1, es decir, la suma de todas las probabilidades.

Tomando en cuenta el proceso anterior, se puede decir que el número de iteraciones para la construcción de un árbol de Huffman con un número de hojas *N,* es *N* (complejidad lineal) y para hacer la búsqueda de un elemento dentro del árbol se tiene una complejidad de *N log N* por las características que presenta un árbol binario [82].

Sea un alfabeto *A* con una distribución de frecuencia *{f(a): a ε A}*. El árbol de

Huffman es construido a partir de una lista de prioridad ordenada en forma descendente con respecto a sus probabilidades con *X* nodos. En la figura 2.1 se muestra el algoritmo a seguir para la construcción del árbol de Huffman.

```
Huffman(A)
{
  s=|A|
  X=A
  Para i=1 hasta s-1
  {
    y=nodo nuevo
    Izq[y]=min(X)
    Der[y]=min(X)
    f[y]=f[Izq[y]]+f[Der[y]]
    Insertar(X,y)
  }
  Devolver min(X)
}
```

Figura 2.1-Algoritmo para la construcción del árbol de Huffman.

Ahora bien, para asignar un código Huffman a un elemento, una vez formado el árbol, se recorre el árbol desde el nodo que representa a dicho símbolo hasta el nodo raíz; por cada nodo recorrido se pone el código asociado, ya sea 1 ó 0, y al final se invierte la serie obtenida. Con la ecuación 2.2 se hace el cálculo del tamaño promedio de los códigos obtenidos [8].

$$E(L) = \sum_{i=1}^{m} l_i f_i(U_i) \tag{2.2}$$

donde l_i es la longitud del símbolo, U_i *y* $f_i(U_i)$ son la probabilidad del símbolo U_i.
Para calcular la eficiencia de una codificación es necesario saber la entropía del flujo original, la cual, se obtiene a partir de la ecuación 2.3.

$$H(x) = -\sum_{i=0}^{m} p_i \log_2 p_i \; con \; \sum p_i = 1 \tag{2.3}$$

y posteriormente sustituir valores en la ecuación 2.4

$$n = \frac{H(x)}{E(L)} \tag{2.4}$$

De igual forma, el cálculo de la redundancia o aparición de los códigos se define por ecuación 2.5.

$$r = 1 - n = 1 - \frac{H(x)}{E(L)} \tag{2.5}$$

El proceso de descompresión es relativamente sencillo, basta con que se tenga conocimiento de la estructura del árbol para hacer el recorrido desde el nodo raíz hasta el nodo que indique el código Huffman para determinar el símbolo de entrada. Además, la codificación de los datos cumple con la particularidad de que los elementos utilizados para sustituir a los originales, no son en ningún caso prefijo de otro, por tal motivo, se puede realizar una decodificación de forma inmediata e inequívoca de los datos.

2.3.2.3 Huffman Dinámico

Este método fue desarrollado por Faller y Gallager [84,85]. Parecido al Huffman estático, pero con la principal diferencia que las frecuencias o probabilidades de los símbolos son determinadas conforme avanza el proceso de compresión. El algoritmo dinámico tiene un mejor desempeño en la mayoría de los casos con respecto a la razón de compresión y una pequeña variante de tiempo. El proceso de compresión de los datos inicia con un árbol sin símbolos y sin probabilidades. Se leerán los símbolos de entrada, se modificarán los símbolos y sus respectivas frecuencias de aparición dinámicamente.

Con respecto al tiempo empleado en el proceso de compresión, el compresor estático es más rápido, tanto para la compresión como para la descompresión de los elementos. Sin embargo, con el dinámico es posible obtener una mejor razón de

compresión.

2.3.2.4 Codificador Aritmético

El principio de este algoritmo fue propuesto por Peter Elías en los años 60s [98]. Se basa en un modelo estadístico y un codificador aritmético. Este método requiere la probabilidad de cada símbolo de aparición para poder realizar el proceso de compresión. Tal algoritmo no se limita al uso de una cantidad entera de bits para realizar la codificación de los símbolos, sino más bien, el resultado arrojado por este compresor es un número real R que se encuentra en el rango $0 \leq R \leq 1$, donde la precisión de R está directamente afectada por la longitud de la cadena. Con esto, se puede representar una misma cadena con diferentes números, siempre y cuando se cumpla la condición que dichos números se encuentren dentro del intervalo en el cual R está limitado.

Fundamentalmente, el proceso del codificador aritmético, consiste en la creación de una secuencia de intervalos anidados en la forma $\Phi_k(S)=[\alpha_k, \beta_k), k=0,1,...,N,$ donde S es la secuencia de la fuente, α_k y β_k son números reales tales que $0 \leq \alpha_k \leq \alpha_{k+1}$ y $\beta_{k+1} \leq \beta_k \leq 1$.

A continuación, los pasos seguidos para realizar la compresión:

1. Calcular las frecuencias de ocurrencia de cada símbolo a comprimir.
2. Establecer rango inicial [límite inferior, límite superior) a [0,1).
3. Dividir intervalo actual según las probabilidades de cada símbolo y una función de proporcionalidad.
4. Dividir lo obtenido hasta el paso 3 en tantos subintervalos como símbolos contenga el alfabeto a comprimir.

Formalmente, las siguientes ecuaciones determinan los subintervalos.

$$\textbf{Rango} = \textbf{Superior} - \textbf{Inferior} \tag{2.6}$$

$$\textbf{Superior} = \textbf{Inferior} + \textbf{Rango} * \frac{CF_{S_{i-1}}}{CF_{S_0}} - 1 \tag{2.7}$$

$$\textbf{Inferior} = \textbf{Inferior} + \textbf{Rango} * \frac{CF_{S_i}}{CF_{S_0}} \tag{2.8}$$

donde *CF* representa un rango de frecuencia y S_i representa la probabilidad de un símbolo *i.* Todos y cada uno de los símbolos estarán dentro del intervalo [0,1), ya que la probabilidad de cada elemento es equivalente al intervalo que éste ocupa. Por ende

$$0 \leq= \sum_{i=1}^{n} P_i \leq 1 \tag{2.9}$$

Ahora bien, la descompresión de los datos codificados está determinada por el valor de código $\hat{v}$ de la secuencia de compresión [10]. Entonces, la secuencia de decodificación es:

$$\hat{S}(\hat{v}) = \{\widehat{s_1}(\hat{v}), \widehat{s_2}(\hat{v}), \ldots \widehat{s_N}(\hat{v})\} \tag{2.10}$$

El proceso de descompresión recupera la información en la misma secuencia en la que fueron codificados y dada la ecuación (2.10), se define una secuencia de valores de códigos normalizados $\{\hat{v}_1, \hat{v}_2, \ldots, \hat{v}_N\}$. Empezando con $\hat{v}_1 = \hat{v}$, se puede encontrar $\hat{s}_k$ de $\hat{v}_k$ y entonces realizar $\hat{v}_{k+1}$ de $\hat{s}_k$ y $\hat{v}_k$.

El descompresor, para su buen funcionamiento, debe conocer los siguientes datos a partir de la compresión:

- El número *R,* obtenido en la última etapa del algoritmo de compresión y representa a los datos comprimidos.
- La longitud de *R*, es decir, la cantidad de símbolos a los cuales representa. Depende de la cantidad de símbolos que tenga la cadena original.
- Todos los símbolos procesados y sus respectivas probabilidades.

Howard *et al.* [13] hacen mención que la característica más importante del

codificador aritmético es su flexibilidad, ya que puede ser usado en conjunción con cualquier modelo que pueda proveerle una secuencia de probabilidades pero con la desventaja que tiende a ser muy lento debido a que requiere al menos de una multiplicación por evento y en algunos casos arriba de dos multiplicaciones y dos divisiones por evento.

2.3.2.5 LZSS

Creado por James Storer y Thomas Szymansky en el año de 1982 [10]. Toman como referencia el algoritmo de compresión LZ77 desarrollado por Lempel y Ziv en 1977 [14]. Tanto LZ77 como LZSS se basan en la compresión con el uso de diccionario.

La estructura principal del LZ77 analiza una secuencia de símbolos (flujo de entrada) por medio de una *ventana deslizante* en la cual existen dos elementos clave. El primero es un *buffer de búsqueda* en donde se almacenarán los códigos o la secuencia que ya ha sido procesada. El segundo es un *buffer de procesamiento* donde se almacenan aquellos símbolos que se van a codificar.

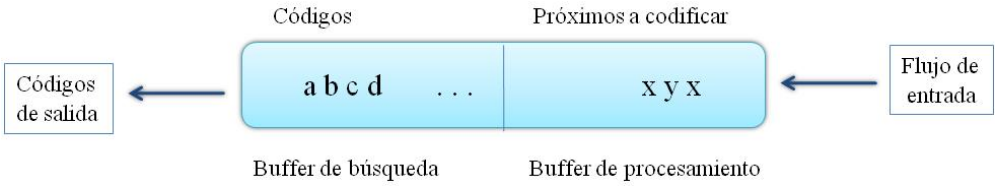

Figura 2.2-Compresor LZ77.

El proceso de compresión está dado por la entrada de símbolos en el *buffer de procesamiento*. Se lee símbolo a símbolo y se va comparando si existe una entrada igual en el *buffer de búsqueda*; de ser así, se sustituye por el código adecuado, el cual es una tripleta ordenada:*[desplazamiento, longitud, símbolo]*. El *desplazamiento* indica el punto de inicio en la coincidencia entre el *buffer de búsqueda* y el *buffer de procesamiento*, la *longitud* dice cuantas posiciones abarca dicha coincidencia y el *símbolo* indica el próximo elemento a procesar.

Al principio de la compresión, ambos buffers están vacíos. Durante el proceso de codificación, el *buffer de búsqueda* y el *buffer de procesamiento* van creciendo hasta su límite, que es establecido con anterioridad. Una vez iniciado el proceso, los símbolos son comparados uno a uno; se busca obtener la coincidencia de datos con longitud más grande, es decir, leído un símbolo y encontrado en el *buffer de búsqueda*, se apunta al siguiente símbolo en el *buffer de procesamiento* y se vuelve a realizar la búsqueda hasta que no haya elementos iguales.

La eficiencia de la compresión está dada por la forma en que se codifican dichas tripletas ordenadas y es ahí donde surge LZSS, que es la mejora del algoritmo LZ77.

El proceso de descompresión es más simple que la compresión. Tanto el LZ77 como el LZSS son algoritmos de compresión asimétricos. Lo anterior implica que estos dos algoritmos o alguna de sus variantes son muy útiles en casos tales donde la compresión de un archivo se haga una o pocas veces pero, que se descomprima dicho archivo en muchas ocasiones.

2.3.2.6 LZW

Desarrollado por Terry Welch en el año de 1984 [15] y es una variante del algoritmo de compresión LZ78 [16].

Para iniciar el proceso de compresión, el LZW almacena en el diccionario todos los posibles símbolos que puedan llegar a ser procesados, es decir, todos los elementos del alfabeto. Con lo anterior se garantiza que el primer carácter procesado siempre va a ser encontrado en el diccionario.

La idea general es buscar símbolo por símbolo y hacer que las entradas del diccionario contengan más entradas conforme avanza el proceso de compresión.

Algoritmo LZW

1. Iniciar el diccionario con todos los símbolos del alfabeto a utilizar.
2. Leer el primer símbolo y devolver su código.
3. Leer el siguiente símbolo concatenado con símbolo anterior, si el símbolo se encuentra, devolver código, si no, crear nueva entrada con dicha concatenación.
4. Si no existen datos a leer, salir. Si existen, repetir paso 3.

Para el proceso de decodificación, el descompresor inicia el diccionario con todos los símbolos del alfabeto a manejar. Basta con leer el código actual y hacer referencia al diccionario para devolver los datos originales. En consecuencia, el descompresor resulta ser más simple que la compresión.

Con la descripción de este algoritmo se culmina la parte que concierne a la compresión de datos en el esquema a proponer de compresión-cifrado para telemedicina. En la siguiente sección se continúa con la introducción de cifrado de datos.

2.4 Cifrado de datos

También conocido como *criptografía* que proviene del griego *kryptós* que significa esconder, y de *graphein* que es escribir.

El cifrado de datos es una técnica que sirve para proteger documentos de un acceso no autorizado, ya que codifica la información de tal suerte que es necesario conocer una o varias claves para poder extraer la información original.

Menezes *et al.* [17] definen a la criptografía como el estudio de técnicas matemáticas relacionado a los aspectos de seguridad de la información tales como la confidencialidad, integridad de los datos, autenticación de la entidad y autenticación

del origen de los datos.

El proceso de hacer ininteligible los datos de un flujo de entrada utilizando algoritmos especiales (texto plano-texto cifrado), se llama *cifrado*; y al proceso inverso, de lo ininteligible a lo entendible (texto cifrado-texto plano) se le conoce como *descifrado*.

La criptografía permite establecer comunicaciones seguras por medio de los siguientes aspectos:

- Confidencialidad. Las entidades que quieran ver la información original deberán estar autorizadas, de lo contrario, no podrán.
- Integridad. Los datos se reciben tal y como fueron originados, sin alteraciones. Es posible detectar inserción, eliminación o sustitución de información.
- Autenticación. Asegura, en una transmisión, que las entidades involucradas sean realmente las que dicen ser. Si el proceso se realiza entre *A* y *B,* el primero verifica que el segundo realmente sea *B,* y viceversa.
- No repudio. Permite que una entidad no niegue que ha formado parte de una transmisión o intercambio de información cifrada.

El cifrado de datos se divide en varios tipos, según su funcionamiento. En [18] se dividen en cifradores simétricos y cifradores asimétricos dependiendo de cómo sea el manejo de la llave. Los cifradores simétricos a su vez se dividen en cifradores de flujo y cifradores de bloques, dependiendo la forma en que realizan el cifrado.

2.4.1 Cifradores Simétricos

Son comúnmente más rápidos al momento de cifrar o descifrar debido a que usan una sola clave para ambos procesos. El receptor debe tener conocimiento de la clave generada por el emisor, por ende, la clave deberá ser transmitida junto con el texto cifrado por el medio.

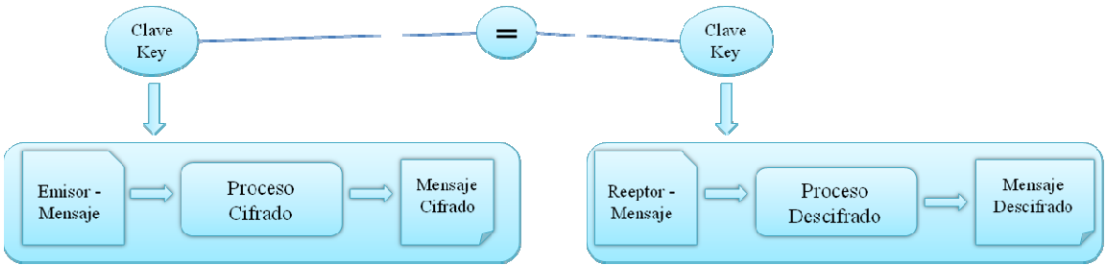

Figura 2.3-Esquema de Cifradores Simétricos.

2.4.2 Cifradores de Flujo

Sea A un alfabeto de q símbolos y permitir que E_e sea un simple cifrador de sustitución con un bloque de longitud l donde $e \quad \varepsilon \quad K$. Sea $m_1\, m_2\, m_3$ una cadena de texto plano y sea $e_1\, e_2\, e_3$ un *keystream* de K. Un cifrador de flujo toma el texto plano y produce un texto cifrado $c_1\, c_2\, c_3$ donde $c_1 = E_{ei}(m_i)$. Si d_i denota el inverso de e_i, entonces $D_{di}(c_i)=m_i$ descifra el texto cifrado.

Estos algoritmos tienen la característica de realizar el proceso de cifrado de los datos bit a bit o byte a byte. Esta peculiaridad los convierte en algoritmos bastante rápidos y en cifradores resistentes a errores, ya que un error no tiende a propagarse. La desventaja principal radica en la dificultad para detectar alguna alteración.

El uso de este tipo de cifradores se recomienda cuando los datos deben de ser procesados un símbolo a la vez o cuando la memoria a utilizar está limitada.

2.4.2.1 RC4

Diseñado por Ron Rivest en el año de 1987. Cifrado de flujo con llave de tamaño variable y operaciones orientadas o diseñadas a trabajar en bytes [19].

RC4 inicialmente genera un flujo pseudoaleatorio de bits llamado *keystream,* que es combinado con el texto plano para poder realizar el proceso de cifrado. RC4 tiene, forzosamente dos entradas, una que define la información a cifrar y otra, la clave con

la que será cifrada. El proceso de combinación del *keystream* con el texto plano se realiza mediante una función tipo *XOR*.

En este algoritmo, el *keystream* es independiente del texto plano. Una llave de longitud variable de 1 a 256 bytes es utilizada para inicializar el vector de estado *S*, llamado también *S-box,* de 256 bytes que contiene los elementos *S[0]*, *S[1]*,…, *S[255]*. En cualquier momento, *S* contiene una permutación de todos los números de 8 bits desde 0 hasta 255. En otras palabras, para generar el *keystream*, RC4 posee un estado interno que consiste en una permutación de todos los 256 posibles símbolos de 8 bits de longitud y dos índices de un byte cada uno.

Es importante mencionar que tanto para cifrar los datos, como para descifrarlos, se realiza el mismo proceso. Entonces, para iniciar el cifrado o descifrado, se genera primeramente un byte *k* de *S* seleccionando una de las 255 entradas. Cada una de las entradas de *S* es inicializada en orden ascendente desde 0 hasta 255.

$$S[0]=0,\ S[1]=1,\ S[2]=2,\ \ldots,\ S[255]=255$$

Posteriormente, un vector temporal *VT* es creado. Entonces, si la longitud de la llave *K* es 256 bytes, *K* es copiado al vector *VT*, de lo contrario, para una *K* con longitud *llave_long,* el primer elemento *llave_long* de *VT* será copiado de *K,* y *K* será repetido tantas veces como sea necesario para llenar *T*.

for i=0 to 255 do
$S[i]=i$;
$VT[i]=K[i \bmod llave\_long]$;

Realizado lo anterior, utilizar *VT* para general la permutación inicial de *S*. Considerar por hecho la inicialización de *S* y se procede a realizar un intercambio de cada uno de sus valores con otro valor según lo establecido por $VT[i]$.

```
j=0;
for i=0 to 255 do
    j=(j+S[i]+T[i]) mod 256;
    swap (S[i], S[j]);
```

Para la generación del flujo se siguen los siguientes pasos:

```
i,j=0;
while (verdadero)
    i=(i+1) mod 256;
    j=(j+S[i] mod 256;
    swap (S[i], S[j]);
    vt=(S[i]+S[j]) mod 256;
    k= S[vt];
```

Para realizar el proceso de cifrado basta con realizar la operación *XOR* al valor *k* con el próximo byte de la información a cifrar.

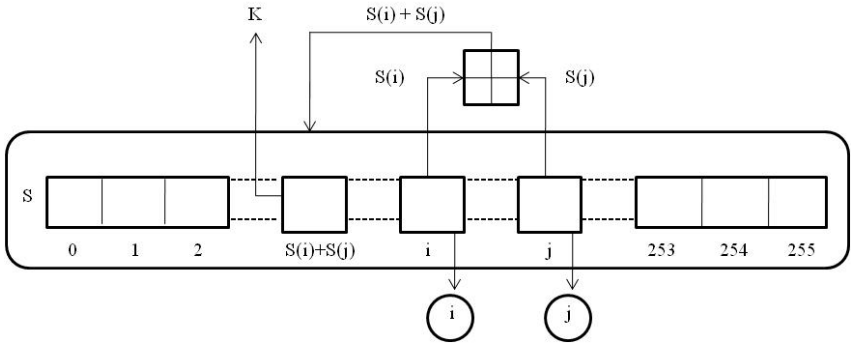

Figura 2.4-Esquema de cifrado RC4.

2.4.3 Cifradores de Bloque

Es un esquema de cifrado donde los datos a cifrar son procesados por *bloques* (un número *X* de bits dependiendo del algoritmo). Por tal motivo, los datos son cifrados por un bloque a la vez, lo cual los convierte en más lentos comparándolos con los cifradores de flujo y además, la propagación de errores es común. La ventaja que brinda este tipo de algoritmos es que es prácticamente imposible introducir nuevos bloques sin que éstos puedan ser detectados.

2.4.3.1 DES

Este es uno de los algoritmos de cifrado más usado en gran diversidad de aplicaciones. El 23 de noviembre de 1976, *Data Encryption Standard* (DES) [20], fue adoptado por el Instituto Nacional de Estándares y Tecnología (NIST) como Estándar Federal de Procesamiento de la Información (FIPS) [21].

DES es una combinación de dos técnicas básicas de cifrado: confusión y difusión (una sustitución seguida por una permutación). Este algoritmo utiliza un tamaño de llave de 56 bits y realiza el cifrado de los datos cifrando con tamaño de bloques de 64 bits. De igual forma que otros cifradores, DES contiene dos entradas, la información a cifrar y la llave con la que será cifrada.

En el cifrado, el texto plano, en bloques de 64 bits, atraviesa un proceso de una permutación inicial para posteriormente ir a un conjunto de 16 etapas o rondas con la misma función, en las cuales se realiza tanto la permutación como sustitución de elementos. En la última etapa, la 16, se arroja un resultado de 64 bits donde ya es procesado el texto plano en conjunción con la llave. Entonces, cada vez que se lee un bloque de 64 bits, después de la permutación inicial, el bloque es dividido en dos partes, de 32 bits cada uno. Cada uno de estos bloques es procesado por las 16 rondas, las cuales contienen las mismas operaciones en cada una de ellas, llamada función F. Por último, al finalizar la ronda 16, se unen los dos bloques de 32 bits divididos anteriormente y se les aplica una permutación que es inversa a la permutación realizada al principio del proceso.

Por el lado de la llave, se le aplica un corrimiento de elementos y de los 56 bits que se tienen originalmente, son tomados 48. Posteriormente, para cada una de las 16 etapas mencionadas anteriormente, una sub llave K_i es generada por la combinación de un corrimiento de elementos y las permutaciones. Con el corrimiento de elementos, las sub llaves generadas serán diferentes en las 16 etapas, caso contrario a la función de

permutación que será la misma en cada una de ellas.

Entonces, la función *F* consta principalmente de cuatro pasos:

1) Expansión.- A cada una de las mitades de 32 bits, se le agregan 8 bits para lograr así, un bloque de 48 bits mediante la expansión de permutación *EP*.
2) Mezcla.- El resultado obtenido en el paso 1, se combina con una sub llave mediante la función lógica *XOR*.
3) Sustitución.- Realizada la mezcla de elementos, dicho bloque es dividido en ocho partes de 6 bits cada uno. Posteriormente, los valores son procesados por las *S-boxes* donde a la salida arroja únicamente 4 bits basados en una función no lineal. En [18] se encuentra mayor detalle de las *S-Boxes.*
4) Permutación.- Dado que se tienen 8 *S-boxes*, se obtiene como resultado 32 valores de salida y se les aplica una función de permutación.

La salida de la función *F* es combinada con el bloque izquierdo mediante una función *XOR*. El resultado de esta operación genera un nuevo bloque de lado derecho; la mitad derecha anterior genera un nuevo bloque izquierdo. Este proceso se repite 16 veces, y de ahí es que sean las 16 rondas del DES.

Si C_i es el resultado de la iteración i, I_i y D_i son las mitades izquierda y derecha de C_i, K_i es la llave de 48 bits para la ronda i, y F es la función que realiza todas las sustituciones, permutaciones correspondientes y opera con *XOR* la llave, entonces, una representación del algoritmo DES está dada por la figura 2.5.

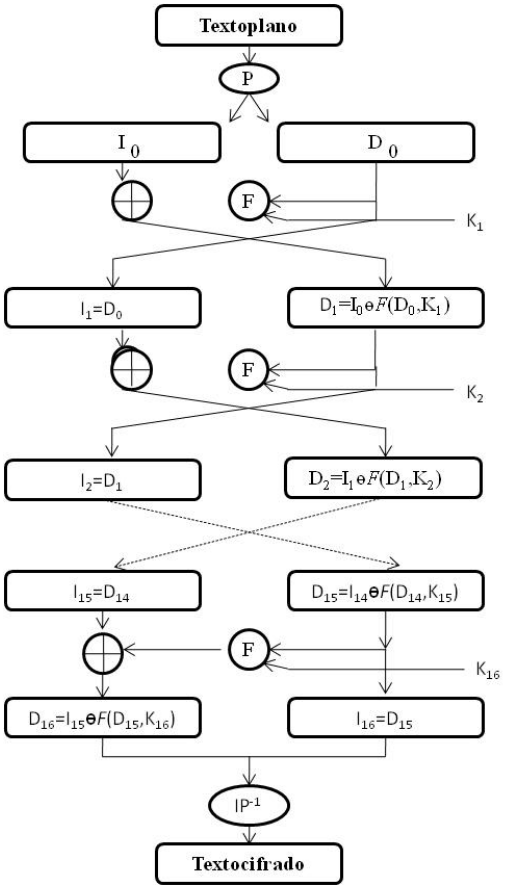

Figura 2.5-Esquema de cifrado DES.

La permutación inicial y la inversa son definidas por tablas establecidas. La entrada a la tabla consiste de 64 bits numerados de 1 a 64. De igual forma, la tabla de permutación, contiene esta misma enumeración. Ahora bien, cada entrada en la tabla de permutación indica la posición de un bit numerado en la salida, el cual también contiene 64 bits.

El proceso de descifrado, usa los mismos pasos que el algoritmo de cifrado, pero utilizando ahora las sub llaves en orden inverso al realizado cuando se cifran los datos.

2.4.3.2 AES

Este algoritmo de cifrado por bloques fue adoptado como estándar para sustituir al DES. El *Advanced Encryption Standard* (AES) fue publicado por NIST el 26 de noviembre del 2001 [22].

Dicho algoritmo, desarrollado por el Dr. Vincent Rijmen y el Dr. Joan Daemen, trabaja con bloques de 128 bits y con un tamaño de llave de 128, 192 y 256 bits.

Los 128 bits del bloque de entrada son copiados a un arreglo *Vector*, el cual es modificado en cada fase del cifrado o descifrado. *Vector* y la llave son almacenados en una matriz, por separado, para su posterior procesamiento. En este caso, se trabaja con una matriz 4x4 elementos bytes para *Vector* y una llave de 128 bits, por lo cual, la llave es copiada y expandida en una matriz que contiene palabras de intercambio. Para 128 bits hay 44 palabras, ya que cada palabra se compone de 4 bytes.

Es importante mencionar que AES se compone de diez etapas, donde cada una de las primeras nueve tiene cuatro sub etapas, y la última, la décima, contiene tres sub etapas. Tales sub etapas, donde se realiza la permutación y sustitución son:

SubBytes.- Cada uno de los bytes es sustituido por otro elemento tomando como referencia una *S-Box*. Con esta fase se obtiene la no linealidad en el cifrado ya que las *S-Box* son creadas con la fusión de una función inversa y una transformación irreversible. Cada byte de *Vector*, es mapeado en un nuevo byte dado lo siguiente: los 4 bits más significativos de un byte son usados para indicar la posición de una fila, y los 4 bits menos significativos, para la columna.

ShiftRows.- Se realiza un corrimiento de elementos, hacia la izquierda del *Vector*, logrando con esto una transposición de elementos. La primera fila no es alterada, la segunda fila, tiene un corrimiento de una posición (un byte); la tercera se recorre dos posiciones; por último, la fila cuatro, tiene un corrimiento de 3 bytes.

MixColums.- Se realiza una combinación de los elementos de las columnas de *Vector* mediante una transformación lineal. Cada valor de la columna de *Vector* es procesado por una función lineal irreversible. Esta fase brinda difusión en el cifrado y además, dicha columna de entrada es tratada como un polinomio GF(2^8). Posteriormente, un módulo (x^4+1) con un polinomio fijo *c(x)* dado por

$$c(x)= \{03\}x^3+\{01\}x^2+\{01\}x+\{02\} \tag{2.11}$$

Cada una de las columnas de *Vector* es procesada individualmente y cada byte de ésta es mapeado a un nuevo valor. Entonces, una transformación dada por *MixColums* en una columna $j(0\leq j\leq 3)$ de *Vector*, puede ser expresado como:

$$\begin{aligned} v'_{0,j} &= (2\cdot v_{0,j}) \oplus (3\cdot v_{1,j}) \oplus v_{2,j} \oplus v_{3,j} \\ v'_{1,j} &= v_{0,j} \oplus (2\cdot v_{1,j}) \oplus (3\cdot v_{2,j}) \oplus v_{3,j} \\ v'_{2,j} &= v_{0,j} \oplus v_{1,j} \oplus (2\cdot v_{2,j}) \oplus (3\cdot v_{3,j}) \\ v'_{3,j} &= (3\cdot v_{0,j}) \oplus v_{1,j} \oplus v_{2,j} \oplus (2\cdot v_{3,j}) \end{aligned} \tag{2.12}$$

AddRoundKey.- En esta fase interviene la combinación de cada elemento de *Vector* con la sub llave, la cual es obtenida a partir de la llave de cifrado original por medio de una función de iteración. Para cada ronda, se obtiene una sub llave diferente. Consecuentemente, *Vector*, con un tamaño de 128 bits es unido a la sub llave, igualmente de 128 bits, mediante el operador *XOR*.

Finalmente, la figura 2.6 muestra el esquema de cifrado de AES.

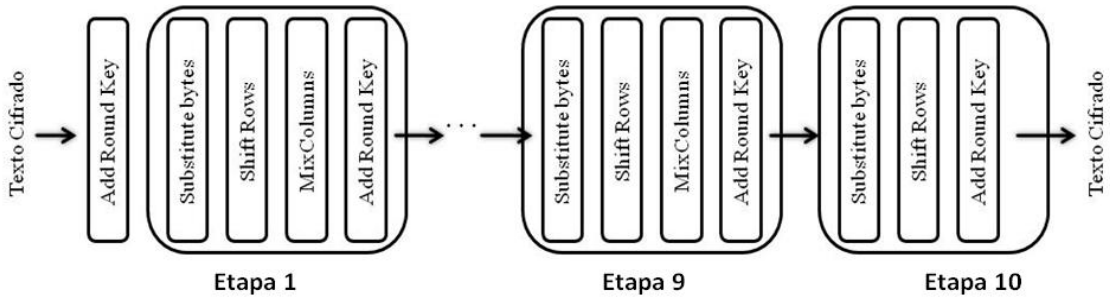

Figura 2.6-Esquema de cifrado AES.

2.4.4 Cifradores Asimétricos

También conocidos como los sistemas de cifrado que manejan dos llaves A_k y B_k. Las dos llaves son diferentes entre sí y a partir de una no se puede deducir la otra. Este tipo de sistemas fue introducido por Diffie y Hellman en el año de 1976.

RSA es uno de los algoritmos que utilizan esta técnica. Dicho algoritmo fue creado en 1977 por Ron Rivest, Adi Shamir y Leonard Adleman [18].

La seguridad que pueden ofrecer estos tipos de cifradores dependerá de la complejidad computacional para poder descubrir alguna de las llaves a partir de la otra. Por ejemplo, qué tan costoso es descubrir la llave privada a partir de la llave pública.

Para efectos de este trabajo, no se abarcará más acerca de este tipo de sistemas ya que, comparados con los cifradores simétricos, son más costosos computacionalmente [23]. Dada su complejidad, también tienen un alto consumo de energía, más que los simétricos [24].

En [25] mencionan que no es conveniente el uso de cifradores asimétricos en sistemas embebidos, no sólo por el tamaño del código, datos y limitantes de procesamiento, sino también por el alto consumo de energía que éstos presentan.

Por lo anterior mencionado, los algoritmos de cifrado asimétricos son descartados, debido a que se pretende diseñar un esquema de compresión-cifrado que pueda ser aplicado incluso en dispositivos con recursos de procesamiento limitados.

2.5 Resumen de Capítulo

En este capítulo se realiza una introducción a las áreas vinculadas a este trabajo de tesis. Se mencionan aspectos de telemedicina, sus inicios y su tendencia. También, se abarca una descripción de las redes de computadoras, tipos de redes existentes y las

diferentes tecnologías inalámbricas, así como sus características.

Por el lado de la compresión de datos, en este capítulo se realiza la descripción de algunos algoritmos y su clasificación. Se menciona la clasificación de los algoritmos de cifrado y la descripción de éstos.

En el siguiente capítulo se hablará acerca de trabajos que relacionen las áreas descritas en este capítulo.

Capítulo 3

Introducción

En este capítulo se menciona trabajo relacionado acerca de la compresión y cifrado de datos que se ha utilizado en el área de telemedicina. Se verán detalles acerca de los esquemas utilizados y hacia donde van dirigidos.

Dado el crecimiento hoy en día de las aplicaciones en telemedicina, surgen nuevas demandas con respecto al desempeño de las mismas. La cantidad de datos manejados en esta área incrementa de forma exponencial, y dicha información es más vulnerable en ambientes inalámbricos.

3.1 Telemedicina

La telemedicina fue propuesta en los años 70's y es definida como el intercambio de información médica usando tecnología de telecomunicaciones tales como el uso de dispositivos de cómputo y redes de computadoras. En [1] definen a la telemedicina como el intercambio de conocimiento y entrega de datos médicos a distancia, usando telecomunicaciones. Entonces, la meta principal de cualquier aplicación de medicina remota, es ayudar física y clínicamente a los médicos a realizar más eficientemente su trabajo y a tener acceso a información médica especializada.

En el capítulo 2 se mencionó que la telemedicina realiza el uso de tecnologías de telecomunicaciones, y, en especial las telecomunicaciones inalámbricas están tomando mucho auge. La tecnología inalámbrica permite cubrir aspectos los cuales, usando tecnología alámbrica, fueran difíciles de lograr. Muchos de los sistemas en telemedicina permiten al usuario final sólo la habilidad para transferir archivos (imágenes, video, audio, grabaciones médicas, etc.) de un lugar a otro [26, 27, 28]. Sólo algunos sistemas brindan más interacción con la localidad remota por medio de audio y/o video trasmitidos en tiempo real [29, 30]. De hecho, los sistemas de

telemedicina inalámbricos tienen un limitado ancho de banda para la transferencia de grandes cantidades de datos que tienden con rapidez, a saturar el medio. Por otro lado, el uso de sistemas inalámbricos presenta vulnerabilidades de seguridad, ya que los datos transmitidos son más susceptibles a ataques de tal suerte que, se vea afectada la confidencialidad e integridad de los mismos.

En la actualidad, existe gran diversidad de sistemas en telemedicina, algunos de ellos requieren más procesamiento de datos que otros. Unos son más complejos o presentan características que los hacen más importantes para el personal médico que otros. Por ejemplo, el monitoreo de latidos en el corazón requiere más procesamiento de datos y tiene más prioridad que el monitoreo de la temperatura en un paciente. No obstante, independientemente del tipo de sistema tratado, en telemedicina se debe contar con aspectos de seguridad para garantizar la confidencialidad de los pacientes y la relación directa con la integridad de los datos, lo cual, es poco común. Un caso en donde se puede apreciar un esquema de seguridad es el brindado por [31], el cual utiliza un sistema de cifrado de datos con el algoritmo RC4 y un proceso de autenticación mediante la dirección MAC para determinar la identidad de una entidad dada. Si bien este sistema de seguridad está aplicado a la transmisión de datos entre sensores con un consumo de energía bajo, pudiera adoptarse en un esquema en el área de telemedicina para los sensores que transmiten datos de índole médico.

Dependiendo del tipo de sistema manejado, la cantidad de información a procesar varía. Cuando se trata de un sistema que maneja gran volumen de información, es necesario considerar ciertas políticas de almacenamiento de la misma y/o maneras de reducirla. Por ello, se debe contar con esquemas de compresión capaz de reducir la información lo más posible, sin afectar considerablemente, según sea el caso, a los datos procesados.

Existe gran variedad de trabajos relacionados con la reducción de información que son aplicados para el área de telemedicina, sin embargo, en su mayoría sólo se

enfocan a determinados tipo de datos, es decir, la variedad de información procesada por un esquema de compresión se encuentra limitado. Por ejemplo, algunos trabajos muestran la compresión para imágenes, ya sean radiografías, encefalogramas, etc.; otros, se enfocan a datos tipo ECG; otros a los tipo audio; por mencionar algunos.

En la literatura, hay pocos trabajos reportados en donde se apliquen ambos procesos, compresión y cifrado. Uno de ellos es el presentado en [32]. En este sistema, para la disminución en el tamaño de datos ECG, previamente a la transmisión de la señal, se aplican redes neuronales. El esquema de compresión utilizado se basa en *wavelets* y el codificador Huffman, logrando así, una compresión de 8 a 1 en ambientes GSM/GPRS. Con respecto a la seguridad manejada, se utiliza el algoritmo de cifrado TEA que permite el cifrado por bloques de 64 bits y con una llave simétrica de 128 bits.

3.2 Compresión en Telemedicina

La mayoría de los trabajos reportados de compresión de datos en telemedicina van enfocados a la compresión de un sólo tipo de datos. Sin embargo, la variedad de datos que se transmiten o almacenan en telemedicina es amplia, con lo cual, varios tipos de datos quedan sin comprimir.

Entre algunos de los ejemplos tenemos a [33] basado en tecnología inalámbrica 3G. Desarrollan un sistema en el cual pueden mandar imágenes de ultrasonido, flujo de ultrasonido, video, sonido y datos para el control de un robot. En este caso, el esquema de compresión utilizado es para el video, ocupando el códec H.263.

Es importante mencionar que los esquemas de compresión empleados son principalmente los que permiten pérdida de información, dejando de lado a los que no lo permiten por su baja compresión con respecto a los primeros como lo menciona [34], donde un análisis comparativo utilizando diferentes tipos de imágenes médicas demuestra que con los algoritmos de compresión con pérdida se tienen mejores

resultados que con los algoritmos sin pérdida. Lo anterior mencionado es apoyado también por [35] que se enfoca principalmente al procesamiento de datos ECG.

En uno de los trabajos recientes, aplican un esquema de compresión en el monitoreo de signos vitales en un paciente realizando una clasificación de los datos ECG para la detección de posibles arritmias en tiempo real [36]. Entonces, dado que los sensores producen gran cantidad de información, los autores utilizan y combinan dos algoritmos de compresión, DPCM (*Differential Pulse Code Modulation*) y BZ2, obteniendo un 88% de compresión que representa un alto y muy buen nivel de compresión.

La contraparte del trabajo anterior, donde de igual forma, se realiza un monitoreo de signos vitales, también de datos tipos ECG, pero utilizando un celular con tecnología 3G y el sistema *Bluetooth*. Sin embargo, como los autores mencionan, un celular tiene un procesador con un desempeño pobre, por tal motivo, no realizan la implementación de algún algoritmo para la compresión de los datos ECG [37].

Un esquema de compresión de imágenes capaz de transferir datos inalámbricamente con la finalidad de que éstos puedan ser procesados por una entidad en un ambiente de desastre y que brinda una arquitectura para búsqueda y rescate de personas en situaciones críticas en las cuales resulta sumamente difícil mover a un paciente es un ejemplo de trabajo con soporte de compresión y ambiente inalámbrico [38].

Hasta el momento, se han mencionado trabajos sin tomar en cuenta la naturaleza de los datos. Ahora, se enfatizarán aquellos trabajos en los cuales hagan uso de la compresión en tipos de datos ECG.

En [35] se presenta una serie de algoritmos de compresión con pérdida de datos que son aplicados a ECG para determinar cuál tiene mejor desempeño con respecto a la reducción de la información. De los cuatro algoritmos evaluados (SAPA2, SKIPPER, TRIM, TPE), se determinó que TPE es mucho más eficiente que los demás.

Por otro lado, en [39] se realiza un estudio de compresión aplicando las siguientes transformadas: Transformada Discreta del Coseno (DCT), Transformada de Hermite (HT) y la Transformada de Legendre (LT). Hay que mencionar que a los coeficientes de cada transformada se le aplica la codificación DPCM. Entonces, los resultados muestran que el mejor comportamiento, con respecto a la reducción de información, es la obtenida aplicando la transformada LT.

Continuando con los algoritmos de compresión con pérdida de información para ECG, [40] muestra otra evaluación de desempeño con los algoritmos FAN [41], AZTEC [42] y ORD que está basado en interpolación lineal. Los resultados muestran el mejor desempeño para el algoritmo ORD ya que es éste el que obtiene una razón de compresión más alta.

Otra forma de realizar la compresión en datos ECG es mediante la utilización de *wavelets*, por tal motivo se menciona [43], donde realizan una comparación de diferentes implementaciones de esquemas de compresión utilizando este tipo de transformadas.

En el trabajo [44] se presenta un nuevo algoritmo para la compresión de datos ECG basado en la extracción local de elementos, filtrado adaptable y de la codificación LZW. La primera parte de este esquema localiza las partes esenciales de la señal: máximos y mínimos; la segunda etapa hace la codificación de los valores obtenidos en la primera mediante la codificación *delta* seguido de la codificación LZW. No obstante, antes de cualquier cosa, se realiza un pre procesamiento aplicando un filtrado de datos mediante *Savitzky-Golay Filter* (SGF) de 6to grado usando una ventana constante de 17 puntos.

Esquema similar es el presentado en [45] donde a diferencia del anterior mencionado, aquí se utiliza *Nearly-Perfect Reconstruction Cosine Modulated Filter Bank* (N-PR CMFB) y posteriormente se aplica el codificador Huffman.

Otro esquema de compresión para datos ECG es el presentado en **[**46**]** que utiliza tecnología GSM y satelital. Dado que realiza la transmisión de datos en tiempo real, es necesario la implementación de un sistema de compresión, por lo cual, realizan la implementación de *Optimal Zonal Wavelet-Based ECG Data Compression* (OZWC). Para este caso, se logra una compresión de 18:1 que resulta bastante aceptable.

Siguiendo en la misma línea de ambientes móviles, se presenta a [47] el cual hace la transmisión de datos ECG. Dicho trabajo realiza la compresión de los datos a partir de la transformada wavelet discreta y posteriormente aplican una modificación del algoritmo de compresión *Run-Length.*

Un caso diferente, es el mostrado por [48] en donde el enfoque de la compresión de los datos ECG se basa en la utilización de tablas Huffman para la transmisión de los mismos en redes telefónicas. El uso de tales tablas de Huffman es opcional, ya que se encuentra configurado por default para trabajar sin ellas. Este trabajo usa el protocolo SCP-ECG [49] para la transferencia de datos tipo ECG.

Otro trabajo relacionado con la compresión de datos para ECG es el presentado en [50] en el cual se aplica la transformada de *Karhunen-Loeve* (KL) con una complejidad de $O((LN)^3)$ y un método basado en *wavelet packets* (WP) [51]. Aplicando este esquema de compresión se obtienen los mejores resultados de toda la literatura, en el orden de 30:1, pero como es de notarse, no puede ser aplicado en ambientes móviles dada su complejidad computacional. De igual forma, el esquema de compresión presentado en [52] basado en la cuantificación vectorial, no puede ser aplicado en ambientes móviles dada su alta complejidad para realizar el proceso de compresión.

Hasta aquí se muestran trabajos que utilizan esquemas de compresión en el procesamiento de los datos aplicados en telemedicina y sus características. Se hace notar el interés y la necesidad del área de telemedicina por tener un mejor

aprovechamiento del medio de transmisión, principalmente el inalámbrico. Por último, se mencionan trabajos basados en la compresión de datos ECG.

3.3 Seguridad en Telemedicina

En sus inicios, la telemedicina sólo se preocupaba por la utilización de los medios de comunicación para lograr objetivos médicos, dejando de lado, cualquier otra necesidad que no se apegara a dicha área. Actualmente, la telemedicina se preocupa, aparte de lo concerniente al área, a otros aspectos que van desde la satisfacción de un paciente, con respecto al servicio otorgado, hasta la seguridad, integridad y confidencialidad de la información manejada. Es por ello que hoy en día se realizan estudios acerca de cómo garantizar lo mencionado anteriormente.

Un trabajo encontrado, con respecto a la seguridad en telemedicina, es el presentado en [53], el cual se enfoca en la autenticación y autorización usando la seguridad NT. Los servicios de seguridad distribuida son brindados por el protocolo de seguridad *NT Lan Manager* (NTLM) soportado tanto por *Windows* como por *Windows NT*. Para el cifrado de los datos se utiliza la tecnología SSL (*Secure Socket Layer*). Otro caso en donde se basan sólo en la seguridad ofrecida por una tecnología ya existente se puede encontrar en [54] que garantiza la integridad de los datos ECG e imágenes tan sólo con el uso del protocolo TCP, es decir, no aplica ningún otro mecanismo diferente al que ya trae implícito dicho protocolo.

Un esquema, igual de simple, es el encontrado en [55] el cual es un dispositivo capaz de procesar doce señales de ECG al mismo tiempo. Las implicaciones de seguridad radican en un pequeño filtrado que se le aplica a los datos, de tal forma que, se elimina la información que identifica a un paciente. También se cuenta con un sistema capaz de hacer la administración de diferentes niveles de acceso a determinada información, es decir, sólo usuarios específicos podrán tener acceso a información determinada. Con la adopción de este esquema se consigue privacidad

para los pacientes.

En [56] se cuenta con un esquema de seguridad el cual hace uso del algoritmo RSA, que es un cifrador de llave pública, en conjunto con DES e IDEA los cuales son algoritmos de cifrado simétrico por bloques. RSA es utilizado para la transmisión de la llave entre las diferentes entidades a utilizar, mientras que DES e IDEA son utilizados para el cifrado de los datos en sí. Dicho esquema es utilizado en un ambiente alámbrico con dispositivos no móviles utilizando una velocidad de transferencia de 10 Mbps en una tecnología Ethernet. Además, este esquema tiene contemplado un sistema de autenticación de usuarios utilizando una pequeña base de datos local.

De igual forma, en [57] se puede encontrar una técnica de seguridad bastante similar a la mencionada anteriormente, ya que utiliza un cifrador asimétrico pero, en esta ocasión es para firmar los mensajes digitalmente mientras que para el cifrado de los datos se utiliza un doble nivel de llave con otro algoritmo de cifrado. También presenta un método de autenticación mediante el uso de credenciales digitales y, dependiendo de esto, se tendrán ciertos permisos. Este sistema en una librería médica digital que está disponible por medio de la web, y adopta medidas de seguridad para mantener la integridad y confidencialidad de los datos.

Siguiendo con este tipo de soluciones, en [58] han desarrollado un sistema que brinda una comunicación segura entre las aplicaciones de un cliente y los servicios de un servidor/middleware. Se utilizan protocolos de intercambio de llaves y RSA. En este trabajo, se puede realizar un monitoreo de signos vitales de un paciente en una aproximación al tiempo real, afirman los autores.

Un esquema similar también se puede observar en [59], el cual protege la información utilizando algoritmos de cifrado de llave pública con la finalidad de

identificar y autenticar *smart cards*. El modo de operación de este esquema es vía web, pudiéndose acceder a la información desde cualquier punto.

Como es de notarse, en los dos últimos trabajos mencionados se utiliza un sólo algoritmo para garantizar la seguridad de los datos y están basados en algoritmos asimétricos o de llave pública. Por otro lado, los primeros trabajos mencionados, se basan en la utilización de dos tipos de algoritmos de cifrado: simétricos y asimétricos.

En todos los trabajos mencionados, a excepción de [31], no consideran la utilización de dispositivos con recursos limitados tales como capacidad de procesamiento, memoria total, etc., ni la utilización de redes inalámbricas, en las cuales, el ancho de banda es limitado. Por tal motivo, en algunos proyectos se presenta la conjunción de dos algoritmos de cifrado, tratando de garantizar aún más la seguridad de los datos manejados, sin embargo, los recursos utilizados por los mismo, pudieran no ser los óptimos para ser usados en ambientes con limitantes así como se aclara en [60] que los pocos trabajos encontrados en esta área, no son aplicables a elementos con ciertas limitantes.

En [61] se mencionan detalles para el monitoreo de signos vitales en pacientes con la utilización de redes inalámbricas *ad hoc*. Dicho trabajo tiene contemplado los requerimientos para el monitoreo de pacientes tales como: la transmisión de signos vitales de rutina y de emergencia, entrega de mensajes en tiempo razonable, conservación de energía, alcance para pacientes fijos y móviles, soporte para dispositivos con capacidad de energía limitada, privacidad y confidencialidad de los datos. En este trabajo, incluso, se mencionan protocolos de administración de energía y *Sleep Strategies* que permiten, a los dispositivos con energía limitada, tener un mayor tiempo de vida.

Con referencia a la seguridad de datos médicos en ambientes móviles, se tiene [62] en el cual se asignan ID’s únicos a los dispositivos con la finalidad de que puedan

autenticarse en el sistema. Existe un proceso de alarma cuando se detecta que un ID es utilizado dos veces al mismo tiempo en el sistema. Aquí se hace mención de diferentes situaciones de seguridad y sus soluciones, sin embargo, no las aplican. También, los autores mencionan que es necesario un estudio más detallado de esquemas de seguridad. Un trabajo similar es encontrado en [63] que agrega un identificador único en los mensajes transmitidos y que además, cuenta con un sistema administrador de grupo. Dicho sistema es llamado AGAPE y su principal desventaja es el consumo relativamente alto de energía, lo que conlleva al poco tiempo de vida cuando es utilizado en entidades móviles.

En la misma línea, para ambientes con limitantes, se presenta el trabajo [60] en el cual hacen uso de un nuevo algoritmo criptográfico simétrico presentado en [64] y asume la existencia de un esquema de distribución de llave robusto y seguro. Este trabajo puede ser usado para tener autenticación, confidencialidad e integridad de los datos transmitidos entre un nodo maestro y los demás nodos secundarios. Para el esquema de distribución de llave, puede ser el mencionado en [65] que sólo puede ser ocupado por cifradores simétricos y que es eficiente con respecto al tiempo y cantidad de memoria utilizados, convirtiéndolo así, en un buen candidato para ser usado en diferentes plataformas.

Otro trabajo es el presentado en [66], el cual está basado en tecnología 3G. Aquí se presenta un esquema de seguridad en el cual, los datos son cifrados utilizando una llave dependiendo de las entidades cercanas. Dicha llave es obtenida aplicando una función *one-way*. Entonces, un mensaje *X* debe ser cifrado como $K_i=hash(K_{i-1})$. Este trabajo está basado en la utilización de un algoritmo de cifrado simétrico e inspirado a partir de [67] el cual provee un administrador de distribución de llaves.

Siguiendo con la búsqueda, se tiene a [68] que hace la utilización de un *Firewall* para el control de tráfico y alertas, cifrado de datos con curvas elípticas embebidas en DESX (*Data Encryption Standard Extended*). Los autores describen una interfaz

inalámbrica para la información clínica basados en una Palm, red digital inalámbrica, transmisión de datos cifrados, servidores de web seguros y respaldo de datos clínicos. También, tienen implementado un sistema en el cual es necesario identificarse mediante un nombre de usuario y una clave para poder acceder a cierta información. Sin embargo, son los mismos autores quienes mencionan errores de seguridad por corregir en dicho trabajo.

Un esquema de seguridad más simple es el encontrado en [69] que protege los datos transmitidos de accesos no autorizados mediante la seguridad ofrecida por la tecnología Bluetooth. Para este caso, se realiza la transmisión de datos tales como saturación de oxígeno y ECG entre sensores y una PDA a una distancia máxima de 10 m.

Uno de los desarrollos software que se encuentra preparado para el soporte de sensores, redes inalámbricas y seguridad es el desarrollado en [70]. El modelo de seguridad empleado en este trabajo brinda cifrado de datos y autenticación de entidades móviles que puede funcionar en ambientes *ad hoc*.

Como se puede notar hasta este momento, existen diferentes trabajos que se enfocan a resolver el problema de la seguridad de los datos en telemedicina. Los esquemas descritos en esta sección presentan diferente complejidad y nivel de seguridad entre cada uno de ellos. Además, tales trabajos abarcan tanto los ambientes móviles como los fijos.

3.4 Aplicaciones para Telemedicina

Anteriormente se mencionaron trabajos relacionados con la compresión de datos en telemedicina, tanto para datos ECG, como para otro tipo de datos. Se mostraron investigaciones conforme a esquemas de seguridad aplicados al área de telemedicina. No obstante, la mayoría de los trabajos mencionados sólo tienen, ya sea el esquema de compresión o el esquema de seguridad, rara vez presentan ambos. Peor aún es la

situación de la existencia de infinidad de trabajos que están dirigidos al área y que no tienen contemplado ningún esquema de compresión de datos, mucho menos, uno de seguridad.

Tal es el caso de [71] en donde se realiza el monitoreo de señales ECG de los pacientes. Este trabajo está enfocado a trabajar en redes celulares 3G y los autores proponen una arquitectura de bajo consumo de energía para un buen desempeño en ambientes móviles.

En la misma línea de investigación se encuentra [72] que propone un sistema integrado (hardware y software) para la adquisición, en tiempo real y de forma inalámbrica, de información cardiaca de pacientes en sus respectivos hogares.

Otro esquema similar se puede apreciar en [73]. En este trabajo se diseña y desarrolla un sistema de monitoreo en tiempo real de pacientes que integra sensores de signos vitales, sensores de localización y una red *ad-hoc* para permitir el monitoreo de forma remota incluso en situaciones de desastre.

De igual forma, otros de los trabajos que se suman a la lista de no presentar esquemas de compresión y/o seguridad, basados en datos ECG, se encuentran [74, 75, 76], [77] que usa Bluetooth, [78] basado en los protocolos GPRS y TCP/IP, [79] y finalmente se menciona a [80] que trabaja en ambientes GSM/GPRS/UMTS.

Así, se puede continuar con una lista interminable de aplicaciones que ofrecen soluciones para ciertos problemas, enfocados al área de telemedicina sin tomar en cuenta la compresión o seguridad de los datos manejados. Entonces, se puede apreciar gran campo de futura investigación que puede aplicarse en telemedicina.

3.5 Resumen de Capítulo

En este capítulo se hizo mención de algunos de los trabajos en el área de telemedicina y que ocupan la compresión de datos y/o esquemas de seguridad para la protección de los mismos.

Es notable que existe una gran deficiencia con respecto a la seguridad de los datos, principalmente, los transmitidos por medios inalámbricos. Quizá se deba a la suposición de que la seguridad es materia de la transmisión de datos y de los protocolos de redes, sin embargo, es bien conocido que existen vulnerabilidades en ellos, por lo cual, es importante considerarlo más aún tratándose de información de pacientes. Por otro lado, la cantidad de información procesada por muchos de los trabajos mencionados, es grande y con tendencia a aumentar más, sin embargo, aún son pocas las investigaciones que se enfocan en la reducción de información para un mejor desempeño en el área de telemedicina. Lo antes mencionado puede ser validado por [81] en donde se realiza un análisis de aplicaciones para telemedicina con más de 125 investigaciones estudiadas, y en las cuales, no hay indicio de esquemas de compresión de datos y/o esquemas de seguridad utilizados.

Existen trabajos en los cuales sí utilizan compresión y/o seguridad de los datos pero es importante tomar en cuenta a qué ambientes están destinados, es decir, hay que hacer ciertas interrogantes tales como: ¿en qué tipo de tecnología de redes puede funcionar?, ¿es factible para ambientes con limitantes de procesador?, ¿cuánta memoria necesitan para realizar el proceso?, ¿qué tan rápido es?, por mencionar algunas. Dadas las respuestas obtenidas a partir de las preguntas anteriores, se puede formar otra clasificación para las aplicaciones en telemedicina.

Finalmente, con lo expuesto en este capítulo, se tiene una idea general de la situación del área de telemedicina con respecto a la reducción de la información que maneja, y a la seguridad de los mismos. También se visualiza la gran cantidad y variedad de

aplicaciones que existen actualmente, por ejemplo, las aplicaciones destinadas a tecnología inalámbrica, y que, resuelven un problema eficientemente, pero, en la mayoría de los casos, dejan de lado la reducción de información y la seguridad de los datos manejados.

Capítulo 4

Introducción

Como se mencionó en el capítulo anterior, se cuenta con una gran variedad de métodos de compresión y de cifrado. En este capítulo se hará referencia el motivo por el cual se eligieron a los algoritmos tratados en esta investigación.

Hay que mencionar que este capítulo justificará la utilización de tales algoritmos en este trabajo.

4.1 Consideraciones

En este trabajo se quiere lograr un esquema de compresión y cifrado de datos que ayude con la problemática existente en el área de telemedicina con el uso de redes inalámbricas. Tal problemática abarca desde el aprovechamiento del medio inalámbrico, hasta la vulnerabilidad que ofrece el mismo en la transmisión de datos. En el primer caso, se busca mandar la mayor cantidad de datos posibles evitando la rápida saturación del mismo. Por otro lado, el segundo punto pretende brindar confidencialidad de información por medios inalámbricos.

Dado lo expuesto en el capítulo 1, acerca de la capacidad de transferencia por un medio inalámbrico, y la vulnerabilidad que tienen éstos al ser transmitidos por el aire, son factores que llevan al estudio de mecanismos que ayudan a eliminar o reducir el impacto negativo propiciado por los mismos.

Por tal motivo, se ha estudiado una serie de algoritmos tanto de compresión como de cifrado de datos, los cuales, para su aplicación a dicho esquema, deberá ser factible, óptimo y transparente para el usuario final.

Para el caso de los algoritmos de compresión de datos, se hace énfasis en la utilización de los algoritmos de *compresión sin pérdida*, dado que permiten recuperar

la información en su forma original, a diferencia de los *compresores de datos con pérdida,* que recuperan una aproximación de la información original, lo cual, pudiera representar problemas al tratar con cierto tipo de información que no tolera alguna pérdida como lo es el *texto.*

A continuación, se mencionan características a tomar en cuenta en el análisis de los algoritmos de compresión y de cifrado para establecer un compromiso de equidad entre ellas.

1. *Rapidez*. Deberá ejecutar el mayor número de instrucciones en el menor tiempo posible.
2. *Razón de compresión*. Aplica para los algoritmos de compresión y es calculada mediante la ecuación 2.1.
3. *Simplicidad*. Realizar procedimiento con operaciones simples ya que, entre mayor complejidad tenga el algoritmo, mayor será el consumo de energía. Por el contrario, entra más simple sea, el consumo de energía será menor.
4. *Recursos a utilizar.* Para que el esquema de compresión y cifrado a utilizar pueda ser implantado en la mayor cantidad de dispositivos, trátese de un equipo de cómputo tradicional o un dispositivo móvil con recursos limitados, entonces, los algoritmos más factibles para este caso serán aquellos que utilicen el menor número de recursos tales como la cantidad de memoria para realizar un procedimiento.
5. *Nivel de Seguridad.* Aplica para los cifradores, los cuales deberán tener una seguridad razonable que complique el acceso a la información de una entidad no autorizada.
6. *Susceptibilidad a errores.* Capacidad con la que cuentan los algoritmos para que los posibles errores generados por factores externos, tales como la falla del medio de transmisión, afecten lo menos posible a la información original.

Tanto para el caso de los compresores, como para los algoritmos de cifrado, los puntos anteriores deberán verse reflejados en la compresión-descompresión y cifrado-descifrado de los datos respectivamente.

4.2 Comparación entre algoritmos de compresión

Dado que el enfoque de este trabajo será en los algoritmos de compresión sin pérdida, los compresores con pérdida de datos quedarán descartados de esta comparación.

En el área de la compresión de datos, en el caso de los algoritmos de compresión sin pérdida de propósito general destacan los siguientes algoritmos: Huffman, Codificador Aritmético, Lempel-Ziv y RLE principalmente. Hay otros algoritmos tales como ABO (*Adaptive Binary Optimization*) que es utilizado para gráficos, MSU para video o FLAC que se utiliza para la compresión de audio. Para efectos de esta investigación se trabajará con los algoritmos de propósito general dado que la telemedicina involucra datos de pacientes, numéricos, etc., donde es necesario no optimizar para un tipo específico de datos.

Primeramente se limita a cinco el número de algoritmos de compresión a analizar: Huffman, Codificador Aritmético, RLE, LZW y LZSS. Lo anterior se debe a que son los compresores sin pérdida de datos de propósito general más utilizados en la actualidad. Es importante mencionar que estos algoritmos presentan características propias que los hacen funcionar de diferente forma, es decir, es posible que para un cierto tipo de dato, un algoritmo se comporte mejor que otros, mientras que en diferente ocasión y con otros factores, suceda lo contrario.

4.2.1 Huffman

Como se menciona en el capítulo 2, este compresor puede trabajar de forma estática y de forma dinámica. En [82] mencionan que la construcción de un código Huffman puede ser realizado en $O(n \log n)$, donde n es el tamaño del alfabeto.

El Huffman Estático es usado más que el Dinámico debido a que el Estático presenta una gran velocidad de procesamiento. Además, el Estático tiene más robustez contra errores, ya que los códigos generados tienden a resincronizarse rápidamente después de un error de transmisión [83]. Por el contrario, el compresor Dinámico es mucho más susceptible a errores de este tipo. Lo anterior se puede recalcar con lo mencionado por Salomon [8] que una de las desventajas más obvias de los códigos de longitud variable es su vulnerabilidad a errores. No obstante, el mismo autor menciona que las personas que usan códigos Huffman están enteradas, desde hace tiempo, que estos códigos se recuperen rápidamente después de un error.

En ambos compresores el espacio necesario para poder ejecutar el proceso está dado por $O(n)$, donde n es el tamaño del alfabeto.

Una comparación entre Huffman y el Codificador Aritmético [82] revela que al momento de comprimir información, el Huffman Estático es por lo menos dos veces más rápido que el Codificador Aritmético, siendo este último en la mayoría de los casos, ligeramente más rápido que el Huffman Dinámico. Por otro lado, en el proceso de descompresión, el Codificador Aritmético es ocho y diez veces más lento que el Huffman Dinámico y el Huffman Estático respectivamente.

Una de las principales desventajas del Huffman Estático se ve reflejada en la compresión de los datos, donde resulta necesario analizar dos veces la información a comprimir, es decir, se realizan dos pasadas. En la primera, se determinan las frecuencias de cada uno de los elementos en el mensaje, seguido por la construcción del árbol Huffman. La segunda etapa codifica la información de acuerdo a lo obtenido en la primera. Esta forma de procesar la información repercute principalmente en la velocidad del algoritmo, lo que puede provocar retardos cuando es usado en redes de comunicaciones [84]. Por otro lado, en el Dinámico no es necesario realizar las dos etapas especificadas para el Estático, ya que las frecuencias se van modificando conforme se leen nuevos datos.

Con el Huffman Dinámico se tiene la ventaja de un ahorro con respecto a la cantidad de información que se transmite por el medio. Dado que las probabilidades de los elementos cambian mientras siga existiendo información a comprimir, no es necesario mandar el árbol de codificación al receptor para que éste sea capaz de descomprimir la información. Por el medio de transmisión solo irá la información comprimida, sin necesidad de información extra, y el receptor será el encargado de decodificarla correctamente. Lo anterior es una gran diferencia con respecto al Estático, ya que éste tiene la necesidad de mandar por el medio de transmisión el árbol de Huffman para decodificar la información, con lo cual se transmite información extra por el medio.

Por el lado de la descompresión, para el Huffman Estático, una vez que ya se cuenta con el árbol de codificación, y que éste es transmitido a la entidad que realizará el proceso de descompresión, es necesario recorrer el árbol partiendo del nodo principal, y dependiendo de los datos comprimidos, determinar el bit correcto siendo 1 ó 0. Se lee consecutivamente hasta que se llegue al final del árbol obteniendo el dato original a partir de los datos codificados. Finalmente, el descompresor devuelve una serie de bits, que representan a la información original. El proceso se repite mientras exista información a decodificar. La descompresión en el Huffman Dinámico varía, ya que no es necesario transmitir el árbol de codificación. En ambos algoritmos, el proceso de descompresión es relativamente sencillo.

En el trabajo de Knuth [85], se menciona una adaptación al Huffman Dinámico que resulta conveniente incluso para ambientes de bajo poder computacional ya que la memoria requerida es de $O((2n-1)c)$, donde c es el costo (en bytes o bits) de almacenar la información de un nodo en memoria. Moffat *et al.* [86] proponen un codificador dinámico usando $n+O(1)$ de espacio de trabajo teniendo las probabilidades de los elementos en forma ordenada. Para otras implementaciones de un método dinámico ver [87, 88, 89, 90].

Es importante mencionar que McIntyre *et al.* [91] destacan el uso de un compresor estático para archivos pequeños o cadenas cortas ya que, el codificador dinámico requiere un gran número de símbolos para que sea eficiente.

Un algoritmo interesante propuesto por Varn [92], reduce el tiempo de comunicación y el consumo de energía. Dicho algoritmo permite optimizar un código considerando un costo relativo de los símbolos de salida. Lo anterior es un factor crítico para ser considerado en dispositivos de poca energía tales como PDAs, celulares, etc.

El codificador Huffman, tanto Estático como Dinámico, brinda generalmente una razón de compresión entre 0.5 y 0.4, lo cual lo hace bastante atractivo ya que puede trabajar con una gran variedad de datos. No obstante, la efectividad de Huffman es sensible a las características de la información procesada. Una de las principales limitantes de estos algoritmos, es que no puede ser usado eficientemente en condiciones cuando la representación de los datos originales es cerca o debajo de 1 bit por símbolo, por ejemplo un alfabeto binario. Por otro lado, Huffman puede producir una compresión cerca de la óptima, pero ésta no puede ser obtenida en situaciones donde la distribución de la fuente sea desconocida.

En la tabla 4.1 se muestran los resultados obtenidos para el proceso de compresión utilizando Huffman Estático, mientras que en la tabla 4.2 se muestran los de Huffman Dinámico.

Archivos	Original Bytes	Comprimido Bytes	Comp Seg	Descomp Seg	RC
alice29	152089	87931	0.031	0.031	0.578
asyoulik	125179	76021	0.031	0.031	0.607
cp	24603	16473	0.001	0.001	0.670
fields.c	11150	7311	0.000	0.000	0.656
grammar.lsp	3721	2413	0.000	0.000	0.648
kennedy	1029744	463395	0.210	0.155	0.450
icet10	426754	250838	0.108	0.078	0.588
plrabn12	481861	275861	0.125	0.078	0.572
ptt5	513216	107121	0.093	0.047	0.209
sum	38240	26489	0.016	0.015	0.693
xargs.1	4227	2838	0.000	0.000	0.671
TOTAL	2810784	1316691	0.615	0.437	0.468

Tabla 4.1-Resultados de Huffman Estático.

Archivos	Original Bytes	Comprimido Bytes	Comp Seg	Descomp Seg	RC
alice29	152089	87718	0.094	0.094	0.577
asyoulik	125179	75902	0.078	0.078	0.606
cp	24603	16325	0.016	0.016	0.664
fields.c	11150	7154	0.016	0.016	0.642
grammar.lsp	3721	2267	0.000	0.000	0.609
kennedy	1029744	443718	0.562	0.562	0.431
icet10	426754	249575	0.250	0.250	0.585
plrabn12	481861	275676	0.265	0.265	0.572
ptt5	513216	106298	0.140	0.140	0.207
sum	38240	25940	0.031	0.031	0.678
xargs.1	4227	2702	0.000	0.000	0.639
TOTAL	2810784	1293275	1.452	1.452	0.460

Tabla 4.2-Resultados de Huffman Dinámico.

Con la utilización de Huffman Estático, se obtiene una razón de compresión de 0.468 con un tiempo de procesamiento de 0.615 segundos para todos los archivos probados. Utilizando Huffman Dinámico, el tiempo de compresión es de 1.452 segundos y la razón de compresión obtenida es 0.460. Se puede apreciar una ligera mejora en la

compresión de los datos utilizando el Dinámico, sin embargo, el tiempo de procesamiento es mejor en el Estático.

Dada las tablas 4.1 y 4.2, se comprueba lo mencionado acerca del desempeño tanto en tiempo, como en la compresión obtenida con la utilización de estos dos algoritmos. Se afirma que la velocidad de compresión por parte de uno, es indiscutiblemente mayor a la del otro, y que, la reducción de la información varía ligeramente entre cada uno de ellos. En el tiempo de descompresión, se presenta el mismo fenómeno que en la compresión: Estático supera a Dinámico.

4.2.2 Codificador Aritmético

Este codificador se caracteriza por su alta razón de compresión, la cual se encuentra, comúnmente, por encima de la ofrecida por el compresor de Huffman. Howard *et al.* [93] afirman que el codificador aritmético dará, en la mayoría de los casos, mejor razón de compresión que los códigos *prefix* tales como Huffman. No obstante, Huffman es más rápido tanto en el proceso de compresión como en la descompresión.

Lo anterior se debe principalmente a la estructuración del algoritmo. Dicho proceso consiste de operaciones más complejas y precisas, de ahí el que ofrezca una mejor razón de compresión, pero, con un lento tiempo de procesamiento. Otra de las ventajas sobre Huffman, es que el Codificador Aritmético calcula la representación de la información al vuelo, en consecuencia, se necesita menos cantidad de memoria para la compresión permitiendo una adaptación en tiempo.

Continuando en la comparación contra el compresor Huffman, el Codificador Aritmético permite estar más cerca del límite establecido por la entropía. El proceso de adaptar el Codificador Aritmético para trabajar de forma dinámica es más sencillo que con el Huffman, pero, a diferencia de éste último, el primero es mucho más lento en el proceso de compresión y menos intuitivo.

El uso del Codificador Aritmético, como se menciona en [11], es recomendable cuando los símbolos *S* de un alfabeto *A* tengan altas probabilidades de aparición; cuando en el modelo *M,* la probabilidad tenga gran variedad de distribución y cuando cada uno de los símbolos sea codificado como la culminación de una secuencia de decisiones de bajo nivel. Ahora bien, entre algunas de sus principales ventajas se tienen:

- Cuando es aplicado a información independiente y está idénticamente distribuida, la compresión de cada símbolo es probablemente óptima.
- Es efectivo para ser aplicado en una gran variedad de situaciones obteniendo diferentes niveles de compresión.
- El proceso fundamental es aritmético, el cual es soportado eficientemente por todos los procesadores de propósito general.
- El Codificador Aritmético, en la mayoría de los casos, da mejor compresión que los códigos prefijos.
- Tiene gran flexibilidad para ser usado con cualquier otro modelo que pueda proveer una secuencia de probabilidades.
- Tiene mejor desempeño en compresión dinámica, especialmente con alfabetos grandes.

No obstante, en la actualidad el Codificador Aritmético no es popular en ambientes donde se requiere gran velocidad de compresión o en dispositivos con recursos limitados de procesamiento. Algunas de sus desventajas son:

- La complejidad de las operaciones aritméticas resulta ser excesiva.
- Las implementaciones eficientes de este codificador son difíciles de entender y por ende, difíciles de implementar.
- En la mayoría de los casos, su implementación es lenta por causa de las multiplicaciones y divisiones requeridas en el proceso. Moffat *et al.* [94], en su implementación mencionan que en la codificación de un símbolo se

requieren dos multiplicaciones y una división, mientras que para la decodificación, tres multiplicaciones y tres divisiones son requeridas.

- Es necesario especificar el fin de un archivo.
- Tiene poca resistencia a los errores ya que los símbolos no son codificados individualmente, entonces, si hay un cambio en un bit, se refleja en el valor arrojado por el codificador, el cual producirá un mensaje totalmente diferente.
- El proceso de decodificación tiene una complejidad computacional más alta que la compresión de los datos.

Se puede notar que este algoritmo es poco factible en ambientes donde existan dispositivos con recursos limitados de procesamiento y almacenamiento. Además, las operaciones aritméticas requieren alta precisión para que no se pierda la calidad y desempeño de la razón de compresión.

Este algoritmo de compresión puede funcionar de forma estática, semi-estática o dinámica. En la primera, las probabilidades para cada uno de los elementos dentro del alfabeto, permanecen de forma constante durante todo el proceso de compresión, dando como resultado un desempeño pobre [95]. La versión semi-estática hace uso de una *pasada* para determinar inicialmente las probabilidades y en algunos casos, esta versión tiene un mejor desempeño que la ofrecida por el codificador dinámico [13]. Finalmente, en el codificador dinámico, la actualización de la probabilidad en los símbolos se realiza conforme vayan apareciendo, es decir, dinámicamente, brindando mejor desempeño que las dos anteriores, pero utilizando una estructura de datos más compleja.

Para la versión semi-estática, la longitud del código resultante está dada por la ecuación 4.1 y para el dinámico por la ecuación 4.2

$$L_{SE} = -log\left[\frac{\left(\prod_{i=1}^{k} c_i!\right)}{t!}\right] \tag{4.1}$$

$$L_D = -log\left[\frac{\left(\prod_{i=1}^{k} c_i!\right)}{n^{\overline{t}}}\right] \tag{4.2}$$

Donde t = tamaño del archivo en bytes; n = número de símbolos en el alfabeto; k = número de símbolos diferentes que aparecen en el archivo; c_i = número de ocurrencias del símbolo $i$to en el archivo; $log\ x = log_2\ x$; $A^{\bar{B}} = (A(A+1)\ldots(A+B-1)$.

La complejidad de este compresor radica principalmente en las operaciones aritméticas utilizadas tal como la división que requiere el uso de *punto flotante*, brindando mayor exactitud pero afectando el tiempo de procesamiento. Además, el uso de elementos de *punto flotante*, hace compleja la implementación de este compresor. Sin embargo, los puntos finales del intervalo se pueden almacenar como un número entero lo suficientemente largo como para representar a un número *flotante* con la desventaja de que existirá una degradación en el desempeño de la compresión por tratar con elementos inexactos. Una versión de un compresor aritmético, utilizando números enteros se encuentra en [93]. Aparte de sustituir los números *flotantes* por enteros, se puede sustituir el proceso de división por un conjunto de multiplicadores especificados en la ecuación 4.4.

El uso de multiplicadores, para sustituir la división tiene el efecto de reducir la complejidad del algoritmo. Sin embargo, esto sólo es aplicable cuando la información tiene una entropía pequeña, hasta 4 bits/símbolo. En caso contrario, y dada la ecuación 4.3, usando la división se obtiene un mejor desempeño en cuanto a tiempo de procesamiento se refiere.

$$C_s \leq (v-b)/l < c_{s+1} \tag{4.3}$$

$$l\ C_s \leq (v-b) < l\ c_{s+1} \tag{4.4}$$

Donde c es el arreglo con distribución acumulativa; v es el valor del código; b y l son el intervalo y la longitud respectivamente.

Diferentes implementaciones de este codificador se encuentran en [94, 96, 97, 98] y en [13] se muestra un análisis matemático de este compresor.

Los resultados obtenidos aplicando el Codificador Aritmético están dados por la tabla 4.3 y la tabla 4.4. En este caso, la razón de compresión para el Codificador Aritmético Estático y Dinámico, es de 0.444 y 0.455 respectivamente. Es notable la pequeña diferencia entre ellos. Ahora bien, el tiempo de compresión va desde los 3.031 segundos para el estático, hasta los 4.722 segundos para el dinámico. La diferencia en los tiempos de descompresión es relativamente proporcional a la diferencia obtenida en la compresión. Resulta notorio, en el estático, que el tiempo de compresión es menor al obtenido en la decodificación de los datos. Para el dinámico, lo anterior no aplica, pero la tendencia, con archivos más grandes, es que el tiempo sea mayor en el proceso de descompresión.

Archivos	Original Bytes	Comprimido Bytes	Comp Seg	Desc Seg	RC
alice29	152089	87336	0.32	0.34	0.574
asyoulik	125179	75723	0.38	0.393	0.605
cp	24603	16298	0.094	0.094	0.662
fields.c	11150	7158	0.047	0.047	0.642
grammar.lsp	3721	2299	0.016	0.016	0.618
kennedy	1029744	433557	0.895	1.215	0.421
icet10	426754	248632	0.325	0.378	0.583
plrabn12	481861	274414	0.367	0.443	0.569
ptt5	513216	74804	0.43	0.67	0.146
sum	38240	24739	0.141	0.141	0.647
xargs.1	4227	2737	0.016	0.016	0.648
TOTAL	2810784	1247697	3.031	3.753	0.444

Tabla 4.3-Resultados de Codificador Aritmético Estático.

Archivos	Original Bytes	Comprimido Bytes	Comp Seg	Desc Seg	RC
alice29	152089	87134	0.391	0.391	0.573
asyoulik	125179	75524	0.407	0.407	0.603
cp	24603	16297	0.094	0.094	0.662
fields.c	11150	7162	0.047	0.047	0.642
grammar.lsp	3721	2302	0.016	0.016	0.619
kennedy	1029744	460207	1.359	1.359	0.447
icet10	426754	249401	0.844	0.844	0.584
plrabn12	481861	273279	0.954	0.954	0.567
ptt5	513216	77964	0.453	0.453	0.152
sum	38240	25635	0.141	0.141	0.670
xargs.1	4227	2740	0.016	0.016	0.648
TOTAL	2810784	1277645	4.722	4.722	0.455

Tabla 4.4-Resultados de Codificador Aritmético Dinámico.

4.2.3 LZW

En el capítulo 2 se menciona que este algoritmo está basado en un diccionario en el cual se almacenan los símbolos que son procesados. De igual forma, se destaca que LZW es una mejora del algoritmo LZ78, que a su vez es una mejora del compresor LZ77. La complejidad del algoritmo LZ77 es $O(n^2)$, mientras que la de LZ78, y por ende la de LZW, es de $O(n)$ donde n es el número de caracteres en la entrada [10]. En [8, 99] se encuentran variantes de los algoritmos basados en Lempel-Ziv.

Los compresores que se basan en diccionario, pueden ser clasificados en estáticos o dinámicos dependiendo de cómo se forma el diccionario. Por otro lado, los diccionarios dinámicos son clasificados en unidireccional y bidireccional. En el primero, todos los apuntadores tienen la misma dirección, mientras que en el segundo, los apuntadores pueden funcionar en dos direcciones. LZW está basado en los diccionarios dinámicos unidireccionales, lo cual permite que la compresión de los datos se pueda realizar en una pasada. De igual forma, se realiza sólo una pasada para el proceso de descompresión de los datos. Lo anterior es factible para que LZW pueda ser aplicado en ambientes donde la capacidad computacional y recursos de las

entidades que vayan a descomprimir la información sea menor que las entidades encargadas de realizar el proceso de compresión. Un punto más a favor, a partir de que la compresión y descompresión de los datos se realiza en una sola pasada, es que dicha característica permite la utilización de este esquema en implementaciones *on-line* donde se quiere tener el menor retardo posible.

LZW es distinguido por su simplicidad lógica, y la cual conlleva a que su implementación no sea costosa. Además, tiene la capacidad de ejecutar el proceso de compresión y descompresión rápidamente.

Michael Dipperstein [100], menciona que el decodificador no requiere ninguna información extra proveniente del compresor para que se pueda realizar la decodificación. Esta característica permite al LZW el poder trabajar en redes de comunicación donde el ancho de banda sea limitado como las redes inalámbricas.

En [100] se realizan una serie de experimentos con diferentes tamaños de códigos, concluyendo que el tamaño del diccionario con mejor desempeño es el representado por 12 bits (4096 posiciones en el diccionario). La comparación estuvo dada por los tamaños de diccionario entre 9 y 15 bits. Hay que tener en cuenta el tamaño del diccionario, ya que éste dará la pauta a la memoria utilizada y al tiempo necesario para realizar la búsqueda de *matches.* Entre más grande sea el diccionario, mayor será el tiempo de búsqueda y mayores los recursos a utilizar. Dicho lo anterior, uno de los principales requisitos del LZW es que se tenga la capacidad de poder almacenar el diccionario de un tamaño *n*, donde *n* es el tamaño de los códigos en bits. Entonces, para almacenar el diccionario se debe tener un espacio necesario de 2^n.

Se mencionó que el tiempo de búsqueda aumenta o disminuye según el tamaño del diccionario. La forma más común de insertar elementos en el diccionario es secuencialmente, el cual tiene una complejidad de *O(N)*, donde *N* representa el número de posiciones en el diccionario. Otra forma de insertar elementos en

posiciones disponibles del diccionario es por medio de un función *hash,* la cual, en el mejor de los casos, puede tener una complejidad de *O(1)*.

Como conclusión acerca del LZW se puede mencionar que es un algoritmo que logra buenos resultados en la razón de compresión y que es relativamente fácil de implementar. Este algoritmo se puede utilizar en aplicaciones que demanden gran velocidad de procesamiento. Por otro lado, LZW también puede ser usado por dispositivos con recursos limitados de procesamiento y almacenamiento. Por último, LZW no crea *overhead* de información extra mandada por el medio para que pueda realizarse la descompresión de los datos.

En comparación con el codificador aritmético, LZW presenta la ventaja de su velocidad, ya que la codificación es extraída directamente del diccionario [93].

Archivos	Original Bytes	Comprimido Bytes	Comp Seg	Descomp Seg	RC
alice29	152089	72322	0.055	0.000	0.476
asyoulik	125179	62976	0.055	0.000	0.503
cp	24603	12228	0.000	0.000	0.497
fields.c	11150	5316	0.000	0.000	0.477
grammar.lsp	3721	2116	0.000	0.000	0.569
kennedy	1029744	419220	0.165	0.055	0.407
icet10	426754	222229	0.055	0.055	0.521
plrabn12	481861	234978	0.055	0.055	0.488
ptt5	513216	70228	0.055	0.055	0.137
sum	38240	30940	0.000	0.000	0.809
xargs.1	4227	2691	0.000	0.000	0.637
TOTAL	2810784	1135244	0.440	0.220	0.404

Tabla 4.5-Resultados de LZW.

Para efectos de analizar la capacidad de compresión del algoritmo LZW, la tabla 4.5 muestra los resultados obtenidos. Véase que se obtiene una razón de compresión de 0.404 que hasta ahora, es la mejor obtenida, superando tanto a Huffman como al Codificador Aritmético, en Estático y Dinámico para cada uno de ellos. Por el lado

del tiempo de compresión, se obtuvo un total de 0.440 segundos y 0.220 segundos para la decodificación de los datos. Tanto en compresión como en tiempo de descompresión, LZW resulta tener mejor desempeño que los resultados antes obtenidos por otros compresores.

4.2.4 LZSS

Es un algoritmo de compresión, que al igual que el LZW, está basado en diccionario. LZSS es una variante del compresor LZ77, por tal motivo, una de las desventajas de este algoritmo es que tiene un tamaño limitado *L* en el buffer de búsqueda. El tamaño de las cadenas *matched* está limitado a *L*-1. El aumento en el tamaño de este buffer produce como resultado mejor razón de compresión, pero a su vez, el tiempo de compresión es mayor.

Una de las ventajas de LZSS es la rapidez con la que realiza la decodificación de los datos y que además, este proceso es realizado por operaciones simples. Entonces, el proceso de descompresión no es costoso en lo que se refiere a velocidad y consumo de energía. Otro punto a favor del LZSS es que es una mejora del algoritmo LZ77, consecuentemente, LZSS tiene buena razón de compresión con un ahorro de memoria utilizada para la codificación.

En la tabla 4.6 se puede constatar la rapidez de descompresión de los elementos. Se obtiene un tiempo de 0.001 segundos, siendo el mejor tiempo para el proceso de decodificación de la información. El tiempo de compresión es de 3.573 segundos con una razón de compresión de 0.367 que es mejor incluso que la presentada por el algoritmo LZW y por ende, que el Huffman y Codificador Aritmético.

Archivos	Original Bytes	Comprimido Bytes	Comp Seg	Descomp Seg	RC
alice29	152089	73117	0.055	0.000	0.481
asyoulik	125179	65551	0.055	0.000	0.524
cp	24603	10941	0.001	0.000	0.445
fields.c	11150	3841	0.000	0.000	0.344
grammar.lsp	3721	1537	0.000	0.000	0.413
kennedy	1029744	288123	2.802	0.000	0.280
icet10	426754	199722	0.220	0.000	0.468
plrabn12	481861	262943	0.220	0.000	0.546
ptt5	513216	105311	0.220	0.000	0.205
sum	38240	17803	0.000	0.000	0.466
xargs.1	4227	2124	0.000	0.000	0.502
TOTAL	2810784	1031013	3.573	0.001	0.367

Tabla 4.6-Resultados de LZSS.

4.2.5 RLE

Este algoritmo es uno de los más sencillos de implementar. Su principal ventaja es que requiere pocos recursos de procesamiento, lo cual lo convierte en un algoritmo útil en ambientes de recursos limitados.

Otra ventaja del RLE es su gran velocidad para realizar la compresión y descompresión de los datos. Tomando en cuenta los algoritmos mencionados anteriormente, RLE es el que mejor desempeño tiene con respecto a la velocidad de procesamiento. En ambas partes del algoritmo, tanto el compresor como el descompresor, se cuenta con una velocidad excesivamente rápida.

No obstante, este algoritmo no tiene una alta razón de compresión para diferentes tipos de alfabetos. RLE sólo es efectivo cuando en la información de entrada, existen por lo menos cuatro elementos repetidos consecutivamente por cada símbolo. Otro forma de trabajo eficiente es cuando el alfabeto tiene pocos símbolos y muchas repeticiones de éstos, principalmente en alfabetos binarios, que, a diferencia del algoritmo de Huffman, RLE da una alta razón de compresión.

Entonces, si el ambiente de aplicación, específicamente el tipo de datos a manejar presenta una gran cantidad de repeticiones consecutivas o un alfabeto pequeño, la utilización de RLE resultará factible, de lo contrario, RLE empeorará el desempeño del esquema generando incluso más información que la original.

En este algoritmo, si un error se presenta, la información puede tener variaciones considerablemente no deseadas ya que se está haciendo una representación de cuántas repeticiones existen de un símbolo dado. No obstante, los errores afectarían realmente a la razón de compresión, ya que, al tratarse de repeticiones, el mensaje podría seguir siendo entendible para el usuario final.

Los resultados obtenidos aplicando RLE se muestran en la tabla 4.7. Tales resultados comprueban la rapidez de procesamiento, tanto en la compresión como en la descompresión, obteniendo tiempos de 0.079 y 0.017 segundos respectivamente. También muestra la baja razón de compresión obtenida, que para este caso, es de 0.849. Es fácil determinar que RLE presenta excelentes tiempos en ambos procesos, pero que, definitivamente, tiene el peor desempeño con respecto a la compresión.

Archivos	Original Bytes	Comprimido Bytes	Comp Seg	Desc Seg	RC
alice29	152089	150989	0.000	0.000	0.993
asyoulik	125179	125193	0.000	0.000	1.000
cp	24603	24617	0.000	0.000	1.001
fields.c	11150	10918	0.000	0.000	0.979
grammar.lsp	3721	3649	0.000	0.000	0.981
kennedy	1029744	1029758	0.031	0.000	1.000
icet10	426754	416293	0.016	0.016	0.975
plrabn12	481861	481875	0.000	0.000	1.000
ptt5	513216	105646	0.015	0.000	0.206
sum	38240	34320	0.016	0.000	0.897
xargs.1	4227	4241	0.000	0.000	1.003
TOTAL	2810784	2387499	0.079	0.017	0.849

Tabla 4.7-Resultados de RLE.

Hasta el momento se han mostrado características, detalles y resultados a tomar en cuenta para la selección del algoritmo de compresión que será ocupado en el esquema de compresión-cifrado. A continuación, se mostrarán aspectos a considerar para los algoritmos de cifrado.

4.3 Comparación entre algoritmos de cifrado

En el lado del proceso de cifrado de datos también existe una clasificación con respecto a los algoritmos, ya que como se mencionaba en el capítulo 2, existe cifrado simétrico y cifrado asimétrico. En el primero, la misma clave se utiliza tanto para cifrar como para descifrar la información, por tanto, ambas entidades deberán tener dicha clave. Por otro lado, en el cifrado asimétrico existe una clave privada y una pública que evita la necesidad de mandarla por el medio de transmisión.

Algunas de las características en el uso de sistemas simétricos son:

- Confidencialidad.
- Autenticación parcial.
- Alta velocidad de cifrado.
- El emisor y el receptor comparten una llave.
- Llaves de corta longitud.
- Nivel de seguridad bueno.

Con un sistema asimétrico se tiene:

- Confidencialidad.
- Autenticación total.
- Diferentes llaves para el emisor y receptor.
- Llaves de gran longitud y con gran tiempo de vida.
- La complejidad computacional es mucho mayor que en un sistema simétrico.
- Los algoritmos asimétricos tienen más alto costo de energía.

- Requieren mayor procesamiento.
- El tamaño de su implementación, supera al código de un cifrador simétrico.

Entonces, mencionadas las características de los algoritmos de cifrado simétricos y asimétricos, es necesario decir que se hará énfasis en el cifrado simétrico por resultar más factible para esta investigación ya que se está considerando que este esquema pueda ser aplicado incluso en ambientes móviles con recursos limitados y que la telemedicina requiere de procesos rápidos y transparentes para el usuario final. Además, se toma en cuenta lo afirmado por [24] acerca de que los cifradores asimétricos tienen un costo de energía más alto que los obtenidos por los simétricos; de lo mencionado en [25] donde los cifradores de llave pública no son aptos para sistemas con recursos limitados; de los resultados obtenidos en [101]; de [23] donde se especifica que los cifradores simétricos tienen básicamente el mismo cómputo tanto en el cifrado como en el descifrado, mientras que en los cifradores asimétricos, el cómputo es diferente para cada uno de ellos; y de [102] donde se menciona que los algoritmos de llave pública, son altamente costosos computacionalmente.

En la figura 4.1 se muestra el consumo de energía de los asimétricos y los simétricos, siendo el segundo el que consume menos energía.

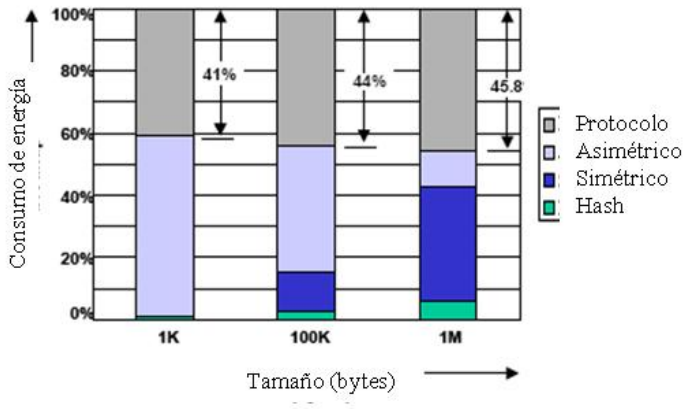

Figura 4.1-Consumo de energía en cifrado simétrico y asimétrico.

Dada la elección de los sistemas simétricos como parte del esquema a utilizar para esta tesis, se tiene que los cifradores simétricos se subdividen en cifrado por bloques y cifrado de flujo. En el caso del cifrado por bloques, como su nombre lo indica, se procesa un tamaño de bloque determinado, dependiendo del algoritmo de cifrado. Caso contrario sucede con los algoritmos de cifrado de flujo, donde la información es tratada por bits o bytes y conforme van llegando, se les aplica la función de cifrado.

Características de cifrado por bloques:

- Alta difusión en el cifrado.
- Fácil detección de alguna posible modificación al criptograma. Dado que se trabaja por bloques, y de existir alguna modificación, ésta sería notable.
- Velocidad de cifrado baja. Depende totalmente del tamaño del bloque.
- Susceptible a errores. De existir algún error en el proceso, dicho error será propagado al bloque.

Características de cifrado de flujo

- Baja difusión en el cifrado.
- Difícil detección de alteraciones.
- Velocidad de cifrado alta.
- Resistente a errores y fácil recuperación a los mismos.

Conociendo las características de los cifradores por bloques y por flujo, se optó por analizar el desempeño de los cifradores por bloques contra los de flujo. Lo anterior debido a que ambos, pueden trabajar en ambientes limitados y en sistemas inalámbricos a pesar de lo mencionado en [103] que menciona que los cifradores de bloque son implementados difícilmente en sistemas con recursos limitados.

Se decidió hacer la elección de AES por ser el estándar de cifrado utilizado en diversidad de aplicaciones hoy en día; DES por ser el estándar anterior de cifrado;

finalmente se eligió a RC4 por formar parte de los protocolos más comunes dentro del ambiente inalámbrico. Los dos primeros algoritmos realizan cifrado de bloques, mientras que el último, cifrado de flujo.

A continuación se mencionarán aspectos de cada uno de estos cifradores, para que en conjunción con lo mencionado en el capítulo 2, se elija el algoritmo de cifrado que trabajará en unión con el algoritmo de compresión de datos.

4.3.1 AES

AES tiene un gran desempeño y un buen nivel de seguridad ofrecido por sus diferentes tamaños de llaves de cifrado: 128,196 y 256 bits. AES trabaja con un tamaño de bloque de 128 bits que permite la detección de errores más fácil que en DES.

En [101] se afirma que el costo de energía, en el proceso de cifrado, es de 1 mJ como máximo y que el consumo de energía en el descifrado es 30% más que el cifrado. La decodificación de datos presenta mayor consumo de energía y un costo más alto debido al proceso de inversión de las funciones [104].

AES puede trabajar con diferentes tamaños de llaves para incrementar el nivel de seguridad. No obstante, incrementando el número de bits para la clave, se incrementa tanto el consumo de energía como el consumo de recursos de procesamiento y almacenamiento de los datos a cifrar ya que, como lo muestra la tabla 4.8, utilizando una llave de 128 bits, una función propia del algoritmo es repetida 10 veces, 12 con una llave de 192 bits y 14 veces para 256 bits, lo que conlleva que, aumentando el nivel de seguridad, aumentan los recursos necesarios para poder llevarlo a cabo.

Tam Key Bits	Tam Bloque Bits	Rondas #
128	128	10
192	128	12
256	128	14

Tabla 4.8-Tamaño de llave, bloque y número de rondas en AES.

En la figura 4.2 se pueden observar los diferentes niveles de consumo de energía para los diferentes tamaños de llave que tiene AES, tanto en el proceso de cifrado como en el descifrado de los datos.

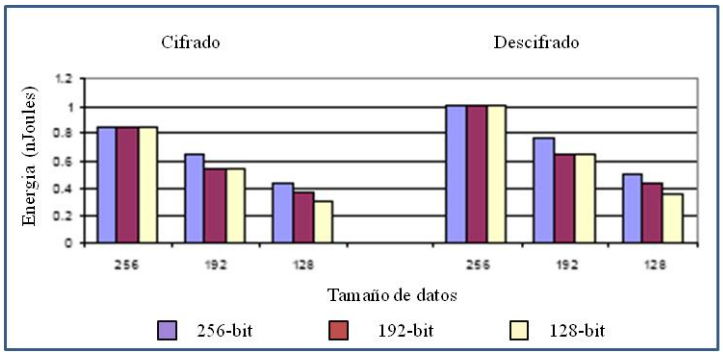

Figura 4.2-Consumo de energía en AES.

AES realiza cerca de 2.5 micro operaciones por ciclo [105] lo que significa que es un cifrado rápido tomando en cuenta el nivel de seguridad que brinda y de las operaciones necesarias para conseguirlo.

El proceso de descifrado, es similar al cifrado, con la diferencia de que las operaciones son realizadas en orden inverso y con unas modificaciones básicas sin alterar la complejidad del mismo, pero sí incrementando ligeramente el tiempo de procesamiento.

Otra de las características de este algoritmo, es que tiene alto paralelismo para ser implementado en hardware, lo cual, incrementaría el tiempo de procesamiento para

realizar el cifrado de los datos. En [106], una comparación entre diferentes implementaciones de AES, muestra diferentes velocidades de cifrado.

Realizando una comparación de AES contra otros algoritmos de cifrado, con respecto a la cantidad de energía consumida en el proceso, se tiene a la figura 4.3 que muestra a AES como el mejor entre DES y RC4.

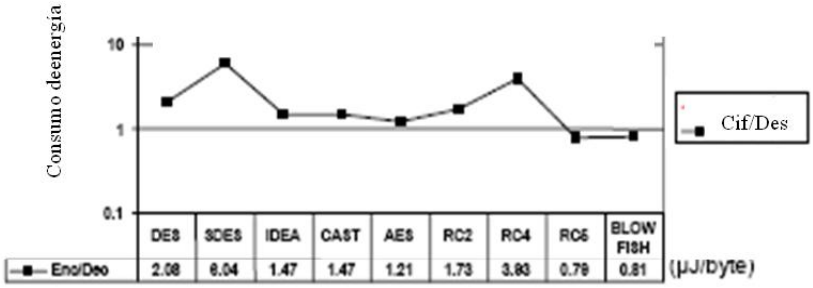

Figura 4.3-Consumo de energía en cifradores.

Finalmente, en la tabla 4.9 se muestran los resultados obtenidos a partir del cifrado de los datos. Los datos son tomados de Canterbury Corpus [107]. Es de notarse que la tabla 4.9 muestra resultados aplicando AES a los datos originales.

Archivos	Tamaño Bytes	Cifrado Seg	Descifrado Seg
alice29	152089	0.031	0.031
asyoulik	125179	0.031	0.031
cp	24603	0.000	0.016
fields.c	11150	0.016	0.016
grammar.lsp	3721	0.000	0.000
kennedy	1029744	0.281	0.266
icet10	426754	0.125	0.109
plrabn12	481861	0.125	0.125
ptt5	513216	0.141	0.141
sum	38240	0.016	0.016
xargs.1	4227	0.000	0.000
TOTAL	2810784	0.766	0.750

Tabla 4.9-Resultados de AES.

Aplicando AES, al conjunto de datos anterior, se obtiene un tiempo total de cifrado de 0.766 segundos, con un promedio de 0.070 segundos por cada archivo. Para el proceso de descifrado se tiene un tiempo promedio de 0.068 logrando un tiempo final de 0.750 segundos para todo el conjunto de datos, los cuales suman 2810784 bytes, logrando así, una velocidad de cifrado de 27.992 Mbps y de 28.592 Mbps en el descifrado.

4.3.2 DES

Algoritmo que ocupa un bloque de 64 bits con una llave de 56 bits ya que los 8 bits restantes, de un total de 64 bits, sirven para revisar la paridad.

El nivel de seguridad ofrecido por DES actualmente es bastante bajo ya que el tamaño de la llave es pequeño comparado con los cifradores modernos que al menos tienen 128 bits para la llave. Por tal motivo, la seguridad en DES tiene una complejidad de 2^n donde n son los 56 bits utilizados para la llave. Para un ataque de fuerza bruta, se tienen que realizar 2^{56} operaciones en el peor de los casos para romper con dicha seguridad.

Tanto el cifrado como el descifrado en este algoritmo, hacen uso de las mismas operaciones, pero, en orden inverso teniendo como resultado que las llaves utilizadas para cada ronda en el cifrado son $K_1, K_2, K_3, \ldots, K_{16}$, y para el descifrado son $K_{16}, K_{15}, K_{14}, \ldots, K_1$. Esta característica, permite que las ocho *S-boxes* utilizadas, ocupen un tamaño de 256 bytes.

Chandramouli *et al.* [23] realizan una comparación entre DES, IDEA y GHOST para determinar el consumo de energía de cada uno de ellos. Como resultado de lo anterior, DES requiere menos consumo de energía. En la figura 4.4 se muestra el comportamiento de DES en diferentes modos de operación.

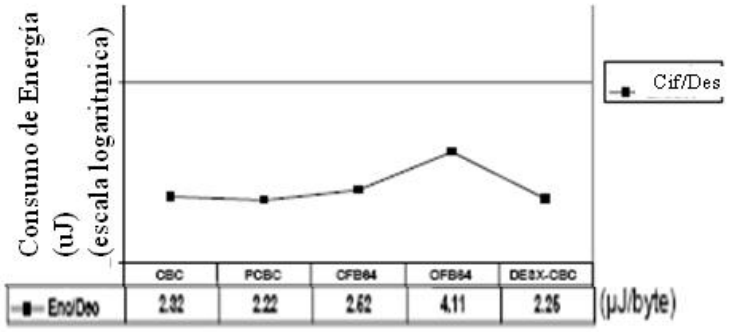

Figura 4.4-Consumo de energía en DES.

Dependiendo del modo de funcionamiento de DES, es el consumo de energía, siendo el modo OFB el que presenta mayor consumo. En la Figura 4.5 se aprecia el desempeño de DES comparado contra el algoritmos RC4 y es notable que DES es superado.

Por último, en la tabla 4.10 se muestran los resultados obtenidos aplicando DES. Se observa un tiempo superior a los obtenidos por AES. Es notable que en el proceso de cifrado y descifrado, los tiempos obtenidos son parecidos, ya que, como se mencionó anteriormente, realizan los mismos procedimientos, pero en orden inverso. Para el cifrado de los datos se obtiene un tiempo de 7.669 segundos, mientras que para el descifrado, 7.767segundos. Entonces, DES tiene una velocidad de cifrado de 2.796 Mbps y 2.760 Mbps para el descifrado.

Este algoritmo ofrece una velocidad de cifrado lenta tomando en cuenta que sólo maneja 56 bits para la llave, lo que lo convierte en un algoritmo poco factible e inseguro para esta investigación.

Archivos	Tamaño Bytes	Cifrado Seg	Descifrado Seg
alice29	152089	0.422	0.438
asyoulik	125179	0.344	0.359
cp	24603	0.078	0.094
fields.c	11150	0.031	0.031
grammar.lsp	3721	0.016	0.016
kennedy	1029744	2.813	2.828
icet10	426754	1.188	1.188
plrabn12	481861	1.200	1.220
ptt5	513216	1.438	1.453
sum	38240	0.125	0.125
xargs.1	4227	0.016	0.016
TOTAL	2810784	7.669	7.767

Tabla 4.10-Resultados de DES.

4.3.3 RC4

Cifrador de flujo que permite procesar información con gran velocidad. Este algoritmo puede ser ocupado con una llave de 40 a 128 bits, teniendo una complejidad de 2^n para encontrar la clave mediante un ataque de fuerza bruta.

En [23] se muestra que el nivel de consumo de energía para RC4 está muy por debajo de DES, es decir, RC4 es mucho más factible en dispositivos con energía de vida limitada. En la figura 4.5 se puede apreciar lo antes mencionado.

Otra de las ventajas de RC4, es que éste funciona de forma independiente al tamaño de la llave que se utilice, es decir, no importando si se usa un tamaño de 40 ó 128 bits, el desempeño de RC4 es el mismo. Esta propiedad no aplica para los algoritmos AES, DES y otros que no son mencionados en esta tesis.

RC4 realiza el cifrado de los datos de forma muy sencilla, con operaciones simples, y, una de sus grandes ventajas, es que tanto en el cifrado como en el descifrado, el algoritmo es el mismo, logrando así, un ahorro de espacio en las entidades a realizar

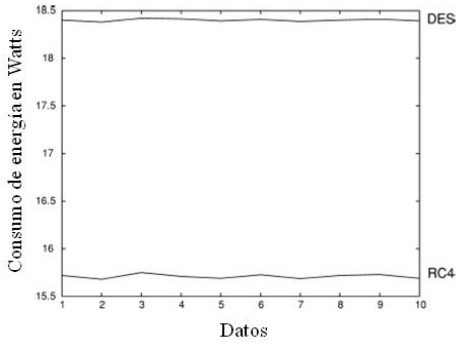

Figura 4.5-Comsumo de energía en RC4 y DES.

esta tarea. Mencionado esto, se tiene que RC4 puede ser implementado usando algunas líneas de código y requiere únicamente 256 bytes de RAM.

Robshaw [108] afirma que RC4 puede cifrar datos rápidamente en una implementación en software superando a varios en su categoría y, claro, a los cifradores de bloques. De igual forma, en [109, 110] se menciona que este algoritmo posee un fuerte diseño para una implementación en hardware para dispositivos móviles dejando de lado lo dicho en [111].

Procediendo a la aplicación de RC4, la tabla 4.11 muestra los resultados dados por el conjunto de datos procesados por este algoritmo.

Archivos	Tamaño Bytes	Cifrado Seg	Descifrado Seg
alice29	152089	0.016	0.016
asyoulik	125179	0.016	0.016
cp	24603	0.016	0.016
fields.c	11150	0.016	0.016
grammar.lsp	3721	0.016	0.016
kennedy	1029744	0.063	0.063
icet10	426754	0.016	0.016
plrabn12	481861	0.031	0.031
ptt5	513216	0.031	0.031
sum	38240	0.016	0.016
xargs.1	4227	0.000	0.000
TOTAL	2810784	0.234	0.234

Tabla 4.11-Resultados de RC4.

Para RC4, el tiempo de cifrado y el de descifrado es el mismo, siendo éste de 0.234 segundos en total para los datos probados con un promedio entre cada uno de ellos de 0.021 para ambos casos. Se cuenta con una velocidad de cifrado y descifrado de 91.64 Mbps

En conclusión, a partir de la tabla 4.11 se afirma que RC4 es, en definitiva, el algoritmo de cifrado más rápido en comparación con AES y DES.

4.4 Selección de algoritmos de compresión y cifrado

El objetivo es brindar un esquema de compresión y cifrado de datos, entonces, se debe buscar la combinación de algoritmos que trabajen de forma óptima cumpliendo las necesidades de las aplicaciones determinadas.

Inicialmente se realizó la prueba de compresión de los algoritmos implementados: Huffman Estático y Dinámico, LZW, LZSS, RLE y Cod. Aritmético Estático y Dinámico con Canterbury Corpus [107].

De los compresores analizados, se tiene que el mejor desempeño con respeto a la razón de compresión es el algoritmo LZSS con 0.367, seguido muy de cerca por LZW con 0.404. Ahora bien, según el tiempo de compresión, el más rápido es RLE con 0.079 segundos, posteriormente LZW con 0.440 y Huffman Estático con 0.615 segundos. Por otro lado, en la descompresión, el más rápido es LZSS con 0.001 segundos, posteriormente RLE con un tiempo de 0.017 segundos, seguido por LZW con 0.220 segundos. Dichos resultados se pueden apreciar en la tabla 4.12.

Dada la tabla 4.12 y a que RLE presenta una razón de compresión bastante pobre, el algoritmo con mejor desempeño, estableciendo un compromiso entre razón de compresión, tiempo de compresión y tiempo de descompresión, es LZW. Como es notable, a éste último se le acerca LZSS en la compresión, pero con un tiempo de compresión más elevado y el algoritmo Huffman Estático con valores más altos.

Algoritmo	RC	Comp Seg	Descomp Seg
LZSS	0.367	3.573	0.001
LZW	0.404	0.440	0.220
RLE	0.849	0.079	0.017
COD. ARIT. EST.	0.444	3.031	3.753
COD. ARIT. DIN.	0.455	4.722	4.722
HUFF. EST.	0.468	0.615	0.437
HUFF. DIN.	0.460	1.452	1.452

Tabla 4.12-Resultados de algoritmos de compresión.

Por tal motivo, se procedió a probar estos tres algoritmos con datos tipo ECG tomados de *ANSI/AAMI EC13 Test Corpus* [112]. En la tabla 4.13 se muestran los resultados obtenidos mediante la aplicación de LZSS, LZW y Huffman Estático respectivamente.

Archivos	Original Bytes	Compresión Bytes	Comp Seg	Descomp Seg	RC
aami3a.dat	86162	27920	0.055	0.000	0.324
aami3b.dat	86284	27795	0.055	0.000	0.322
aami3c.dat	86288	30908	0.055	0.000	0.358
aami3d.dat	69022	23654	0.055	0.000	0.343
aami4a.dat	130422	37414	0.055	0.000	0.287
aami4a_d.dat	130428	37366	0.055	0.000	0.286
aami4a_h.dat	130414	37442	0.055	0.000	0.287
aami4b.dat	130476	43883	0.055	0.000	0.336
aami4b_d.dat	130478	43876	0.055	0.000	0.336
aami4b_h.dat	130474	43922	0.055	0.000	0.337
TOTAL	1110448	354180	0.549	0.001	0.319

a)LZSS.

Archivos	Original Bytes	Compresión Bytes	Comp Seg	Descomp Seg	RC
aami3a.dat	86162	24538	0.055	0.010	0.285
aami3b.dat	86284	24351	0.055	0.010	0.282
aami3c.dat	86288	27805	0.055	0.010	0.322
aami3d.dat	69022	21834	0.055	0.010	0.316
aami4a.dat	130422	34066	0.055	0.010	0.261
aami4a_d.dat	130428	32682	0.055	0.010	0.251
aami4a_h.dat	130414	34623	0.055	0.010	0.265
aami4b.dat	130476	41442	0.055	0.010	0.318
aami4b_d.dat	130478	41716	0.055	0.010	0.320
aami4b_h.dat	130474	42654	0.055	0.010	0.327
TOTAL	1110448	325711	0.549	0.100	0.293

b)LZW.

Archivos	Original Bytes	Compresión Bytes	Comp Seg	Descomp Seg	RC
aami3a.dat	86162	41059	0.016	0.016	0.477
aami3b.dat	86284	40098	0.016	0.016	0.465
aami3c.dat	86288	45474	0.016	0.016	0.527
aami3d.dat	69022	36361	0.016	0.016	0.527
aami4a.dat	130422	68375	0.031	0.016	0.524
aami4a_d.dat	130428	67789	0.031	0.016	0.520
aami4a_h.dat	130414	68697	0.031	0.016	0.527
aami4b.dat	130476	72218	0.031	0.016	0.553
aami4b_d.dat	130478	71802	0.031	0.016	0.550
aami4b_h.dat	130474	72756	0.031	0.016	0.558
TOTAL	1110448	584629	0.250	0.156	0.526

c)Huffman Estático.

Tabla 4.13-Resultados de LZSS, LZW y Huffman Estático.

Probando con los datos de ECG, y limitando los compresores a los tres antes mencionados, se tiene que el mejor desempeño, según la compresión obtenida, lo ofrece el algoritmo LZW con una razón de compresión de 0.293 y con tiempos de 0.549 y 0.100 segundos para compresión y descompresión respectivamente. Posteriormente, LZSS ofrece 0.319 como razón de compresión y 0.549 segundos en la compresión y 0.001 en la descompresión. No obstante, superando a ambos, en tiempo de compresión de 0.250 segundos, pero con razón de compresión de 0.526 y tiempo de descompresión de 0.156 segundos se encuentra el Huffman Estático.

Los algoritmos LZSS y LZW son los que tienen un mejor compromiso entre razón de compresión y tiempo de procesamiento. Por tal motivo, se realiza una prueba de estos dos compresores utilizando archivos ECG de mayor tamaño. En la tabla 4.13d se observa que el LZSS sigue teniendo una buena razón de compresión, sin embargo, el tiempo de procesamiento, tanto para la compresión como para la descompresión ha aumentado de forma considerable.

Archivos	Original Bytes	Compresión Bytes	Comp Seg	Descomp Seg	RC
aami3a3.dat	3198759	1021679	1.978	0.055	0.319
aami3b3.dat	3211301	1018368	2.088	0.055	0.317
aami3c3.dat	3228765	1129642	1.813	0.055	0.350
aami3d3.dat	3211553	1071616	1.813	0.110	0.334
aami4a3.dat	3782238	446518	0.824	0.110	0.118
TOTAL	16632616	4687823	8.516	0.385	0.282

Tabla 4.13d. Resultados de LZSS.

Por el contrario, en la tabla 4.13e se muestran los resultados obtenidos aplicando el algoritmo LZW. Se puede apreciar que los tiempos de compresión y descompresión para este algoritmo son bajos incluso para archivos de mayor tamaño. La razón de compresión sigue siendo alta.

En la tabla 4.13f se hace una comparación con respecto a la complejidad computaciones y espacial tanto de LZSS como de LZW. Se aprecia que el algoritmo LZW requiere menos capacidad computacionl.

Archivos	Original Bytes	Compresión Bytes	Comp Seg	Descomp Seg	RC
aami3a.dat	3198759	859317	0.330	0.165	0.269
aami3b.dat	3211301	850441	0.275	0.165	0.265
aami3c.dat	3228765	975492	0.330	0.165	0.302
aami3d.dat	3211553	940611	0.275	0.220	0.293
aami4a.dat	3782238	5836	0.165	0.110	0.002
TOTAL	16632616	3631697	1.374	0.824	0.218

Tabla 4.13e. Resultados de LZW.

Algoritmo	Complejidad	Memoria
LZSS	O (n^2)	$9n_w+2^n$
LZW	O (n)	2^n

Tabla 4.13f. Complejidad computacional y espacial de LZSS y LZW.

Entonces, dado los resultados obtenidos, lo mostrado en la tabla 4.13f y las características antes mencionadas, se hace elección del algoritmo de compresión LZW por ser el que se adapta mejor al esquema ya que presenta una buena razón de

compresión en un tiempo aceptable y un mejor tiempo de descompresión. LZW tiene gran flexibilidad para el manejo de diferentes tipos de datos. Además, LZW puede funcionar incluso en ambientes con recursos limitados.

Archivos	Original Bytes	Cifra Seg	Descifra Seg
aami3a.dat	86162	0.015	0.015
aami3b.dat	86284	0.015	0.015
aami3c.dat	86288	0.015	0.015
aami3d.dat	69022	0.010	0.010
aami4a.dat	130422	0.031	0.031
aami4a_d.dat	130428	0.031	0.031
aami4a_h.dat	130414	0.031	0.031
aami4b.dat	130476	0.031	0.031
aami4b_d.dat	130478	0.031	0.031
aami4b_h.dat	130474	0.031	0.031
TOTAL	1110448.000	0.243	0.243

a)RC4.

Archivos	Original Bytes	Cifra Seg	Descifra Seg
aami3a.dat	86162	0.016	0.016
aami3b.dat	86284	0.016	0.016
aami3c.dat	86288	0.016	0.016
aami3d.dat	69022	0.031	0.031
aami4a.dat	130422	0.031	0.031
aami4a_d.dat	130428	0.031	0.016
aami4a_h.dat	130414	0.031	0.031
aami4b.dat	130476	0.031	0.031
aami4b_d.dat	130478	0.031	0.031
aami4b_h.dat	130474	0.031	0.031
TOTAL	1110448.000	0.266	0.250

b)AES.

Archivos	Original Bytes	Cifra Seg	Descifra Seg
aami3a.dat	86162	0.234	0.234
aami3b.dat	86284	0.250	0.250
aami3c.dat	86288	0.250	0.250
aami3d.dat	69022	0.203	0.203
aami4a.dat	130422	0.375	0.375
aami4a_d.dat	130428	0.375	0.375
aami4a_h.dat	130414	0.375	0.375
aami4b.dat	130476	0.358	0.358
aami4b_d.dat	130478	0.375	0.375
aami4b_h.dat	130474	0.375	0.375
TOTAL	1110448.000	3.171	3.171

c)DES.

Tabla 4.14-Resultados de RC4, AES y DES.

Posteriormente, una vez seleccionado el algoritmo de compresión, es necesario determinar el algoritmo de cifrado. Entonces, dada la tabla 4.14 se procede a analizar el desempeño de AES, DES, RC4. Es notable que el procedimiento con mayor velocidad de procesamiento es RC4 teniendo un tiempo final de 0.243 segundos en el cifrado y descifrado. Siguiendo de cerca, se encuentra el algoritmo AES con tiempos de 0.266 y 0.250 para el cifrado y descifrado respectivamente. Por último, con un pobre desempeño, está DES.

La disputa se encuentra entre AES y RC4, donde anteriormente ya se describieron sus características, tanto en este capítulo como en el capítulo 2. Entonces, continuando el análisis, AES es el cifrador más seguro hasta ahora, superando en ese aspecto a RC4. Pero, este último, tiene la peculiaridad de subir el nivel de seguridad aumentando el tamaño de la llave utilizada y no afectar el desempeño del mismo. Por el contrario, en AES, sí se afecta el desempeño incrementando el tamaño de la llave, lo cual se refleja directamente en el consumo de energía ya que, en un principio, AES tiende a

consumir menos, pero ésta aumenta y supera a RC4 si el tamaño de la llave es elevado. Dicho fenómeno está representado en la figura 4.6.

Otro punto a favor de RC4, es la gran capacidad por producir información cifrada, es decir, tiene gran velocidad de procesamiento. Y, con respecto a AES, RC4 lo supera fácilmente y tiende a notarse más conforme aumenta la cantidad de información a cifrar. La figura 4.7 muestra la velocidad de cifrado de AES y RC4. Al principio, con poca información, AES tiende a ser mejor. Posteriormente, con el incremento de los datos, RC4 presenta mejor velocidad de cifrado.

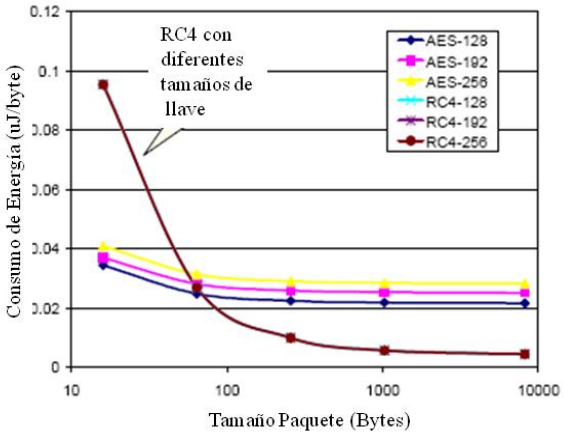

Figura 4.6-Consumo de energía en AES y RC4 con diferentes tamaños de llave.

En [110] se hace mención de aspectos interesantes acerca de los cifradores de bloque y de flujo. Destacan principalmente que los cifradores de flujo son usados comúnmente en aplicaciones de transferencia de información para conexiones seguras, mientras que los cifradores de bloque, son usados típicamente para almacenar información. Se menciona que los cifradores de bloque son usados generalmente para la transferencia de datos en ambientes alámbricos.

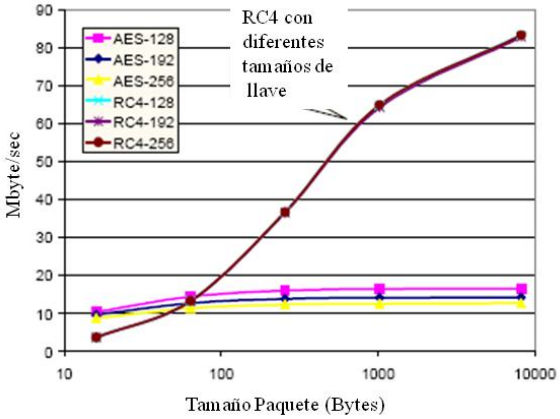

Figura 4.7-Velocidad de cifrado en AES y RC4.

Otro factor importante a considerar, es que RC4 y AES son cifradores de flujo y bloques, respectivamente, por lo cual, no hay que olvidar las características propias de los mismos. El cifrado de bloques tiene la desventaja de que en caso de existir un error, éste tiende a propagarse, característica que en el cifrado de flujo sólo se ve afectado un elemento y no todo el conjunto. En el caso de AES se trabaja con bloques de 128 bits lo cual representa16 bytes, y, la alteración de uno de estos bloques pudiera hacer de la información una representación totalmente diferente dependiendo de la información que se esté manejando. Entonces, tomando en cuenta que se hará manejo de información, la cual no es aceptable su modificación ya que pudieran surgir errores de interpretación por parte el usuario final al momento de analizarla, se procede a la elección del algoritmo RC4 para el trabajo en conjunto con el algoritmo de compresión LZW.

4.5 Compresión-cifrado de datos

Ya que se cuenta con el algoritmo de compresión y cifrado, se hacen pruebas aplicando el algoritmo de compresión a los datos ECG y posteriormente, el algoritmo de cifrado para determinar el desempeño que tienen éstos. En este punto, el cifrado será aplicado a los archivos comprimidos por LZW. La tabla 4.15 muestra los resultados obtenidos.

Analizando la tabla 4.15, se confirma la velocidad de cifrado de RC4 logrando el cifrado y descifrado de los datos comprimidos en un tiempo de 0.095 segundos.

En la tabla 4.16 se muestra el tiempo trascurrido desde que se aplica la compresión de los datos con LZW, hasta el cifrado de los mismos con RC4. Se obtiene un tiempo de 0.535 segundos en el proceso de compresión y cifrado para el conjunto de datos.

			RC4	
Archivo	**Original Bytes**	**LZW Bytes**	**Cifrado Seg**	**Descifrado Seg**
alice29	152089	72322	0.007	0.007
asyoulik	125179	62976	0.008	0.008
cp	24603	12228	0.008	0.008
fields.c	11150	5316	0.007	0.007
grammar.lsp	3721	2116	0.009	0.009
kennedy	1029744	419220	0.025	0.025
icet10	426754	222229	0.008	0.008
plrabn12	481861	234978	0.015	0.015
ptt5	513216	70228	0.004	0.004
sum	38240	30940	0.013	0.013
xargs.1	4227	2691	0.000	0.000
TOTAL	2810784	1135244	0.095	0.095

Tabla 4.15-Resultados de LZW-RC4.

Archivos	**LZW Bytes**	**Tiempo Comp+Cifrado**	**Tiempo Cifrado**	**Tiempo Compresión**
alice29	72322	0.062	0.007	0.055
asyoulik	62976	0.063	0.008	0.055
cp	12228	0.008	0.008	0.000
fields.c	5316	0.008	0.007	0.000
grammar.lsp	2116	0.009	0.009	0.000
kennedy	419220	0.190	0.025	0.165
icet10	222229	0.063	0.008	0.055
plrabn12	234978	0.070	0.015	0.055
ptt5	70228	0.059	0.004	0.055
sum	30940	0.013	0.013	0.000
xargs.1	2691	0.000	0.000	0.000
TOTAL	1135244	0.535	0.095	0.44

Tabla 4.16-Resultados de tiempo en LZW-RC4.

En las tablas 4.17 y 4.18 se muestran los resultados obtenidos aplicando LZW y RC4 a los datos ECG. Para estos tipos de archivo, se puede notar en la tabla 4.17 que se obtiene una velocidad de cifrado y descifrado aproximadamente de 35 Mbps aplicando RC4 a los datos comprimidos con LZW.

Finalmente, en la tabla 4.18 se puede apreciar el tiempo consumido desde el proceso de compresión hasta la culminación del cifrado de los datos. Para los datos probados y con la utilización de LZW y RC4, se obtuvo un tiempo total de 0.620 segundos. Tal resultado implica que se está procesando información con una velocidad de 13.664 Mbps.

Archivos	Original Bytes	LZW Bytes	Cifra Seg	Descifra Seg
aami3a.dat	86162	24538	0.004	0.004
aami3b.dat	86284	24351	0.004	0.004
aami3c.dat	86288	27805	0.005	0.005
aami3d.dat	69022	21834	0.003	0.003
aami4a.dat	130422	34066	0.008	0.008
aami4a_d.dat	130428	32682	0.008	0.008
aami4a_h.dat	130414	34623	0.008	0.008
aami4b.dat	130476	41442	0.010	0.010
aami4b_d.dat	130478	41716	0.010	0.010
aami4b_h.dat	130474	42654	0.010	0.010
TOTAL	1110448	325711	0.071	0.071

Tabla 4.17-Resultados de LZW-RC4.

Archivos	LZW Bytes	Tiempo Comp+Cifrado	Tiempo Cifrado	Tiempo Compresión
aami3a.dat	24538	0.059	0.004	0.055
aami3b.dat	24351	0.059	0.004	0.055
aami3c.dat	27805	0.060	0.005	0.055
aami3d.dat	21834	0.058	0.003	0.055
aami4a.dat	34066	0.063	0.008	0.055
aami4a_d.dat	32682	0.063	0.008	0.055
aami4a_h.dat	34623	0.063	0.008	0.055
aami4b.dat	41442	0.065	0.010	0.055
aami4b_d.dat	41716	0.065	0.010	0.055
aami4b_h.dat	42654	0.065	0.010	0.055
TOTAL	325711	0.620	0.071	0.549

Tabla 4.18-Resultados de tiempo en LZW-RC4.

4.6 Resumen de Capítulo

En este capítulo se menciona una serie de aspectos a tomar en cuenta para la elección de los diferentes algoritmos de compresión y cifrado de datos tratados en esta tesis.

También, se analizan los resultados obtenidos por cada uno de los algoritmos mencionados. Consecuentemente, es en este capítulo donde se determina cuáles serán los algoritmos, de compresión y cifrado, que trabajarán en conjunto y formarán parte del esquema compresión-cifrado para este caso.

La selección de los algoritmos, de compresión y cifrado realizado en este capítulo, se basó en aquellos que mejor se adaptaran al esquema. Para el algoritmo de compresión se buscó el compromiso entre razón de compresión y tiempo de compresión y descompresión. Para el algoritmo de cifrado, se consideró la seguridad que éstos brindaban y la velocidad de cifrado y descifrado. Igualmente, en ambos tipos de algoritmos se abarcaron detalles de complejidad computacional y recursos necesarios para llevar acabo dicho proceso.

Hasta este momento se aporta un algoritmo de compresión y un algoritmo de cifrado de datos, que con base en su análisis y los resultados obtenidos, son factibles de utilizar con datos ECG para su transmisión y/o almacenamiento.

En el próximo capítulo se busca realizar un pre procesamiento a los datos de tal manera que, se pueda incrementar la razón de compresión obtenida por el algoritmo LZW e incluso poder compararla con algoritmos de compresión con pérdida. Lo anterior con la finalidad de tener un mejor aprovechamiento de los medios de transmisión y almacenamiento de información ya que la cantidad de datos ECG generada es muy demandante.

Con el pre procesamiento de los datos se busca manejar la información de tal manera que pueda ser mejor aprovechada por el algoritmo de compresión y así, obtener una alta y mejor razón de compresión.

Capítulo 5

Introducción

En este capítulo se trabajará con los algoritmos de compresión y cifrado seleccionados dados por las características y resultados mostrados en el capítulo 4. Se describen diferentes módulos creados que realizan un pre procesamiento de los datos con la finalidad de mejorar la razón de compresión obtenida anteriormente. Es importante mencionar que el *pre procesamiento* es una serie de pasos aplicados a la información original antes de que ésta sea enviada al algoritmo de compresión. Entonces, a partir de la figura 1.2, se tiene que ahora el bloque *Preprocesamiento* estará formado por el módulo creado y elegido en este capítulo.

En el capítulo 4 se eligió a un compresor para trabajar con datos ECG, no obstante, se quiere mejorar la razón de compresión ofrecida por dicho algoritmo. Para lograr esto, se realiza una serie de modificaciones en el manejo de los datos por parte del compresor, lo que afecta principalmente al tiempo de ejecución y razón de compresión.

Se busca aumentar la razón de compresión para aprovechar más el medio de transmisión de datos a utilizar y los medios de almacenamiento. Obteniendo una compresión más alta, se consigue un mayor número de datos ECG que pueden ser monitoreados de forma paralela. En consecuencia, pudiera aumentar la cantidad de pacientes que transmitan este tipo de información. Igualmente, se lograría un ahorro de espacio en los dispositivos de almacenamiento ya que la cantidad de información tipo ECG tiende a aumentar rápidamente. Por ejemplo, si un paciente genera sólo un canal de señal electrocardiográfica de 12 bits con resolución de 750 Hz., estará produciendo 2.8 gigabytes de información mensualmente. Comúnmente se maneja de 3 a 12 canales por paciente, es decir, hasta 34 gb de datos ECG por un paciente en un mes. Multipliquemos esa cantidad por el número de pacientes que se tienen que

monitorear. Como se puede notar, resulta necesario aumentar la razón de compresión de los datos.

Entre algunas consideraciones tomadas en cuenta, está que, para datos ECG existe una gran repetición de elementos, la cual es aprovecha por LZW. Es necesario mencionar que entre un dato y otro, existe, en muchas ocasiones, poca diferencia con respecto al valor numérico. Entonces, se puede aprovechar dicha característica para mejorar la razón de compresión.

Los datos a probar son tipo ECG digitalizados con una resolución de 12 bits y tomados de [112].

Finalmente, en este capítulo se presentan resultados obtenidos aplicando RC4 a los datos ECG ya comprimidos con el módulo creado.

En las siguientes secciones se describen cada uno de los módulos creados.

5.1 Byte-Byte

En este caso, los valores se leen consecutivamente, de acuerdo al orden de llegada al compresor. No obstante, los datos ECG no son procesados en su totalidad, es decir, no se toman los 12 bits correspondientes, sino que, se toman 8 bits [2] para determinar el primer valor del compresor. Posteriormente, se toman otros 8 bits para que se concatene con el byte anterior y se verifique la existencia de esta cadena en el diccionario. En caso de existir la cadena en el diccionario, se procede a sustituirla por el código correspondiente. En caso contrario, la cadena se agrega al diccionario en una posición disponible determinada por una función *hash*.

Con este procedimiento se reduce la cantidad de información en el diccionario, ya que manejando los 12 bits originales de los datos ECG, el diccionario, en primera instancia debiera tener un tamaño de 2^n, donde n es el número de bits. Entonces, trabajando con 12 bits se tienen 2^{12}=4096 posiciones, más aparte las posiciones que

deberá tener disponibles para lograr una buena compresión. Con el procedimiento *byte-byte* se tiene un diccionario de tamaño igual 2^{12}pero con 256 posiciones ocupadas en el inicio y el resto de las posiciones se encuentran libres.

Como se puede apreciar en la tabla 5.1 la razón de compresión obtenida, para los datos probados, es alta ya que se logra una reducción de más del 70% con respecto a la información original. El tiempo total invertido para lograr la compresión en este caso fue de 0.549 segundos, con un promedio de 0.055 segundos por cada archivo utilizado, lo que significa que cada dato será comprimido en el orden de los microsegundos.

Archivos	Original bytes	Comp bytes	Comp Seg	RC
aami3a.dat	86162	24538	0.055	0.285
aami3b.dat	86284	24351	0.055	0.282
aami3c.dat	86288	27805	0.055	0.322
aami3d.dat	69022	21834	0.055	0.316
aami4a.dat	130422	34066	0.055	0.261
aami4a_d.dat	130428	32682	0.055	0.251
aami4a_h.dat	130414	34623	0.055	0.265
aami4b.dat	130476	41442	0.055	0.318
aami4b_d.dat	130478	41716	0.055	0.320
aami4b_h.dat	130474	42654	0.055	0.327
TOTAL	1110448	325711	0.549	0.293

Tabla 5.1-Resultados de *byte -byte.*

5.2 12-Bits

Se realiza un manejo de los datos con una longitud de 12 bits [48]. Se leen dos bytes para formar una palabra. Dado que se trabaja con palabras de 12 bits, la comparación si existe o no en el diccionario es realizada con el ancho de dicha palabra. Resulta conveniente decir que el primer byte leído, para formar los 12 bits, es el elemento

menos significativo. Entonces, el segundo byte leído es el más significativo de la expresión.

La principal desventaja de utilizar los 12 bits totales de la señal, es la gran cantidad de recursos con la que se debe contar para poder realizar el proceso de compresión eficientemente. El espacio requerido para este ancho de palabra es de 2^n posiciones sin contemplar las posiciones que deben estar libres.

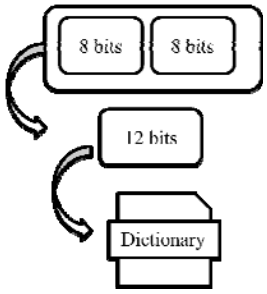

Figura 5.1-Procesamiento de 12 bits.

A continuación se muestran resultados acerca de la razón de compresión y de la velocidad de procesamiento utilizando los 12 bits de la cadena. Es importante mencionar que trabajando con los 12 bits directamente, se pudiera tener una buena razón de compresión siempre y cuando, el tamaño del diccionario fuera lo suficientemente grande, más grande que 2^{12}.

Como se puede apreciar, la razón de compresión, comparada con el proceso de *byte-byte* es menor, entonces, utilizando *12-bits*, se tiene un peor desempeño. Lo anterior se debe principalmente a que en este caso se utiliza un diccionario con 8192, es decir, 13 bits.

Archivos	Original bytes	Comp bytes	Comp Seg	RC
aami3a.dat	86162	46673	0.055	0.542
aami3b.dat	86284	46739	0.110	0.542
aami3c.dat	86288	46741	0.055	0.542
aami3d.dat	69022	37389	0.055	0.542
aami4a.dat	130422	70647	0.989	0.542
aami4a_d.dat	130428	70650	0.989	0.542
aami4a_h.dat	130414	70643	0.934	0.542
aami4b.dat	130476	70676	0.989	0.542
aami4b_d.dat	130478	70677	0.989	0.542
aami4b_h.dat	130474	70675	0.934	0.542
TOTAL	1110448	601510	6.099	0.542

Tabla 5.2-Resultados de *12-bits.*

El tiempo de procesamiento de los datos es mucho mayor, tratándose de 6.099 segundos en la compresión contra los 0.549 utilizados por *byte-byte.* Con lo anterior se afirma que entre más grande sea el diccionario, más es el tiempo de búsqueda y por ende, mayor el tiempo de compresión. Además, el incremento del tiempo radica principalmente en que es necesario leer dos bytes, concatenarlos y posteriormente procesarlos, a diferencia de *byte-byte* donde los elementos son procesados directamente como palabras con longitud de un byte.

Entonces, utilizando los 12 bits originales de la señal ECG, se obtiene una razón de compresión de 0.542 con un tiempo de procesamiento para el banco de pruebas de 6.099 segundos, con lo que este proceso resulta poco factible para su aplicabilidad ya que el tiempo de procesamiento es alto y la razón de compresión es baja si se compara contra *byte-byte.*

Los módulos siguientes son creados con la finalidad de aumentar la razón de compresión obtenida hasta ahora. Se enfoca en diferentes formas de manejar la información, para así obtener los mejores resultados posibles.

5.3 12-Bits-Swap

Al igual que en el procedimiento anterior, en este proceso es necesario leer dos bytes para formar los 12 bits correspondientes, sólo que ahora se invierten los elementos con el fin de verificar si es posible una mejora en la razón de compresión. Entonces, el primer byte leído ahora será el elemento más significativo y el segundo byte será el menos significativo.

En este procedimiento es necesario aplicar una función de corrimiento a nivel lógico de cuatro posiciones en la parte más significativa para evitar un traslape de valores.

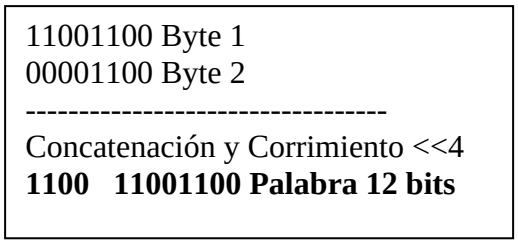

Figura 5.2-Procesamiento 12 bits con *swap*.

Dada la figura 5.2, los ocho bits del primer byte son ocupados en su totalidad, y del segundo, sólo se ocupan los primeros cuatro bits para formar la palabra original de 12 bits. Se concatena el primer byte con el segundo, pero aplicando un corrimiento de cuatro posiciones al primer byte leído, ocupando este último la posición más significativa de la palabra.

A diferencia del procedimiento *12-bits*, en este procedimiento se invierten los elementos pretendiendo obtener un mejor resultado en la razón de compresión y dada la tabla 5.3 se puede notar sólo una pequeña mejora con respecto al proceso *12-bits* y con un ligero aumento en el tiempo de procesamiento.

En este procedimiento se obtiene una razón de compresión de 0.526 contra 0.542 del proceso *12-bits*. El tiempo de operación tiene un incremento de 0.109 segundos con

respecto a *12-bits*, teniendo como resultado un tiempo final de 6.208 segundos para la compresión de los datos.

En la aplicación de este método no se tiene una mejora significativa en los parámetros de razón de compresión y de velocidad de procesamiento, los cuales son los dos más importantes a mejorar en esta investigación.

Archivos	Original bytes	Comp bytes	Comp Seg	RC
aami3a.dat	86162	45930	0.055	0.533
aami3b.dat	86284	44887	0.164	0.520
aami3c.dat	86288	42663	0.110	0.494
aami3d.dat	69022	34353	0.110	0.498
aami4a.dat	130422	69541	0.934	0.533
aami4a_d.dat	130428	66316	0.989	0.508
aami4a_h.dat	130414	70374	0.934	0.540
aami4b.dat	130476	69422	0.934	0.532
aami4b_d.dat	130478	70677	0.989	0.542
aami4b_h.dat	130474	70311	0.989	0.539
TOTAL	1110448	584474	6.208	0.526

Tabla 5.3-Resultados de *12-Bits-Swap*.

Para mayor detalle de los resultados obtenidos mediante *12-Bits-Swap* ver la tabla 5.3. En este caso, no hay una compresión tan efectiva debido a la relación existente entre el tamaño de palabra y el tamaño de diccionario utilizado.

5.4 Byte-Byte-Swap

Dado el procedimiento *12-Bits-Swap* y el comportamiento que éste presenta, se realiza la inversión de elementos, pero sin el corrimiento de los mismos ya que se analizará con el manejo de byte a byte.

Primeramente se lee la información, dicha tarea se realiza elemento a elemento, donde un elemento se compone de ocho bits. Posteriormente y a consecuencia de que se requiere la inversión de elementos, es necesario leer dos datos consecutivos.

Leídos los dos elementos, se realiza la inversión de posiciones con la característica de que esta vez, no será necesario ejecutar un corrimiento de posiciones ya que los datos son procesados directamente con bytes. Aplicando lo anterior, el tiempo de ejecución es mejor con respecto a *12-Bits-Swap*.

Una vez realizada la inversión de elementos: el primer byte leído fungiendo como la parte más significativa de la palabra, y el segundo byte como el menos significativo; éstos son enviados al compresor LZW.

En las pruebas realizadas con *Byte-Byte-Swap* no hay cambios con respecto al tiempo de procesamiento en *Byte-Byte*. No obstante, el cambio se obtuvo en el desempeño de la compresión. Para este caso, se obtuvo un valor de 0.296 en la razón de compresión con un tiempo de procesamiento de 0.549 segundos, mientras que en *Byte-Byte* se obtiene una compresión de 0.293. En este caso, la inversión de elementos no repercute directamente en el tiempo. Es importante decir que el tiempo de ejecución de este proceso puede variar si se utiliza una entidad de prueba más limitada o con mejores recursos de procesamiento. Los resultados pudieran diferir de forma mínima.

La tabla 5.4 aporta los resultados obtenidos aplicando el proceso *Byte-Byte-Swap*. Esto con la finalidad de analizar el comportamiento durante el proceso, principalmente los aspectos de compresión y tiempo de ejecución para poder determinar si presenta mejores resultados a los presentados por *Byte-Byte* y *12-Bits-Swap*.

La complejidad de este procedimiento, al igual que la presentada por *Byte-Byte,* está dada por la complejidad del algoritmo LZW, ya que sólo se realiza la inversión de bytes. En consecuencia, sólo cambia la asignación de valores antes de que los datos sean tomados por el compresor. *Byte-Byte* no tiene operaciones extras, ya que no se realiza ninguna operación antes de que la información sea comprimida, a diferencia de *Byte-Byte-Swap* que sí las tiene.

Se puede apreciar en la tabla 5.4 que este procedimiento no es mejor a *Byte-Byte* en lo que concierne a la razón de compresión. Pero, sí presenta mejores resultados, tanto en la compresión como en el tiempo de procesamiento, con respecto a *12-Bits* y *12-Bits-Swap*.

Archivos	Original bytes	Comp bytes	Comp Seg	RC
aami3a.dat	86162	24781	0.055	0.288
aami3b.dat	86284	24444	0.055	0.283
aami3c.dat	86288	28054	0.055	0.325
aami3d.dat	69022	22080	0.055	0.320
aami4a.dat	130422	34383	0.055	0.264
aami4a_d.dat	130428	33064	0.055	0.254
aami4a_h.dat	130414	34959	0.055	0.268
aami4b.dat	130476	41919	0.055	0.321
aami4b_d.dat	130478	42166	0.055	0.323
aami4b_h.dat	130474	43315	0.055	0.332
TOTAL	1110448.000	329165.000	0.549	0.296

Tabla 5.4-Resultados de *Byte-Byte-Swap*.

5.5 División

Debido al manejo de valores de 12 bits y la necesidad de leer 2 bytes, en este proceso se trata por separado a cada uno de estos bytes.

Dado que ahora se tratará independientemente cada byte, es necesario especificar una forma de identificarlos. Entonces, se hará referencia con el nombre de *Bytes-Pares* a los bytes 2,4,6,…, *n* y con *Bytes-Nones* a la secuencia 1,3,5,…,*n*-1, donde *n* es un número par.

Dicho lo anterior, el tratamiento de la información se realiza de tal manera que los *Bytes-Nones* y los *Bytes-Pares* son procesados de forma independiente uno de otro, es decir, se realiza una distinción clara entre estos dos elementos para que posteriormente puedan ser procesados por el compresor.

El compresor, recibirá los *Bytes-Nones* y los *Bytes-Pares*, primero uno y posteriormente el otro, no importando el orden. En este caso, lo que realmente importa es que ambos elementos sean separados unos de otros.

Es importante mencionar, que haciendo la división de información en *Bytes-Nones*, *Bytes-Pares*, y comprimirlos, se obtiene que en los *Bytes-Nones*, dado que se ocupan los 8 bits, hay menor probabilidad de que existan elementos repetidos, característica no favorable para el compresor LZW. Por otro lado, en los *Bytes-Pares*, el comportamiento es diferente, ya que sólo se ocupan 4 de los 8 bits disponibles, entonces, hay una mayor probabilidad de repeticiones. Además, en el tipo de información manejada para este caso, los elementos presentan gran cantidad de repetición o poca variación entre sí en los bits más significativos.

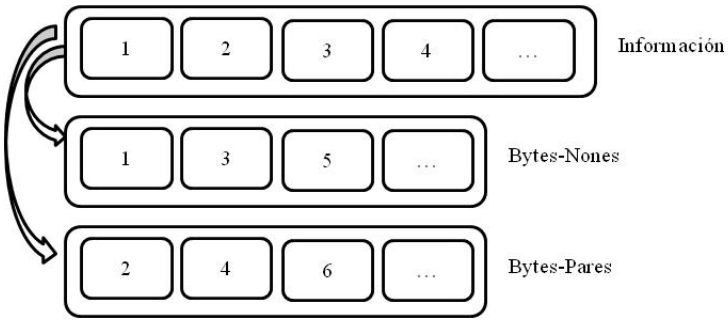

Figura 5.3-Proceso de división en *Bytes-Pares* y *Bytes-Nones*.

La división de la información está representada por la figura 5.3. Una vez dividida la información, es enviada al compresor, ya sea primero los nones o los pares, terminando uno, se limpia el diccionario e inmediatamente después se manda la otra secuencia.

Es notable que se pudiera hacer uso de dos diccionarios para reducir el tiempo de procesamiento, pero se necesitaría una capacidad de memoria mínima de 8192 posiciones trabajando con 12 bits para cada uno de los dos diccionarios. Dado que se está considerando las limitaciones en el espacio de almacenamiento, se procede a la

reutilización de un mismo diccionario teniendo como resultado un compromiso entre velocidad de procesamiento y memoria a utilizar.

Para este proceso se obtiene una razón de compresión de 0.231, la cual representa una mejora del 56.1 y 21.9 % con respecto a los dos últimos procedimientos mencionados anteriormente. El tiempo de compresión, como era de esperarse, incrementó. No obstante, este procedimiento arroja un compromiso aceptable entre razón de compresión y velocidad de procesamiento.

Haciendo una comparación entre *División* y *Byte-Byte* se tiene 0.231 y 0.293 para cada uno de los procedimientos en razón de compresión y con tiempos de procesamiento de 0.825 y 0.549 respectivamente. En este caso se puede apreciar la factibilidad en razón de compresión, pero la desventaja con respecto al tiempo de procesamiento. Comparando contra *12-bits,* el módulo *División* tiene una mejora del 86% con respecto al tiempo de procesamiento y un 57% en la razón de compresión

Es importante determinar prioridades, ya sea con respecto al tiempo consumido o a la reducción máxima de cierta información, es decir, establecer si lo más importante es la velocidad de procesamiento o la reducción de datos.

Aplicando *División* como parte del esquema de compresión, se obtiene una mejora en la capacidad para reducir la información a un costo que se ve reflejado en el incremento del tiempo.

En la tabla 5.5 se muestran los resultados obtenidos aplicando *División* y en la tabla 5.6 se muestran detalles de dicho proceso y resultados obtenidos en cada uno de los archivos ya divididos.

Como se puede apreciar en la tabla 5.6, los *Bytes-Pares* tienen el mejor comportamiento al momento de ser comprimidos. Los archivos probados eran del

mismo tamaño tanto en los nones como en los pares, es decir, si los *Bytes-Nones* tenían 10 bytes, los *Bytes-Pares* también.

Archivos	Original bytes	Comp bytes	Comp Seg	RC
aami3a.dat	86162	20590	0.083	0.239
aami3b.dat	86284	20423	0.083	0.237
aami3c.dat	86288	23707	0.083	0.275
aami3d.dat	69022	18136	0.083	0.263
aami4a.dat	130422	26403	0.083	0.202
aami4a_d.dat	130428	26115	0.083	0.200
aami4a_h.dat	130414	26560	0.083	0.204
aami4b.dat	130476	31519	0.083	0.242
aami4b_d.dat	130478	31585	0.083	0.242
aami4b_h.dat	130474	31846	0.083	0.244
TOTAL	1110448	256884	0.825	0.231

Tabla 5.5-Resultados de *División.*

Archivos	Bytes Nones	Bytes Pares	Total bytes
aami3a.dat	20011	579	20590
aami3b.dat	19816	607	20423
aami3c.dat	22713	994	23707
aami3d.dat	17503	633	18136
aami4a.dat	25191	1212	26403
aami4a_d.dat	24894	1221	26115
aami4a_h.dat	25342	1218	26560
aami4b.dat	30385	1134	31519
aami4b_d.dat	30540	1045	31585
aami4b_h.dat	30717	1129	31846
TOTAL	247112	9772	256884

Tabla 5.6-Detalle en resultados de *División.*

Entonces, se comprueba lo antes mencionado acerca de una alta probabilidad de repetición en la segunda parte de las palabras de 12 bits. En el mejor de los casos, para el ejemplo anterior, se tiene una mejora de un 95% dada la compresión del archivo con los pares respecto a la compresión obtenida por los nones.

5.6 División-LZW-RLE

En el proceso anterior, se observa que en los elementos pares existe mayor repetición de información debido a que éstos representan los bits más significativos y que sólo se ocupan cuatro de los ocho disponibles. La repetición de tales datos pudiera ser mejor aprovechada por el algoritmo RLE.

A partir de lo anterior mencionado, se realiza el siguiente procedimiento llamado *División-LZW-RLE*. En este proceso, al igual que en *División*, se realiza la división de los datos de tal manera que queden los *Bytes-Pares* por un lado y los nones por otro para su procesamiento.

En este caso, se hace uso de la utilización de dos compresores, el LZW y el RLE. Como se mencionó en el capítulo 4, el algoritmo RLE es muy rápido y los recursos utilizados por éste son mínimos. Entonces, los archivos nones son comprimidos utilizando el algoritmo LZW, mientras que para los pares, se utiliza el RLE.

En este caso, la compresión de los archivos, tanto los nones como los pares, se puede realizar de forma paralela, ya que los compresores utilizados en este proceso no comparten recursos y tampoco necesitan información extra. Entonces, mientras se realiza la compresión con LZW, el compresor RLE realiza su trabajo. Lo descrito anteriormente tiene como resultado un ahorro de tiempo, ya que no se tiene que esperar a que culmine la compresión de unos datos para empezar la compresión de otros.

División-LZW-RLE hace un manejo de los datos con un tamaño de 8 bits para poder hacer la división de los mismos de forma correcta y separarlos en nones y pares. Los datos son tratados en su forma original sin la necesidad de aplicarle un corrimiento lógico como en el caso de *12-Bits-Swap.*

Dada la tabla 5.7, aplicando este procedimiento se obtiene una razón de compresión de 0.233 la cual es aceptable si la comparamos contra la obtenida utilizando los 12 bits. Por otro lado, el tiempo de procesamiento es de 0.600 que compite directamente con el tiempo obtenido en *Byte-Byte* que es de 0.825 segundos y con una razón de compresión de 0.231. Nuevamente se confirma la necesidad por establecer una prioridad entre tiempo de procesamiento y reducción de información para poder elegir el mejor procedimiento.

Los resultados obtenidos por *División-LZW-RLE* son consecuencia que los *Bytes-Nones* son procesados con el compresor LZW y que los *Bytes-Pares* trabajan con RLE. La principal desventaja en este procedimiento es que las repeticiones, en la mayoría de los casos, no se encuentran consecutivamente uno de otro. Por tal motivo, la compresión no varía considerablemente respecto a la obtenida en *Byte-Byte*. De suceder el caso contrario, donde en la segunda parte de la palabra, las repeticiones se encuentren de forma consecutiva, la razón de compresión sería bastante alta. Los resultados aplicando este procedimiento se ven reflejados en la tabla 5.7.

Archivos	Original bytes	Comp bytes	Comp Seg	RC
aami3a.dat	86162	20254	0.060	0.235
aami3b.dat	86284	20101	0.060	0.233
aami3c.dat	86288	24072	0.060	0.279
aami3d.dat	69022	17926	0.060	0.260
aami4a.dat	130422	26889	0.060	0.206
aami4a_d.dat	130428	26592	0.060	0.204
aami4a_h.dat	130414	27040	0.060	0.207
aami4b.dat	130476	31891	0.060	0.244
aami4b_d.dat	130478	31800	0.060	0.244
aami4b_h.dat	130474	32223	0.060	0.247
TOTAL	1110448	258788	0.600	0.233

Tabla 5.7-Resultados de *División-LZW-RLE*.

Archivos	Bytes Nones	Bytes Pares	Total Bytes
aami3a.dat	20011	243	20254
aami3b.dat	19816	285	20101
aami3c.dat	22713	1359	24072
aami3d.dat	17503	423	17926
aami4a.dat	25191	1698	26889
aami4a_d.dat	24894	1698	26592
aami4a_h.dat	25342	1698	27040
aami4b.dat	30385	1506	31891
aami4b_d.dat	30540	1260	31800
aami4b_h.dat	30717	1506	32223
TOTAL	247112	11676	258788

Tabla 5.8-Detalle en resultados de *División-LZW-RLE.*

En la tabla5.8 se observa con mayor detalle los resultados obtenidos con *División-LZW-RLE.* Por el lado de los *Bytes-Pares* en el proceso *División* se tiene un tamaño total de 9772 bytes, mientras que *División-LZW-RLE* obtiene un total de 11676 bytes. La diferencia radica principalmente en la capacidad del LZW de aprovechar más las repeticiones de elementos aunque éstos no se encuentren de forma consecutiva. No obstante, *División-LZW-RLE* pudiera tener mejor comportamiento en cuanto al tiempo de procesamiento se refiere. Lo anterior es considerando un banco de pruebas más grande, ya que *División-LZW-RLE* no necesita esperar el procesamiento total de los elementos nones o pares, a diferencia de *División*, que forzosamente tiene que terminar el procesamiento de un elemento para empezar el otro y así no utilizar más recursos de memoria. Entonces, con el uso de *División-LZW-RLE* se obtiene una razón de compresión factible con un buen tiempo de procesamiento y además, los recursos requeridos le permiten ser ocupado en ambientes donde éstos sean limitados.

5.7 Different

El procesamiento de la información para este procedimiento consiste, de igual forma, en la división de los datos de tal manera que se tenga el mismo esquema que en

División: elementos nones y pares. Pero, a diferencia, se realizará un procedimiento extra a los elementos nones.

El proceso de división de elementos va acompañado, sólo para los *Bytes-Nones*, de una comprobación de datos de tal manera que el elemento actualmente procesado será comparado con el dato inmediatamente posterior. Lo anterior con la finalidad de detectar cercanía de valores entre la información.

A continuación se muestra una descripción detallada de dicho procedimiento:

Se lee la cadena de entrada, la cual consta de 12 bits, se leen dos bytes los cuales son divididos de forma independiente uno de otro, por un lado los *Bytes-Nones* y por otro los *Bytes-Pares*, que son descritos en el proceso *División*. Dado los elementos nones únicamente, se leerá byte a byte, pero, el valor leído actualmente será almacenado temporalmente en memoria para su posterior comparación con el elemento siguiente. La comparación de estos elementos tiene la finalidad de que si el elemento en memoria es igual o con una diferencia de valor ya sea más grande o más chico en una unidad, se sustituirá el valor leído por el que se encuentra en memoria, de lo contrario, el valor del elemento en memoria será actualizado por el valor leído actualmente para su posterior comparación con el siguiente elemento. El proceso se realiza mientras exista información posterior al dato leído. Si la diferencia excede en una unidad a los elementos comparados, el cambio de valor no se realiza y los elementos se conservan como originalmente fueron creados. En caso contrario, el cambio de valores se realiza para poder lograr la repetición de más elementos en forma consecutiva.

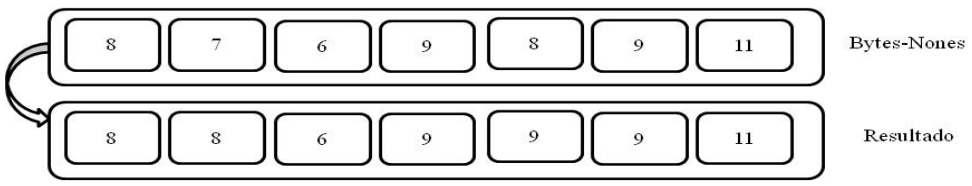

Figura 5.4-Comparando y sustituyendo valores.

Teniendo en cuenta la figura 5.4, el cambio de valor en la comparación se realiza con un máximo de una unidad con respecto de los valores comparados para así evitar la alteración significativa de información original. De igual forma, se procedió a realizar esta comparación en los elementos nones, pero fue omitida en este procedimiento, ya que de aplicarse esta operación en *Bytes-Pares,* la alteración de los datos es alta y en el momento de hacer la representación de los datos, juntando los dos bytes leídos (12 bits de la palabra), existirían anomalías. Tales anomalías, por ejemplo, pudieran repercutir directamente en el diagnóstico realizado por el personal médico. Dicha comparación es aplicada en el procedimiento *Different-LZW-RLW* descrito más adelante.

Como se puede notar en la figura 5.5, el algoritmo presentado, tiene una complejidad proporcional a la cantidad de datos entrantes, pero con operaciones elementales tales como la comparación de elementos y la asignación de valores por lo cual se puede afirmar que este procedimiento está dado por *O(n)*, donde *n* es el número de elementos a leer.

```
Puntero_Archivo=0
While !EOF
{
Leer(dato_none1)
Almacena_Memoria(dato_none1)
Puntero_Archivo++
Leer(dato_none2)
Compara_iguales(Almacena_Memoria,dato_none2)
  If Compara_iguales is TRUE Then
     dato_none2=Almacena_Memoria
}
```

Figura 5.5-Algoritmo de comparación entre elementos.

En la tabla 5.9 se muestran los resultados obtenidos donde se puede apreciar una razón de compresión alta la cual es de 0.181. Dicho resultado se obtiene en un tiempo de 1.240 segundos para el conjunto probado. Es fácil notar que la capacidad de

reducción de información mediante este procedimiento tiene un mejor desempeño a un tiempo de operación razonable.

Archivos	Original bytes	Comp bytes	Comp Seg	RC
aami3a.dat	86162	13738	0.090	0.159
aami3b.dat	86284	13348	0.090	0.155
aami3c.dat	86288	16823	0.090	0.195
aami3d.dat	69022	12390	0.070	0.180
aami4a.dat	130422	20239	0.150	0.155
aami4a_d.dat	130428	19957	0.150	0.153
aami4a_h.dat	130414	20341	0.150	0.156
aami4b.dat	130476	27823	0.150	0.213
aami4b_d.dat	130478	27775	0.150	0.213
aami4b_h.dat	130474	28117	0.150	0.215
TOTAL	1110448.000	200551.000	1.240	0.181

Tabla 5.9-Resultados de *Different.*

Haciendo una comparación entre *Different*, *Byte-Byte* y *12-Bits* se tiene que el segundo de ellos logra una razón de compresión de 0.293 con un tiempo de procesamiento de 0.549 segundos, mientras que *12-Bits* brinda 0.542 como razón de compresión en un tiempo de 6.099 segundos. Por otro lado, *Different* en un tiempo de 1.240 segundos logra la razón de compresión de 0.181, lo que significa que por un incremento de tiempo, se está mejorando la razón de compresión dada por *Byte-Byte*. Irremediablemente, *12-Bits* presenta el peor desempeño y el motivo de comparación es que muchos de los datos ECG son de un tamaño de palabra de 12 bits.

Trabajo	Algoritmos	RC
[42]	AZTEC	0.100
[44]	Cortes	0.210
[113]	Fourier	0.135
[44]	TP	0.500
[35]	SAPA2	0.098

Tabla 5.10-Algoritmos y razón de compresión de algoritmos *lossy*.

Otro punto a favor de *Different*, es que la razón de compresión se acerca a la obtenida por algoritmos de compresión tales como los mencionados en [113] y los cuales son algoritmos de compresión de tipo *lossy*, diferentes a los tratados en esta investigación. Tales algoritmos son aplicados en datos tipo ECG.

Archivos	Bytes Nones	Bytes Pares	Total Bytes
aami3a.dat	13159	579	13738
aami3b.dat	12741	607	13348
aami3c.dat	15829	994	16823
aami3d.dat	11757	633	12390
aami4a.dat	19027	1212	20239
aami4a_d.dat	18736	1221	19957
aami4a_h.dat	19123	1218	20341
aami4b.dat	26689	1134	27823
aami4b_d.dat	26730	1045	27775
aami4b_h.dat	26988	1129	28117
TOTAL	190779	9772	200551

Tabla 5.11-Detalle en resultados de *Different.*

Para mayor detalle de los resultados logrados con *Different* se observa la tabla 5.11 en la que se pueden constatar los valores obtenidos para cada uno de los elementos *Bytes-Nones* y *Bytes-Pares.*

5.8 Different-LZW-RLE

Hasta este punto, se puede afirmar, que *Different* es el proceso con mejor desempeño en la reducción de información, con un incremento despreciable en el tiempo de ejecución con respecto a sus contendientes.

Dado que el proceso descrito con anterioridad presenta los mejores resultados, y, analizando las tablas de detalle, en donde se realiza el proceso de división de elementos nones y pares, se visualiza que en los elementos pares se puede obtener un mejor aprovechamiento.

Los elementos pares, por sus características: posición que ocupan al momento de formar una palabra de 12 bits y la alta probabilidad de repetición de elementos entre sí; permite que sea factible y posible la obtención de mejores resultados que se expresan directamente en el desempeño de la compresión.

Prácticamente se realiza una serie de tareas muy parecida a la del proceso *Different*: leer la información, dividir en elementos pares y nones, diferencias entre valores (sólo para elementos nones) y comprimir. La etapa de comprimir la información en *Different* está determinada por el algoritmo LZW tanto en las entidades nones como en las pares. En *Different-LZW-RLE*, la diferencia de valores (tercer paso en *Different*) se aplica a los datos nones y a los pares, ya que, como se mencionó con anterioridad, los elementos pares, tienden más a la repetición. Además, la compresión de los datos se realiza mediante la aplicación del compresor LZW, en los nones, y, el algoritmo de compresión RLE, en los pares.

La aplicación del algoritmo RLE a los valores pares tiene como finalidad encontrar el mayor número de repeticiones de forma consecutiva dado que se les aplica la comparación de valores con una diferencia máxima de una unidad entre sí para poder sustituirlo. Entonces, si los elementos comparados difieren a lo mucho en una unidad, se sustituye el segundo valor por el primero comparado, lo que conlleva a que existan más elementos repetidos seguidos unos de otros. Caso tal se pude ver en la figura 5.6.

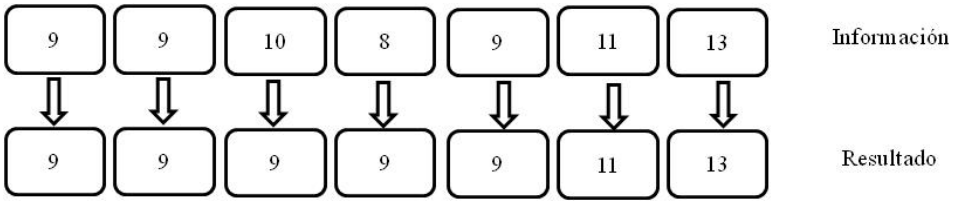

Figura 5.6-Ejemplo de sustitución de valores.

La figura 5.6 muestra cómo se logra obtener repeticiones de elementos a partir de la comparación y sustitución de valores. Dichos resultados son aprovechados por el

compresor RLE al encontrarse secuencias de datos con el mismo valor de forma consecutiva.

En la tabla 5.12 se muestran los resultados obtenidos con *Different-LZW-RLE*.

Archivos	Original bytes	Comp bytes	Comp Seg	RC
aami3a.dat	86162	13162	0.060	0.153
aami3b.dat	86284	12744	0.060	0.148
aami3c.dat	86288	15832	0.060	0.183
aami3d.dat	69022	11760	0.060	0.170
aami4a.dat	130422	19030	0.060	0.146
aami4a_d.dat	130428	18739	0.060	0.144
aami4a_h.dat	130414	19126	0.060	0.147
aami4b.dat	130476	26692	0.060	0.205
aami4b_d.dat	130478	26733	0.060	0.205
aami4b_h.dat	130474	26991	0.060	0.207
TOTAL	1110448	190809	0.600	0.172

Tabla 5.12-Resultados de *Different-LZW-RLE.*

Como se puede apreciar en la tabla 5.12, la compresión obtenida es de 0.172 con un tiempo para lograrla de 0.6 segundos. *Different* brinda una razón de compresión de 0.181 en un tiempo de 1.240segundos. A partir de lo último mencionado, el mejor desempeño tanto en tiempo como en la reducción de los datos es generado por *Different-LZW-RLE*, superando a *Different* en la compresión de los datos y lográndolo en la mitad del tiempo.

En la tabla 5.13 se puede comprobar el comportamiento y la reducción de información en los *Bytes-Pares.* Tomando como referencia a *Different*, se observa que en los elementos pares, el valor más pequeño obtenido por la compresión es de 579 bytes que compite directamente contra los 3 bytes obtenidos mediante *Different-LZW-RLE*. Por otro lado, en los elementos nones de ambos procedimientos: *Different y Different-LZW-RLE*, los resultados obtenidos son iguales, ya que el procedimiento es el mismo y la compresión es llevada en ambos casos por el algoritmo LZW.

Archivos	Bytes Nones	Bytes Pares	Total Bytes
aami3a.dat	13159	3	13162
aami3b.dat	12741	3	12744
aami3c.dat	15829	3	15832
aami3d.dat	11757	3	11760
aami4a.dat	19027	3	19030
aami4a_d.dat	18736	3	18739
aami4a_h.dat	19123	3	19126
aami4b.dat	26689	3	26692
aami4b_d.dat	26730	3	26733
aami4b_h.dat	26988	3	26991
TOTAL	190779	30	190809

Tabla 5.13-Detalle en resultados de *Different-LZW-RLE*.

El tiempo de procesamiento en este caso se debe a que el algoritmo de compresión RLE es muy rápido, como se menciona en el capítulo 2 y 4. Además, los recursos necesarios para la compresión son mínimos.

Ciertamente, se podría decir que *Different-LZW-RLE* es el procedimiento con los mejores resultados presentados en esta investigación. No obstante, existe un detalle muy significativo que afecta directamente a los datos cuando éstos son recuperados para su interpretación. Sucede que, a pesar del desempeño obtenido, no se puede aplicar este proceso a datos en los cuales, el margen de error en el momento de recuperación, es bajo. Debido principalmente a la utilización de la comparación e intercambio de valores entre los elementos pares, *Different-LZW-RLE* presenta una desventaja para este caso.

Al momento en que se está modificando o alterando la información de los elementos pares, se afecta directamente a la palabra que se forma con los 12 bits originales con una diferencia de 256 unidades. Consecuentemente, al momento de unir un elemento impar con su correspondiente elemento par, para conseguir 12 bits, la parte menos

significativa (non) es la que pudiera, en un momento dado, tener algún tipo de alteración controlada, específicamente, una unidad de diferencia.

En este caso, tanto *Different* como *Different-LZW-RLE* modifican valores en los *Bytes-Nones* sí y sólo sí la diferencia resultante de la comparación entre un elemento y otro consecutivo, es mayor o menor en sólo una unidad, de lo contrario, la información no se altera. Ahora bien, aplicando el mismo procedimiento al byte más significativo de una palabra, se tendrá el mismo comportamiento. La diferencia significativa radica al momento de juntar la parte menos significativa con la parte más representativa formando una palabra de 12 bits.

En el caso de la modificación de los elementos nones, y no modificar los pares, al momento de formar la palabra de 12 bits, la información se vería afectada con respecto a su magnitud, la cual aumentaría o disminuiría sola una unidad. Modificando los pares y alterando o no los nones, al juntarlos, la información varía radicalmente. Véase figura 5.7.

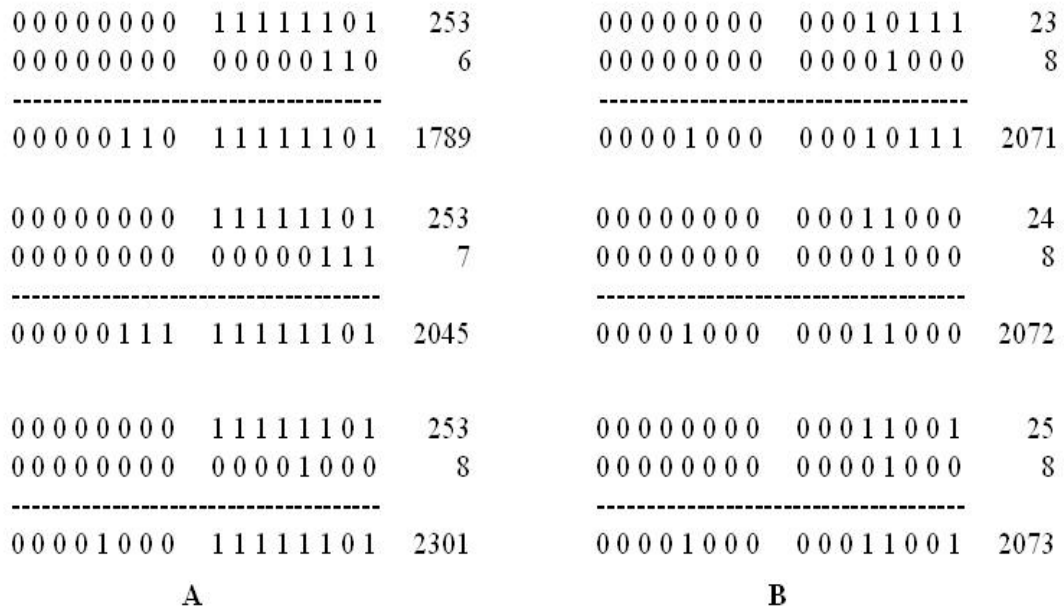

Figura 5.7-Modificación de *Bytes-Pares* (A) y *Bytes-Nones* (B).

En la figura 5.7 A se realiza la modificación a los elementos pares y no a los nones. Resulta evidente la variante entre los resultados aún considerando que el cambio de valores sólo se realiza en una unidad. En el primer ejemplo, donde 253 (non) y 6 (par), se tiene un resultado de 1789 que varía en 256 unidades con lo obtenido en el segundo ejemplo donde 253 (non) y (7). El mismo caso aplica para los valores 253 (non) y 8 (par). Si bien es notorio que el patrón de diferencia entre los elementos de la figura 5.7 A es de 256 unidades, no se encontró una implementación eficiente que pudiera adaptarse a tal esquema y determinar con exactitud qué elementos incrementaban y cuáles decrementaban.

El poder determinar con certeza el tipo de cambio que sufre un dato específico resulta crucial, ya que en el momento de ser representado como información descomprimida, se pudieran realizar los cambios pertinentes con respecto a su magnitud y así, evitar errores de interpretación por parte del usuario final. Los métodos probados para esto, incrementan el uso de recursos computacional y tiempo de procesamiento, haciendo que *Different-LZW-RLE* tenga un peor desempeño comparándolo con otro procedimiento de los antes ya mencionados. Además, la razón de compresión tiende a disminuir.

Entre alguna de las técnicas probadas para poder tener un control de la variación de los datos, es la de tener un símbolo especial que indicara si los elementos procesados aumentaban su valor o lo disminuían. La implementación de lo anterior trae como resultado que la información tuviera un incremento con lo que respecta a su tamaño original. Si bien pudieran existir otras técnicas para no incrementar el tamaño original, y así perder desempeño en la compresión, es cierto que dichos procedimientos repercutirían directamente en el tiempo para ejecutar dicha tarea.

Por otro lado, en la figura 5.7 B, se observa que la modificación a los *Bytes-Nones*, no afecta considerablemente a la variante entre los elementos. Sean los resultados finales: 2071, 2072 y 2073, donde los elementos nones son 23,24 y 25

respectivamente, es notoria la diferencia entre los elementos y otro, y, si el valor a comparar es el 24, existen dos posibles candidatos a sufrir una alteración de valor: 23 y 25. Entonces, realizando el proceso de modificar los elementos cercanos, con una diferencia de una unidad, 23 y 25 se convertirían en 24 prácticamente. El valor asociado a estos tres nuevos elementos (24,24,24) combinado con su respectivo elemento par (8) sería de 2072. Con esto se logra que en la representación de la información, no haya confusión por el usuario final al momento de analizar los datos. Además, no es necesaria la utilización de elementos de control que determinen si un elemento aumentó, disminuyó o si su valor permanece igual, con lo cual se gana tiempo de procesamiento, disminución en la utilización de recursos y una razón de compresión que no se ve afectada por dicho control.

5.9 Divide2Different-LZW

Analizando otras posibles situaciones tratando de reducir aún más la información procesada y mejorar el desempeño de cualquiera de los métodos antes mencionados, se optó por, de nueva cuenta, realizar un tratado diferente a los elementos pares.

Lo anterior se debe a que los elementos pares presentan la característica de tener posiciones libres dentro de su representación digital, específicamente 4 posiciones, lo que implica que dentro de un byte, pudieran almacenar dos elementos.

Realizando la implementación, es necesario leer dos elementos en los datos pares y posteriormente aplicarle un corrimiento de cuatro posiciones al segundo elemento leído para su posterior interacción con el primer dato procesado mediante el operador OR y así juntar ambos elementos dentro de un mismo byte.

En la figura 5.8 se aprecia claramente la disposición de los elementos, y cómo, dado dos valores que están representados por dos bytes, son representados en uno solo. Consecuentemente a esto, la cantidad de información original es reducida a la mitad, es decir, *n/2*, donde *n* representa al total de los elementos pares. El grado de

complejidad de este procedimiento radica únicamente en el corrimiento de posiciones y la operación OR. Posteriormente, la información será procesada por el algoritmo de compresión LZW.

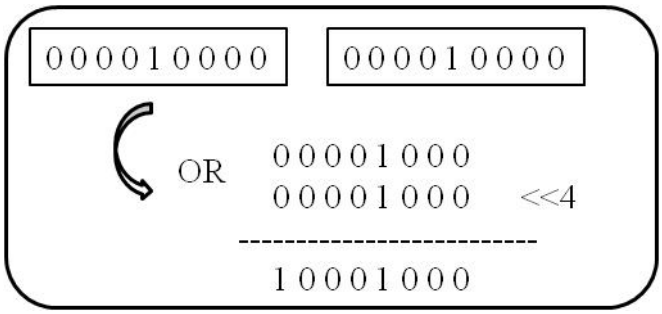

Figura 5.8-Proceso de juntar dos bytes en uno.

Análogamente, los *Bytes-Nones* son tratados de igual forma que en *Different* y *Different-LZW-RLE*. Ahora bien, los resultados obtenidos aplicando esta técnica son mostrados en la tabla 5.14 y los detalles de los pares y nones, en la tabla 5.15.

Archivos	Original bytes	Comp bytes	Comp Seg	RC
aami3a.dat	86162	13622	0.110	0.158
aami3b.dat	86284	13227	0.110	0.153
aami3c.dat	86288	16705	0.110	0.194
aami3d.dat	69022	12273	0.110	0.178
aami4a.dat	130422	20176	0.155	0.155
aami4a_d.dat	130428	19894	0.155	0.153
aami4a_h.dat	130414	20290	0.155	0.156
aami4b.dat	130476	27764	0.155	0.213
aami4b_d.dat	130478	27696	0.155	0.212
aami4b_h.dat	130474	28075	0.155	0.215
TOTAL	1110448	199722	1.369	0.180

Tabla 5.14-Resultados de *Divide2Different-LZW*.

Archivos	Bytes Nones	Bytes Pares	Total Bytes
aami3a.dat	13159	463	13622
aami3b.dat	12741	486	13227
aami3c.dat	15829	876	16705
aami3d.dat	11757	516	12273
aami4a.dat	19027	1149	20176
aami4a_d.dat	18736	1158	19894
aami4a_h.dat	19123	1167	20290
aami4b.dat	26689	1075	27764
aami4b_d.dat	26730	966	27696
aami4b_h.dat	26988	1087	28075
TOTAL	190779	8943	199722

Tabla 5.15-Detalle en resultados de *Divide2Different-LZW*.

La razón de compresión obtenida es de 0.180 con un tiempo de procesamiento de 1.369 segundos. En la tabla 5.15 es notable que los *Bytes-Pares* presentan un valor menor numéricamente con respecto a *Different.* No obstante, haciendo una comparación contra el proceso *Different*, existe una diferencia mínima en la razón de compresión, y de igual forma, una diferencia despreciable en el tiempo de procesamiento.

5.10 Divide2Diferent-LZW-RLE

Otra técnica probada a partir del procedimiento *Divide2Different-LZW* con la finalidad de obtener una mejor compresión es la de *Divide2Different-LZW-RLE*. En este caso se aplica, en los elementos pares, el compresor RLE para analizar el comportamiento que tienen los datos con dicho algoritmo.

Una vez realizado el proceso de juntar dos elementos en uno sólo, es decir, reducir la información a la mitad, los datos son comprimidos por *Run Lenght Encoding*. Los *Bytes-Nones* tienen el mismo procesamiento descrito en *Divide2Different-LZW*.

Archivos	Original bytes	Comp bytes	Comp Seg	RC
aami3a.dat	86162	13510	0.069	0.157
aami3b.dat	86284	13107	0.069	0.152
aami3c.dat	86288	17644	0.069	0.204
aami3d.dat	69022	12258	0.069	0.178
aami4a.dat	130422	21319	0.069	0.163
aami4a_d.dat	130428	21091	0.069	0.162
aami4a_h.dat	130414	21415	0.069	0.164
aami4b.dat	130476	28822	0.069	0.221
aami4b_d.dat	130478	28548	0.069	0.219
aami4b_h.dat	130474	29175	0.069	0.224
TOTAL	1110448	206889	0.686	0.186

Tabla 5.16-Resultados de *Divide2Different-LZW-RLE.*

En la tabla 5.16 se aprecia que esta técnica implementada tiene un desempeño ligeramente menor con respecto a la compresión obtenida, ya que logra una razón de compresión de 0.186. Sin embargo, el tiempo en la que se realiza la compresión de los datos, es por debajo de otros esquemas mencionados anteriormente, con un tiempo total de 0.686 segundos. De nueva cuenta, la velocidad de procesamiento se debe principalmente al compresor RLE.

Se consideró que con *Divide2Different-LZW-RLE* se iba a obtener una mejor compresión de la información, pero como demuestra la tabla 5.16, no es así. Por el lado del tiempo de compresión se esperaban los resultados obtenidos, y que este procedimiento fuera mejor que su predecesor.

Analizando más a detalle el comportamiento de *Divide2Different-LZW-RLE,* la tabla 5.17 muestra cómo en un principio se obtienen mejores resultados, posteriormente la tendencia es de tener un crecimiento en la cantidad de información manejada.

Archivos	Bytes Nones	Bytes Pares	Total Bytes
aami3a.dat	13159	351	13510
aami3b.dat	12741	366	13107
aami3c.dat	15829	1815	17644
aami3d.dat	11757	501	12258
aami4a.dat	19027	2292	21319
aami4a_d.dat	18736	2355	21091
aami4a_h.dat	19123	2292	21415
aami4b.dat	26689	2133	28822
aami4b_d.dat	26730	1818	28548
aami4b_h.dat	26988	2187	29175
TOTAL	190779	16110	206889

Tabla 5.17-Detalle en resultados de *Divide2Different-LZW-RLE*.

Como comentario final se puede mencionar que *Divide2Different-LZW-RLE* no tuvo el desempeño deseado, ya que se obtiene una razón de compresión por encima de la obtenida en *Divide2Different-LZW*, cuando se esperaba lo contrario.

5.11 Comparación en Compresión

Una serie de procedimientos fueron aplicados a los datos de trabajo, cada uno con sus respectivos tiempos de procesamiento y la razón de compresión que podían alcanzar en un momento dado.

Para efectos del esquema de compresión aplicado a telemedicina, es importante tener una relación de compromiso entre el tiempo de ejecución necesario y la capacidad de tener una alta compresión con respecto a la información original. Entonces, se debe poner énfasis en aquella rutina que se adapte mejor a las necesidades de tiempo y compresión.

Se describieron una serie de procedimientos, conjunción de algoritmos de compresión y pre procesamiento aplicado a la información con el fin de obtener como resultado la mínima cantidad de datos, es decir, una reducción de los datos lo más alto posible. A

continuación se muestra en la tabla 5.18 los resultados obtenidos por los diferentes métodos aplicados.

Método	**RC**	**Comp Seg**
Byte-Byte	0.293	0.549
12-Bits	0.542	6.099
12-Bits-Swap	0.526	6.208
Byte-Byte-Swap	0.296	0.549
Division	0.231	0.825
Division-LZW-RLE	0.233	0.600
Different	0.181	1.240
Different-LZW-RLE	0.172	0.600
Divide2Different-LZW	0.180	1.369
Divide2Different-LZW-RLE	0.186	0.686

Tabla 5.18-Resultados de los módulos de pre procesamiento.

Si se tuviera que elegir un método tan sólo por la razón de compresión obtenida, éste sería definitivamente *Different-LZW-RLE*. Pero como ya se mencionó, este procedimiento altera de forma significativa la información. Por otro lado, si tan sólo se basara en el tiempo de procesamiento para la elección de un método, se elegiría a *Byte-Byte.*

Caso inverso, la compresión con el peor desempeño se obtiene utilizando *12-Bits* y *12-Bits-Swap* que tienen un tiempo de ejecución de 0.542 y 0.526 segundos respectivamente, siendo de igual forma, los peores tiempos obtenidos.

Ahora bien, los que presentan un equilibrio entre razón de compresión y tiempo de procesamiento son: *Different* con un tiempo de 1.240 segundos y 0.181 de compresión, *Divide2Different-LZW* con 0.180 de compresión en un tiempo de 1.369 segundos, *Divide2Different-LZW-RLE* que obtiene en 0.686 segundos la reducción a 0.186.

Es importante recordar que la elección de los algoritmos LZW y RLE se debe a la reducida complejidad que presentan, así como los pocos recursos que se utilizan para poder realizar el proceso de compresión. A partir de ellos, se realizaron las implementaciones de los procesos descritos en este capítulo.

En la figura 5.9 se realiza una comparación entre tres diferentes procedimientos: *Different*, *Byte-Byte* y *12-Bits.* Como se puede observar, el último de ellos presenta la peor razón de compresión, mientas que *Byte-Byte* figura en la segunda posición con una razón de compresión de 0.293. Finalmente, *Different*, con una razón de compresión de 0.181, se ubica como en el mejor procedimiento de la comparación.

Otro detalle que se puede observar en la misma figura, es cómo la compresión de los datos varía entre cada uno de los procedimientos. Visto de otra forma, si se quisiera realizar el proceso de compresión a unos datos, en este caso, para información de 12 bits tal cual, el resultado sería una reducción de información del 54.2%. No obstante, con un pre procesamiento de la información se llega a obtener hasta un 18.1% de compresión.

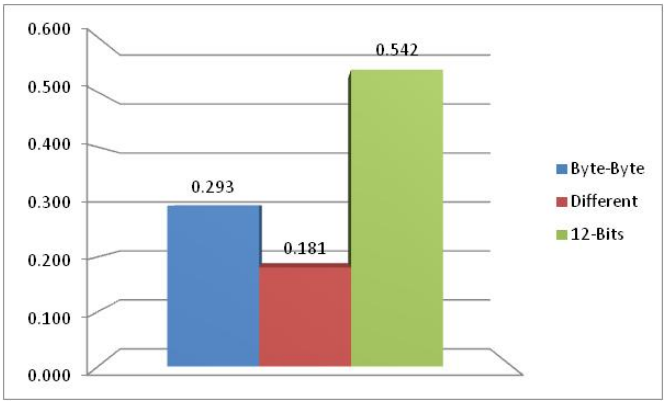

Figura 5.9-Comparación entre tres métodos de pre procesamiento (razón de compresión).

Por el lado del tiempo de ejecución, en la figura 5.10 se observan los tres diferentes tiempos obtenidos. Primeramente, el tiempo más alto y por ende el peor, es el arrojado por *12-Bits* con un tiempo de 6.099 segundos. *Different* aporta un tiempo de

1.240 segundos y finalmente, el mejor tiempo es conseguido aplicando *Byte-Byte*. No hay que perder de vista la relación compresión-tiempo. Es notable la diferencia de tiempos para cada uno de estos procedimientos así como lo es notable su compresión. Por lo cual, *Different* presenta un mejor compromiso entre razón de compresión, tiempo de ejecución y recursos necesarios para llevarlo a cabo. Si bien se pudiera refutar lo anterior anteponiendo a *Byte-Byte* como el mejor procedimiento, también se pudiera decir que la compresión obtenida por este último, pudiera ser alcanzada por un compresor comercial, aunque claro, con el uso de más recursos computacionales.

Los recursos necesarios para la realización de dichas tareas varían entre un procedimiento y otro. Por ejemplo, en el caso de *Byte-Byte* los recursos utilizados se limitan al almacenamiento en memoria del diccionario utilizado, es decir, dependiendo del tamaño del diccionario, será la utilización de memoria, claro, no dejando de lado las operaciones básicas del compresor.

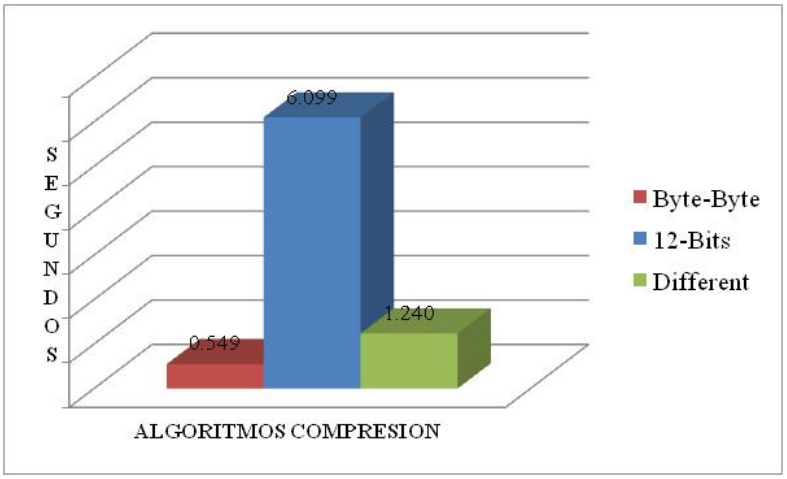

Figura 5.10- Comparación entre tres métodos de pre procesamiento (tiempo de compresión).

Different, aparte de considerar el tamaño del diccionario y las operaciones del compresor LZW, tiene en cuenta la división de los elementos pares y nones, así como la comparación de elementos en los *Bytes-Nones.* Ni en la división de elementos ni en la comparación de datos hay un incremento de complejidad con respecto al algoritmo,

sin embargo, es necesario la utilización de algunas variables extras. Pero, el procedimiento con mayores recursos utilizados es *12-bits*, donde un diccionario de mayor tamaño es necesario para una buena compresión. También es necesario la utilización de funciones de corrimiento y concatenación de elementos por lo que se necesitan más variables auxiliares, además, al momento de realizar una búsqueda en el diccionario, el tiempo es más alto.

En la figura 5.11 se muestra el comportamiento de los procedimientos descritos en este capítulo. La gráfica muestra la razón de compresión obtenida por cada uno de ellos y se puede realizar una comparación con respecto a la razón de compresión en la cual *12-Bits* tiene el peor desempeño y *Different-LZW-RLE* el mejor. Es notable la diferencia en la compresión cuando la información es tratada directamente como una palabra de 12 bits. Los demás procedimientos, con diferente pre procesamiento para la información presentan variación en la razón de compresión obtenida entre cada uno de ellos.

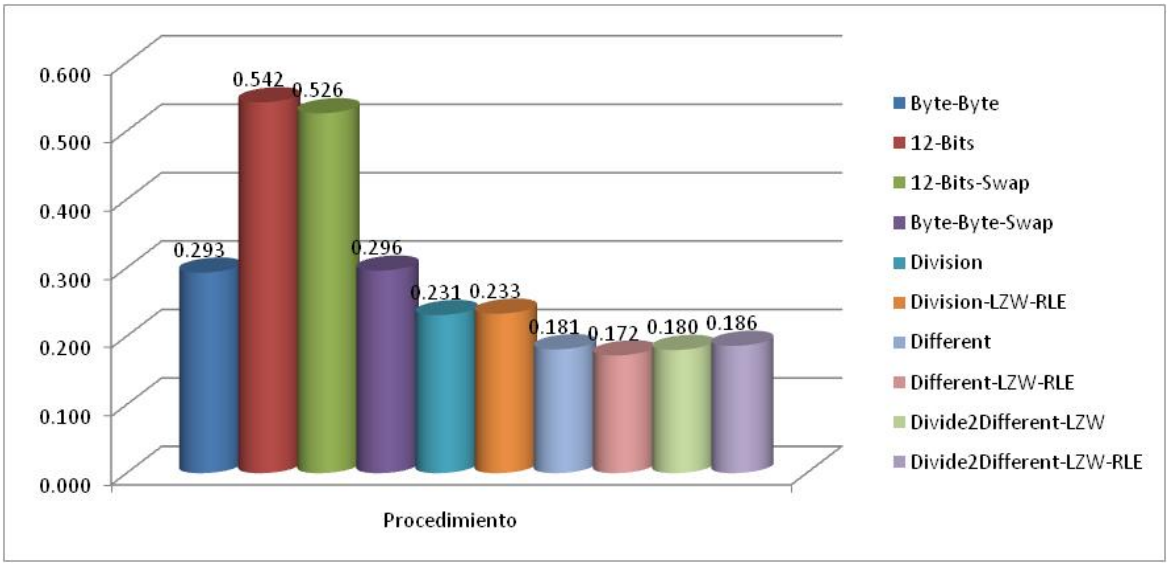

Figura 5.11-Resultados de los módulos de pre procesamiento (razón de compresión).

Con respeto al tiempo de procesamiento, en la figura 5.12 se puede identificar claramente que el mejor tiempo de ejecución es obtenido por *Byte-Byte* y *Byte-Byte-Swap* con un tiempo de 0.549 segundos en ambos casos. El peor desempeño se consigue aplicando *12-Bits-Swap* por lo que la aplicación de este último no es recomendable, además de que los resultados arrojados en la compresión también son malos. De nueva cuenta es detectable el incremento en el tiempo cuando la información se trata con procedimientos que involucran el procesamiento de 12 bits.

No hay que perder de vista que en la mayoría de los casos, una mayor compresión requiere un tiempo de ejecución más alto y los recursos necesarios para llevar a cabo dicha tarea, se incrementan.

Finalmente, en esta sección del capítulo, se realiza un enfoque principalmente a tres procedimientos: *12-Bits*, *Byte-Byte* y *Different*, ya que la comparación de todos los procedimientos pudiera resultar redundante.

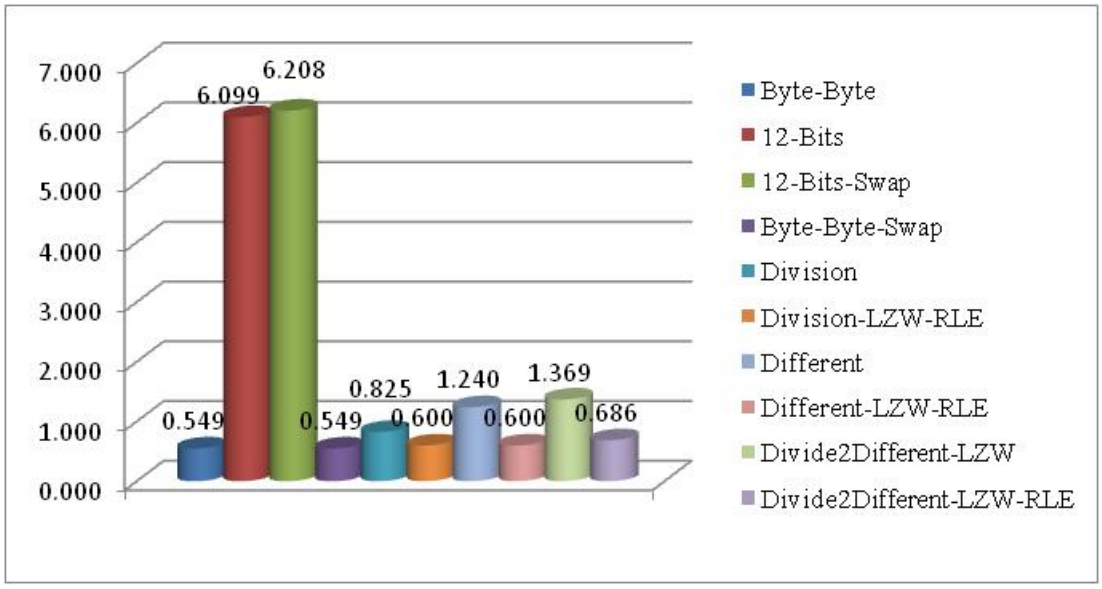

Figura 5.12-Resultados de los módulos de pre procesamiento (tiempo de compresión).

Además, la comparación realizada busca detectar el peor de los casos, el caso medio y el mejor desempeño en sus casos base en compresión-tiempo. Si bien se pudiera

mencionar que el mejor tiempo de ejecución en conjunción con la mejor compresión se obtiene aplicando *Different-LZW-RLE,* se hace mención del problema con dicho procedimiento anteriormente. No obstante, el procedimiento que tiene un desempeño equiparable comparado con *Different* es el procedimiento *Divide2Different-LZW* el cual arroja una compresión de 0.180 en un tiempo de 1.369 segundos por lo cual, será el procedimiento que formará parte del esquema de compresión y cifrado. Como es notorio, se está poniendo más énfasis en la capacidad de reducción de los datos. Además, este procedimiento pudiera ser aplicado a otro tipo de datos en donde no necesariamente existieran repeticiones de forma consecutiva, detalle que no puede darse en *Divide2Different-LZW-RLE.*

Retomando lo mencionado al principio de este capítulo acerca de que un sólo paciente puede generar hasta 34 gb de información ECG mensualmente, se tiene que, aplicando el módulo *Divide2Different-LZW* se puede reducir la información de 34 hasta 6 gb aproximadamente. Lo anterior significa que se están reduciendo los datos más de 5 veces, implicando un gran ahorro de recursos, tanto de espacio como de transmisión. Para este caso se tiene un ahorro de 28 gb que pudieran ocuparse para el almacenamiento de otros datos.

También, se puede hacer la comparativa entre *Divide2Different-LZW*y los algoritmos de compresión *con pérdida* de la tabla 5.10. Como es de notarse, la razón de compresión obtenida por *Divide2Different-LZW* se acerca a la obtenida por los algoritmos *con pérdida*, incluso, en algunos casos, la supera.

5.12 Comparación en Cifrado

A continuación se muestra la comparación entre los resultados obtenidos mediante la aplicación de los algoritmos de cifrado.

Hay que tener en cuenta que un algoritmo de cifrado, a diferencia de un algoritmo de compresión, es independiente de la información, es decir, no importa qué tipo de dato

se esté manejando, el tiempo de cifrado será el mismo y éste dependerá directamente de la cantidad de información a cifrar.

En el capítulo 4 se determina qué algoritmo de cifrado es el que va a formar parte del esquema, sin embargo, en este capítulo se muestran resultados obtenidos con los diferentes algoritmos de cifrado tratados, con la finalidad de afirmar la decisión tomada anteriormente.

Como primer elemento se muestra el algoritmo *DES*.

Archivos	Original Bytes	Cifra Seg	Descifra Seg
aami3a.dat	86162	0.234	0.234
aami3b.dat	86284	0.250	0.250
aami3c.dat	86288	0.250	0.250
aami3d.dat	69022	0.203	0.203
aami4a.dat	130422	0.375	0.375
aami4a_d.dat	130428	0.375	0.375
aami4a_h.dat	130414	0.375	0.375
aami4b.dat	130476	0.358	0.358
aami4b_d.dat	130478	0.375	0.375
aami4b_h.dat	130474	0.375	0.375
TOTAL	1110448	3.171	3.171

Tabla 5.19-Resultados de DES en Telemedicina.

En la tabla 5.19 se observa el comportamiento de DES, teniendo un tiempo de ejecución de 3.171 segundos. Nótese que el cifrado en dicha tabla se aplicó a la información original, es decir, sin un procesamiento de compresión previo. Pero, como en todo esquema de compresión y cifrado de datos, el primer proceso a ejecutarse es el de compresión.

Aplicando la compresión como primer procedimiento, antes que el cifrado, se logra, obviamente, eliminar la redundancia de información, y, la seguridad aumenta para el cifrado debido a que es menos susceptible a ataques para descubrir cierto patrón en

los datos. Dicho lo anterior, en la tabla 5.20 se presenta el tiempo de cifrado para los datos después de un proceso de compresión.

Archivos	Original Bytes	Cifra Seg	Descifra Seg	Compre+Cifrado Seg
aami3a.lzw	13622	0.037	0.037	0.147
aami3b.lzw	13227	0.038	0.038	0.148
aami3c.lzw	16705	0.048	0.048	0.158
aami3d.lzw	12273	0.036	0.036	0.146
aami4a.lzw	20176	0.058	0.058	0.213
aami4a_d.lzw	19894	0.057	0.057	0.212
aami4a_h.lzw	20290	0.058	0.058	0.213
aami4b.lzw	27764	0.076	0.076	0.231
aami4b_d.lzw	27696	0.080	0.080	0.235
aami4b_h.lzw	28075	0.081	0.081	0.236
TOTAL	199722	0.570	0.570	1.939

Tabla 5.20-Resultados de *Divide2Different-LZW* y DES.

Con el algoritmo DES se obtiene un tiempo de cifrado de 0.570 segundos al procesar la información después de que ésta ha sido comprimida, en este caso por *Divide2Different-LZW*. Otro factor importante, es el tiempo final que se toma para realizar el proceso de compresión y cifrado en conjunto. Para el caso de *Divide2Different-LZW* con el cifrador DES, se logra un tiempo de 1.939 para los datos probados.

En los siguientes párrafos se analizan los dos cifradores restantes que fueron elegidos para efectos de esta investigación.

En la tabla 5.21 se observa el comportamiento de RC4 con los datos originales, sin compresión, mientras que en la tabla 5.21, con un tiempo de ejecución de 0.044 segundos, se muestra al algoritmo RC4 en conjunción con *Divide2Different-LZW*. Con este algoritmo, por las características mencionadas en el capítulo 2 y 4, se esperaba un procesamiento rápido. Comparado con DES, existe una diferencia de 0.526 segundos. Ahora, el tiempo combinado entre compresión y cifrado, para este

caso, es de 1.413 segundos representando el 72.87% del tiempo total de DES con la compresión correspondiente.

Archivos	Original Bytes	Cifra Seg	Descifra Seg
aami3a.dat	86162	0.015	0.015
aami3b.dat	86284	0.015	0.015
aami3c.dat	86288	0.015	0.015
aami3d.dat	69022	0.010	0.010
aami4a.dat	130422	0.031	0.031
aami4a_d.dat	130428	0.031	0.031
aami4a_h.dat	130414	0.031	0.031
aami4b.dat	130476	0.031	0.031
aami4b_d.dat	130478	0.031	0.031
aami4b_h.dat	130474	0.031	0.031
TOTAL	1110448	0.243	0.243

Tabla 5.21-Resultados de RC4 en Telemedicina.

Archivos	Original bytes	Cifra Seg	Descifra Seg	Compre+Cifrado Seg
aami3a.lzw	13622	0.002	0.002	0.112
aami3b.lzw	13227	0.002	0.002	0.112
aami3c.lzw	16705	0.003	0.003	0.113
aami3d.lzw	12273	0.002	0.002	0.112
aami4a.lzw	20176	0.005	0.005	0.160
aami4a_d.lzw	19894	0.005	0.005	0.160
aami4a_h.lzw	20290	0.005	0.005	0.160
aami4b.lzw	27764	0.007	0.007	0.162
aami4b_d.lzw	27696	0.007	0.007	0.162
aami4b_h.lzw	28075	0.007	0.007	0.162
TOTAL	199722	0.044	0.044	1.413

Tabla 5.22- Resultados de *Divide2Different-LZW* y RC4.

De igual forma, en la tabla 5.23 se muestran los resultados obtenidos a los datos sin compresión. Posteriormente, en la tabla 5.24 se hace referencia a los resultados obtenidos aplicando el algoritmo de cifrado AES. Se obtiene un tiempo de 0.048

segundos para el cifrado de los datos con una diferencia de 0.004 y de 0.522 segundos comparado contra RC4 y DES respectivamente.

En el caso de los tiempos obtenidos por DES y AES, la diferencia es notable, pero, comparado con RC4, el tiempo de cifrado es mínimo. Por tal motivo, se realizó el cifrado con archivos de datos más grandes (en bytes). Entonces, el archivo a cifrar tiene un tamaño de 1,575,000 bytes. Primeramente, para el cifrado RC4 se obtiene un tiempo de 0.078 segundos y, con AES el tiempo de cifrado es de 0.421 segundos. En [114] se afirma la superioridad de RC4 sobre AES con respecto a la velocidad de cifrado.

Archivos	Original Bytes	Cifra Seg	Descifra Seg
aami3a.dat	86162	0.016	0.016
aami3b.dat	86284	0.016	0.016
aami3c.dat	86288	0.016	0.016
aami3d.dat	69022	0.031	0.031
aami4a.dat	130422	0.031	0.031
aami4a_d.dat	130428	0.031	0.016
aami4a_h.dat	130414	0.031	0.031
aami4b.dat	130476	0.031	0.031
aami4b_d.dat	130478	0.031	0.031
aami4b_h.dat	130474	0.031	0.031
TOTAL	1110448	0.266	0.250

Tabla 5.23- Resultados de AES en Telemedicina.

Se puede concluir que el algoritmo RC4 es el que cifra los datos con mayor velocidad, y aunque con datos pequeños AES se encuentra cerca de dicha velocidad, ésta tiende a ser menor que la obtenida con RC4, cuando se trata de datos de mayor tamaño.

Archivos	Original bytes	Cifra Seg	Descifra Seg	Compre+Cifrado Seg
aami3a.lzw	13622	0.002	0.002	0.112
aami3b.lzw	13227	0.002	0.002	0.112
aami3c.lzw	16705	0.003	0.003	0.113
aami3d.lzw	12273	0.006	0.006	0.115
aami4a.lzw	20176	0.005	0.005	0.160
aami4a_d.lzw	19894	0.005	0.002	0.160
aami4a_h.lzw	20290	0.005	0.005	0.160
aami4b.lzw	27764	0.007	0.007	0.162
aami4b_d.lzw	27696	0.007	0.007	0.162
aami4b_h.lzw	28075	0.007	0.007	0.162
TOTAL	199722	0.048	0.046	1.417

Tabla 5.24- Resultados de *Divide2Different-LZW* y AES.

5.13 Resumen de Capítulo

En este capítulo se hace hincapié en las diferentes formas en que fueron tratados los datos para obtener una compresión que resultara ser la mínima.

Se verifica el comportamiento de los algoritmos de cifrado y se muestran los resultados obtenidos aplicando éstos después del proceso de compresión.

Los algoritmos a utilizar en el esquema de compresión-cifrado para efectos de esta investigación son: *Divide2Different-LZW* como compresor, y *RC4* como algoritmo de cifrado. Lo anterior como resultado de lo obtenido, mostrado y analizado en este capítulo.

Aplicando *Divide2Different-LZW* se logra obtener una mejor razón de compresión con un pre procesamiento de la información. Se buscó que dicho pre procesamiento no afectara considerablemente el tiempo de ejecución ni la necesidad en la utilización de grandes recursos computaciones. Se recalca que con la utilización de este módulo se logra reducir, en un factor de 5.6, la cantidad de información ECG emitida por un paciente en un mes.

Entonces, llegado este punto, se conoce qué algoritmos de compresión y cifrado de datos se pueden utilizar en el manejo de datos tipo ECG. Tales algoritmos se eligieron a partir de los mejores resultados obtenidos y de su factibilidad para ser usados en diferentes ambientes. Además, es en este capítulo donde se aporta un módulo que maneja la información de tal suerte que ésta sea aprovechada de mejor forma por el algoritmo de compresión obteniendo mejores resultados en la razón de compresión.

En el siguiente capítulo se mencionan las conclusiones obtenidas a partir de los resultados logrados y de lo descrito en esta tesis.

Capítulo 6

Introducción

En este capítulo se revisan los objetivos de este trabajo y se dan las conclusiones a las que se llegó a partir de los resultados obtenidos. Se describe brevemente las aportaciones hechas con la realización de este proyecto y, se especifica trabajo a futuro a realizar.

6.1 Objetivos de tesis

Principalmente se tiene el objetivo de diseñar un esquema de compresión y cifrado de datos que pueda ser adoptado por aplicaciones en telemedicina utilizando principalmente datos de ECG. Este objetivo principal surge a partir del gran crecimiento en el área de telemedicina, con esto, también aumenta la cantidad de datos procesados y paralelamente, se vuelven más vulnerables cuando se utiliza en conjunción con tecnología inalámbrica.

Con este trabajo de tesis, se propone que algoritmos de compresión y cifrado trabajen en conjunto para poder funcionar en ciertas aplicaciones telemédicas.

Inherentemente se tiene el objetivo de reducir la cantidad de información mediante un algoritmo de compresión y añadir seguridad a los mismos, que en este caso, se manejará un esquema de cifrado de datos para garantizar su confidencialidad y la privacidad de las aplicaciones en telemedicina.

Ahora bien, el esquema que se pretende crear debe buscar la forma para generar una razón de compresión alta y casi comparable con aquellos algoritmos de compresión con pérdida de información realizando un compromiso entre capacidad de reducción de datos y velocidad para realizar dicho proceso. Además, este esquema tendrá como otra característica primordial, que pueda ser aplicado en la mayor cantidad de

ambientes posibles, es decir, que tenga contemplado el poder trabajar eficientemente incluso en ambientes con recursos limitados.

6.2 Objetivos alcanzados

Con la realización de este trabajo se aporta una solución, a lo que hasta ahora es sumamente demandante por telemedicina: reducción de información y seguridad de los mismos.

Se realiza el análisis de diversos algoritmos de compresión y de cifrado de datos, desde su funcionamiento hasta la evaluación de los resultados obtenidos. Para el caso de los algoritmos de compresión, se aporta que algoritmos son óptimos para un determinado tipo de datos, así como el mejor escenario en el que presentan su mejor desempeño, tomando en cuenta el factor de compresión obtenido y la velocidad de compresión. Lo anterior es sin dejar de lado el análisis de los recursos necesarios para poder obtener los mejores resultados.

Por otro lado, para los algoritmos de cifrado, se realiza la evaluación y desempeño de los mismos conforme al nivel de seguridad que éstos puedan brindar y la rapidez del proceso de cifrado y de descifrado. De igual forma, se abarcan detalles de los recursos necesarios para óptimos resultados.

Tanto para los algoritmos de compresión como de cifrado, se toma en cuenta la flexibilidad que éstos presentan para interactuar en situaciones donde la capacidad de procesamiento y/o recursos disponibles estén controlados, es decir, que tales algoritmos tengan la capacidad de poder operar en situaciones adversas con respecto a recursos y aún así, brindar resultados factibles para el esquema compresión-cifrado.

Se evalúa el desempeño de los algoritmos trabajando en forma individual con determinados datos, y con los mismos datos, también se evalúa el desempeño de los algoritmos trabajando en forma conjunta, que, para este caso, es la evaluación de un

algoritmo de compresión trabajando a la par con un algoritmo de cifrado. Este mismo tipo de evaluación es realizado para todas las combinaciones posibles de los algoritmos candidatos al esquema compresión-cifrado.

En este trabajo se crear un esquema de compresión y cifrado de datos para garantizar la reducción de éstos así como su confidencialidad. Tal esquema está creado para que pueda funcionar en ambientes con limitantes de recursos. Con lo anterior se logra que el nuevo esquema compresión-cifrado tenga mayor cobertura con respecto a los ambientes en los que pueda trabajar eficientemente. Así también, con esta propuesta, se logra una factibilidad para que dicho esquema funcione con diferentes tecnologías utilizadas para la transmisión, almacenamiento y procesamiento de la información.

Mencionando primero que la variedad de datos que pueden enviarse es amplia, este trabajo se enfoca a datos de ECG por ser uno de los tipos de datos de mayor interés y que presentan más generación de información para ser procesado en ambientes inalámbricos. Entonces, en este trabajo se crean una serie de procedimientos para alcanzar una razón de compresión más alta con respecto a la obtenida anteriormente, sacrificando ligeramente el tiempo consumido para llegar a los resultados deseados. Con la creación e incursión de estos procedimientos, se obtiene una reducción de información, que para el caso de datos tipo ECG, puede ser comparado con algoritmos de compresión *con pérdida* de información, alcanzando una alta reducción y haciendo una representación de los datos originales con sólo un quinto de la misma aproximadamente. Entonces, dada una cantidad X de datos ECG, éstos pueden ser representados con $X/5$.

En resumen, este esquema de compresión-cifrado viene a dar una solución para el área de telemedicina con el manejo de datos ECG. Este esquema puede ser adoptado por una gran diversidad de aplicaciones, ya que como se mencionó anteriormente, puede trabajar en diferentes ambientes tecnológicos.

6.3 Conclusiones

En los últimos años, el área de telemedicina ha tenido un gran crecimiento y con ello, nuevas soluciones son requeridas, principalmente en el manejo de datos ECG. La cantidad de datos manejados también ha crecido considerablemente. Consecuentemente, al manejar gran volumen de información, repercute directamente en la seguridad de éstos. Sin embargo, aún existe un rezago con respecto a la seguridad y reducción de la información en dicha área. Por tal motivo, es necesaria la implantación de un esquema que permita la reducción de información y que a su vez, adhiera seguridad a los datos transmitidos.

En la evaluación de los algoritmos de compresión, la mejor relación razón de compresión-velocidad es el aportado por LZW. Ahora, con respecto a los algoritmos de cifrado evaluados, AES y RC4 presentan resultados similares, sin embargo, se elije a RC4 ya que probándolo con datos de mayor tamaño, la velocidad de procesamiento es mucho más rápido que el presentado por AES. No obstante, hay que considerar que posiblemente AES sea más seguro, ya que, como se mencionó en el capítulo 2, AES es el estándar de cifrado ocupado actualmente en muchas aplicaciones.

Los algoritmos seleccionados, de acuerdo a los resultados obtenidos, para trabajar en conjunción son: LZW para la compresión y RC4 para el cifrado de los datos.

Para obtener una mayor razón de compresión, se crean una serie de procedimientos, determinando que *Divide2Different-LZW* es el que ofrece mejores resultados.

Entonces, por último, el esquema de compresión-cifrado estará conformado por el algoritmo *Divide2Different-LZW* para la compresión y *RC4* para el cifrado.

Como conclusión general, se puede afirmar que el nuevo esquema de compresión y cifrado de datos aplicado en ECG presentado en este trabajo de tesis, tendrá un

impacto tal que, se abrirán nuevas líneas de investigación con respecto a la reducción y seguridad de la información para dicha área.

6.4 Trabajo Futuro

Como un proyecto a realizar en un futuro se tiene la idea de un esquema automatizado y autoreconfigurable, que dependiendo del tipo de dato procesado, haga una elección de un algoritmo de compresión y de un algoritmo de cifrado, ya que no todos los datos requieren el mismo nivel de seguridad y/o velocidad de procesamiento.

A partir del algoritmo de compresión seleccionado en esta investigación, realizar una implementación que maximice el tiempo de búsqueda de una palabra en el diccionario utilizando *CAM*s (*Content-Addressable Memory*).

También, un trabajo a futuro es la implementación del esquema de compresión-cifrado presentado en este trabajo en una arquitectura hardware, siendo esta última un FPGA o un DSP, según convenga. Con lo anterior se pretende la implementación física en dispositivos móviles.

Otra tarea interesante de realizar, a partir de los dos puntos anteriores, sería la paralelización de los algoritmos de compresión y cifrado elegidos para el sistema autoreconfigurable en hardware. Donde, ciertamente, habría una mejora con respecto a la velocidad de procesamiento.

Algo sin duda a mejorar de este trabajo es la razón de compresión. Tratar de obtener mejores resultados sin que esto afecte considerablemente el desempeño del sistema. También, pudiera incorporarse, de ser posible, un sistema de detección y corrección de errores para tipos de datos específicos.

Finalmente, a partir de este trabajo de investigación, se pudiera crear un esquema

compresión-cifrado que aplicara para datos multimedia tales como video y audio.

Bibliografía

[1]. C. Pattichis, E. Kyriacou, S. Voskarides, M. Pattichis, R. Istepanian and C. Schizas, *"*Wireless Telemedicine Systems: An Overview*", IEEE Antenna's and Propagation Magazine*, Vol. 44, No. 2, Abril 2002.

[2]. D. Cypher, N. Chevrollier, N. Montavont, and N. Golmie, "Prevailing over Wires in Healthcare Environments: Benefits and Challenges", *IEE Communications Magazine*, Vol. 44, No. 4, Abril 2006.

[3]. U. Varshney and S. Sneha, "Patient Monitoring using Ad Hoc Wireless Network", *IEEE Communications Magazine,* Vol. 44, No. 4, April 2006.

[4]. Y. Xiao, .X. Shen, B. Sun and L. Cai, "Security and Privacy in RFID and Applications in Telemedicine", *IEEE Communications Magazine*, Vol. 44, No. 4, Abril 2006.

[5]. M. Cermack, "Monitoring and Telemedicine Support in Remote Environments and in Human Space Flight", *British Journal of Anaesthesia,* Mayo 2006.

[6]. Y. Choi, J. Krause, H. Seo, K. Capitan, J. Madison and K. Chung, "Telemedicine in the USA: Standarization through Information Management and Technical Applications", *IEEE Communications Magazine,* Vol. 44, No. 4, April 2006.

[7]. A. Tanenbaum, *Computer Networks*. 4th ed., Prentice Hall PTR, 2003.

[8]. D. Salomon, *Data Compression: The Complete Reference*, 3rd ed. Springer-Verlag, 2004.

[9]. M. Morales-Sandoval, C. Feregrino-Uribe, "Hardware Architecture for Elliptic Curve Cryptography and Lossless Data Compression", *15th International Conference on Electronics, Communications and Computers (CONIELECOMP'05)*, Vol. 00, February 2005.

[10]. K. Sayood, *Lossless Compression Handbook,* 1st ed., CA: Academic Press, 2002.

[11]. Witten, A. Moffat and T. Bell, *Managing Gigabytes: Compressing and Indexing Documents and Images*. 2nd ed., CA: Academic Press, 1999.

[12]. S. W. Smith, *Digital Signal Processing: A Practical Guide for Engineers and Scientists,* Newnes; Book and CD ROM ed., Septiembre 2002.

[13]. P. Howard and J. Vitter, "Analysis of Arithmetic Coding for Data Compression", *Data Compression Conference*, Abril 1991.

[14]. J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression",
IEEE Transactions on Information Theory, Vol. 23, Mayo 1977.

[15]. T. Welch, "A Technique for High-Performance Data Compression", *IEEE Computer,* Vol. 17, No. 6, Junio 1984.

[16]. J. Ziv and A. Lempel, "Compression of Individual Sequences via Variable-Rate
Coding," *IEEE Transactions on Information Theory*, Vol. 24, 1978.

[17]. A. Menezes, P. van Oorschot and S. Vanstone, *Handbook of Applied Cryptography,* 1st ed., 1996.

[18]. W. Stallings, *Cryptography and Network Security*. 4th ed., NJ: Prentice Hall, 2005.

[19]. R. Rivest, "The RC4 Encryption Algorithm", *RSA Data Security* , Inc, Marzo 1992.

[20]. National Bureau of Standards (U.S.), "Data encryption standard (DES)", *Federal*
Information Processing Standards Publication 46, National Technical Information
Service, Springfield, VA, Abril 1977.

[21]. B. Schneier, *Applied Cryptography*. 2nd ed., NY: John Wiley & Sons, 1996

[22]. National Institute of Standards and Technology (NIST), "Advanced Encryption Standard (AES)", *Federal Information Processing Standards Publications 197 (FIPS197),* Noviembre 2001.

[23]. R. Chandramouli, S. Bapatla, K. Subbalakshmi and R. UMA, "Battery Power-Aware Encryption", *ACM Transactions on Information and System Security*, Vol. 9, No. 2, Mayo 2006.

[24]. N. Potlapally, S. Ravi, A. Raghunathan and N. Jha, "Analyzing the Energy Consumption of Security Protocols", *International Symposium: Low Power Electronics and Design (ISLPED'03)*, Agosto 2003.

[25]. R. Venugopalan, P. Ganesan, P. Peddabachagari, A. Dean, F. Mueller, M. Sichitiu, "Encryption Overhead in Embedded Systems and Sensor Networks Nodes: Modeling and Analysis", *International Conference on Compilers, Architecture and Synthesis for Embedded Systems,* Noviembre 2003.

[26]. D. Caramella, "Teleradiology: State of the Art in Clinical Environment", *European Journal of Radiology*, Vol. 22, No. 3, Junio 1996.

[27]. Bae and S. Olariu, "Design and Evaluation of a FuzzyLogic-Based Location Management Method. For Wireless Mobile Networks", *4th International Workshop of Mobile Computing,* Junio 2003.

[28]. J. Udupa, P. Saha, "Fuzzy Connecteness and Image Segmentation", *IEEE,* Vol. 91, No. 10, Octubre 2003.

[29]. W. Scott, D. Bluemke, W. Mysko, D. Weller, G. Kelen, R. Reichle, J. Weller, J. Gitlin, "Interpretation of Emergency Department Radiographs by Radiologists and Emergency Medicine Physicians: Teleradiology Workstation versus Radiograph Readings", *Radiology,* 1995.

[30]. C. Heneghan, A. Sclafani, J. Stern, J. Ginsburg, "Telemedicine Applications in Otolaryngology", *IEEE Engineering in Medicine and Biology Magazine,* Vol. 18, No. 4, Julio 1999.

[31]. F. Hu, C. Xiaojun, “Security in Wireless Actor & Sensor networks (WASN): Towards a Hierarchical Re-keying Design”, *International Conference on Information Technology: Coding and Computing ITCC’05*, Vol. 2, Abril 2005.

[32]. J. Quero, M. Elena, J. Segovia, C. Tarrida, J. Santana, C. Santana, “CardioSmart: Cardiological Monitoring Intelligent system using GPRS”, *IEEE Latin America Transactions,* Vol. 3, No. 2, Abril 2005.

[33]. S. Garawi, R. Istepanian, M. Abu-Rgheff, “3G Wireless Communications for Mobile Robotic Tele-ultrasonography Systems”, *IEEE Communications Magazine,* Vol. 44, No. 4, Abril 2006.

[34]. M. Ansari and R. Anand, “Analysis of Medical Image Compression Techniques and their Performance Evaluation for Telemedicine”, *International Conference on Cognition and Recognition,* 2003.

[35]. C. Lamberti, P. Coccia, “ECG Data Compression for Ambulatory Device”, *Computers in Cardiology*, September 1988.

[36]. M. Bagues, J. Bermudez, A. Burgos, A. Goni, A. Illarramendi, J. Rodriguez and A. Tablado, “An Innovative System that Runs on a PDA for a Continuous Monitoring of People”, *19th IEEE Symposium on Computer-Based Medical Systems,* Junio 2006.

[37]. T. Sakamoto, H. Wang and D. Wei, "Performance Evaluation of Twelve-Lead Electrocardiogram Telemonitor Utilizing 3G Cellular Phone", *Sixth IEEE international Conference on Computer and information Technology,* Vol. 00, Septiembre 2006.

[38]. S. Olariu, K. Maly, E. Foudriat and S. Yamany, “Wireless Support for Telemedicine in disaster Management, *Tenth international Conference on Parallel and Distributed Systems,* Volume 00, Julio 2004.

[39]. Olmos Salvador, P. Laguna, P. Caminal, R. Jané, “Compresión de datos de la señal ECG mediante transformadas ortogonales”, *XIII Reunión Anual de la Agrupación Española de Bioingeniería*, Octubre 1995.

[40]. R. Nygaard, G. Melnikov, A. Katsaggelos, “Rate distortion optimal ECG signal compression”, *International Conference on Image Processing,* Vol. 2, Octubre 1999.

[41]. D. DiPersio and R. Barr, “Evaluation of the FAN Method of Adaptive Sampling on Human Electrocardiograms”, *Medical and Biological Engineering and Computing,* Vol. 23, No. 5, Septiembre 1985.

[42]. J. Cox, F. Nolle, H. Fozzard, and G. Oliver Jr., "AZTEC: a Preprocessing Program For real-time ECG Rhythm Analysis," *IEEE Transactions on Biomedical Engineering*, Vol. 15, No. 2, 1968.

[43]. A. Bilgin, M. Marcellin and M. Altbach, “Wavelet Compression of ECG Signals by JPEG2000”, *IEEE Conference on Data Compression* , Marzo 2004.

[44]. M. Negoita, L. Goras "On a compression algorithm for ECG signals", *13th European signal processing conference*, Septiembre 2005.

[45]. M. Blanco-Velasco, F. Cruz-Roldan, J. Godino-Llorente and K. Barner, “Efficient ECG Compression Based on M-Channel Maximally Decimated Filter Banks”, *13th European Signal Processing Conference,* 2005.

[46]. S. Pavlopoulos, R. Istepanian, E. Kyriacou, D. Koutsouris, “Optimal Wavelet Biosignal Compression for Mobile multi-purposetelemedicine”, *IEEE EMBS International Conference on Information Technology Applications in Biomedicine,* 2000.

[47]. A. Alshamali, “A Mobile Telecardiology System Using TETRA Standards”, *International Conference on Information Technology: Computers and Communications*, Abril 2003.

[48]. C. Zywietz, V. Mertins, D. Assanelli, C. Malossi, “Digital ECG Transmission from Ambulance Cars with Application of the European Standard Communications Protocol SCP-ECG”, *Computers in Cardiology,* Septiembre 1994.

[49]. CEN TC251, ENV 1064:1994: *Standard Communication Protocol for Computer Assisted Electrocardiography (SCP-ECG)*. http://www.tc251wgiv.nhs.uk/pages/pdf/censcp019.pdf (revisado Septiembre 27, 2007).

[50]. O. Salvador., D. Echeverria, R. Jané y P. Laguna, "Compresión de señales ECG multiderivacionales mediante expansiones ortogonales optimas: WP y KL", *Comunicaciones del XV Congreso anual de la Sociedad Española de Ingeniería Biomédica*, 1997.

[51]. B. Bradie, B., "Wavelet Packet-Based Compression of Single Lead ECG", *IEEE Transactions on Biomedical Engineering*, Vol. 43, No. 5, Mayo 1996.

[52]. J. Sánchez-Céspedes, G. Bernal-Ruiz, "Compresión de la Señal Electrocardiográfica (E.C.G)", *Umbral Científico, 2004.*

[53]. S. Vasudevan, "Secure telemedicine system for home health care", Morgantown, W. Va: [West Virginia University Libraries], 2000.

[54]. Y. Chu and A. Ganz*, "*Mobile Telemedicine System Using 3G Wireless Networks", <http://www.touchbriefings.com/pdf/1251/ACF7326.pdf>. Septiembre 2006, (revisado Septiembre 2007).

[55]. Medtronic, Philips Medical Systems, Welch Allyn Medical Products & ZOLL Medical Corp, "An exclusive supplement to JEMS", *Journal of Emergency Medical Services,* http://www.emstelemedicine.com/images/July06-ECGSupplement.pdf (revisado Septiembre 2007).

[56]. L. Makris, N. Argiriou, and M. Strintzis, "Network Access and Data Security Design for Telemedicine Applications", *2nd IEEE Symposium on Computers and Communications,* Julio 1997.

[57]. Papadakis V. Chrissikopoulos, D. Polemi, "A Secure web-based Medical Digital Library Architecture Based on TTPs", *16th International Conference of Medical Infobahn in Europe – XVI MIE*, 2000.

[58]. C. Sima, R. Raman, R. Reddy, W. Hunt and S. Reddy, "Vital signs Services for Secure Telemedicine Applications", *AMIA Symp,*1998.

[59]. R. Reddy, R. Raman1, S. Reddy, K. Cleetus, W. Hunt, I. Lapshin and W. Beam, “Secure Collaborative Telemedicine in Rural West Virginia”, *NLM Symposium on Telemedicine and Telecommunications: Options for the New Century*, Marzo 2001.

[60]. C. Poon, Z. Yuan-Ting and B. Shu-Di, “A Novel Biometrics Method to Secure Wireless Body Area Sensor Networks for Telemedicine and M-health”, *IEEE* Communications Magazine, Vol. 44, No. 4, Abril 2006.

[61]. U. Varshney, S. Sneha, "Patient Monitoring using Ad Hoc Wireless Networks: Reliability and Power Management", *IEEE Communications Magazine,* Vol.44,No.4, Abril 2006.

[62]. X. Yang, S. Xuemin, B. Sun, C. Lin, "Security and Privacy in RFID and Applications in Telemedicine", *IEEE Communications Magazine,* Vol.44, No.4, Abril 2006.

[63]. D. Bottazzi, A. Corradi, R. Montanari, "Context-aware Middleware Solutions for anytime and anywhere Emergency Assistance to Elderly People", *IEEE Communications Magazine,* Vol.44, No.4, Abril 2006.

[64]. S. Bao, Y. Zhang and L. Shen, “A New Symmetric Cryptosystem of Body Area Sensor Networks for Telemedicine”, *Conference the Japan Society of Medical Electronics & Biological Engineering*, Vol. 44, Julio 2005.

[65]. E. Blaß, M. Zitterbart, “An Efficient Key Establishment Scheme for Secure Aggregating Sensor Networks”, *ACM Symposium on information, Computer and Communications Security*, Marzo 2006.

[66]. F. Hu, J. Tillett, J. Ziobro and N. K. Sharma, “Secure Tree-Zone-Based Wireless Sensor Networks for Telemedicine Applications”, *IEEE GLOBECOM*, 2003.

[67]. S. Basagni, K. Herrin, E. Rosti and D. Bruschi, “Secure Pebblenets”, *2nd ACM international Symposium on Mobile Ad Hoc Networking &Amp, Computing,* Octubre2001.

[68]. R. Duncan, M. Shabot, “Secure Remote Access to a Clinical Data Repository Using a Wireless Personal Digital Assistant (PDA)”, *AMIA Symp,* Noviembre 2000.

[69]. J. Becker, D. Gebauer, L. Maier-Hein, M. Schwaibold, J. Schöchlin, A. Bolz, “The Wireless Monitoring of Vital Parameters: A Design Study”, *Biomed Tech* , 2002.

[70]. K. Lorincz, D. Malan, T. Fulford-Jones, A. Nawoj, A. Clavel, V. Shnayder, G. Mainland, and M. Welsh, “Sensor Networks for Emergency Response: Challenges and Opportunities", *IEEE Pervasive Computing,* Vol.3, No.4, Octubre 2004.

[71]. F. Hu, Y. Wang and H. Wu, “Mobile Telemedicine Sensor Networks with Low-energy Data Query and Network Lifetime Considerations”, *IEEE Transactions on Mobile Computing*, Vol. 5, No. 4, Abril 2006.

[72]. G. Mendoza, B.Tran, "In-home Wireless Monitoring of Physiological Data for Heart Failure Patients", *24th Annual Conference and the Annual Fall Meeting of the Biomedical Engineering Society,* Vol.3, Octubre 2002.

[73]. T. Gao, D. Greenspan, M. Welsh, R. Juang, A. Alm, "Vital Signs Monitoring and Patient Tracking Over a Wireless Network,"*27th Annual International Conference of the Engineering in Medicine and Biology Society,* Septiembre 2005.

[74]. B. Lo, S. Thiemjarus, R. King, and G. Yang, “Body Sensor Network - A Wireless Sensor Platform for Pervasive Healthcare Monitoring”, *3rd International Conference on Pervasive Computing*, Mayo 2005.

[75]. T. Fulford-Jones, W. Gu-Yeon, M. Welsh, "A Portable Low-power Wireless two-lead EKG System", *26th Annual International Conference of the IEEE* , Vol.1, Septiembre 2004.

[76]. R. Fensli, E. Gunnarson, T. Gundersen, “A Wearable ECG-Recording System for Continuous Arrhythmia Monitoring in a Wireless Tele-Home-Care Situation”, *18th IEEE Symposium on Computer-Based Medical Systems,* Junio 2005.

[77]. J. Ottenbacher, S. Römer, C. Kunze, U. Großmann and Wilhelm Stork, “Integration of a Bluetooth Based ECG System into Clothing”, *Eighth international Symposium on Wearable Computers,* Volume 00, Octubre 2004.

[78]. R. Fenslia, E. Gunnarsonb, O. Hejlesen, “A wireless ECG System for Continuous Event Recording and Communication to a Clinical Alarm Station", *IEEE 26th Annual International Conference of the Engineering in Medicine and Biology Society*, Vol.1, Septiembre 2004.

[79]. D. Cruz, E. Barros, "Vital Signs Remote Management System for PDAs", *8th Euromicro Conference on Digital System Design*, Septiembre 2005.

[80]. J. Rodriguez, A. Goni, A. Illarramendi, "Real-time classification of ECGs on a PDA," *IEEE Transactions on Information Technology in Biomedicine,* Vol.9, No.1, Marzo 2005.

[81]. W. Tohme, S. Olsson, "Developments and Directions of Technology Assessment in Telemedicine", *Fourth International Conference on Image Management and Communications*, Agosto1995.

[82]. A. Booksieinl, S. Klein, T. Raita, “Is Huffman coding dead?”, *16th Annual international ACM SIGIR Conference on Research and Development in information Retrieval,* Julio 1993.

[83]. D. Lelewer and D. Hirschberg, “Data compression”, *ACM Comput. Surv.* Septiembre 1987.

[84]. J. Vitter, “Design and Analysis of Dynamic Huffman Codes”, *Journal ACM* , Vol 34, No. 4, Octubre 1987.

[85]. D. Knuth, “Dynamic Huffman Coding”, *Journal Algorithm,* Vol. 6, No. 2, Junio1985.

[86]. A. Moffat and J. Katajainen, “In-Place Calculation of Minimum-Redundancy Codes”, *4th international Workshop on Algorithms and Data Structures,* Vol. 955, Agosto1995.

[87]. J. Vitter, "Dynamic Huffman Coding,"*ACM Trans. Math. Software*, Vol. 673, Junio 1989.

[88]. D. Jones, “Application of Splay Trees to Data Compression”, *Commun. ACM* , Vol. 31, No. 8, Agosto 1988.

[89]. N. Faller, "An Adaptive System for Data Compression," *Record of the 7th Asilomar Conference on Circuits, Systems and Computers,* 1973.

[90]. R. Gallager, "Variations on a theme by Huffman”, *IEEE Transactions on Information Theory,* Vol. 24, No. 6, Noviembre 1978.

[91]. D. McIntyre and M. Pechura, “Data Compression Using Static Huffman Code-Decode Tables”, *Commun. ACM*, Vol. 28, No. 6, Junio 1985.

[92]. B. Varn. “Optimal Variable Length Codes (Arbitrary Symbol Cost and Equal Code Word Probability)”. *Information Control*, Junio 1971.

[93]. P. Howard and J. Vitter, "Arithmetic Coding for Data Compression", *IEEE*, Vol. 82, No. 6, Junio 1994.

[94]. A. Moffat, R. Neal, I. Witten, "Arithmetic coding revisited," *Data Compression Conference,* Marzo 1995.

[95]. T. Bell, J. Cleary and I. Witten, *Text Compression Book,* Prentice Hall, 1990.

[96]. P. Howard and J. Vitter, “Practical Implementations of Arithmetic Coding”, *Image and Text Compression*, 1992.

[97]. A. Said, “Comparative Analysis of Arithmetic Coding Computational Complexity”, *Hewlett-Packard Laboratories Report*, 2004.

[98]. Witten, R. Neal and J.Cleary, “Arithmetic Coding for Data Compression”, *Commun. ACM,* Vol. 30, No. 6, Junio 1987.

[99]. G. Held, T. Marshall, *Data and Image Compression: Tools and Techniques*, 4th ed., John Wiley & Sons, 1996.

[100]. Michael Dipperstein, “Lempel Ziv Welch (LZW) Encoding Discussion and Implementation”, http://michael.dipperstein.com/lzw/index.html (revisado Septiembre 2007).

[101]. A. Hodjat and I. Verbauwhede, “The Energy Cost of Secrets in Ad-Hoc Networks”, *IEEE 5th Workshop on Wireless Communications and Networking,* 2002.

[102]. D. Carman, P. Kruus and B. Matt, “Constraints and Approaches for Distributed Sensor Network Security”, Tech. Rep. 00-010, *NAI Labs*, Septiembre 2000.

[103]. J. Großschädl, S. Tillich, C. Rechberger, M. Hofmann and M. Medwed, “Energy Evaluation of Software Implementations of Block Ciphers Under Memory Constraints”, *10th Conference on Design, Automation and Test in Europe,* Abril 2007.

[104]. K. Ssenyonjo, 2004. http://www.cs.bath.ac.uk/~amb/CM30076/ projects.bho /2004-5/Ssenyonjo_Kafuuma-2004-5.pdf (revisado Agosto 2007).

[105]. K. Aoki and H. Lipmaa, “Fast implementations of the AES candidates”, *3rd AES Conference*, Abril 2000.

[106]. H. Qin, T. Sasao and Y. Iguchi, “An FPGA Design of AES Encryption Circuit with 128-bit Keys”, *15th ACM Great Lakes Symposium on VLSI,* Abril 2005.

[107]. *Canterbury Corpus*, http://corpus.canterbury.ac.nz/descriptions/#cantrbry (revisado Septiembre 2007).

[108]. M. Robshaw, "Stream Ciphers," Technical Report TR701, Ver. 2.0, *RSA Laboratories*, 1995. http://citeseer.ist.psu.edu/article/robshaw95stream.html (revisado Septiembre 2007).

[109]. N. Sklavos, G. Selimis and O. Koufopavlou, “FPGA Implementation Cost and Performance Evaluation of IEEE 802.11 Protocol Encryption Security Schemes”, *Journal of Physics: Conference Series,* Vol. 10, 2005.

[110]. H. Kim, J. Han, S. Cho, “An Efficient Implementation of RC4 Cipher for Encrypting Multimedia Files on Mobile Devices”, *ACM Symposium on Applied Computing,* Marzo 2007.

[111]. E. Dawson, H. Gustafson, M. Henricksen, B. Millan, “Evaluation of RC4 Stream Cipher”, *Information Security Research Centre,* Julio 2002.

[112]. *ANSI/AAMI EC13 Test Corpus,* http://www.physionet.org/physiobank/database/aami-ec13/ (revisado Agosto 2007).

[113]. R. Horspool and W. Windels, “An Approach to ECG Compression”, *IEEE Seventh Symposium on Computer-Based Medical Systems*, Junio 1994.

[114]. *Botan BSD-licensed library*, http://botan.randombit.net/bmarks.html (revisado Agosto 2007.)

[115]. David A. Huffman, “A Method for the Construction of Minimum-Redundancy Codes”, *IEEE Proccedings of the IRE*,Vol. 40, Septiembre 1952.