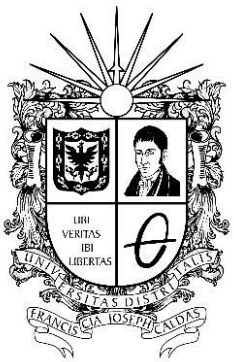
DISEÑO E IMPLEMENTACIÓN DE UN ALGORITMO DE CIFRADO HÍBRIDO PARA SISTEMAS DE COMUNICACIÓN CRÍTICA



UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Autor
JULIAN ERNESTO URREGO PIÑEROS
JORGE ALEXANDER ALZA RAMIREZ

UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS
Facultad Tecnológica
Ingeniería en Telecomunicaciones
Bogotá D.C. Septiembre, 2022

DISEÑO E IMPLEMENTACIÓN DE UN ALGORITMO DE CIFRADO HÍBRIDO PARA SISTEMAS DE COMUNICACIÓN CRÍTICA



UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Autor
JULIAN ERNESTO URREGO PIÑEROS 20151373221
jeurregop@correo.udistrital.edu.co
JORGE ALEXANDER ALZA RAMIREZ 20182373015
jaalzar@correo.udistrital.edu.co

Modalidad
Monografía

Presentado para optar al título de: Ingeniero en Telecomunicaciones

Director
Ing. Ernesto Cadena Muñoz, PhD

UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS
Facultad Tecnológica
Ingeniería en Telecomunicaciones
Bogotá D.C. Septiembre, 2022

AGRADECIMIENTOS

Le agradecemos principalmente a Dios, por permitirnos la fuerza, la sabiduría, el entendimiento y la inteligencia para poder afrontar los retos de la vida, del entorno académico y profesional, a nuestros Padres, Madres, Amigos por ese apoyo incondicional que nos han dado desde el día que iniciamos con este sueño y proyecto de vida, por ser esa parte de nuestras vidas que no nos ha dejado desfallecer en la consecución del mayor de logro de nuestras vidas, a La Universidad Distrital Francisco José de Caldas, al cuerpo de docentes quienes nos han formado como profesionales y ciudadanos íntegros, responsables, a los compañeros de la universidad que han compartido nuestro recorrido de aprendizaje durante el cual nos permitió aprender también de su experiencias vividas, al Ingeniero Ernesto Cadena Muñoz el cual nos ha guiado en el desarrollo de la propuesta de grado como Tutor, y en especial mención a Diana Marcela Alza Ramírez y María Camila Díaz Vega, por sus aportes tan relevantes, auténticos y claves para no permitirnos desfallecer en la ejecución de este trabajo de grado.

Para todos ellos y ellas nuestro más sincero agradecimiento y dedicatoria de este trabajo de grado.

Julian Ernesto Urrego Piñeros
Jorge Alexander Alza Ramírez

Tabla de Contenido

Índice de Figuras 7
Índice de Tablas 10
Glosario 11
Lista de Abreviaturas y Siglas 13
ABSTRACT 15
1. INTRODUCCION 16
2. FORMULACIÓN DEL PROBLEMA 18
3. JUSTIFICACIÓN 19
4. OBJETIVOS 20
4.1. General 20
4.2 Específicos 20
5. MARCO DE REFERENCIA 21
5.1 Antecedentes 21
5.1.1 Antecedentes modelo de cifrado simétrico basado en atractores caóticos 21
5.1.2 Modelo híbrido de cifrado y descifrado para imágenes usando rsa, aes y afinamiento con la operación XOR. 21
5.1.3 Cifrado de datos segura por medio de combinación de AES, RSA Y HMAC 21
5.1.4 Demostración USRP N210 de detección de banda ancha y clasificación de modulación jerárquica ciega 22
5.1.5 Análisis de fugas del transmisor cuando se opera USRP (N210) en modo dúplex 22
5.1.6 Mejora de la privacidad en MANET basado en SDN mediante el uso de un algoritmo de cifrado y descifrado híbrido 22
5.1.7 Modelo de cifrado híbrido basado en AES, RSA y TWOFISH para el cifrado Bluetooth 23
5.1.8 Cifrado de voz mediante múltiples mapas caóticos con rápida transformación de Fourier 24
5.1.9 Un nuevo algoritmo de cifrado de voz dinámico basado en el mapa caótico de Lorenz sobre el protocolo de internet 24
5.1.10 Métricas de rendimiento efectivas para sistemas de comunicación multimedia de misión crítica 25
5.1.11 Estudio y análisis para la selección de una tecnología a implementar en la actualización de la red PPDR de la policía metropolitana de Bogotá “mebog” 25
5.1.12 Un flujo caótico simple con función sinusoidal hiperbólica y su aplicación al cifrado de voz 26

5.1.13 Cifrado tire Pressure Monitoring System para mejorar la seguridad vehicular 26
5.1.14 Analizador de espectro basado en SDR basado en GNU Radio de código abierto 26
5.1.15 Implementación de estación base GSM recepción de señales LTE aplicando radio definido por software 27
5.2 Marco Teórico 28
5.2.1 Cifrado 28
5.2.1.1 Cifrado de los Datos 28
5.2.1.2 Clave Digital 28
5.2.1.3 Cifrado Simétrico 29
5.2.1.3.1 Data Encryption Standard (DES) 29
5.2.1.3.2 Advanced Encryption Standard (AES) 31
5.2.1.3.3 AES y sus modos de cifrado 33
5.2.1.4 Cifrado Asimétrico 34
5.2.2 Python 34
5.2.2.1 Lenguaje interpretado o de script 35
5.2.2.2 Tipado Dinámico 35
5.2.2.3 Fuertemente Tipado 35
5.2.2.4 Multiplataforma 35
5.2.2.5 Orientado a Objetos 35
5.2.2.6 ¿Por Qué Python? 36
5.2.2.7 Librería PYCRYPTODOME 36
5.2.3 Radio Definida por Software (SDR) 37
5.2.3.1 Procesamiento Digital de Señal (DSP) 37
5.2.3.2 Características de la Radio Definida por Software (SDR) 37
5.2.3.3 Niveles de SDR 38
5.2.4 ASK (Codificación por Desplazamiento de Amplitud) 39
5.2.4.1 Modulador ASK 40
5.2.4.2 Métodos de Demodulación 40
5.2.4.3 Método de Demodulación: Detección Síncrona 40
5.2.5 USRP N210 41
5.2.6 Sistemas de Comunicaciones para Misión Crítica 41
5.2.6.1 ¿Qué es Misión Crítica? 42
5.2.6.2 ¿CÓMO SE IMPLEMENTA UN SISTEMA DE ESTE TIPO? 42

5.2.7 Métricas de Rendimiento de la Red 42
5.2.7.1 Tasa Media de Éxito (THROUGHPUT) 43
5.2.7.2 Retardo Promedio (DELAY) 43
5.3. MARCO LEGAL 44
6. METODOLOGIA 47
6.1 Scripts de Trabajo 52
6.2 Autenticación de Llave Pública RSA y Cifrado de Llave de Sesión AES 52
6.3 Cifrado de Texto Plano con Algoritmo de Cifrado Simétrico AES 57
6.4 Capa de Comunicación – GNU Radio 62
6.4.1 Transmisor 62
6.4.2 Receptor 64
6.5 Técnica de Relleno 66
6.5.1 Técnica de Recorte 69
7. RESULTADOS 70
7.1 Cálculo del Delay 70
7.2 Cálculo del Throughput 77
7.3 Cálculo del Jitter 83
7.4. Análisis de Resultados 85
8. CONCLUSIONES Y RECOMENDACIONES 85
9. BIBLIOGRAFIA 87
10. ANEXOS 90

Índice de Figuras

Fig 1. Esquema comparativo entre cifrado no homomórfico-cifrado. [18]……..23
Fig 2. atractor de Lorenz. [22]……..25
Fig 3. Estructura General de Feistel en DES. [28]……31
Fig 4. Estructura General AES. [28]…….33
Fig 5. Señales de la Modulación ASK.[37]……..39
Fig 6. Modulador ASK. [37]…….40
Fig 7. Demodulador ASK detección sincrónica. [37]……40
Fig 8. Comparación de una señal con Jitter y una señal sin Jitter. [42]…….....44
Fig 9. Interfaz de usuario donde se carga el archivo de texto en claro y contraseña para desencriptar la llave pública RSA. [autoría propia]……48
Fig 10. Esquema de cifrado asimétrico de llave de sesión AES. [autoría propia]……48
Fig 11. Esquema de cifrado simétrico de texto en claro. [autoría propia]……49
Fig 12. Metodología de trabajo. [autoría propia]……..50
Fig 13. Máquinas usadas para el proyecto. [autoría propia]…….51
Fig 14. Script gen_rsa_keys.py. [autoría propia]……..54
Fig 15. Interfaz gráfica para la autenticación. [autoría propia]…….54
Fig 16. Script Des_Public_Key.py. [autoría propia]…….55
Fig 17. Script Encryption.py**.** [autoría propia]……..56
Fig 18. Script Desencryption.py. [autoría propia]……... 57
Fig 19. Interfaz de usuario solicitando la ubicación del archivo a encriptar. [autoría propia]……...58
Fig 20. Script Encryption.py. [autoría propia]…….59
Fig 21. Script desencryption.py. [autoría propia]……60
Fig 22. Reproducción de archivo .JPG, .JPEG, o .PNG. [autoría propia]……61
Fig 23. reproducción de archivo .mp3 o .wav. [autoría propia]……...62
Fig 24. Transmisor GNU Radio. [autoría propia]…….63
Fig 25. Señal modulada ASK. [autoría propia]…….64

Fig 26. Receptor GNU radio. [autoría propia]……..65
Fig 27. Señal recibida ASK. [autoría propia]……66
Fig 28. Señal cuadrada recuperada. [autoría propia]……66
Fig 29. Diagrama flujo de la capa de comunicación. [autoría propia]……..67
Fig 30. Filler.py. Script de relleno. [autoría propia]……68
Fig 31. Contenido final del archivo encriptado sin rellenar. [autoría propia]……68
Fig 32. Contenido final del archivo. [autoría propia]……69
Fig 33. Script slicer.py. Técnica de recorte. [autoría propia]……70
Fig 34. Preparación del filtro dirección IP de origen. [autoría propia]……71
Fig 35. Preparación del filtro dirección IP de destino. [autoría propia]……72
Fig 36. Filtro aplicado en el transmisor. [autoría propia]……72
Fig 37. Toma del tiempo en el primer paquete del transmisor. [autoría propia]……73
Fig 38. Toma del tiempo en el último paquete del receptor. [autoría propia]……73
Fig 39. Cálculo del Delay. [autoría propia]……..74
Fig 40. Delay promedio de cada una de las muestras. [autoría propia]……74
Fig 41. Script Master-TX.py. [autoría propia]……..75
Fig 42. Tiempo de Ejecución de algoritmo de cifrado en el Transmisor y Receptor. [autoría propia]……...76
Fig 43. Delay + Tiempo de ejecución. [autoría propia]……76
Fig 44. Menú de Toma de datos de Delay y Throughput. [autoría propia]……..77
Fig 45. Toma del Throughput en Wireshark, en el apartado Average bytes/s ó Average bits/s. [autoría propia]……..78
Fig 46. Toma del Throughput por muestra. [autoría propia]……78
Fig 47. Throughput Promedio Transmisor. [Autoria propia]……79
Fig 48. Throughput Promedio Receptor. [autoría propia]……79
Fig 49. Throughput en el receptor en relación con las 10 muestras tomadas. [autoría propia]……...80
Fig 50. Throughput en el receptor en relación con las 10 muestras tomadas. [autoría propia]……...81
Fig 51. Throughput vs Delay en la recepción. [autoría propia]……....81

Fig 52. Comparación del Throughput. [Autoria propia]……82
Fig 53. Jitter en el Transmisor. [autoría propia]……..83
Fig 54. Jitter en el Receptor. [autoría propia]……84

Índice de Tablas

Tabla 1. Funcionalidad librería PYCRYPTODOME. [32]……...36
Tabla 2. Organización de los scripts de envío de llave pública encriptada rsa. [autoría propia]……...52
Tabla 3. Organización de los scripts desencriptación de llave pública y posterior envío de la información cifrada de manera híbrida. [autoría propia]……..52
Tabla 4. Primera ejecución simultánea. [autoría propia]……..53
Tabla 5. Segunda ejecución simultánea. [autoría propia]…….53
Tabla 6. Ejecución simultánea proceso de cifrado simétrico aes. [autoría propia]……...57
Tabla 7. Muestras usadas para la transmisión. [autoría propia]……70
Tabla 8. Direcciones ip de los dispositivos usados. [autoría propia]……71

Glosario

Concepto	Explicación Breve y Detallada del Concepto
Llave Pública	Es un archivo con información usada en métodos criptográficos conocidos como certificados, la llave publica es la parte de información que puede ser publicada y usada para cifrar.
Llave Privada	La llave privada está protegida por contraseña en un archivo de base de datos de manera que sólo el propietario pueda acceder a ésta para descifrar mensajes cifrados utilizando la clave pública correspondiente.
Criptografía	es un método de protección de la información y las comunicaciones mediante el uso de códigos, de modo que solo aquellos a quienes está destinada la información puedan leerla y procesarla. El prefijo «cripta-» significa «oculto» o «bóveda» – y el sufijo «-grafía» significa «escritura»
Encriptación	La encriptación es un método de codificación de datos (mensajes o archivos) de modo que solo las partes autorizadas puedan leer la información o acceder a ella. La encriptación utiliza algoritmos complejos para codificar la información que se envía.
Cifrado	El cifrado es un método habitual en la criptografía (la técnica que consiste en escribir mensajes en clave), es una codificación del contenido del mensaje, protegiéndolo.
DES	Algoritmo de cifrado basado en un secreto compartido (clave), el cual cifra el texto en bloques de 64 bits, Utiliza claves de 56 bits.
AES	Algoritmo de cifrado basado en un secreto compartido (clave), el cual cifra el texto en bloques de 128 bits y Utiliza claves de 128, 192 o 256 bits.
RSA	Algoritmo de cifrado de llave público-privada, que se basa en la dificultad de calcular números primos de gran magnitud.
Codificación	Consiste en traducir los valores obtenidos en el proceso de cuantificación (ya son valores digitales) al sistema binario, mediante la utilización de una serie de códigos preestablecidos.

Modulación	La modulación es la transmisión de información cambiando las propiedades de una forma de onda (señal portadora) con una señal moduladora que contiene la información.
Throughput	Es la velocidad con la que la información pasa de un lugar a otro
Delay	Es el retardo que se produce en un sistema de comunicación desde el origen hasta el destino
Jitter	Es la variación del retardo en el tiempo. Esto afecta significativamente las comunicaciones, ya que un paquete posterior puede llegar antes que un paquete anterior.

Lista de Abreviaturas y Siglas

Sigla/Abreviatura	Significado
DES	Estándar de Cifrado de Datos
AES	Estándar de Cifrado de Avanzado
RSA	Estándar de Cifrado de Llave Pública y Privada
FM	Modulación por Frecuencia
ASK	Codificación por Desplazamiento de Amplitud
FSK	Codificación por Desplazamiento de Frecuencia
PSK	Codificación por Desplazamiento de Fase
QAM	Codificación por Desplazamiento de Amplitud y Cuadratura
QoS	Calidad de Servicio
ACK	Acuse de Recibo

RESUMEN

En el presente documento se evidencia el desarrollo del trabajo de grado el cual consta de la ejecución de un algoritmo de cifrado híbrido para sistemas de comunicación de misión crítica a través de radio definido por software.

El algoritmo se desarrolla mediante el lenguaje de programación Python el cual permite realizar una autenticación asimétrica RSA. Se realiza el envío de la llave RSA (pública y privada) cifrada por una contraseña del receptor al transmisor, el transmisor desbloquea la llave publica con la cual se cifra mediante RSA la llave de sesión AES, la cual cifra el texto plano, luego se cifra el texto plano a través del algoritmo AES (cifrado simétrico). En el receptor se descifra la llave de sesión mediante la llave privada RSA, con la cual se usa a su vez para descifrar el texto plano.

Para este trabajo se toman muestras con archivos de imagen (.jpg o .png) que tienen un tamaño entre los 44.4 kB hasta los 226 kB, los cuales al llegar se abren automáticamente.

La comunicación se ejecuta mediante el software GNU Radio en el cual se usa una técnica de modulación ASK a una rata de símbolo de 195,312 kbps, al momento de realizar esta comunicación se ejecuta por el dispositivo USRP-N210, dispositivo implementado para desarrollo de Radio Definida por Software, ya que permite delegar las funciones de hardware al código usado por la herramienta GNU Radio a través de Python y C++.

Se realiza las mediciones de rendimiento throughput, delay, jitter, evaluando el desempeño del algoritmo, por lo cual estos datos se ven afectados solo con la rata de bit ajustada en los diagramas de bloques de GNU Radio. El throughput varía con valores muy leves ya que está condicionado al empaquetamiento y la información de control que consume el protocolo UDP a través de la comunicación entre el computador y el USRP-N210 para su funcionamiento.

ABSTRACT

In this document, the development of the degree work is evidenced, which consists of the execution of a hybrid encryption algorithm for mission-critical communication systems through software-defined radio.

The algorithm is developed through the Python programming language which allows RSA asymmetric authentication. The RSA key is sent (public and private) encrypted by a password from the receiver to the transmitter, the transmitter unlocks the public key with which the AES session key is encrypted using RSA, which encrypts the plain text, then Flat text through the AES algorithm (symmetrical encryption). In the receiver the session key is deciphered by the private key RSA, with which it is used in turn to decipher the flat text.

For this work, samples are taken with image files (.jpg or .png) that have a size between 44.4 kB and 226 kB, which are opened automatically when they arrive.

The communication is executed through the GNU Radio software in which an ASK modulation technique is used at a symbol rate of 195,312 kbps, at the time of carrying out this communication it is executed by the USRP-N210 device, a device implemented for the development of Defined Radio by Software, as it allows hardware functions to be delegated to the code used by the GNU Radio tool through Python and C++.

Performance measurements throughput, delay, jitter are performed, evaluating the performance of the algorithm, for which these data are affected only with the bit rate adjusted in the GNU Radio block diagrams. The throughput varies with very slight values since it is conditioned to the packaging and the control information that the UDP protocol consumes through the communication between the computer and the USRP-N210 for its operation.

1. INTRODUCCION

En los sistemas de comunicación crítica se requiere que la información que se transmita sea real y verídica, que tenga los principios de integridad, disponibilidad y confiabilidad; ya que sean estas las únicas instituciones que puedan acceder a esta información, pues es vital para los casos eventuales de emergencia. [1]. Además, son más propensas a ataques cibernéticos, debido a que no están en un medio confinado, esto las hace vulnerables a ataques de playback o de reproducción, los cuales interceptan las comunicaciones y la retransmiten. [2], los ataques de Man-In-The Middle o de Intermediario que además de interceptar la información la pueden manipular y pueden reenviar información fraudulenta. [3].

Con el fin de mitigar este riesgo se desarrollará un sistema de cifrado híbrido que constará de autenticación asimétrica y de cifrado de contenido simétrico, que primeramente logre establecer un circuito de comunicación seguro y dos, logre un flujo eficiente de los datos para disminuir la latencia y obtener mejores tiempos de respuesta. Se desarrollará en el lenguaje de programación de Python que se implementará para equipos de Radio Definida por Software (SDR) en los dispositivos USRP-N210.

Con la evolución de la Televisión Digital Terrestre y la tecnología celular con tecnologías como LTE por su despliegue tecnológico y gran capacidad de penetración, optimización del ancho de banda y del espectro, es viable y valedero el uso de encriptación digital, como algoritmos RSA ó AES que requieren plena exactitud de los datos de la información cifrada para poder ser descifrados. [4], pero que por sí solos tienen un defecto, RSA es muy seguro pero tiene en su contra el procesamiento de la técnica de encriptación, pero AES aunque si es óptimo en el consumo de recursos, puede ser inseguro por el uso de única llave pública. Las técnicas de cifrado híbrido donde RSA encripta solamente la llave de sesión AES, brinda mayor seguridad, pero también garantizando un óptimo desempeño de los recursos de procesamiento de la encriptación. [5].

En la historia de la humanidad, a medida que se ha ido desarrollando la tecnología se puede evidenciar las distintas técnicas de cifrado como el algoritmo César, Feistel, entre otros. La invención de la radio por el italiano Guglielmo Marconi, revolucionó de manera histórica la forma de comunicarnos y nuevamente, se abre la necesidad de encriptar la información que por allí pasa, mientras que en el campo de la encriptación la gran y prestigiosa máquina de la Alemania Nazi del código enigma, sin lugar a duda, fue oro puro con lo cual le permitía controlar la guerra hasta la invención de la computadora, para poder descifrar su costoso código criptográfico. Por el avance de la tecnología se hace bastante necesario el desarrollo de sistemas de cifrado analógicos y digitales más robustos, como es el caso de los algoritmos

DES y AES de cifrado simétrico, el algoritmo de claves Diffie-Heltman y últimamente el desarrollo de algoritmos de clave pública y privada como el RSA.

En la actualidad, se ha desarrollado un algoritmo para telefonía llamado criptofonía, con el cual enmascara las voces, volviendo el sonido irreconocible y en el receptor puedan volver a ser construidas. [6].

2. FORMULACIÓN DEL PROBLEMA

En Internet y en todos los sistemas de comunicaciones se maneja un gran volumen de datos y es parte esencial garantizar la seguridad, confiabilidad y disponibilidad de dicha información. [7]. Además de los crímenes y atentados, el robo de información, filtración de datos, organizaciones delictivas internacionales, requieren una aproximación distinta. La profesionalización de las entidades de seguridad pública y servicios de emergencia implica un entrenamiento avanzado y especializado de los policías, gendarmes y elementos policiales, así como un equipamiento moderno que acompañe los esfuerzos por brindar protección a la ciudadanía. [8]. Estos competidores maliciosos intervienen con los ataques de denegación de servicios (DoS) y los ataques de denegación de servicio distribuido (DDoS) que afecta un sinfín de equipos limitando la disponibilidad de los datos con tan solo inundar de mensajes los sistemas comprometidos. [9].

En los sistemas de comunicación crítica hay ciertos requisitos exigibles como lo son la confidencialidad robusta (que sólo las entidades autorizadas tengan conocimiento del intercambio de información, por ejemplo utilizando técnicas de esteganografía (su misión es ocultar la propia información-datos, de modo que lo que no se ve es más difícil de robar o manipular) y anonimato (que no se sepa el emisor, el receptor, ni la vinculación entre emisores y receptores) y además que sólo las entidades autorizadas tengan acceso al contenido de la información intercambiada, por ejemplo utilizando técnicas de cifrado, fragmentación, manipulación a ciegas, etcétera. [1].

Otro problema, es que cuando se activan las labores de emergencia, éstas podrían ser fácilmente manipuladas porque se propagan en el aire, atraviesan muros y paredes, son más propensas al ataque de playblack o de reproducción, destacada en el párrafo anterior con los ataques de DoS y de intermediario, el cual puede afectar con la integridad y disponibilidad de los datos. [2], [3].

3. JUSTIFICACIÓN

Debido a que en el país, no se ha implementado tecnología de punta para sistemas de comunicación crítica, como es el caso de PPDR (Protección Pública de Atención a Desastres), la tecnología que se maneja es muy precaria, tanto como la policía como las fuerzas militares manejan un sistema troncalizado Motorola P25, ya que no cuenta con interoperabilidad entre los equipos que manejan diferentes cuerpos de socorro y desastres, su banda es muy limitada, por lo que es muy común ver comunicación half-duplex con el método Push-To-Talk, y su limitada red no permite una integración con la red de operadores móviles del país. Además, su sistema está permanentemente estableciendo autenticado, esto afectando la latencia del servicio de PPDR. [10],[11].

El propósito del proyecto es usar la menor cantidad de infraestructura posible, así procuramos que no se tengan que realizar conexiones a estaciones base, sino que los dispositivos puedan albergar un flujo de la información mediante las llamadas redes ad hoc. Esto permitirá mejorar el sistema de seguridad sin sacrificar la latencia en estos sistemas. [12].

Realizar el proyecto mejora y garantiza la confidencialidad, integridad y disponibilidad de los datos, que sean exclusivamente los organismos de socorro y defensa los únicos que puedan manipular y acceder a la información y que ningún tercero, atacante o persona ajena a dichas comunicaciones pueda interceptar o manipular las comunicaciones. Esto es crucial, puesto que la información que se maneja es clasificada, puesto que hay bases de datos de sospechosos y criminales en los organismos de seguridad y pueden ser manipulables por estas personas, y si tratamos con los organismos de emergencia, la información debe ser confiable, integral y disponible para que dichos organismos puedan actuar sin ningún tipo de información errónea, que sea eficaz y actuar con información veraz en el corto tiempo posible. [1].

4. OBJETIVOS

4.1. General

Desarrollar un algoritmo de cifrado híbrido para sistemas de comunicación de misión crítica mediante radio definido por software.

4.2 Específicos

- Diseñar un sistema de autenticación asimétrico para aplicaciones de sistemas de comunicación crítica e implementarlo en un dispositivo Radio Definida por Software.
- Diseñar un sistema de cifrado simétrico para el texto plano en aplicaciones de sistemas de comunicación crítica e implementarlo en un dispositivo Radio Definida por Software.
- Evaluar las variables como throughput, delay, jitter, texto plano y texto cifrado antes y después de aplicar el método de cifrado.

5. MARCO DE REFERENCIA

5.1 Antecedentes

5.1.1 Antecedentes modelo de cifrado simétrico basado en atractores caóticos

En el artículo se presenta un modelo de cifrado simétrico extensible con el fin de realizar comunicaciones digitales, aprovechando el caos generado por los sistemas dinámicos no lineales, todo esto es debido al aumento en la capacidad de procesamiento de las máquinas y los desarrollos en los algoritmos de búsqueda combinatoria que disminuyen el tiempo para descifrar fraudulentamente la información por tal razón se plantea la necesidad de generar nuevas formas de codificar la información y así lograr su transmisión segura, por lo anterior se propone en el esquema de cifrado desarrollado, se toma el sistema caótico y se divide en dos sub sistemas autónomos llamados maestro y esclavo, una vez sincronizados los dos subsistemas, se envía el mensaje por medio del método de la tabulación ASCII y el receptor lo representa por medio de la tabulación ASCII. [13].

5.1.2 Modelo híbrido de cifrado y descifrado para imágenes usando rsa, aes y afinamiento con la operación XOR.

El modelo propuesto describe un modelo de cifrado simétrico en el siguiente orden, cifrado por RSA, seguido por AES y finaliza con afinación con XOR. Se realiza la exposición de los datos sobre los diferentes algoritmos, para AES se usa el tamaño de llave de 128 bits (10 rondas).

Los resultados experimentales del modelo propuesto por los autores muestran que el cifrado tiene rendimiento aleatorio gracias a mediciones de parámetros como MSE, ED, MD y PSNR por lo que se puede concluir que su seguridad es de nivel complejo. Este modelo híbrido a diferencia de lo propuesto como desarrollo en este proyecto presenta un método adicional de cifrado, optimizado para ser usado sobre imágenes, aun así, no contempla el desarrollo a alto nivel de una solución nueva, como tal se hace uso de modelos existentes y acoplarlo a un objetivo específico y la protección de llaves por medio de sistemas adicionales. [14].

5.1.3 Cifrado de datos segura por medio de combinación de AES, RSA Y HMAC

El modelo propuesto depende de 3 funciones híbridas que son combinadas entre ellas para producir un sistema de protección fuerte. La función AES es usada para cifrar los archivos de manera simétrica, el algoritmo asimétrico RSA es usado para cifrar el password AES y HMAC para cifrar la autenticación entre cliente-servidor o cliente-cliente.

Los datos recorren el modelo bajo los tres niveles de seguridad usando los 3 algoritmos en combinación.

- El receptor usa RSA para generar las parejas de llave (pública y privada) y envía el valor de la llave pública al emisor.

- El emisor usa el algoritmo AES para cifrar los datos usando llaves cifradas de 256 bits.
- RSA es usado para cifrar la llave simétrica con la pública del receptor.
- HMAC con SHA256 es usado para calcular la MAC que incluye la combinación de función hash con una llave de criptografía secreta.
- Al texto cifrado producido por la codificación del texto plano se le añade MAC desde el texto cifrado.

El receptor usa la llave almacenada (llave privada) para descifrar la clave AES, y después hallar la información usando esta.
El modelo propuesto usa 3 tipos de cifrado de manera combinada en determinado orden con el fin de explotar las ventajas de cada uno de los algoritmos implementado para la construcción de un sistema altamente seguro, resaltando que la ejecución general de cifrado es simple y rápida con bajos requisitos computacionales.

Analizando este modelo con el proyecto propuesto se presenta similitud ya que se ofrece un método que puede ser acoplado a cualquier sistema de comunicación a diferencia que usa el cifrado con HMAC como capa adicional de seguridad haciéndolo más complejo donde el paso de llaves también contempla el cifrado, pero como principal diferencia en el proyecto propuesto se establece un método de paso seguro de llaves por medio de servicios externos y no capas de cifrado adicionales. [15].

5.1.4 Demostración USRP N210 de detección de banda ancha y clasificación de modulación jerárquica ciega

En este trabajo, se clasifican de manera jerárquica las señales AM, FM, 4/16/64-QAM, 2/4/8-PSK, DSS y FSK. Se usan patrones de energía y estaciones ciclo estacionarias para determinar las frecuencias de la señal receptora. En los trabajos más complejos como la modulación digital, se incluye la prueba de Kuiper. Estos trabajos fueron realizados con código echo en C++ y en GNU Radio. [16].

5.1.5 Análisis de fugas del transmisor cuando se opera USRP (N210) en modo dúplex

En los radios USRP-N210 suele haber fallas de auto-interferencia en comunicación full-dúplex que pueden añadir y distorsionar la señal recibida. En este trabajo se puede observar la pequeña falla en estos dispositivos y la manera de cómo se puede suprimir este efecto. [17].

5.1.6 Mejora de la privacidad en MANET basado en SDN mediante el uso de un algoritmo de cifrado y descifrado híbrido

Se desarrolla un esquema de cifrado basado en MANET (Redes ad hoc) [12]. las cuales pueden usar nodos en los mismos dispositivos para evitar poner infraestructura extra, utilizando la tecnología de redes definidas por software (SDR), donde se desarrolla un

modelo de cifrado monomórfico, este tipo de cifrado se destaca por actuar y hacer operaciones con los datos cifrado en sí sin tener que descifrarlos.

La justificación para usar cifrado homomórfico se basa en el hecho que, al poder operar con los datos cifrados sin tener que descifrarlos, es más difícil para los atacantes obtener algún tipo de información y garantiza la confiabilidad de la información, ya que estos datos pueden ser descifrados en servidores externos para realizar operaciones de consulta y funcionamiento. el esquema de cifrado homomórfico y no homomórfico. Se evidencia en la Fig. 1. [18].

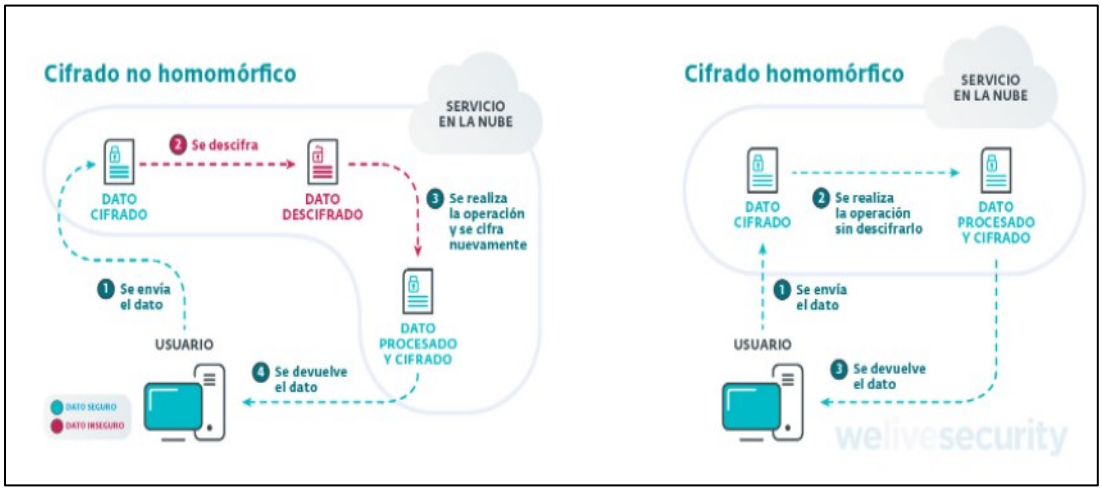


Fig 1. Esquema comparativo entre cifrado no homomórfico-cifrado. [18]

Es por esto por lo que el trabajo de Miriyala y Satya, se basa en cifrado homomórfico y doble descripción usada sobre redes definidas por software (SDN) en un sistema de red ad hoc MANET. Para garantizar la seguridad de la información se usa sistemas de doble clave, incluyendo resultados intermedios, también se obtendrá un mínimo gasto de recursos de cómputo y de comunicación. [19].

5.1.7 Modelo de cifrado híbrido basado en AES, RSA y TWOFISH para el cifrado Bluetooth

Se usa el algoritmo AES con una llave de 128 bits para la codificación del mensaje, se codifica el mensaje nuevamente usando TWOFISH con la misma llave de 128 bits, finalmente, la llave generada al inicio será cifrado por RSA con una llave de 1024 bits para protección sobre la transferencia.

En el receptor el proceso de descripción trabaja en orden inverso al proceso de cifrado, generando conexiones más seguras para la conexión realizada.

El modelo híbrido presentado en esta propuesta para uso sobre comunicación en bluetooth no tiene contemplado el no repudio sobre el texto cifrado, así como la autenticidad del origen porque se generó una función hash y firmas digitales como una de sus ventajas.

Se observa similitud con el proyecto planteado y como diferencia este proyecto usa twofish como método adicional de cifrado en el paso de llaves de AES, desde nuestra propuesta se hace énfasis en el paso seguro por medio de servicios que logren proteger esa primera comunicación entre los dispositivos. [20].

5.1.8 Cifrado de voz mediante múltiples mapas caóticos con rápida transformación de Fourier

Los algoritmos de cifrado de voz son muy importantes para mejorar la seguridad que está en permanente peligro de los ciber-ataques, para ello Alemami et al., desarrolla un algoritmo en MATLAB basado en cuatro pasos: haciendo una transformada rápida de Fourier (FFT) y luego de esto se obtiene el valor absoluto, en el segundo paso se crea un algoritmo de mapa lógico, el cual inicia un proceso de confusión de la señal, en el tercer paso se crea un algoritmo de mapa Seno, que aprovecha la función seno clave para el proceso de difusión, después de haber obtenido ambos resultados se ejecuta un XOR entre las dos, obteniendo los datos codificados. La efectividad del método se prueba con el coeficiente de correlación de Pearson, el cual define resultados entre -1 a 1. Entre más alto sea el valor absoluto la correlación será más alta y si es muy próximo a cero será lo contrario. Por consiguiente, se obtienen valores muy próximos a cero. [21].

5.1.9 Un nuevo algoritmo de cifrado de voz dinámico basado en el mapa caótico de Lorenz sobre el protocolo de internet

El modelo caótico de Lorenz propuesto realiza el cifrado basado en el modelo del atractor de Lorenz visto en la Fig 2. ya que pequeños cambios producidos en las condiciones iniciales varían el resultado final. Este algoritmo se basa primeramente en la generación de llaves las cuales al tener mayor número y longitud eleva la seguridad ante los posibles ataques y la posterior codificación basada en dicho atractor, los resultados demuestran una covarianza muy baja y muy buenos niveles de entropía que comparados con los algoritmos AES disminuye el retardo de procesamiento del algoritmo en cuestión. [22]. Tal como se evidencia en la Fig 2.

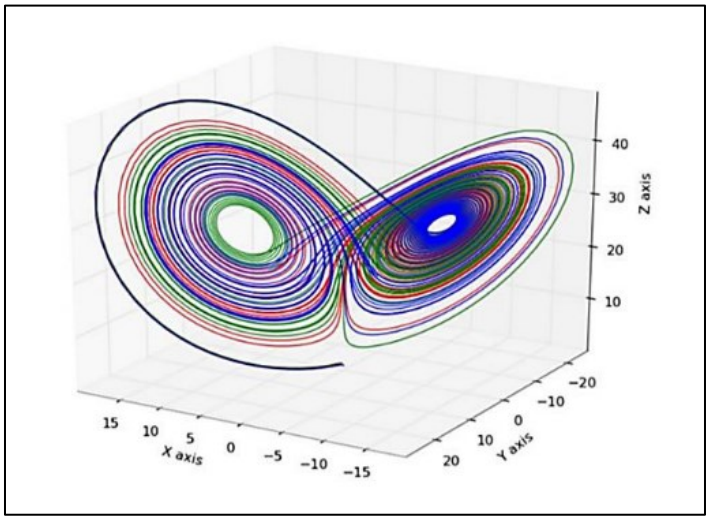


Fig 2. atractor de Lorenz. [22]

5.1.10 Métricas de rendimiento efectivas para sistemas de comunicación multimedia de misión crítica

En este trabajo se describe cómo se maneja la comunicación con el protocolo IMS (Subsistema Multimedia IP) y los protocolos de señalización SIP que son los que configuran los registros de usuario y acceso al servidor del sistema. Se describe en una fuerte necesidad de distinguir los servicios de comunicación de misión crítica de los tradicionales, ya que, los segundos son fundamentales en este tipo de escenarios y que no ocasiones de ningún modo la pérdida de la red. Cabe resaltar dentro de la investigación dos propuestas fuertes y es usar tecnología dedicada o de ancho de banda comercial. En la primera, se despegará red y hardware exclusivo para las comunicaciones y en el segundo escenario, se compartiría la red con la tradicional, pero ofreciendo un espectro mayor de ancho de banda usando Redes Definidas por Software o Función de Virtualización de Redes, en este escenario vale la pena señalar el camino hacia novedosos algoritmos de cifrado y compresión que puedan mejorar los procesos de autenticación y la rata de transmisión con paquetes más cortos y qué darían prioridad en los procesos de KPI (Indicadores Clave de Procesos), en los cuales se examinan la QoS de cualquier red y la forma en cómo se empaqueta los paquetes y se asocian en las distintas capas del modelo OSI. El gran desafío parte de la capa física primordialmente y desde allí se trabaja y fortalece las capas superiores. [23].

5.1.11 Estudio y análisis para la selección de una tecnología a implementar en la actualización de la red PPDR de la policía metropolitana de Bogotá “mebog”

En la tésis de Fadul, se puede evidenciar en trabajo de investigación con respecto a qué la tecnología se puede implementar para uso de la red Protección Pública y Atención de Desastres (PPDR) para la Policía Metropolitana de Bogotá y otros organismos de emergencia

militares, y de socorro como salud y bomberos. En este trabajo se hace un estudio de las diversas tecnologías como LTE OPERADOR, LTE PRIVADO, P25, DRM, TETRA e IDEN, concluyendo que la que mejor rendimiento tuvo fue la de LTE OPERADOR, ya que, por su fácil interoperabilidad, desarrollo de tecnología de punta enfocado hacia estas tecnologías, bajas latencias, bajas pérdida de paquetes y montado sobre IP puede desplegar una infinidad de servicios como la calidad de las llamadas, soporte de mapas y video streaming, sin importar la cobertura. Habrá que preguntarse que suele ser mejor, advirtiendo factores económicos, políticos y regulatorios con respecto a una red dedicada que usaría LTE PRIVADO para su operación o usar LTE OPERADOR compartido con la red de telefonía desplegada actualmente y con la posible traba que puedan ofrecer los operadores de red celular. [10].

5.1.12 Un flujo caótico simple con función sinusoidal hiperbólica y su aplicación al cifrado de voz

El trabajo hecho por Mobayen et. al., consiste en realizar un sistema de cifrado con un flujo caótico con funciones sinusoidales hiperbólicas. Aquí se estudia su efectividad en el cifrado de voz al generar números aleatorios sujetos a un algoritmo llamado RNG usado para cifrar el sonido, y estudiando dos características especiales de los atractores ocultos (con punto de equilibrio y sin punto de equilibrio), donde se evalúan en diagramas de bifurcación y los exponentes de Lyapunov, lo que hace que este sistema sea demasiado complejo e ideal para labores de esquemas criptográficos. A partir de este estudio se propone usar atractores caóticos para el desarrollo de tecnologías futuras y el hardware correspondiente para ello. [24].

5.1.13 Cifrado tire Pressure Monitoring System para mejorar la seguridad vehicular

En este trabajo se pretende evaluar la viabilidad de implementar un protocolo TMPS **(**Tire-Pressure Monitoring System) o “sistema de monitorización de la presión de neumáticos, que sea más seguro basado en el registro de desplazamiento de retroalimentación lineal (LSFR). Este modelo se evaluó tanto en sistemas radio definida por Software mediante el módulo Universal Serial Radio Peripheral (USRP) N210 y Arduino de bajo costo para demostrar la eficiencia en la amenaza del sistema. [25].

5.1.14 Analizador de espectro basado en SDR basado en GNU Radio de código abierto

Para la realización del proyecto se lleva a cabo la construcción de un sistema de detección de espectro de 2,3 GHz a 2,7 GHz, destinado a ser utilizado en Cognitive Radio. usando un USRP SDR de 12 bits, cuyas muestras fueron guardadas y luego analizadas dentro de Matlab. Otra configuración, esta vez con un dongle de bajo costo RTL-SDR de 8 bits, conectado a una Raspberry Pi capturada, demodulada y mostró las señales de transmisión en AM y FM, con el algoritmo de gestión escrito en Python, Además, utilizando una Raspberry Pi2, se diseñó un sistema para monitorear las porciones no utilizadas (llamadas "blancas") del

espectro UHF, con el extremo frontal de RF realizado por un analizador de espectro RF Explorer de bajo costo.

Los SDR no necesitan necesariamente una computadora para funcionar, se conectó una unidad RTL a una Raspberry Pi2, en la cual se ejecuta en el sistema operativo Raspbian, que muestrea y demodula la transmisión de señales AM y Señales FM, también se implementó un receptor canalizado, para portadoras de radar entrantes, en GNU Radio, utilizando RTL SDR para receptores de alerta temprana, con filtros de paso de banda seleccionando diferentes sub-bandas cubriendo un ancho de banda de 2,4 MHz alrededor de 940 MHz. Un analizador de espectro de banda AM, basado en un Si570VCO (oscilador controlado por voltaje) y administrado por LabVIEW, puso a disposición las muestras digitales en línea por un servidor web, para ser utilizado con fines educativos. Se utilizó otro RTL-SDR como analizador de espectro, conectado esta vez a un teléfono móvil Android con una aplicación Java, ambas unidades conectadas mediante un cable USB. Otra aplicación, usando USRP y GNU Radio, monitorea la banda alrededor de 2,5 GHz para encontrar ranuras vacías para Cognitive Radio, utilizando un algoritmo de detección de energía. La transmisión de FM y la televisión digital se monitorearon y desmodularon usando RTL y USRP SDRs, cuyos datos son enviados por TCP/IP por una Raspberry Pi para ser analizados remotamente. Un sistema llamado SwepSense fue ideado para monitorear un gran ancho de banda de espectro electromagnético dividido en las subbandas 50 MHz-2,2 GHz; 400 MHz- 4,4 GHz y 1,2 GHz-6 GHz, con alta resolución temporal, basado en un USRP cuyo chip de oscilador local estaba conectado a un circuito generador de diente de sierra. Para cubrir dos sub-bandas distantes, se utilizaron dos unidades RTL para operar como un espectro de bajo costo en un sistema de monitoreo, cada uno cubriendo una banda específica, controlado dentro del entorno GNU Radio. [26].

5.1.15 Implementación de estación base GSM recepción de señales LTE aplicando radio definido por software

La Universidad Santo Tomás actualmente no cuenta con toda la infraestructura de telecomunicaciones que permita a sus estudiantes realizar prácticas de laboratorio en el área de la telefonía móvil. Por lo anterior, este trabajo de investigación tiene como finalidad la implementación de una estación base de telefonía móvil celular (TMC), GSM 2G y el desarrollo de un sistema de recepción de señales LTE 4G, todo esto mediante el uso de radio definido por software y herramientas de software libre. La implementación de la estación base genera y gestiona el recurso inalámbrico emitido por ella misma, con el fin de aprovisionar servicios de mensajería de texto corto y llamada de voz. Por otra parte, el sistema de recepción de señales LTE 4G fue desarrollado en el software GNU Radio, lo que permite replicar de una forma didáctica y educativa, todo el proceso de generación y recepción de servicios básicos de telecomunicaciones móviles. [27].

5.2 Marco Teórico

5.2.1 Cifrado

La criptografía o cifrado se designa a un procedimiento que traduce un texto sin formato (plain text o texto plano) en una secuencia ininteligible de caracteres mediante una clave. El objetivo es que el contenido del texto secreto resultante o criptograma (texto cifrado) solo sea accesible para aquellos que disponen de la clave para descifrarlo. Aunque expresiones tales como "texto plano" o "texto cifrado" provengan de la estrategia militar, los métodos criptográficos pueden ser también aplicados a otro tipo de información electrónica como mensajes de voz, archivos de imagen o códigos de programación, además de a mensajes de texto.

El cifrado tiene como finalidad proteger archivos, unidades de disco o directorios de intrusiones o transmitir datos de forma confidencial. Ya en la antigüedad se utilizaban métodos criptográficos sencillos que se reducían en primera instancia a la codificación de la información que se quería proteger, permutando caracteres aisladosxz|z , palabras o frases enteras del texto plano del mensaje (cifrado por transposición o permutación) o sustituyendo los caracteres por combinaciones alternativas (cifrado por substitución).
Para decodificar un texto cifrado es necesario que el destinatario conozca la regla por la cual se ha cifrado el texto. En el cifrado por transposición, las permutaciones suelen llevarse a cabo a partir de una matriz (matriz de transposición) que ha de conocerse o poderse reconstruir. El cifrado por sustitución se basa en una ordenación tabular de caracteres y cifras en forma de código secreto. [28].

5.2.1.1 Cifrado de los Datos

Un método de cifrado se compone fundamentalmente de dos elementos, un algoritmo criptográfico y una o más claves secretas. Mientras que el algoritmo describe el método de cifrado (p. ej., "desplaza cada letra a lo largo de la secuencia del alfabeto"), la clave proporciona el parámetro ("C = tres posiciones"). Esto es lo que permite definir al cifrado como un procedimiento por el cual se entrega un texto en claro y una clave al algoritmo criptográfico y se obtiene un texto cifrado. [28].

5.2.1.2 Clave Digital

En la codificación moderna se utilizan secuencias de bits como claves digitales. Un criterio esencial para la seguridad del cifrado es la longitud en bits de las claves, la cual indica la medida logarítmica del número de combinaciones posibles que pueden usarse. También se habla en este caso de espacio de posibilidades de claves o espacio de claves, el cual, cuanto más amplio, más resistente es contra los llamados ataques de fuerza bruta (probar una tras otra todas las combinaciones posibles).

El código César permite explicar este tipo de ataque informático. El espacio de claves en este método de codificación es de 25, lo que equivale a una longitud de clave por debajo de 5. Esto significa que alguien que busque descifrar esta clave, solo necesita probar 25 combinaciones, lo que convierte a este método de cifrado en un obstáculo muy débil. Por el contrario, los métodos más modernos utilizan claves capaces de generar más combinaciones. El Advanced Encryption Standard o AES, por ejemplo, permite escoger entre longitudes de clave de 128, 192 o 256 bits, lo cual aumenta el abanico de claves posibles.

Un método muy común para calcular las claves desde las contraseñas lo representa la función derivada de claves PBKDF2 (Password-Based Key Derivation Function 2), método por el cual se añade una secuencia semi aleatoria de bits, el denominado valor de sal, a la contraseña y, utilizando funciones hash criptográficas, se deriva la secuencia de bits con la longitud deseada.

El método PBKDF2, el cálculo matemático se ejecuta en varias iteraciones (fases idénticas o rondas), para proteger la clave así generada contra ataques de fuerza bruta. El valor de sal dificulta aún más un intento de descifrar la contraseña con tablas arcoíris o tablas rainbow, un patrón de ataque que permite derivar una contraseña de un valor hash almacenado. Otros procedimientos de hash de contraseñas son los llamados sctypt, bcrypt y LM-Hash, aunque este último está considerado obsoleto y poco seguro. [28].

5.2.1.3 Cifrado Simétrico

Si la clásica criptografía simétrica utiliza el alfabeto para traducir un texto plano en un texto cifrado, el cifrado informático se realiza mediante bits. En este caso se diferencia entre el cifrado de flujo y el de bloque por el algoritmo que utilizan:

Cifrado en flujo (stream): el cifrado se realiza por bits y se basa en claves muy largas. Cada bit del texto sin formato se correlaciona con un bit en el flujo de claves, de tal forma que se crea un carácter cifrado.

Cifrado de bloques: en este algoritmo los bits que se han de cifrar se agrupan en bloques de longitud fija, que se corresponden con otros diferentes. Resulta un cifrado de longitud fija.
Los algoritmos de la criptografía simétrica se basan habitualmente en operaciones tan sencillas como la substitución y la permutación, que hoy se aplican al texto en claro de forma combinada y en rondas o iteraciones consecutivas, así como en las adiciones, las multiplicaciones, la aritmética modular y las operaciones XOR. [28].

5.2.1.3.1 Data Encryption Standard (DES)

Se trata de un método de cifrado desarrollado por IBM en la década de 1970 y estandarizado por el Instituto Nacional de Estándares y Tecnología (NIST) norteamericano en 1977. Como primer método de cifrado informático y tan seguro como permitían los estándares de la época, DES sienta las bases de la criptografía moderna. Libre de derechos de patente, hoy se

considera ya obsoleto debido a la reducida longitud de 64 bits (56 bits útiles) de las claves. Ya en 1994, el sistema criptográfico fue descifrado en 50 días con 12 estaciones HP-9735 en un primer criptoanálisis experimental. El estándar actual de la tecnología ha hecho a una clave DES tan vulnerable que un ataque de fuerza bruta la descifra en unos pocos segundos. Este algoritmo simétrico trabaja en bloques de bits (cifrado de bloques): el texto en claro se subdivide en bloques de 64 bits, cada uno de los cuales se cifra con una clave de 64 bits. Se traduce así cada texto de 64 bits en un texto cifrado de 64 bits. En realidad, el algoritmo solo utiliza 56 bits pues los ocho restantes se usan para comprobar la paridad, siendo descartados después. En DES, de esta forma, solo se dispone de 56 bits efectivos para el cifrado.
El algoritmo de cifrado en DES se estructura en una red Feistel que combina sustituciones y permutaciones en 16 rondas. Este esquema, que toma su nombre del empleado de IBM Horst Feistel, se puede describir en cuatro características:

1. Permutación de entrada: el bloque de texto plano de 64 bits se somete a una permutación de entrada que modifica la secuencia de los bits. El resultado se escribe en dos bloques de 32 bits, uno izquierdo (L) y uno derecho (R).

2. Permutación de la clave: Los 56 bits de la clave relevantes para el cifrado también se someten a una permutación, para después ser divididos en dos bloques de 28 bits (C y D). De estos dos bloques C y D se genera una clave de ronda (subclave) para cada ronda. Para generar esta clave, los bits de ambos bloques se desplazan cada uno en 1 o 2 bits cíclicamente hacia la izquierda. Esto debe garantizar que en cada ronda de cifrado se crea una subclave diferente. Seguidamente los dos bloques C y D resultan en una subclave de 48 bits.

3. Rondas de cifrado: cada ronda recorre los pasos a) hasta d). En cada vuelta el cifrado involucra a un bloque R y a una subclave. De momento el bloque L permanece fuera del proceso.

3.1 Expansión: la mitad del bloque derecho de 32 bits R se expande a un bloque de 48 bits mediante la permutación de expansión. Para ello, los 32 bits se dividen en grupos de 4 bits y se permutan o se duplican.

3.2 Unión mediante una operación XOR del bloque de datos y la subclave: en cada ronda el bloque R expandido a 48 bits se une a una subclave de 48 bits mediante una operación XOR, dando como resultado otro bloque de 48 bits.

3.3 Cajas-S o cajas de substitución: tras mezclarlo con la subclave, este bloque de 48 bits se divide en ocho bloques de 6 bits. Estos bloques son procesados por las cajas de substitución, que substituyen los 6 bits de entrada por 4 de salida, que se agrupan en un bloque de 32 bits.

3.4 Unión con XOR del bloque R con el bloque L: con cada ronda, el resultado de las cajas-S se mezcla con el hasta ahora inutilizado bloque L mediante XOR. De ello

resulta un bloque de 32 bits que, como nuevo bloque R, entra en la segunda ronda. El bloque R de la primera ronda es el bloque L de la segunda.

4. Permutación de salida: Una vez se han efectuado las 16 rondas, los bloques L y R se agrupan en un bloque de 64 bits y se someten a una permutación de salida inversa a la de entrada. El texto resultante es un mensaje cifrado. Para descifrar un texto codificado según DES se sigue el mismo esquema en sentido inverso, tal como se evidencia en la Fig 3. [28].

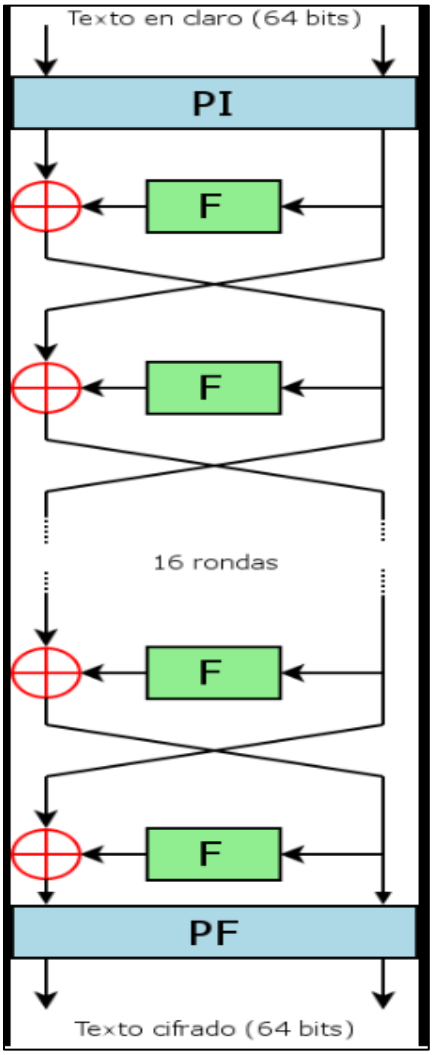


Fig 3. Estructura General de Feistel en DES. [28]

5.2.1.3.2 Advanced Encryption Standard (AES)

En la década de 1990 se puso de relieve que el estándar de cifrado DES usado hasta ese momento ya no estaba a la altura del desarrollo tecnológico, de tal manera que se hacía necesario contar con otro más moderno. El algoritmo Rijndael, desarrollado por los entonces estudiantes Vincent Rijmen y Joan Daemen, se consolidó como el sucesor tras ganar la

convocatoria que lanzó el NIST en 1997 para desarrollar el AES, un estándar de cifrado adaptado a la información sensible del siglo XXI. El NIST lo certificó como AES a finales del año 2000.

El sucesor de DES también subdivide el texto plano en bloques, pero el nuevo AES soporta claves de 128, 192 y 256 bits y, en lugar de utilizar bloques de 64 bits, el estándar avanzado utiliza bloques más grandes de 128 bits, que se cifran en varias rondas consecutivas con ayuda de una red de substitución-permutación (SPN). Como su antecesor, AES también usa en cada ronda una nueva subclave, que se deriva de forma recursiva de la clave inicial y se une al bloque a cifrar con XOR. El funcionamiento del algoritmo se puede subdividir grosso modo en cuatro fases:

1. Expansión de la clave: para que AES pueda utilizar una nueva subclave en cada ronda, derivada por recursión de la primera clave, esta se ha de expandir a una longitud que permita generar el número necesario de subclaves de 128 bits. Así, cada subclave se basa en un fragmento de la clave de salida expandida. El número de subclaves necesarias comprende el número de rondas (R), incluida la ronda final y una subclave para la ronda previa (R + 1).

2. Ronda previa: en la ronda previa el bloque de 128 bits se transpone en una tabla bidimensional o matriz (state) y se une con la primera subclave mediante XOR (Key Addition). La tabla comprende 4 columnas y 4 líneas. Cada casilla contiene 1 byte (8 bits) del bloque que se ha de cifrar.

3. Rondas de cifrado: el número de rondas depende de la longitud de clave utilizada, esto es, 10 rondas en AES128, 12 en AES192 o 14 en AES256. En cada ronda tienen lugar las siguientes operaciones:

   3.1 SubBytes: se trata de una substitución alfabética simple, en la cual cada byte del bloque original se sustituye por un equivalente mediante una caja-S.
   3.2 ShiftRows: en la transformación ShiftRow los bytes de las casillas de la matriz se desplazan cíclicamente hacia la izquierda.
   3.3 MixColumns: esta transformación contenida en el algoritmo AES consiste en mezclar los datos dentro de las columnas de la tabla. Este paso se basa en recalcular cada casilla, para lo cual las columnas se someten a una multiplicación matricial. Los resultados se unen mediante XOR.
   3.4 KeyAddition: al final de cada ronda se produce una nueva KeyAddition, es decir, una unión con la subclave, que se apoya, como en la etapa inicial, en la unión por XOR del bloque de datos o los bytes de la matriz con la subclave actual.

4. Etapa final: Esta es la última ronda de cifrado que, al contrario que las rondas anteriores, no contiene transformaciones del tipo MixColumns, comprendiendo únicamente las operaciones SubBytes, ShiftRows y KeyAddition. El resultado de esta ronda final es el texto cifrado.

En la Fig 4. Se evidencia como AES ha demostrado una elevada seguridad debido a su algoritmo, sin conocerse bis dato ningún ataque relevante. La longitud de clave de 128 bits anula la eficacia de los ataques de fuerza bruta y las operaciones ShiftRows y MixColumns son las responsables de la mezcla óptima de los bits, porque, en definitiva, cada bit depende de la clave. Por su sencilla implementación y gran velocidad, AES se utiliza como estándar de cifrado en WPA2, SSH e IPSec y su algoritmo se usa para cifrar archivos comprimidos como 7-Zip o RAR. [28].

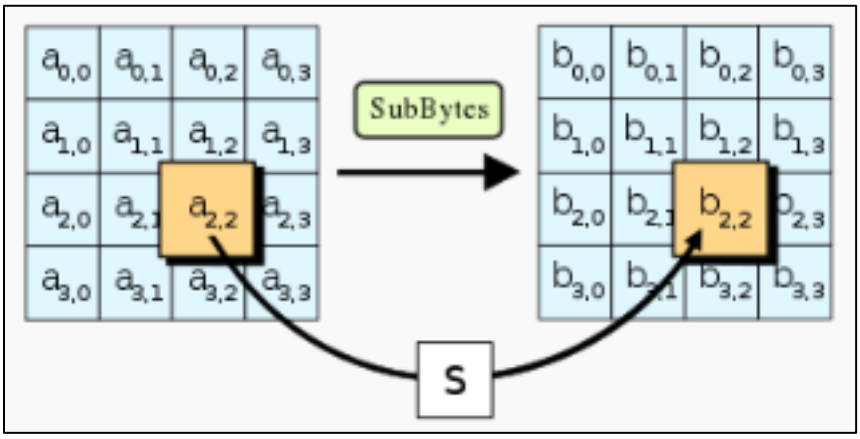


Fig 4. Estructura General AES. [28]

5.2.1.3.3 AES y sus modos de cifrado

AES no es un simple algoritmo que se le pasa un dato de entrada y devuelva una salida, él funciona con modos de operación los cuales son varios, pero aquí una pequeña lista de los muchos que se podrían usar:[29].

1) **Cipher-block chaining (CBC)**En el modo cipher-block chaining (CBC), a cada bloque de texto plano se le aplica la operación **XOR** con el bloque cifrado anterior antes de ser cifrado. De esta forma, cada bloque de texto cifrado depende de todo el texto en claro procesado hasta este punto. Para hacer cada mensaje único se utiliza asimismo un vector de inicialización. [29].

2) **Counter (CTR)**

Al igual que OFB, el modo contador convierte una unidad de cifrado por bloques en una unidad de flujo de cifrado. Genera el siguiente bloque en el flujo de claves cifrando valores sucesivos de un *contador*. El contador puede ser cualquier función sencilla que produzca una secuencia de números donde los resultados se repiten con muy baja frecuencia. Si bien la operación más usada es un contador, el modo CTR tiene características similares al OFB, pero permite también usar una propiedad de acceso aleatorio para el descifrado. [29].

3) **Electronic Code-Book (ECB):**

El más sencillo es el modo *electronic codebook* (ECB), en el cual los mensajes se dividen en bloques y cada uno de ellos es cifrado por separado utilizando la misma clave K. La desventaja de este método es que a bloques de texto plano o claro idénticos les corresponden bloques idénticos de texto cifrado, de manera que se pueden reconocer estos patrones como guía para descubrir el texto en claro a partir del texto cifrado. De ahí que no sea recomendable para protocolos cifrados. [29].

4) **Galois/Counter Mode (GCM): En criptografía, Galois/Counter Mode (GCM)** es un modo de operación para los cifrados de bloques criptográficos de clave simétrica ampliamente adoptados para su rendimiento. Las tasas de rendimiento de GCM para canales de comunicación de alta velocidad de última generación se pueden lograr con recursos de hardware económicos. La operación es un algoritmo de cifrado autenticado diseñado para proporcionar autenticidad de datos (integridad) y confidencialidad. GCM se define para cifrados de bloques con un tamaño de bloque de 128 bits. [29].

5) **Galois Message Authentication Code (GMAC)**

es una variante de autenticación solamente del GCM que puede formar un código de autenticación de mensajes incremental. Tanto GCM como GMAC pueden aceptar vectores de inicialización de longitud arbitraria. [29].

5.2.1.4 Cifrado Asimétrico

Los algoritmos asimétricos son diferentes a los simétricos en un sentido muy importante. Cuando se genera una clave simétrica, simplemente se escoge un número aleatorio de la longitud apropiada. Al generar claves asimétricas el proceso es más complejo.

Los algoritmos asimétricos se llaman asimétricos porque en lugar de usar una sola clave para realizar la codificación y la decodificación, se utilizan dos claves diferentes: una para cifrar y otra para descifrar. Estas dos claves se encuentran asociadas matemáticamente, cuya característica fundamental es que una clave no puede descifrar lo que cifra.

Cuando se completa la generación de una clave asimétrica se define una clave de cifrado (clave pública) y una clave de descifrado (clave privada); la primera puede ser conocida por todo el mundo, pero, de otro lado se debe tener mucho cuidado en ocultar la clave privada. Las claves asimétricas tienen la sorprendente propiedad de que lo que se está cifrando con una clave sólo se puede descifrar con la otra. [30].

5.2.2 Python

Python, es un lenguaje de programación creado por Guido van Rossum a principios de los años 90 cuyo nombre está inspirado en el grupo de cómicos ingleses “Monty Python”. Es un lenguaje similar a Perl, pero con una sintaxis muy limpia y que favorece un código legible.

Se trata de un lenguaje interpretado o de script, con tipado dinámico, fuertemente tipado, multiplataforma y orientado a objetos. [31].

5.2.2.1 Lenguaje interpretado o de script

Un lenguaje interpretado o de script es aquel que se ejecuta utilizando un programa intermedio llamado intérprete, en lugar de compilar el código a lenguaje máquina que pueda comprender y ejecutar directamente una computadora (lenguajes compilados).
La ventaja de los lenguajes compilados es que su ejecución es más rápida. Sin embargo, los lenguajes interpretados son más flexibles y portables.

Python tiene, no obstante, muchas de las características de los lenguajes compilados, por lo que se podría decir que es semi interpretado. En Python, como en Java y muchos otros lenguajes, el código fuente se traduce a un pseudo código máquina intermedio llamado byte code la primera vez que se ejecuta, generando archivos .pyc o. pyo (bytecode optimizado), que son los que se ejecutarán en sucesivas ocasiones. [31].

5.2.2.2 Tipado Dinámico

La característica de tipado dinámico se refiere a que no es necesario declarar el tipo de dato que va a contener una determinada variable, sino que su tipo se determinará en tiempo de ejecución según el tipo del valor al que se signe, y el tipo de esta variable puede cambiar si se le asigna un valor de otro tipo. [31].

5.2.2.3 Fuertemente Tipado

No se permite tratar a una variable como si fuera de un tipo distinto al que tiene, es necesario convertir de forma explícita dicha variable al nuevo tipo previamente. Por ejemplo, si tenemos una variable que contiene un texto (variable de tipo cadena o string) no podremos tratarla como un número (sumar la cadena “9” y el número 8). En otros lenguajes el tipo de la variable cambiaría para adaptarse al comportamiento esperado, aunque esto es más propenso a errores. [31].

5.2.2.4 Multiplataforma

El intérprete de Python está disponible en multitud de plataformas (UNIX, Solaris, Linux, DOS, Windows, OS/2, Mac OS, etc.) por lo que si no utilizamos librerías específicas de cada plataforma nuestro programa podrá correr en todos estos sistemas sin grandes cambios. [31].

5.2.2.5 Orientado a Objetos

La orientación a objetos es un paradigma de programación en el que los conceptos del mundo real relevantes para nuestro problema se trasladan a clases y objetos en nuestro programa. La ejecución del programa consiste en una serie de interacciones entre los objetos.

Python también permite la programación imperativa, programación funcional y programación orientada a aspectos. [31].

5.2.2.6 ¿Por Qué Python?

Python es un lenguaje que todo el mundo debería conocer. Su sintaxis simple, clara y sencilla; el tipado dinámico, el gestor de memoria, la gran cantidad de librerías disponibles y la potencia del lenguaje, entre otros, hacen que desarrollar una aplicación en Python sea sencillo, muy rápido y, lo que es más importante, divertido. [31].

La sintaxis de Python es tan sencilla y cercana al lenguaje natural que los programas elaborados en Python parecen pseudocódigo. Por este motivo se trata además de uno de los mejores lenguajes para comenzar a programar. [31].

Python no es adecuado sin embargo para la programación de bajo nivel o para aplicaciones en las que el rendimiento sea crítico. [31].

Algunos casos de éxito en el uso de Python son Google, Yahoo, la NASA, Industrias Light & Magic, y todas las distribuciones Linux, en las que Python cada vez representa un tanto por ciento mayor de los programas disponibles. [31].

5.2.2.7 Librería PYCRYPTODOME

Es un paquete o librería de Python autónomo primitivo criptografico de bajo nivel el cual es compatible con Python 2.7, Python 3.5 y posteriores, y PyPy. [32].

5.2.2.7.1. Funcionalidades de la Librería PYCRYPTODOME

En la Tabla 1. Se observa las diferentes funciones que se encuentran en la librería pycryptodome.

Paquete	**Descripción**
Cifrado criptográfico	Módulos para proteger la **confidencialidad**, es decir, para cifrar y descifrar datos (ejemplo: AES).
Crypto.Firma	Módulos para asegurar **la autenticidad**, es decir, para crear y verificar las firmas digitales de los mensajes (ejemplo: PKCS#1 v1.5).
Cripto.Hash	Módulos para la creación de **resúmenes** criptográficos (ejemplo: SHA-256).
Crypto.PublicKey	Módulos para generar, exportar o importar *claves públicas* (ejemplo: RSA o ECC).
Protocolo criptográfico	Módulos para facilitar las comunicaciones seguras entre las partes, en la mayoría de los casos aprovechando las primitivas criptográficas de otros módulos (ejemplo: el esquema de uso compartido secreto de Shamir).
Cripto.IO	Módulos para tratar codificaciones comúnmente utilizadas para datos criptográficos (ejemplo: PEM).
Crypto.Random	Módulos para generar datos aleatorios.
Crypto.Util	Rutinas de propósito general (ejemplo: XOR para cadenas de bytes).

Tabla 1. Funcionalidad librería PYCRYPTODOME. [33].

5.2.3 Radio Definida por Software (SDR)

El concepto SDR básicamente hace referencia al hecho de utilizar un software para controlar casi todas las funciones de un dispositivo hardware de comunicaciones o de sensores tipo radar. Esta tecnología SDR es muy flexible y fácil de usar lo que permite y habilita la creación de dispositivos inalámbricos interoperables con diversas ventajas, entre las que se puede resaltar:

- Múltiples modos de operación
- Reconfiguración
- Actualizaciones Over-The-Air
- Menor coste de desarrollo

El concepto SDR básicamente lo que hace es acercar el código lo máximo posible a la antena, convirtiendo así problemas hardware en problemas software, ya que gracias a la tecnología con la que cuenta (tecnología software, tecnología de hardware de procesado de señal, tecnología RF) como se muestra en la siguiente Figura, donde la unión entre un hardware y un software hacen realidad la consecución de este concepto. (E&Q Engineering, 2010). [34].

5.2.3.1 Procesamiento Digital de Señal (DSP)

Este es un término que se relaciona bastante con el concepto SDR y básicamente es una técnica que toma señales que usualmente están en forma analógica, y las convierte en datos digitales que luego pueden ser analizados. Este análisis es ejecutado en forma digital ya que una señal cuando 31 ha sido reducida a valores numéricos discretos, sus componentes pueden ser aislados, analizados y reordenados más fácilmente que en su primitiva forma analógica. Prácticamente DSP es la transformación de una señal en un hardware digital según reglas previamente definidas las cuales son introducidas en el hardware a través de un software que puede o no manejar lenguajes tanto de alto como de bajo nivel. En la siguiente Figura se muestra un esquema muy simple de lo que trata DSP. (Hernández, 2004). [34].

5.2.3.2 Características de la Radio Definida por Software (SDR)

Existen diferentes conceptos del término SDR es por ello que sólo se hará mención a 3de sus definiciones más importantes. [35].

- SDR es una radio sustancialmente definida por software y cuyo comportamiento en la capa física del modelo OSI puede ser considerablemente alterado a través de cambios en su software.
- SDR es un término utilizado para describir software de control para una aplicación de radio el cual provee técnicas de modulación, operaciones de banda angosta y banda ancha, funciones de seguridad en comunicaciones y requerimientos de forma de onda.
- SDR es un sistema de comunicación por radio en donde los componentes que se han implementado por hardware (por ejemplo, mezcladores, filtros, amplificadores,

moduladores, demoduladores, detectores, etc.,) ahora se implementarán utilizando software en una computadora u otros dispositivos SDR es una tecnología creada para mejorar la interoperabilidad entre diferentes servicios. La tecnología SDR está compuesta de software y hardware, puede ser reconfigurada dinámicamente para habilitar comunicaciones entre una amplia variedad de normas de comunicaciones, protocolos y radio enlaces. SDR permite crear dispositivos inalámbricos y equipo de redes multibanda y multifuncionales, que pueden ser dinámicamente reconfigurados, o a través de actualizaciones de software y reconfiguraciones de hardware. [35].

5.2.3.3 Niveles de SDR

Con el fin de establecer una clasificación de los equipos de radio, el Wireless Innovation Forum ha definido 5 niveles de utilización de software dentro del radio para controlar o realizar funciones de la capa física, determinando la frontera entre hardware y software en el equipo: [36].

- Nivel 0: Radio construido utilizando hardware solamente, no puede cambiarse por software. En este nivel no hay software, ni en el control ni en la realización propia de las funciones de operación del radio.
- Nivel 1 Radio controlado por software con limitaciones en cuanto a funciones controlables. Se controlan algunas como nivel de potencia, interconexiones, etc. pero nunca modo o frecuencia.
- Nivel 2: En este nivel una parte significativa del radio es configurable por software. Usualmente se utiliza el término Radio Controlado por Software (SCR). Existe control de software de ciertos parámetros como frecuencia, modulación, generación/detección de forma de onda, seguridad, etc. La etapa de RF permanece en hardware y no puede ser reconfigurada. Es importante destacar que el software en este tipo de radios sólo controla funciones que están implementadas de modo físico dentro del radio, a hardware únicamente.
- Nivel 3: Este nivel agrupa todos los radios en los que al menos una de sus funciones está definida por software, incluyendo lo que se conoce como Radio Definido por Software Ideal (ISR) donde la frontera entre la parte configurable y la no configurable se encuentra muy cercana a la antena, y la etapa final de RF es configurable. Se puede decir que el ISR es completamente programable.
- Nivel 4: Los equipos pertenecientes a esta categoría reciben la denominación de Ultimate Software Radio (USR) y constituyen una etapa superior al ISR puesto que no solo son completamente programables, sino que además son capaces de soportar un amplio rango de funciones y frecuencias simultáneamente. Dentro de este nivel se encuentran los teléfonos celulares con soporte para varios estándares. [36].

5.2.4 ASK (Codificación por Desplazamiento de Amplitud)

La señal ASK (también conocida como OOK) es una técnica de modulación digital en la que la información se encuentra modulada en amplitud. Es un caso particular de AM. ASK fue una de las primeras técnicas de modulación empleadas y antecedió a los sistemas analógicos de comunicación.

$$\int ASK(T) = f(t)\cos(\omega_c t)$$
$$\bullet \text{ si } f(t) = 0V \rightarrow fASK(t) = 0; \quad (1)$$
$$\bullet \text{ si } f(t) = AV \rightarrow fASK(t) = A\cos(\omega_c t);$$

Consiste en la conmutación de una señal senoidal portadora cos(ω_ct) de alta frecuencia a través de su encendido y apagado mediante una señal binaria unipolar f(t), la cual es el mensaje que se desea transmitir con niveles de tensión de 0 a A voltios y ancho de bit τb. Su función se representa en la siguiente fórmula.

En la Fig 5. Señales de la Modulación ASK. [37] se muestra la señal de información f(t) y la señal modulada ASK respectivamente. [37].

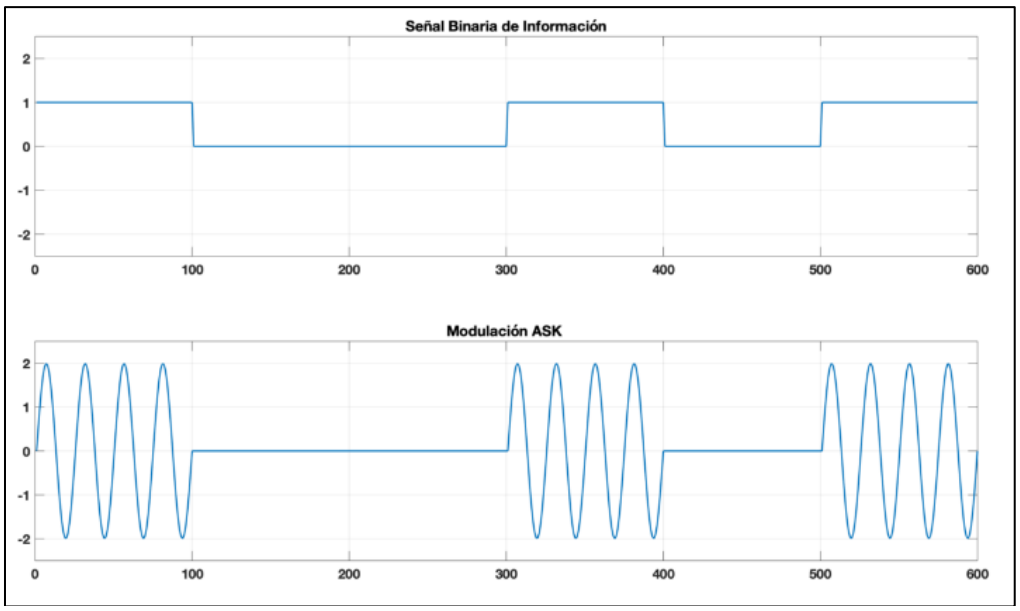


Fig 5. Señales de la Modulación ASK. [37]

5.2.4.1 Modulador ASK

La señal ASK se obtiene mediante el proceso de modulación que se muestra en la Fig 6, en la que se cuenta con un generador de señal senoidal, el mensaje binario y un filtro pasa bandas. [37].

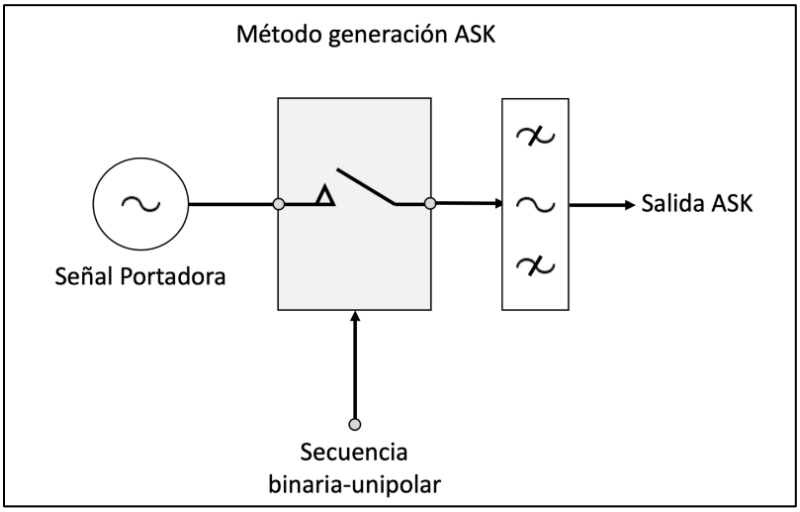


Fig 6. Modulador ASK. [37]

5.2.4.2 Métodos de Demodulación.

Puesto que la modulación ASK es un caso particular de la modulación AM, los demoduladores de AM son también válidos para la detección de señales ASK.

De esta manera, van a existir muchos métodos comunes de demodulación: detección síncrona y detección de envolvente. [37].

5.2.4.3 Método de Demodulación: Detección Síncrona.

Consiste en multiplicar (analógicamente) la señal ASK que llega al demodulador, por la señal portadora que se genera localmente en el receptor. La señal multiplicada resulta pasa por un filtro pasa bajas con frecuencia de corte menor que 2ω_c.

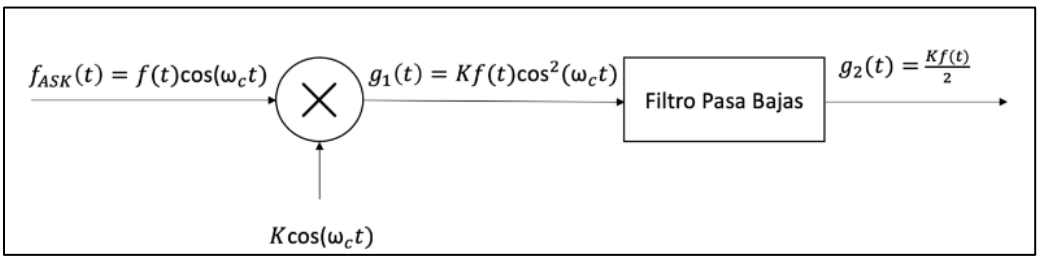


Fig 7. Demodulador ASK detección sincrónica. [37]

Este tipo de demodulación presenta de frecuencia y fase. Una solución para evitar este tipo de problemas consistiría en transmitir la señal problemas portadora junto con la señal moduladora (ASK), y recuperarla posteriormente en el receptor. Esta portadora puede ser transmitida de dos formas diferentes:

1) Enviada simplemente junto con la señal f_ASK(t). Para recuperar la señal portadora sería necesario hacer un paso filtrado de banda muy estrecha, que dejara pasar solamente la portadora, lo cual es difícil de conseguir.

2) Mandarla desplazada en frecuencia con respecto a la señal modulada. De esta manera la recuperación (filtrado paso de banda) es más sencilla. Por el contrario, presenta el inconveniente de que, al estar desplazada la señal portadora en el espectro de la frecuencia, ocupa un mayor ancho de banda. [37].

5.2.5 USRP N210

El USRP N210 proporciona capacidad de procesamiento de alto ancho de banda y alto rango dinámico. El USRP N210 está diseñado para aplicaciones de comunicaciones exigentes que requieren este tipo de desarrollo rápido. La arquitectura del producto incluye un FPGA Xilinx® Spartan® 3A-DSP 3400, ADC dual de 100 MS / s, DAC dual de 400 MS / sy conectividad Gigabit Ethernet para transmitir datos hacia y desde procesadores host. Un diseño modular permite que el USRP N210 funcione de CC a 6 GHz, mientras que un puerto de expansión permite sincronizar y utilizar varios dispositivos de la serie USRP N210 en una configuración MIMO. También se puede usar un módulo GPSDO opcional para controlar el reloj de referencia USRP N210 dentro de 0.01 ppm del estándar mundial de GPS. El USRP N210 puede transmitir hasta 50 MS / s hacia y desde aplicaciones host. Los usuarios pueden implementar funciones personalizadas en la estructura FPGA, o en el softcore RISC de 32 bits integrado. El USRP N210 proporciona un FPGA más grande que el USRP N200 para aplicaciones que exigen lógica, memoria y recursos DSP adicionales. La FPGA también ofrece el potencial de procesar hasta 100 MS / s tanto en la dirección de transmisión como en la de recepción. El firmware FPGA se puede recargar a través de la interfaz Gigabit Ethernet. [38].

5.2.6 Sistemas de Comunicaciones para Misión Crítica

Las comunicaciones de voz de misión crítica responden a situaciones en las cuales es fundamental salvar vidas, proteger la integridad física de las personas y mantener la operación de procesos industriales de alta relevancia.

Durante una catástrofe, desastre natural o ante una situación que afecte la seguridad pública, éstas ofrecen una respuesta rápida y eficiente a sus requerimientos de comunicación basados en sistemas seguros, robustos, interoperables y de alta disponibilidad. [39].

5.2.6.1 ¿Qué es Misión Crítica?

Llamamos aplicaciones de misión crítica a aquellas aplicaciones que tienen un impacto muy importante en el funcionamiento de la organización y cuya falla traería efectos terribles, lo que en Seguridad Pública se traduce en riesgo de vida.

Tanto en situaciones de rutina como extremas, los encargados de la seguridad pública necesitan comunicaciones confiables para hacer su trabajo y mantener a los ciudadanos y las comunidades seguras. Es decir, hablamos de un contexto en donde en algunos casos hay que tomar decisiones en fracción de segundos, y las comunicaciones no pueden fallar porque hay vidas en juego y cada segunda cuenta.

En la actualidad hay dos estándares de comunicaciones que cumplen con los requisitos de misión crítica: el APCO 25 o P25 y el TETRA. El primero es un estándar americano y el otro es un estándar europeo. No hay una tecnología mejor que la otra, sino que ambas tienen ventajas y desventajas según el tipo de aplicación. Por eso es necesario entender los requerimientos del cliente para poder definir la solución óptima en cada caso. [40].

5.2.6.2 ¿CÓMO SE IMPLEMENTA UN SISTEMA DE ESTE TIPO?

Recopilación de datos: es fundamental tener en cuenta la geografía del lugar y el área de cobertura, y recopilar todos los datos del usuario como ser modo de operación, cantidad de terminales y grupos, posibles lugares para implementar sitios de repetición, necesidad de transferencia de datos (localización de vehículos y personas), integración con sistemas existentes (sistema de atención de emergencias 911), etc.

Diseño de la solución: una vez recopilada esta información se define la tecnología y lugar de sitios de repetición, generando un área de cobertura en un software especial el cual va a desplegar en un mapa el área de trabajo para las distintos terminales (portátiles y móviles).

Este diseño incluye la infraestructura de los sitios, como ser Shelters, mástiles, sistemas de respaldo de energía, etc.

Implementación de la solución: una vez aceptado el diseño por parte de del cliente, se hacen los ajustes necesarios de la solución verificando algunos temas de campo, y se empieza con la infraestructura donde estarán los sitios de repetición los cuales darán cobertura a este sistema de misión crítica. Luego se seguirá con la instalación del equipamiento de repetición y más tarde la puesta en marcha de la solución. [40].

5.2.7 Métricas de Rendimiento de la Red

la calidad del servicio (QoS por sus siglas en inglés) establece los parámetros o métricas de una red de comunicaciones que miden la calidad y disponibilidad del sistema y de acuerdo

con estos, se garantiza o no, que el sistema es capaz de cubrir las necesidades de una aplicación en específico.

Algunos de estos parámetros se muestran a continuación. Cabe destacar que dependiendo de la aplicación un parámetro puede resultar más relevante que otro. [41].

5.2.7.1 Tasa Media de Éxito (THROUGHPUT)

Es la tasa media de éxito de la entrega de mensajes en un canal de comunicaciones. Está definido como el número total de bits de información exitosamente enviados sobre el tiempo que tardo el sistema en transmitir esa información, como se muestra en la ecuación: [41].

$$Thoughput = \frac{TIP * \sum PEE}{TDS} bps \quad (2)$$

Donde:

- PEE = Paquete exitosamente enviado
- TIP = Tamaño de la información de cada paquete,

sin tener en cuenta los bits de encabezado que añade el modelo OSI.

- TDS = Tiempo de simulación

5.2.7.2 Retardo Promedio (DELAY)

Es el espacio de tiempo entre el envío de un paquete o mensaje y su recepción por parte del destinatario (ecuación 3). Para su correcta medición se deben sumar todos los tiempos en que el mensaje dura ser procesado por los nodos intermedios en la transmisión. La ecuación 8 hace referencia al delay promedio de la red, al tener en cuenta, el delay de todos los paquetes recibidos correctamente. [41].

$$Delay = T.recibido - T.transmitido$$

$$Delay.prom = \frac{\sum \text{Delay paquetes recibidos}}{\text{Número total de paquetes recibidos}} \quad (3)$$

5.2.7.3. Fluctuación del Retardo (JITTER)

Uno de los principales problemas a los que está sometida una señal binaria transmitida a alta velocidad es lo que conocemos como jitter. La Unión Internacional de Telecomunicaciones (UIT) en su norma G. 701 define el jitter como las variaciones de corta duración no cumulativas de una señal digital con respecto a sus posiciones ideales en el tiempo. El punto de referencia tomado para la observación de dichas variaciones puede ser cualquier punto

conveniente y fácilmente identificable, como, por ejemplo, el borde izquierdo o derecho de un pulso o cualquier instante de muestreo. Es decir, el jitter (cuya traducción al español es temblor) se genera cuando el cruce por cero de una señal no se produce en el instante exacto en el que debería producirse. Por ejemplo, el tiempo de bit, t b, de una señal transmitida a 100 Mbps (ó Mb/s) es de 100 ns (nanosegundos), el mismo que puede calcularse simplemente al evaluar el inverso del ancho de banda, AB: de modo que, el cambio de un bit de uno a cero o viceversa debería idealmente producirse exactamente en múltiplos de 100 ns, pero si no es así, los múltiples adelantos o retardos en las transiciones generan el fenómeno conocido como jitter, mismo que puede provocar una degradación en el rendimiento de la transmisión al introducir errores en la lectura de los bits y desfases incontrolados en las señales digitales. [42].

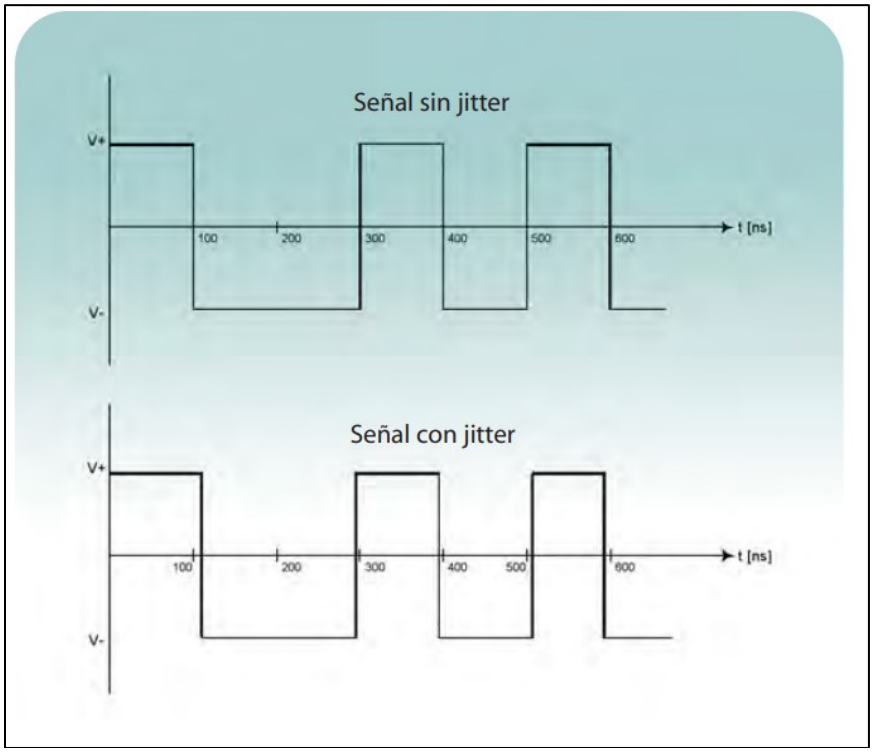


Fig 8. Comparación de una señal con Jitter y una señal sin Jitter. [42]

5.3. MARCO LEGAL

Para realizar el desarrollo de aplicaciones móviles se debe tener en consideración los criterios legales, para evitar sanciones legales, proteger la App para evitar la malversación de esta.

Algunas normas de la IEEE para la electrónica y los dispositivos digitales. [43].

- IEEE 12207 “Norma para la Tecnología de la Información - Software Procesos del ciclo de vida”

- IEEE 830 “prácticas recomendadas para la creación de especificaciones de requisitos de software”
- IEEE 1028 “revisión de software antes del lanzamiento formal”
- IEEE 1233 “Guía para el desarrollo de especificaciones de requisitos de software”. [43].

El espectro radioeléctrico es considerado a nivel mundial como un recurso limitado, por tal razón el uso de este espectro radioeléctrico es regulado de manera local en cada país.

Para Colombia el espectro radioeléctrico es asignado a la aplicación y ejecución de varias políticas públicas ejecutadas por el ministerio de las telecomunicaciones, las políticas son las siguientes:

Permisos para el uso del Espectro Radioeléctrico a través de mecanismos de Selección Objetiva: Proceso mediante el cual el Ministerio de Tecnologías de la Información y las Comunicaciones otorga permiso para el uso del espectro radioeléctrico dentro del territorio nacional en las bandas y rangos de frecuencias según la disponibilidad y planeación del espectro. [44].

Permiso para el uso del Espectro Radioeléctrico para la atención y prevención de situaciones de emergencia: Es el mecanismo a través del cual el Ministerio de las Tecnologías de la Información y las Comunicaciones otorgar o modificar permisos para el uso del espectro radioeléctrico para la defensa nacional, atención y prevención de situaciones de emergencia y seguridad pública en las bandas y rangos de frecuencias según la disponibilidad y planeación del espectro. [44].

Permisos temporales para el uso del espectro radioeléctrico: Este permiso otorgar directamente permisos para el uso temporal del espectro radioeléctrico, para garantizar la continuidad en la prestación del servicio de provisión de redes y servicios de telecomunicaciones. [44].

En Colombia a partir de la constitución política de 1991 existen leyes para la utilización y asignación del espectro:

- **Artículo 75**

El espectro electromagnético es un bien público inalienable e imprescriptible sujeto a la gestión y control del Estado. Se garantiza la igualdad de oportunidades en el acceso a su uso en los términos que fije la ley.

Para garantizar el pluralismo informativo y la competencia, el Estado intervendrá por mandato de la ley para evitar las prácticas monopolísticas en el uso del espectro electromagnético. [45].

- **Artículo 76**

La intervención estatal en el espectro electromagnético utilizado para los servicios de televisión estará a cargo de un organismo de derecho público con personería jurídica, autonomía administrativa, patrimonial y técnica, sujeto a un régimen legal propio. Dicho organismo desarrollará y ejecutará los planes y programas del Estado en el servicio a que hace referencia el inciso anterior. [46].

- **Ley 1341 o Ley de TIC**

Los servicios de radiodifusión sonora en Colombia se deben amparar en los valores primarios de la nación con este fin se puede dar democracia e igualdad. En el momento de transmitir por medio de los servicios de radiodifusión sonora no podrá hacerse transmisiones que atenten en contra la vida y leyes del país. [46].

- **Ley 1273 de 2009**

"El 5 de enero de 2009, el Congreso de la República de Colombia promulgó la Ley 1273 "Por medio del cual se modifica el Código Penal, se crea un nuevo bien jurídico tutelado denominado "De la Protección de la información y de los datos" y se preservan integralmente los sistemas que utilicen las tecnologías de la información y las comunicaciones, entre otras disposiciones. Dicha ley tipifica como delitos una serie de conductas relacionadas con el manejo de datos personales, por lo que es de gran importancia que las empresas se blinden jurídicamente para evitar incurrir en alguno de estos tipos penales". [47].

- **Ley 1266 de 2008**

Por la cual se dictan las disposiciones generales del habeas data y se regula el manejo de la información contenida en bases de datos personales, en especial la financiera, crediticia, comercial, de servicios y la proveniente de terceros países y se dictan otras disposiciones". [48].

- **Ley 1581 de 2012**

Mediante la Ley 1581 de 2012 se expidió el Régimen General de Protección de Datos Personales, el cual, de conformidad con su artículo 1, tiene por objeto "desarrollar el derecho constitucional que tienen todas las personas a conocer, actualizar y rectificar las informaciones que se hayan recogido sobre ellas en bases de datos o archivos, y los demás derechos, libertades y garantías constitucionales a que se refiere el artículo 15 de la Constitución Política; así como el derecho a la información consagrado en el artículo 20 de la misma. [49].

- **Ley 1928 de 2018**

Mediante la presente Ley, el Gobierno colombiano ha incorporado a su cuerpo normativo interno el Convenio sobre Ciberdelincuencia, adoptado el 23 de noviembre de 2001 en Budapest y en vigor desde julio de 2004. [50].

- **Ley 527 de 1999**

Por medio de la cual se define y reglamenta el acceso y uso de los mensajes de datos, del comercio electrónico y de las firmas digitales, y se establecen las entidades de certificación y se dictan otras disposiciones. [51].

6. METODOLOGIA

Para la realizar el proyecto consta de dos bloques fundamentales los cuales son el transmisor y el receptor en el cual se utiliza el equipo USRP-N210 para realizar la comunicación inalámbrica, para lograr la encriptación se utiliza el programa Python en el cual se desarrolla algoritmo de cifrado encargado de encriptar, y desencriptar la información enviada en el transmisor y receptor respectivamente de manera híbrida, es decir, primero se realiza la encriptación con autenticación (protección por contraseña) de la llave de sesión AES de manera asimétrica con el algoritmo RSA , ya que este algoritmo al ser muy seguro por manejar llave pública para encriptar puede ir por un medio no seguro y llave privada que sólo se usa para desencriptar y está alojada en el receptor, pero en contraste sacrifica rendimiento solo se encripta la llave de sesión, con la cual se va a cifrar la información; y este a su vez se usa en el algoritmo AES para cifrar la información de texto en claro que es de mayor rendimiento, pero que, por sí solo es inseguro y más vulnerable a ataques.

Se hace el envío previo de la llave publica cifrada con contraseña por el medio no seguro del receptor al transmisor. En el transmisor se crea una interfaz de usuario, el cual además de cargar el archivo a cifrar se encarga de capturar la contraseña de desencriptación de la llave pública RSA, como se puede detallar en la Fig 9.



Fig 55. Interfaz de usuario donde se carga el archivo de texto en claro y contraseña para desencriptar la llave pública RSA. [autoría propia]

A continuación, se describe de una manera detallada el funcionamiento del algoritmo:

- Cifrado de llave de sesión AES con el algoritmo asimétrico RSA. Se evidencia en la Fig 10.

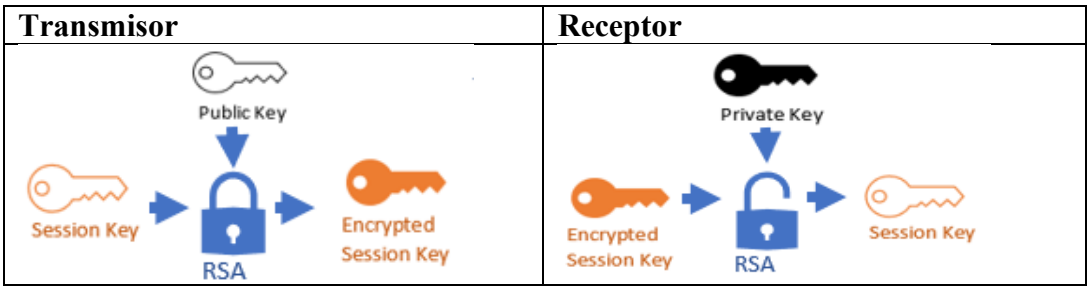


Fig 56. Esquema de cifrado asimétrico de llave de sesión AES. [autoría propia]

- Cifrado de la información de texto en claro por el algoritmo simétrico AES. Se evidencia en la Fig 11.

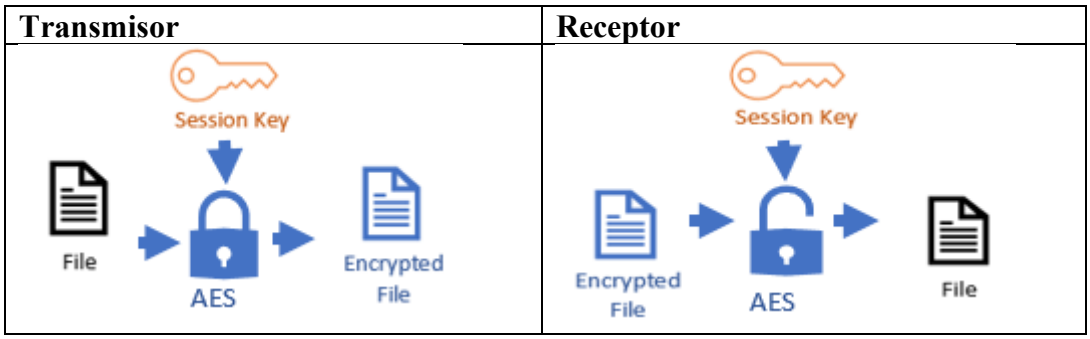


Fig 57. Esquema de cifrado simétrico de texto en claro. [autoría propia]

Para realizar la comunicación se desarrolla un diagrama de bloques en el programa GNU Radio el cual permite interactuar con los equipos USRP-N210. Para ello se realiza un flujograma con modulación ASK, es primordial que se haga con un sistema de modulación digital ya que la información del texto cifrado tiene que llegar con los bits idénticos, tanto en el transmisor como en el receptor. Este algoritmo al detectar, aunque sea un error de un bit en toda la trama de bits del contenido del texto cifrado provoca error y no descifra.

Para este proyecto, se usó modulación digital ASK ya que fue el método más sencillo de interpretar e implementar, sin embargo, se puede desarrollar con cualquier tipo de modulación digital que garantice la integridad de los bits tanto en la transmisión como en la recepción.

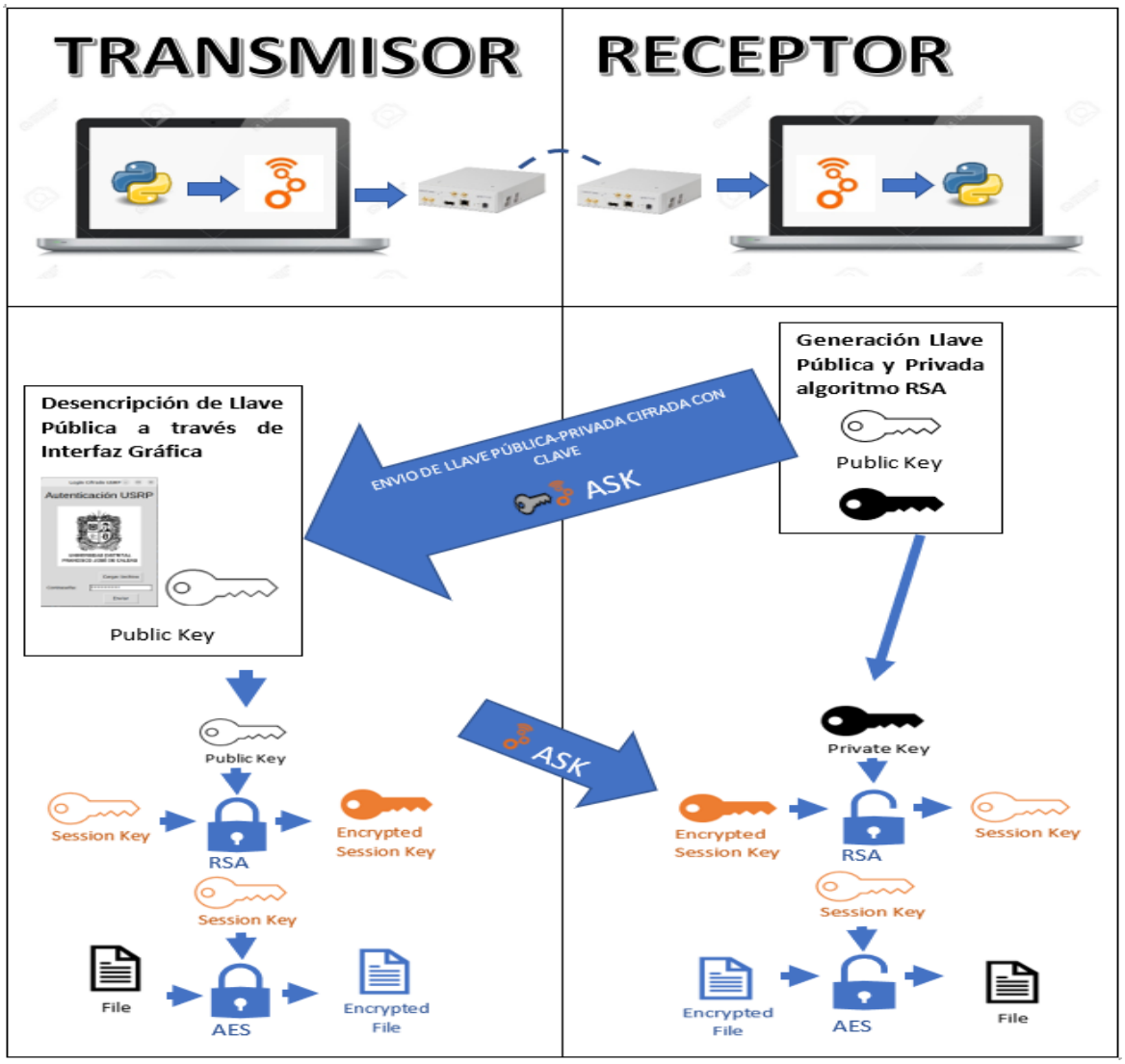


Fig 58. Metodología de trabajo. [autoría propia]

En la Fig 12, se evidencia el envío, primero la llave pública RSA encriptada, la cual se desencripta por contraseña, luego con esa llave se encripta la llave de sesión AES, y siguiente se cifra la información en AES. En el receptor, se desencripta la llave de sesión con la llave privada y con esa llave se desencripta la información

El algoritmo de encriptación, desencriptación y la interfaz gráfica se ejecutan en Python versión 3.104 64-bit, la comunicación entre los dispositivos de transmisión y recepción USRP-N210 se desarrolló en el programa GNU Radio versión 3.10.1.1.

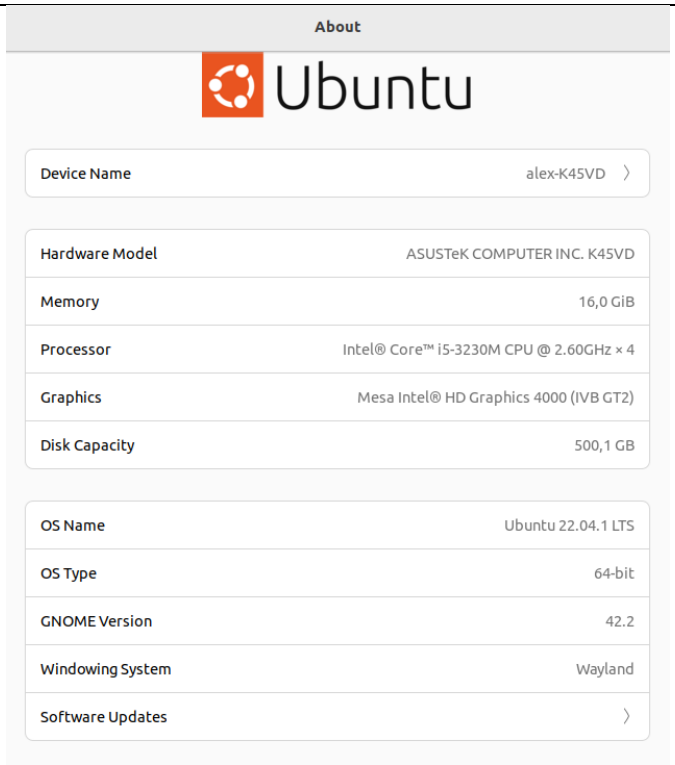
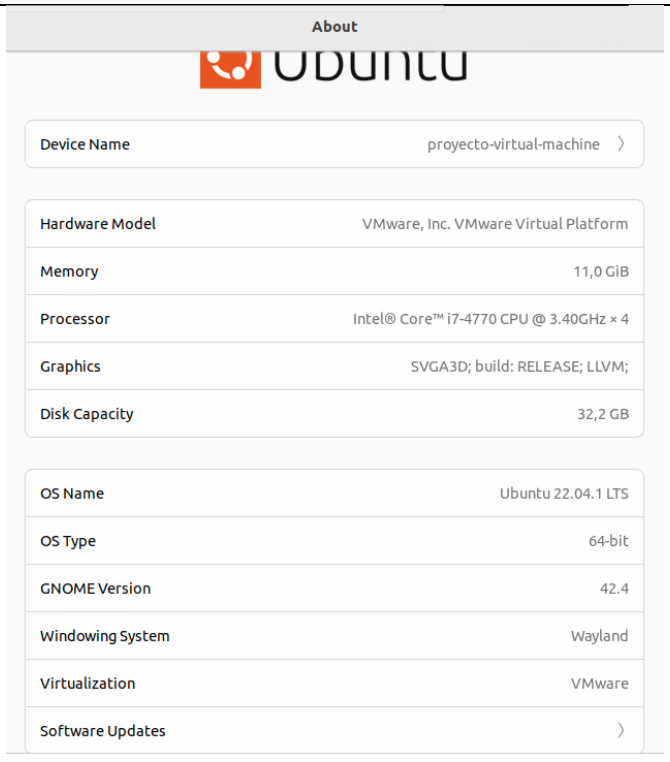
Transmisor (Máquina Nativa)	
Receptor (Máquina Virtual)	

Fig 59. Máquinas usadas para el proyecto. [autoría propia]

6.1 Scripts de Trabajo

Para el desarrollo de la metodología de trabajo, se desarrollaron un total de 17 scripts, 7 para el envío de la llave pública del Receptor al Transmisor, en los cuales se ejecuta momento previo al proceso previo del envío de la información.

SCRIPT MAESTRO	MASTER KEY TX	MASTER KEY RX
FASE DE ENCRIPTACION *(Scripts)*		• Gen_RSA_Keys.py
FASE DE COMUNICACIÓN *(Scripts)*	• slicer2.py • PublicKeyRX.py	• Filler2.py • PublicKeyTX.py

Tabla 2. Organización de los scripts de envío de llave pública encriptada RSA. [Autoría propia]

Para la desencriptación de la llave pública, la encriptación de la información de texto en claro de manera híbrida (RSA/AES), envío de la información cifrada y posterior desencriptación en el receptor se usan los siguientes 10 scripts, los cuales se ejecutan una vez para el proceso de envío como se exhibe en la Tabla 3. Organización de los Scripts Desencriptación de llave pública y posterior envío de la información cifrada de manera híbrida.

SCRIPT MAESTRO	MÁSTER TX	MÁSTER RX
FASE DE ENCRIPTACION *(Scripts)*	• GUI.py • Des_Public_Key.py • Encryption.py	• Desencryption.py
FASE DE COMUNICACIÓN *(Scripts)*	• Filler.py • USRP_TX.py	• Slicer.py • USRP_RX.py

Tabla 3. Organización de los Scripts Desencriptación de llave pública y posterior envío de la información cifrada de manera híbrida. [autoría propia]

6.2 Autenticación de Llave Pública RSA y Cifrado de Llave de Sesión AES

Para este apartado se toman 13 scripts organizados de la manera como se ilustra en la Tabla 4 y Tabla 5. De los cuales, solo 5 se usan para la fase de encriptación.

El Transmisor se debe ejecutar primero para recibir la llave RSA encriptada, y unos segundos después se debe ejecutar el receptor enviando dicha llave, tal como se evidencia en la Tabla 4.

SCRIPT MAESTRO	MASTER KEY TX	MASTER KEY RX
FASE DE ENCRIPTACION *(Scripts)*		• Gen_RSA_Keys.py
FASE DE COMUNICACIÓN *(Scripts)*	• slicer2.py • PublicKeyRX.py	• Filler2.py • PublicKeyTX.py

Tabla 4. Primera ejecución simultánea. [autoría propia]

Primero se ejecuta el receptor que estará preparado para recibir, unos segundos después se ejecuta el transmisor que enviará la información cifrada, tal como se evidencia en la Tabla 5.

SCRIPT MAESTRO	MÁSTER TX	MÁSTER RX
FASE DE ENCRIPTACION *(Scripts)*	• GUI.py • Des_Public_Key.py • Encryption.py	• Desencryption.py
FASE DE COMUNICACIÓN *(Scripts)*	• Filler.py • USRP_TX.py	• Slicer.py • USRP_RX.py

Tabla 5. Segunda ejecución simultánea. [autoría propia]

A continuación, se detallarán los algoritmos primordiales para este proceso, exceptuando la fase de comunicación que se explicará más adelante. Para el diseño de los siguientes algoritmos, se tuvo en cuenta la librería **Pycryptodome,** donde se toman los diseños sugeridos.

- **Gen_RSA_Keys.py:** En este algoritmo se genera las llaves pública y privada RSA en el receptor:

```
1  from Crypto.PublicKey import RSA
2
3  secret_code = "Proyecto123"
4  key = RSA.generate(2048)
5  encrypted_key = key.export_key(passphrase=secret_code, pkcs=8,
6                                 protection="scryptAndAES128-
   CBC")
7
8  private_key = key.export_key()
9  file_out = open("private.pem", "wb")
10 file_out.write(private_key)
11 file_out.close()
12
13 file_out = open("rsa_key.bin", "wb")
14 file_out.write(encrypted_key)
15 file_out.close()
```

```
16
17 print(key.publickey().export_key())
```

Fig 60. Script gen_rsa_keys.py. [autoría propia]

En la Fig. 14. se puede evidenciar en el algoritmo en la variable secret_code se guarda la la contraseña y por medio de las líneas 5 y 6 se genera la llave pública y privada encriptada con dicha clave.

Una vez generada la llave publica y llave privada encriptada con una contraseña la cual se evidencia en la variable `secret_code = "Proyecto123" ,` estas llaves se guardan en un archivo el cual tiene como nombre `rsa_key.bin`, este archivo se envía desde el receptor hacia el transmisor por medio del programa GNU Radio.

- **GUI.py *(Interfaz Gráfica):*** Por medio de la interfaz gráfica le pedimos al usuario que ponga la contraseña para poder obtener la clave pública RSA en el transmisor. Una vez puesta esta contraseña, se pueden ejecutar varias comunicaciones sin tener que volver a poner dicha contraseña. La contraseña se exporta a un archivo `password.txt.`



Fig 61. Interfaz gráfica para la autenticación. [autoría propia]

En la Fig 15, Se evidencia la Interfaz Gráfica como se realiza la autenticación la cual se hace por contraseña para la obtención de la clave pública RSA en el transmisor.

- **Des_Public_Key.py:** Mediante este algoritmo se obtiene la llave publica a través de la contraseña ingresada por el usuario en la interfaz gráfica, esta se obtiene importando el archivo `password.txt` y la llave importando el archivo `rsa_key.bin.` y luego se exporta en el archivo receiver.pem. tal como se evidencia en la Fig 16.

```
1  from Crypto.PublicKey import RSA
2
3  with open("password.txt",encoding='utf-8') as f:
4      secret_code = f.read()
```

```
encoded_key = open("rsa_key.bin", "rb").read()
key = RSA.import_key(encoded_key, passphrase=secret_code)

#print(key.publickey().export_key())
public_key = key.publickey().export_key() #Genera una llave pública
print(public_key)
file_out = open("receiver.pem", "wb")
file_out.write(public_key)
file_out.close()
```

Fig 62. Script Des_Public_Key.py. [autoría propia]

- **Encryption.py:** Por medio de este algoritmo Fig 17, Se puede evidencias como se cifra la llave de sesión con la llave pública RSA. Luego se crea el archivo `encrypted_data.bin` donde se guarda el texto cifrado, luego se importa la llave pública en la variable `recipient_key`, posteriormente se genera la llave de sesión AES `session_key`, en `enc_session_key` se encripta la llave de sesión con la llave pública y se guardan las variables en el archivo `encrypted_data.bin`.

```
from Crypto.PublicKey import RSA
from Crypto.Random import get_random_bytes
from Crypto.Cipher import AES, PKCS1_OAEP

#Importa el archivo a cifrar
data = open("data.bin", "rb")
datos = data.read()

# Crea el archivo en donde se va a guardar el texto cifrado
file_out = open("encrypted_data.bin", "wb")

# Importa la llave pública del archivo receiver.pem
recipient_key = RSA.import_key(open("receiver.pem").read())
# Genera una llave de sesión AES
session_key = get_random_bytes(16)

# Encrypt the session key with the public RSA key
# Crea un objeto para cifrar la llave de sesión con la llave pública
cipher_rsa = PKCS1_OAEP.new(recipient_key)
#Encripta la llave de sesión
enc_session_key = cipher_rsa.encrypt(session_key)

# Encrypt the data with the AES session key
# Crea un objeto AES para el cifrado con la llave de sesión
cipher_aes = AES.new(session_key, AES.MODE_EAX)
# Cifra la información con un tag del nombre de archivo y su contenido
#enc_NameFile, tag = cipher_aes.encrypt_and_digest(nameFile)
ciphertext, tag = cipher_aes.encrypt_and_digest(datos)

# Exporta la información en un archivo binario
# Guarda la llave pública encriptada, cipher.nonce, tag, el texto cifrado
```

```
[file_out.write(x) for x in (enc_session_key,
cipher_aes.nonce,tag,ciphertext)]
file_out.close()

print("Archivo cifrado exitosamente")
```

Fig. 63 Script Encryption.py**.** [autoría propia]

En la Fig 17, Resalta el proceso de encriptación de la llave de sesión AES.

- **Desencryption.py:** Mediante este algoritmo se descifra la llave de sesión con la llave privada alojada en el receptor. Se evidencia como se realiza la extracción del archivo encriptado mediante la sentencia `file_in = open ("encrypted_data.bin", "rb")`, despues se importa la llave privada en la variable `private_key`, se extrae el archivo encriptado y se divide en cuatro variables, en la primera variable `enc_session_key` es la llave de sesion encriptada. Con la llave privada se desencripta la llave de sesión AES.

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import AES, PKCS1_OAEP
import subprocess

file_in = open("encrypted_data.bin", "rb")

# Importa la llave privada
private_key = RSA.import_key(open("private.pem").read())

#Importa la llave de sesión encriptada, el nonce, el tag,
# el nombre de archivo y el texto cifrados
enc_session_key, nonce, tag, ciphertext = \
    [file_in.read(x) for x in (private_key.size_in_bytes(
    ), 16, 16, -1)]

# Decrypt the session key with the private RSA key
# Crea un objeto con el cual descifra la llave de sesión con la
llave privada
cipher_rsa = PKCS1_OAEP.new(private_key)
# Descifra la llave de sesión
session_key = cipher_rsa.decrypt(enc_session_key)

# Decrypt the data with the AES session key
# Crea un objeto para descifrar con el nonce traído del archivo
encriptado
#cipher_aes = AES.new(session_key, AES.MODE_EAX, nonce)
cipher_aes = AES.new(session_key, AES.MODE_EAX, nonce)
# Descifra la información
#nameFile = cipher_aes.decrypt_and_verify(enc_NameFile, tag)
data = cipher_aes.decrypt_and_verify(ciphertext, tag)

nameFile, data = data[:30],data[30:-1]

#Decodifica el nombre de archivo
nameFile = nameFile.decode("utf-8")
```

```
34
35  #Busca y recorta el nombre de archivo
36  index = 0
37  for i in nameFile[::-1]:
38      if(i == "/"):
39          break
40      index = index - 1
41
42  #Sobreescribe y cierra el archivo
43  file_out = open(nameFile[index:], "wb")
44  file_out.write(data)
45  file_out.close()
46
47  #Reproduce el archivo
48  if nameFile[-3:] == "jpg" or nameFile[-3:] == "peg" or nameFile[-
    3:] == "png":
49      print(subprocess.run("eog " + nameFile[index:], shell=True))
50  elif nameFile[-3:] == "mp3" or nameFile[-3:] == "wav":
51      print(subprocess.run("totem " + nameFile[index:], shell=True))
52
53  print("Archivo descifrado exitosamente")
```

Fig 64. Script Desencryption.py. [autoría propia]

En la Fig 18, Se resalta el proceso de desencriptación de la llave de sesión AES.

6.3 Cifrado de Texto Plano con Algoritmo de Cifrado Simétrico AES

Para este apartado se toman 9 scripts organizados de la manera como se ilustra en la Tabla 6. de los cuales, solo 3 se usan para la fase de encriptación.

SCRIPT MAESTRO	MÁSTER TX	MÁSTER RX
FASE DE ENCRIPTACION *(Scripts)*	• GUI.py • Encryption.py	• Desencryption.py
FASE DE COMUNICACIÓN *(Scripts)*	• Filler.py • USRP_TX.py	• Slicer.py • USRP_RX.py

Tabla 6. Ejecución Simultánea proceso de cifrado simétrico AES. [autoría propia]

Primero se ejecuta el receptor que estará preparado para recibir, unos segundos después se ejecuta el transmisor que enviará la información cifrada, tal como se evidencia en la Tabla 6.

A continuación, se detallarán los algoritmos primordiales para este proceso, exceptuando la fase de comunicación que se explicará más adelante. Para el diseño de los siguientes

algoritmos, se tuvo en cuenta la librería **Pycryptodome,** donde se toman los diseños sugeridos.

- **GUI.py *(Interfaz Gráfica):*** En la misma interfaz de usuario usada para autenticarse por contraseña se le pide al usuario que seleccione un archivo del sistema, tamaño recomendado menor a 400 kB, tal como se evidencia en la Fig 19.

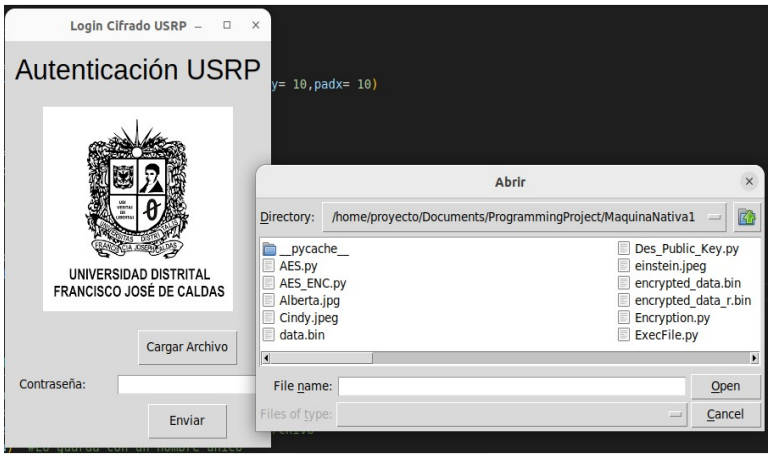


Fig 65, Interfaz de usuario solicitando la ubicación del archivo a encriptar. [autoría propia]

- **Encryption.py:** Por medio de este algoritmo se implementa la encriptación del texto claro en AES. Se importa el archivo `data.bin`, en el cual se encuentra el texto plano. Se encripta la información del texto plano en las variables `ciphertext`, `tag` y `cipher_aes.nonce` con la llave de sesión `session_key`, mediante las líneas 32 y 33 se guardan las variables en el archivo `encrypted_data.bin`.

```
1  from Crypto.PublicKey import RSA
2  from Crypto.Random import get_random_bytes
3  from Crypto.Cipher import AES, PKCS1_OAEP
4
5  #Importa el archivo a cifrar
6  data = open("data.bin", "rb")
7  datos = data.read()
8
9  # Crea el archivo en donde se va a guardar el texto cifrado
10 file_out = open("encrypted_data.bin", "wb")
11
12 # Importa la llave pública del archivo receiver.pem
13 recipient_key = RSA.import_key(open("receiver.pem").read())
14 # Genera una llave de sesión AES
15 session_key = get_random_bytes(16)
16
17 # Encrypt the session key with the public RSA key
18 # Crea un objeto para cifrar la llave de sesión con la llave pública
19 cipher_rsa = PKCS1_OAEP.new(recipient_key)
20 #Encripta la llave de sesión
21 enc_session_key = cipher_rsa.encrypt(session_key)
```

```
22
23  # Encrypt the data with the AES session key
24  # Crea un objeto AES para el cifrado con la llave de sesión
25  cipher_aes = AES.new(session_key, AES.MODE_EAX)
26  # Cifra la información con un tag del nombre de archivo y su
    contenido
27  #enc_NameFile, tag = cipher_aes.encrypt_and_digest(nameFile)
28  ciphertext, tag = cipher_aes.encrypt_and_digest(datos)
29
30  # Exporta la información en un archivo binario
31  # Guarda la llave pública encriptada, cipher.nonce, tag, el texto
    cifrado
32  [file_out.write(x) for x in (enc_session_key,
    cipher_aes.nonce,tag,ciphertext)]
33  file_out.close()
34
35  print("Archivo cifrado exitosamente")
```

Fig 66. Script Encryption.py. [autoría propia]

En la Fig 20, Se evidencia el proceso de encriptación del texto en claro, de la variable datos.

- **Desencryption.py:** Por medio de este algoritmo se desencripta el texto en claro después de haber sido desencriptada la llave de sesión por medio de la llave privada RSA. Se evidencia como se realiza la extracción del archivo encriptado mediante la sentencia `file_in = open("encrypted_data.bin", "rb")`, despues se importa la llave privada en la variable `private_key`, se extrae el archivo encriptado y se divide en cuatro variables, en la primera variable `enc_session_key` es la llave de sesión encriptada, la segunda variable `nonce` y la tercera variable `tag` que se utilizan para desencriptar el texto plano, y la variable `ciphertext` que es en donde se encuentra el contenido del archivo encriptado. Con la variable `cipher_aes` se crea un objeto con el cual se puede desencriptar el texto cifrado en la variable `data`. Hasta aquí se ha realizado la desencriptacion correctamente.

  Con la variable `data` se toman los primeros 30 Bytes para tomar la ruta original del archivo en la variable `nameFile`, luego se convierte el tipo de dato de Bytes a String con la funcion `decode("utf-8") y` se hace un ciclo For para tomar el nombre del archivo. Los Bytes restantes se guardan en la variable `data` que son los que finalmente obtienen la informacion del archivo. En la linea 43 con la funcion `open` se nombra el archivo con el extraido del For, con la variable `write` se guarda el contenido del archivo. Finalmente se cierra el archivo. Otra opción adicional a este Script es reproducir el archivo mediante los if finales que toman el tipo de archivo y según el tipo de archivo, si es una imagen o foto .JPG o .JPEG (ver **Error! Reference source not found.**) la visualiza, si es un audio o canción tipo MP3 la reproduce (ver **Error! Reference source not found.**).

```
1  from Crypto.PublicKey import RSA
2  from Crypto.Cipher import AES, PKCS1_OAEP
3  import subprocess
```

```
4
5  file_in = open("encrypted_data.bin", "rb")
6
7  # Importa la llave privada
8  private_key = RSA.import_key(open("private.pem").read())
9
10 #Importa la llave de sesión encriptada, el nonce, el tag,
11 # el nombre de archivo y el texto cifrados
12 enc_session_key, nonce, tag, ciphertext = \
13 [file_in.read(x) for x in (private_key.size_in_bytes(
14 ), 16, 16, -1)]
15
16 # Decrypt the session key with the private RSA key
17 # Crea un objeto con el cual descifra la llave de sesión con la
   llave privada
18 cipher_rsa = PKCS1_OAEP.new(private_key)
19 # Descifra la llave de sesión
20 session_key = cipher_rsa.decrypt(enc_session_key)
21
22 # Decrypt the data with the AES session key
23 # Crea un objeto para descifrar con el nonce traído del archivo
   encriptado
24 #cipher_aes = AES.new(session_key, AES.MODE_EAX, nonce)
25 cipher_aes = AES.new(session_key, AES.MODE_EAX, nonce)
26 # Descifra la información
27 #nameFile = cipher_aes.decrypt_and_verify(enc_NameFile, tag)
28 data = cipher_aes.decrypt_and_verify(ciphertext, tag)
29
30 nameFile, data = data[:30],data[30:-1]
31
32 #Decodifica el nombre de archivo
33 nameFile = nameFile.decode("utf-8")
34
35 #Busca y recorta el nombre de archivo
36 index = 0
37 for i in nameFile[::-1]:
38 if(i == "/"):
39 break
40 index = index - 1
41
42 #Sobreescribe y cierra el archivo
43 file_out = open(nameFile[index:], "wb")
44 file_out.write(data)
45 file_out.close()
46
47 #Reproduce el archivo
48 if nameFile[-3:] == "jpg" or nameFile[-3:] == "peg" or nameFile[-
   3:] == "png":
49 print(subprocess.run("eog " + nameFile[index:], shell=True))
50 elif nameFile[-3:] == "mp3" or nameFile[-3:] == "wav":
51 print(subprocess.run("totem " + nameFile[index:], shell=True))
52
53 print("Archivo descifrado exitosamente")
```

Fig 67. Script Desencryption.py. [autoría propia]

En la Fig 21, Se resalta en rojo el proceso de desencriptación del texto en claro, en verde la extracción del nombre y contenido del archivo y en azul la opción de reproducir el archivo si es de tipo .JPG, .JPEG, .PNG, .WAV, .MP3. tal como se evidencia en la **Error! Reference source not found.**. Se simulan con imágenes ya que, por la rata de bit manejada, es muy posible manejar este tipo de datos de para información crítica cómo fotos, etc. También se pueden manejar por su corto ancho de banda mensajes de texto muy parecido a la mensajería instantánea. Para este ejercicio se simuló con imágenes que se abren una vez recibida en la recepción



Fig 68. Reproducción de archivo .JPG, .JPEG, o .PNG. [autoría propia]

En la Fig 22, Se evidencia el archivo seleccionado el cual se abre una vez que se haya recibido.

También el uso esencial son los archivos de imagen, o enviar audio en tiempo real, que es la manera más práctica e histórica de manejar estos sistemas, sin embargo, en este caso, se debe manejar por modulación digital, ya que el cifrado es de tipo digital, para este ejercicio, dado al corto ancho de banda que contamos es muy posible enviar audio en tiempo real comprimido en formato .mp3, para este ejercicio se simuló enviando archivos de audio mp3 que se abren inmediatamente después de haberlo enviado ver Fig 23.

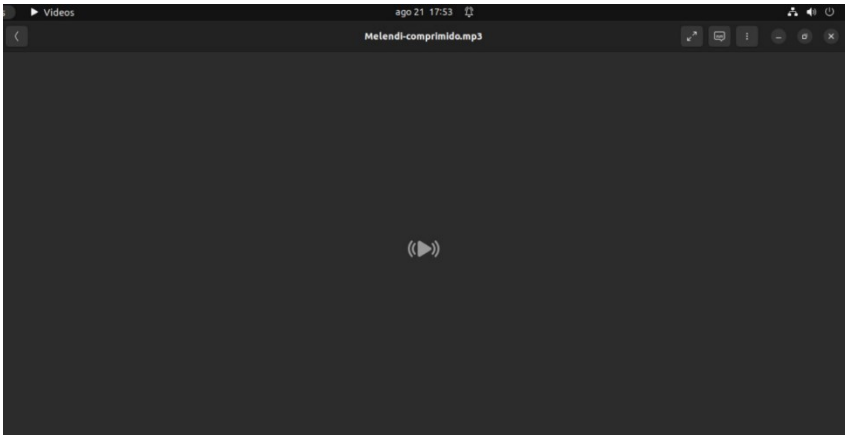


Fig 69. Reproducción de archivo .MP3 o .WAV. [autoría propia]

En la Fig 23, Se evidencia el archivo seleccionado el cual se abre una vez que se haya recibido.

6.4 Capa de Comunicación – GNU Radio

Para el proceso de la comunicación se usa el software GNU Radio que produce un archivo de Python tanto en emisor y el receptor. Aunque el presente trabajo su objetivo principal es la encriptación de la información que pasa a través de los radios, es fundamental garantizar la integridad de la información encriptada, puesto que, si el archivo en donde contiene la información cifrada llega con un error de bit, este no puede ser desencriptado en la recepción, por lo que se ve supremamente necesario implementar modulación digital, ya que este tipo de modulación logra recrear de manera idéntica en su organización binaria dicho archivo cifrado. La modulación que se usa en este caso es la modulación ASK, ya que se resolvió por un tipo de modulación sencilla de interpretar e implementar, pero que, sin lugar a duda, se puede implementar cualquier tipo de modulación que logre garantizar la integridad de la información del texto cifrado y así optimizar el proceso de comunicación.

6.4.1 Transmisor

En el transmisor se importa el archivo `encrypted_data_r.bin` donde está el archivo encriptado adecuado para la comunicación por GNU Radio a través del bloque **File Source**, después se separa la información por paquetes cada 256 bytes mediante el bloque **Stream to Tagged Stream**, por medio del bloque **Protocol Formatter** se le asigna una cabecera binaria de 32 bits, esto permitirá en la recepción sincronizar la información y detectar en donde a comenzar a capturar los paquetes, luego en el bloque **Tagged Stream Mux** se une la cabecera con el payload, hasta aquí se adapta la información para ser sincronizada en la recepción.

A partir de aquí se toma esta información y se adapta para el proceso de modulación, que en este caso es ASK. En el bloque **Unpack K Bits** se transforma la información de bytes a un

bit por símbolo, el bloque **Repeat**, toma cada muestra binaria y la interpola 8 veces a una velocidad de muestreo de 1,56 MHz, se prepara esta información para ser multiplicada por una señal senoidal de 781.5 kHz, por lo cual se crea la señal modulada ASK remuestreada a una tasa de 3.152 MHz lista para ser enviada por el dispositivo de radio USRP N-210.

Este flujograma maneja una rata de símbolo de 195.312 kbps. En la Fig 24, Se puede observar el diagrama de bloques del transmisor realizado:

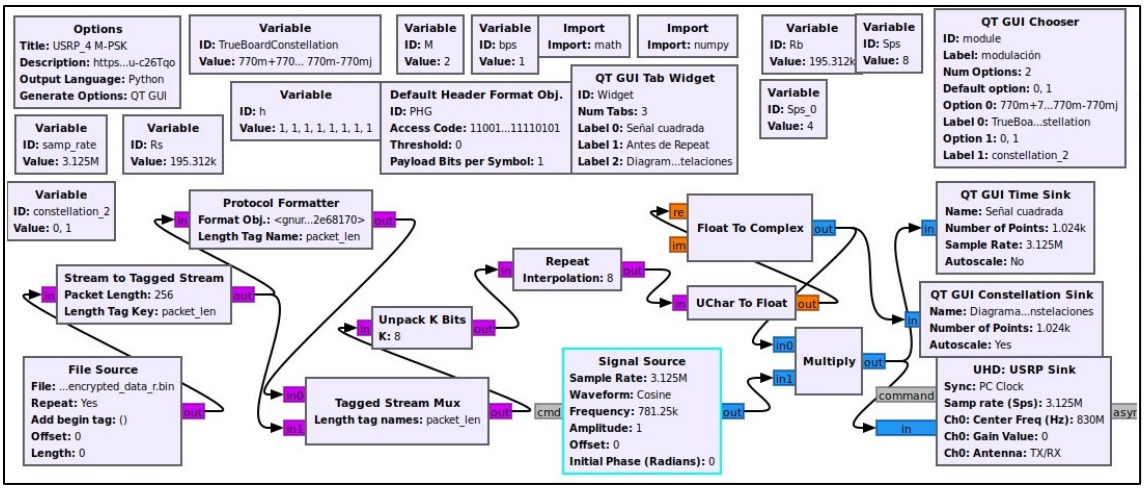


Fig 70. Transmisor GNU Radio. [autoría propia]

- **File Source:** En este bloque importa el archivo encriptado.
- **Stream to Tagged Stream:** En este bloque se genera los paquetes cada 256 Bytes, bajo el `packet_len`.
- **Protocol Formatter:** En este Bloque toma un objeto el cual se puede tomar del bloque llamada **Default Header Format Obj.** donde obtiene un código de acceso binario el cual se le va a agregar al encabezado de cada carga útil o payload de 256 Bytes en el cual fue divido el archivo.
- **Tagged Stream Mux:** En este Bloque une el encabezado del **Protocol Formatter** con el payload.
- **Ubpack K Bits:** En este bloque se convierte la información en Bytes a binario.
- **Repeat:** En este bloque interpola cada bit en ocho muestras.
- **Signal Source:** En este bloque se genera una señal coseno que va a representar a la portadora su frecuencia de transmisión es de 781.25 kHz.
- **Multiply:** En este bloque se multiplica la portadora con la señal cuadrada, generando una señal modula ASK.
- **Uchar To Float:** En este bloque se convierte la información en Bytes a dato tipo Flotante.
- **Float to Complex:** En este bloque la información de dato tipo Flotante a dato tipo Complejo.

- **UHD USRP Sink:** En este bloque se encarga de enviar la señal modulada al dispositivo USRP_N210. Se utiliza la frecuencia de transmisión de 830 MHz.

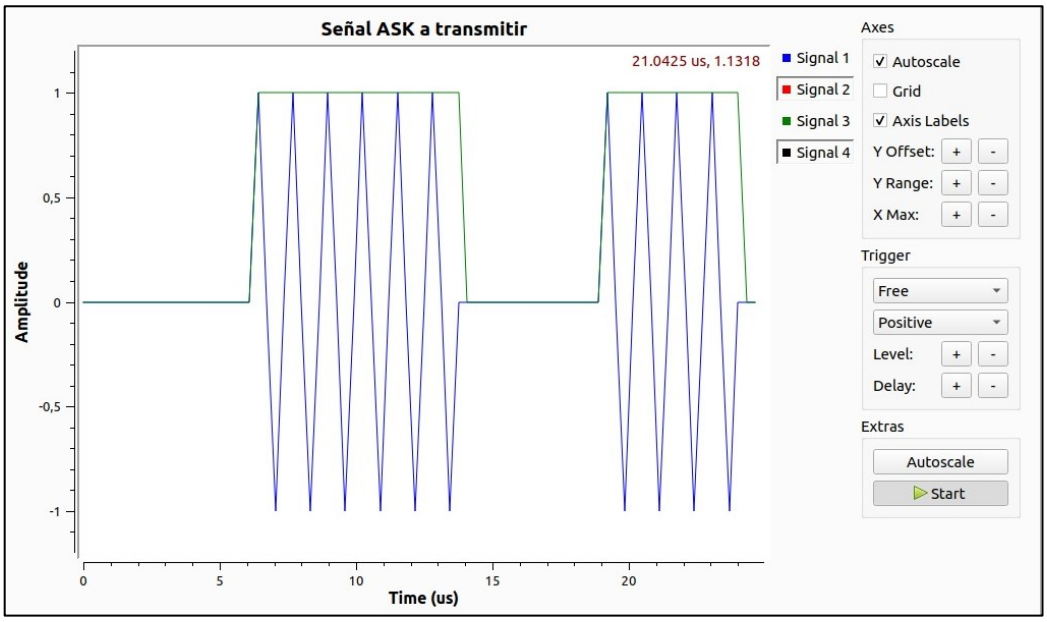


Fig 71. Señal modulada ASK. [autoría propia]

En la Fig 25, Se videncia la señal cuadrada en color verde vs la señal modulada en azul en la Modulación ASK.

6.4.2 Receptor

En la recepción, se toma la señal en el medio electromagnético sintonizando a la frecuencia de 830 MHz con una tasa de muestreo de 3.125 MHz desde el bloque **UHD: USRP Source,** luego se toman dichas muestras, se convierte a tipo flotante tomando el valor real para poder graficar la señal ASK recibida (véase en la Fig.15), también de las muestras se convierte a tipo de dato flotante pero esta vez no tomando su valor real sino su valor absoluto multiplicado al cuadrado para poder empezar el proceso de **demodulación**: Se hace pasar la señal a través del bloque **Band Pass Filter** para eliminar cualquier señal de ruido y de defectos que pueda tener la recepción USRP, se emplea el bloque **Threshold** para detectar la amplitud de la señal y poder traducir en 1 los valores altos y en 0 los valores bajos o que no emite señal, luego se hace pasar por el bloque **Clock Recovery MM** que sincroniza el stream y lo decima a una tasa de 8, volviendo a obtener la rata de bit de 195,312 kbps.

Una vez obteniendo la señal demodulada se procede a sincronizar los bits con el bloque **Correlate Access Code - Tag Stream** que una vez, detecta el código de acceso en la cabecera comienza a capturar los paquetes, luego en **Tagged Stream Align** alinea y ordena los paquetes uno por uno para evitar y/o minimizar errores, después en el bloque **Pack K**

Bits convierte la información de bits a bytes y en el bloque **File Sink** se obtiene en la recepción el texto cifrado en el archivo `encrypted_data_r.bin`.

En la Fig 26, Se puede observar el diagrama de bloques del receptor realizado:

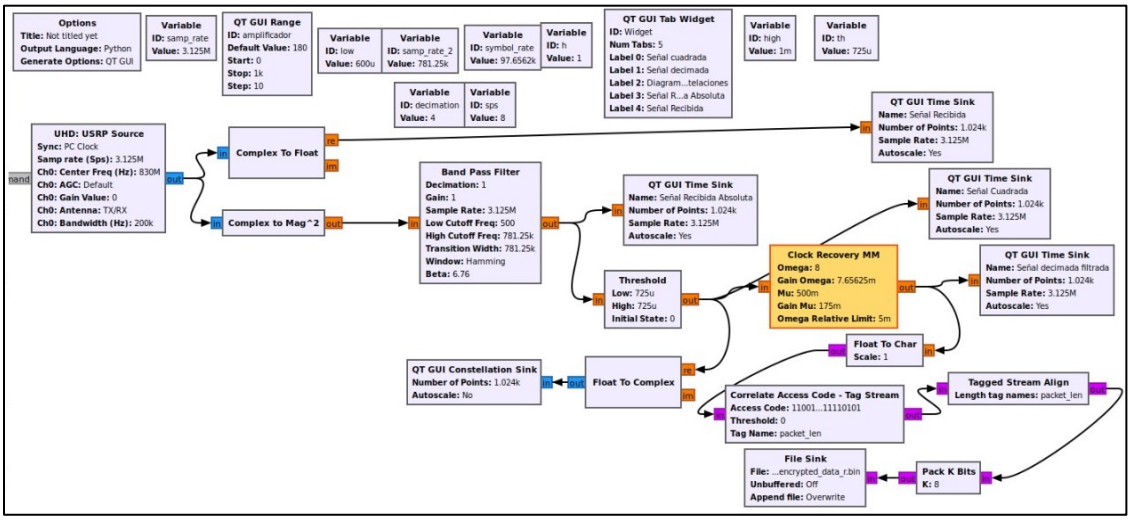


Fig 72. Receptor GNU radio. [autoría propia]

- **UHD: USRP Source:** En este bloque se realiza la recepción de la señal a través del dispositivo USRP-N210, la frecuencia de Recepción es de 830 kHz.
- **Band Pass Filter:** En este bloque se ejecuta el filtrado de la señal eliminando el ruido.
- **Complex to Mag^2:** En este bloque se convierte el tipo de dato Complejo al valor absoluto al cuadrado de este convertido en dato tipo Flotante.
- **Complex To Float:** En este bloque separa y convierte el dato de tipo Complejo en sus variables real e imaginaria de tipo Flotante. Se usa solo para visualizar la señal de entrada.
- **Threshold:** En este bloque toma un valor de umbral especificado. Si el valor de la señal sobrepasa High en la salida se obtiene 1, si el valor es menor que Low en salida se obtiene 0. Aquí se obtiene la señal cuadrada.
- **Clock Recovery MM:** En este bloque sincroniza el número de muestras de la señal cuadrada y la décima a una sola muestra.
- **Float to Char:** En este bloque los datos tipo Flotante se convierten en datos tipo Byte.
- **Correlete Access Code – Tag Stream:** En este bloque se sincroniza los paquetes con el Access Code, una vez el Bloque detecta este código empieza a tomar lo valores binarios. Y elimina la cabecera.
- **Tagged Stream Align:** En este bloque se alinean los paquetes en orden de llegada así mejorando la reconstrucción del archivo.
- **Pack k Bits:** En este bloque se convierte los Binario a Bytes.
- **File Sink:** En este bloque genera el archivo recibido a partir de los datos de entrada.

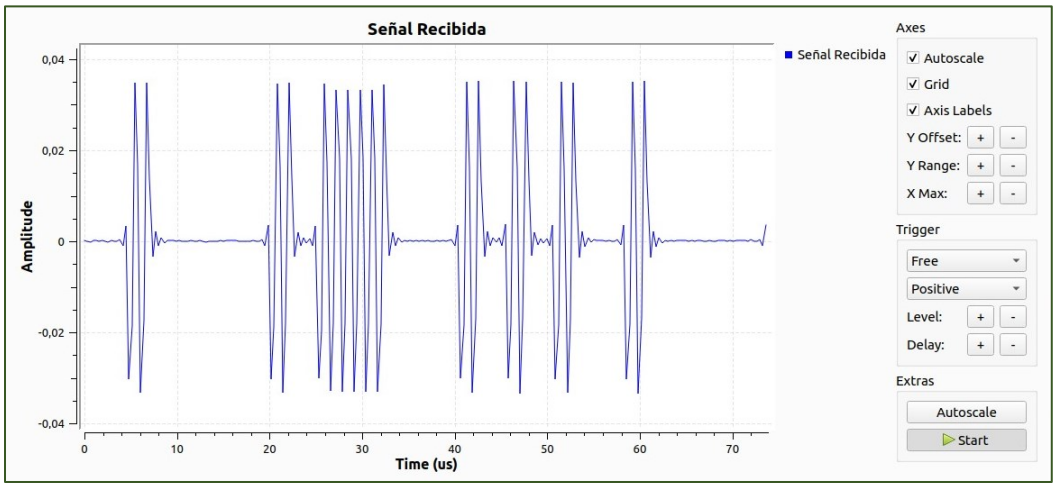


Fig 73. Señal recibida ASK. [autoría propia]

En la Fig 27, Se evidencia la señal obtenida en el dispositivo USRP-N210, modula por el método ASK.

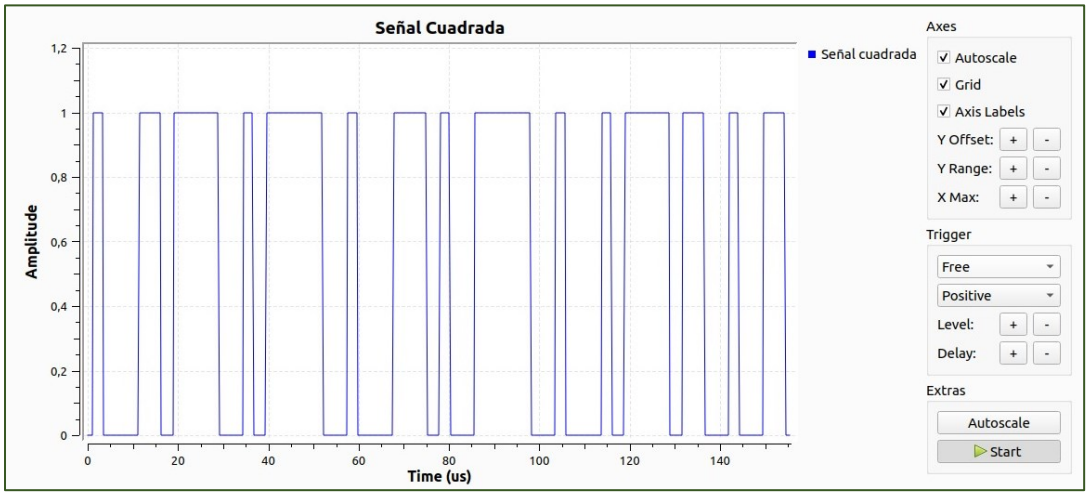


Fig 74, Señal cuadrada recuperada. [autoría propia]

En la Fig 28, Se evidencia la señal obtenida a realizar la demodulación por el método ASK, ya se encuentra lista para realizar la sincronización de paquetes.

6.5 Técnica de Relleno

Debido a que el bloque entre **Stream to Tagged Stream** no envía el último paquete ya que este no completa la longitud indicada, el archivo llega incompleto al receptor, por lo que se hace necesario aplicar una técnica de relleno en la transmisión antes de ejecutar el programa de GNU Radio, y a su vez aplicar otra técnica de recorte en la recepción después de ejecutar el programa de recepción de GNU Radio, como se evidencia en la Fig 29.

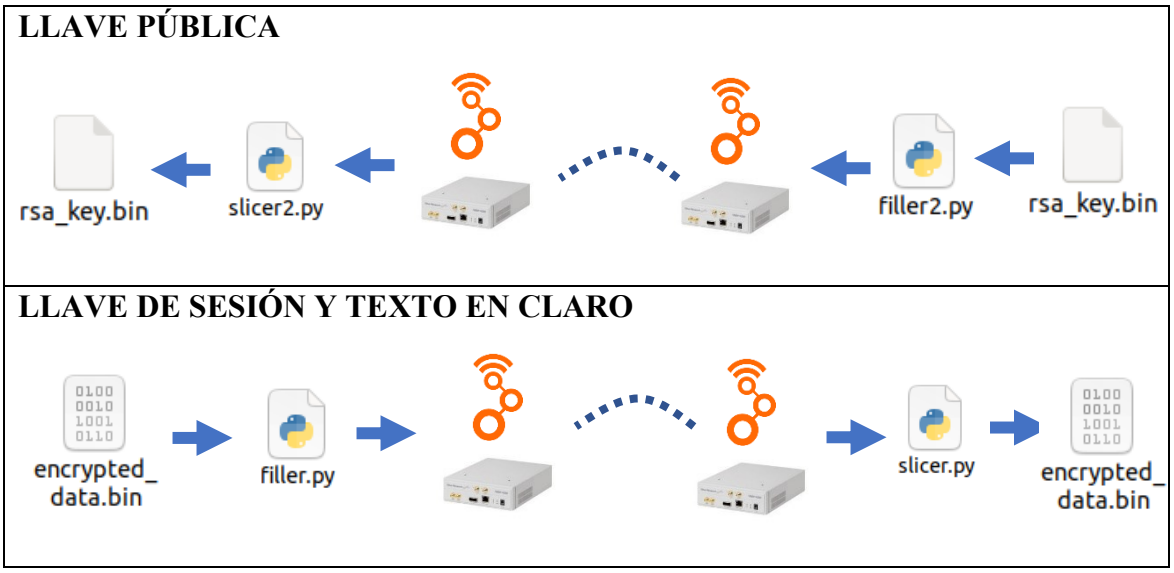


Fig 75. Diagrama flujo de la capa de comunicación. [autoría propia]

En la técnica de relleno se realiza a través de un Script llamado filler.py, básicamente "rellena" de un carácter aleatorio para cuadrar el último paquete y pueda ser enviado.

```
1  import os #Librería para consultar ruta del archivo
2  import math
3  import secrets #Librería que emplea números aleatorios para
   criptografía
4
5  #Consulta el tamaño del archivo
6  file_size = os.path.getsize(r'encrypted_data.bin')
7
8  fileSizeConcat = math.ceil(file_size / 256) * 256
9  #print(fileSizeConcat)
10
11 #Determina el tamaño de relleno
12 sizeConcat = fileSizeConcat - file_size
13 print(f'Tamaño del relleno: {sizeConcat}')
14
15 #Genera un número aleatorio entre 0 y 127
16 random_number = secrets.randbelow(127)
17
18
19 print(f'Numero aleatorio: {random_number}')
20 print(f'Caracter aleatorio: {chr(random_number)}')
21
22 #Genera un número de carácteres del tamaño de relleno
23 concat = (chr(random_number))*(sizeConcat+1)
24 #print(concat)
25
26 #Convierte el relleno en bytes
27 concat2 = bytes(concat, encoding='utf-8')
```

```
28 #print(concat2)
29
30 #Lee el archivo sin relleno
31 with open("encrypted_data.bin","rb") as f1:
32     data = f1.read()
33
34 #Concatena el contenido del archivo y el relleno
35 data = data + concat2
36
37 #Sobrescribe el archivo con el relleno
38 with open("encrypted_data_r.bin","wb") as f1:
39     data = f1.write(data)
```

Fig 76. Filler.py. Script de relleno. [autoría propia]

Como se puede evidenciar en la Fig 30, El archivo original `encrypted_data.bin` finaliza con el número hexadecimal 48, pero para completar el tamaño del último paquete genera aleatoriamente un carácter ASCII entre 0 y 127 y rellena el paquete con el carácter '[' representado en el número hexadecimal 5B como se evidencia en la Fig. 31. y renombrado como `encrypted_data_r.bin.`

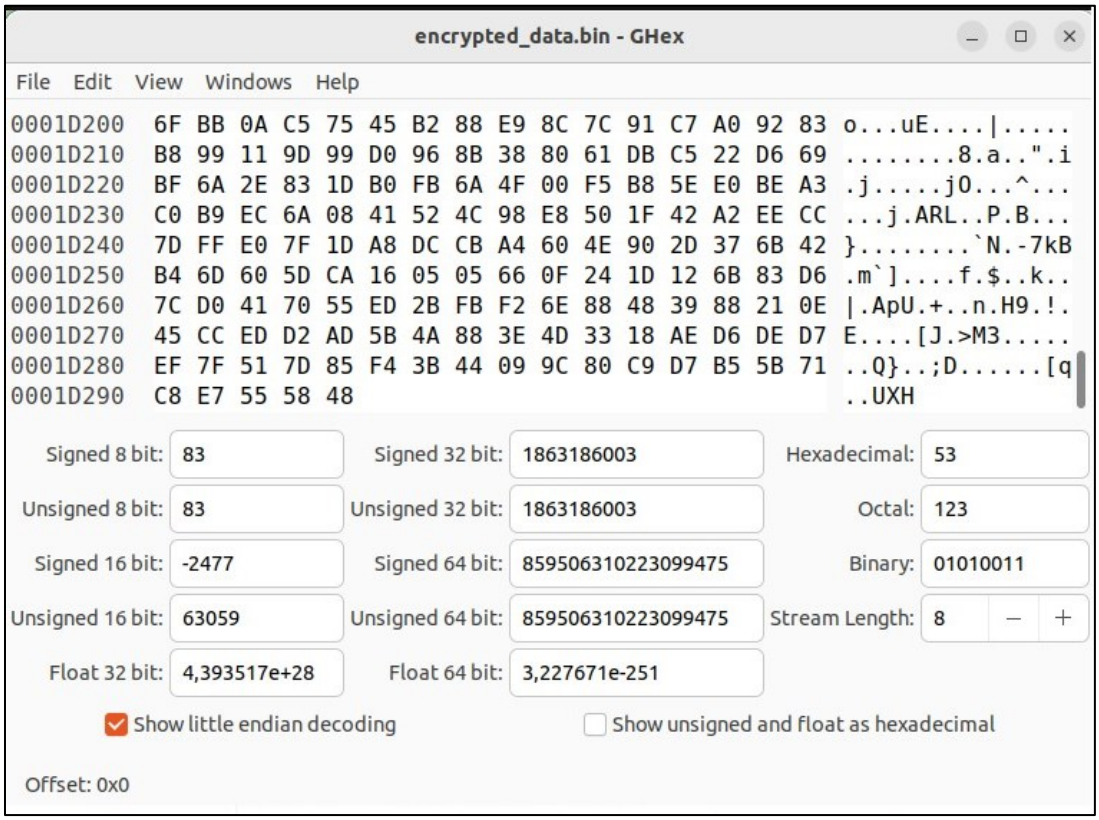


Fig 77. Contenido final del archivo encriptado sin rellenar. [autoría propia]

Nótese que se rellena con el carácter '[', en la Fig 32.

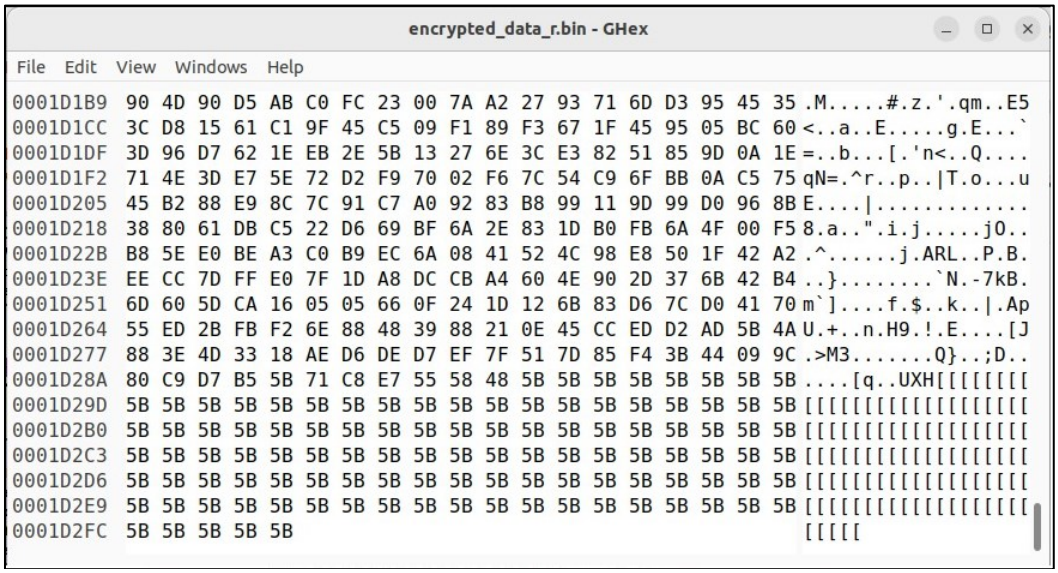


Fig 78. Contenido final del archivo. [autoría propia]

6.5.1 Técnica de Recorte

Se realiza con el script Slicer.py, este script recorta y elimina el relleno que se usó para la transmisión, su técnica consiste en buscar el primer Byte distinto de fin a inicio del contenido del archivo, por ejemplo, en la Fig. 33. El script recorre el carácter '[', hasta que encuentra el carácter 'H' y recorta el archivo hasta ese carácter dejándolo igual al original.

```
1  index = 0
2  data = ""
3
4  #Lee el archivo rellenado
5  with open("encrypted_data_r.bin","rb") as f2:
6      data = f2.read()
7
8  #Aplica un for para saber en qué índice debe quitar el relleno
9  for i in range(len(data)-2,-1,-1):
10     index = index + 1
11     if data[i] != data[i-1]:
12         break
13
14 # for i in data[::-1]:
15 #     index = index + 1
16 #     if i != 32:
17 #         break
18
19 print(f'Tamaño a quitar:{index+1}')
20
21 #Sobrescribe el archivo original
22 with open("encrypted_data.bin","wb") as f2:
23     #f2.write(data[:len(data)-(index-1)])
```

```
24      f2.write(data[:len(data)-(index+1)])
```

Fig 79. Script slicer.py. Técnica de recorte. [autoría propia]

Contenido final del archivo encriptado con Relleno. Nótese que se rellena con el carácter '['. Tal como se evidencia en la Fig 33.

Aunque la técnica es ingeniosa hay una probabilidad *p* de que el carácter de relleno coincida con el ultimo carácter del archivo, el ultimo carácter del archivo tiene 256 posibilidades frente a la 127 que el carácter de relleno.

$$p = \frac{1}{\text{probabilidad último carácter}} * \frac{1}{\text{probabilidad carácter de relleno}}$$
$$p = \frac{1}{256} * \frac{1}{128} = \frac{1}{32512} \approx 0,000003076 \quad (4)$$

La probabilidad de que coincida es de 0.000003076.

7. RESULTADOS

Para ello, se procede a tomar 50 muestras, 10 por cada archivo y éstos son los archivos que se usaron en la transmisión.

 colombia.png 44,4 kB	 tierra.jpg 96,9 kB	 lena.jpg 128,8 kB
 ibra.jpeg 190,8 kB	 Alberta.jpg 226,4 kB	

Tabla 7. Muestras usadas para la transmisión. [autoría propia]

7.1 Cálculo del Delay

Para ello se empleó el software Wireshark, el cual tomó las variables del Throughput, Delay y Jitter. Para este apartado se toman las capturas en el transmisor como en el receptor durante el tiempo de ejecución del programa y se filtran los paquetes en relación con las direcciones IP como se ilustra en la Tabla 8.

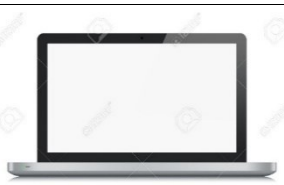
Transmisor		
Dirección IP	192.168.10.1	192.168.10.2
Receptor		
Dirección IP	192.168.10.5	192.168.10.11

Tabla 8. Direcciones IP de los dispositivos usados. [autoría propia]

Se prepara la captura en el Transmisor con el filtro con dirección el primer filtro **IP Source** 192.168.10.1:

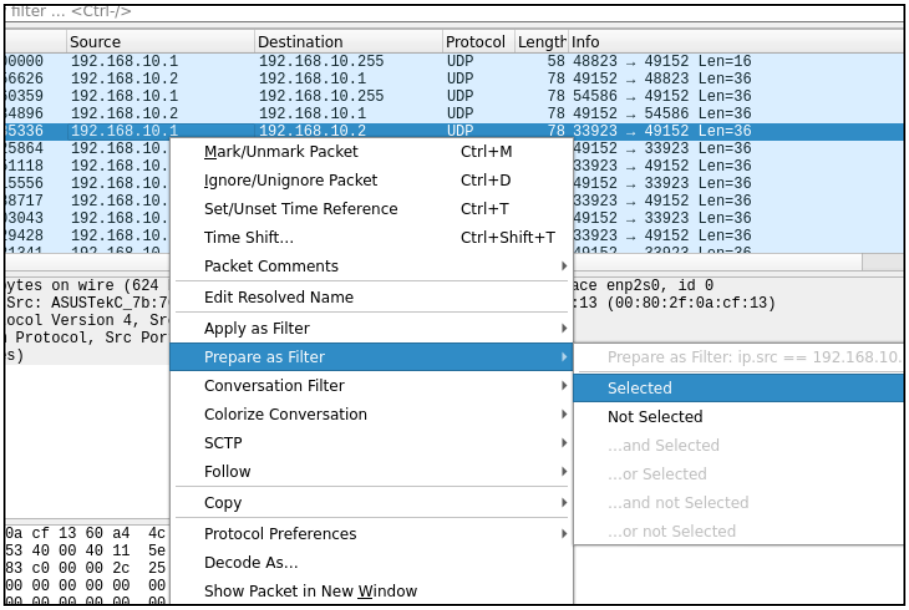


Fig 80. Preparación del filtro dirección IP de origen. [autoría propia]

Se prepara el segundo filtro **IP Destination** 192.168.10.2:

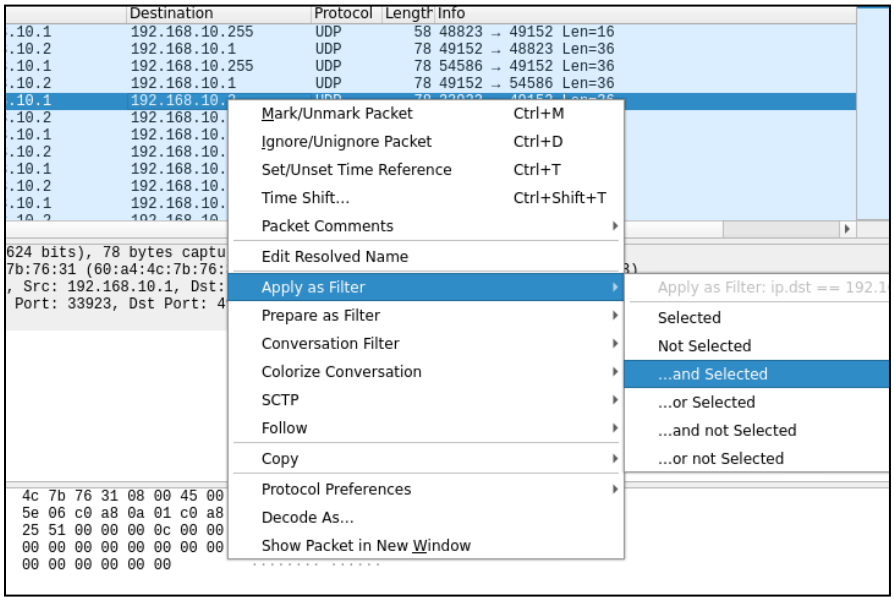


Fig 81. Preparación del filtro dirección IP de destino. [autoría propia]

En la **Error! Reference source not found.** En las Fig 34 y Fig 35, Se evidencia el proceso que se debe realizar para asignar el filtro correspondiente en el transmisor, el cual se observa aplicado el filtro en la Fig 36.

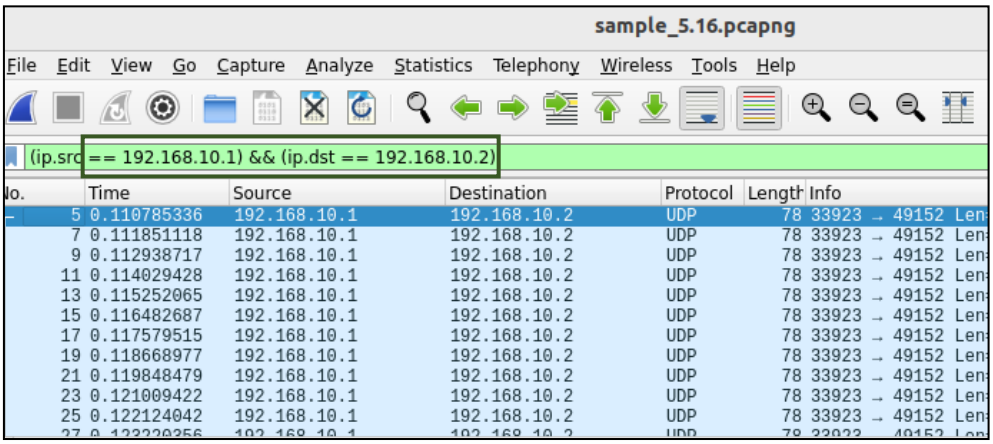


Fig 82. Filtro aplicado en el transmisor. [autoría propia]

Luego ubicamos el primer paquete en el transmisor y en la parte inferior se puede encontrar las características del paquete. En **frame**, ubicamos **Arrival Time** donde se toma la hora exacta. Lo mismo se hace en el receptor en el último paquete, como se muestra en la Fig 37.

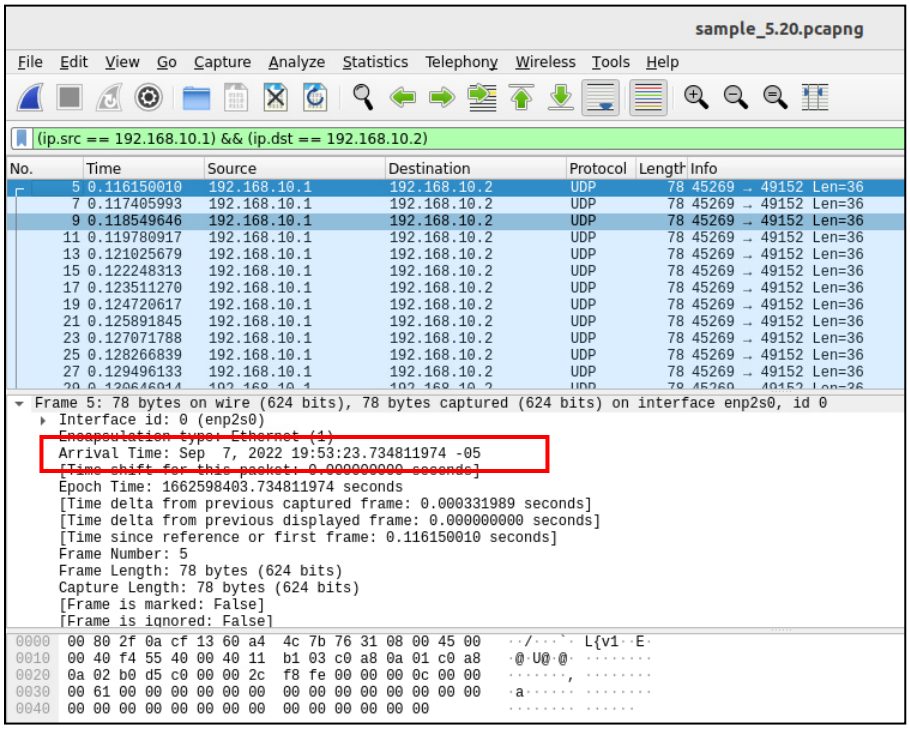


Fig 83. Toma del tiempo en el primer paquete del transmisor. [autoría propia]

Luego, se toma el último paquete del receptor como se observa en la Fig 38.

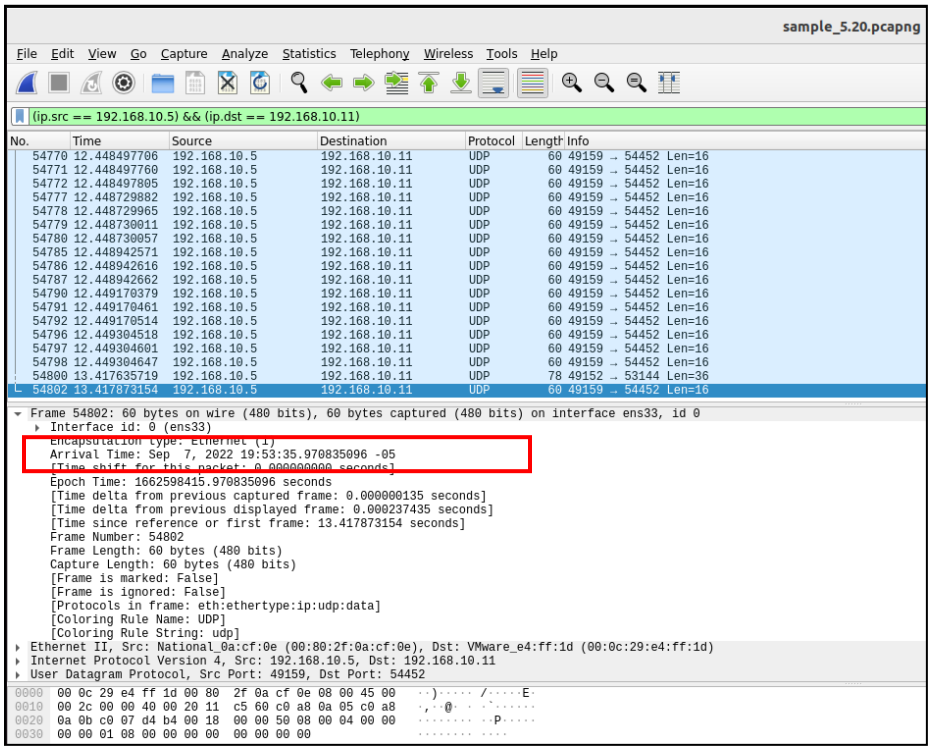


Fig 84. Toma del tiempo en el último paquete del receptor. [autoría propia]

Después se realiza la resta de estos dos tiempos con ayuda de la herramienta Excel como se puede ver en la Fig 39, La cual nos indica el cálculo del Delay, derivado de la diferencia temporal entre el último paquete del receptor y el primer paquete del transmisor.

Tiempo de Ejecución	Tiempo primer paquete	Jitter	Tiempo de Ejecución	Tiempo de último paquete	Throughput	Jitter	Delay	Jitter
0.23535	59:48.1	0.001077	0.077865	59:54.0	33056	0.000351	00:05.9	0.000
0.23691	05:26.0	0.0014	0.08409	05:32.1	32696	0.000351	00:06.1	0.000
0.22995	08:49.9	0.001007	0.08807	08:56.1	30600	0.000378	00:06.2	0.000
0.19838	11:38.8	0.000995	0.07304	11:43.6	35264	0.000327	00:04.8	0.000
0.21222	36:59.8	0.001082	0.09727	37:05.8	30392	0.00038	=H9-E9	0.000
0.22859	36:28.5	0.001077	0.09543	36:34.3	34296	0.000338	00:05.8	0.000
0.19849	39:59.2	0.001287	0.09659	40:05.2	29912	0.000544	00:06.0	0.000
0.22002	45:37.4	0.001098	0.08733	45:43.3	28956	0.000387	00:05.9	0.000
0.22564	52:22.6	0.001145	0.09693	52:29.2	37280	0.000314	00:06.6	0.000
0.20554	05:28.9	0.001015	0.08425	05:34.4	34472	0.000337	00:05.5	0.000
0.219109	0.875184258	0.001118	0.0880865	0.875252214	32692.4	0.000371	00:05.9	0.000

Fig 85. Cálculo del Delay. [autoría propia]

De acuerdo con la anterior operación se genera la Fig 40.

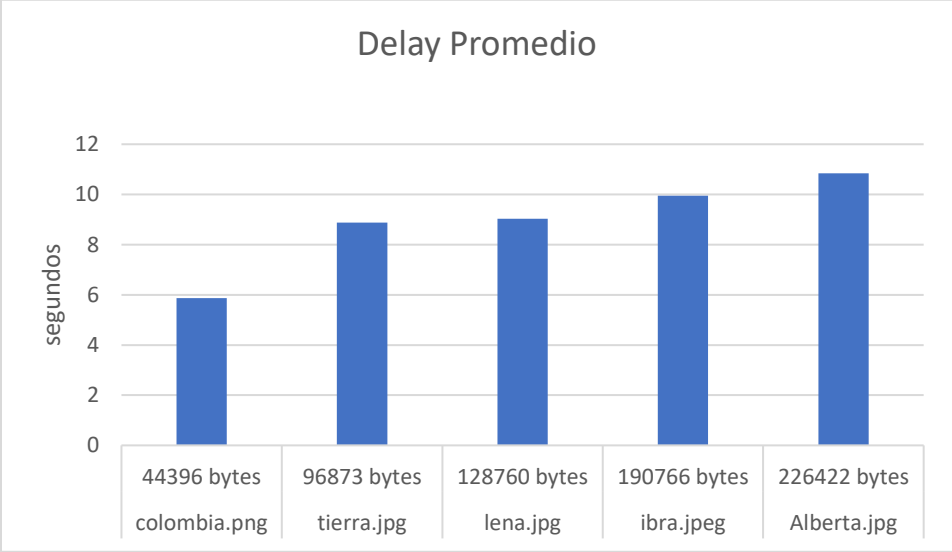


Fig 86. Delay promedio de cada una de las muestras. [autoría propia]

Como se puede observar la variación del delay, aunque hay una estrecha relación con el tamaño del archivo, es poco significativa variando desde casi los 6 segundos con el archivo menos pesado hasta los 11 segundos con el archivo más pesado.

Otra de las muestras significativas es el tiempo de procesamiento de cifrado tanto en el transmisor como en el receptor. Para ello se usó la librería *time* el cual ejecuta las sentencias

`time.process_time()` el cual toma el tiempo exacto de ejecución cuando es ejecutado esa línea de código, omitiendo todos los recursos que pueda utilizar el sistema en segundo plano. Se captura el tiempo de ejecución tanto en la transmisión como en la recepción de la siguiente manera:

```
1  import time
2  import os
3
4  exec(open("GUI.py").read())
5
6  start = time.process_time()
7
8  exec(open("Des_Public_Key.py").read())
9
10 exec(open("Encryption.py").read()) #Cifrado RSA
11
12 exec(open("filler.py").read()) #Acondiciona el archivo para GNU
13 Radio
14
15 end2 = time.process_time()
16
17 #exec(open("USRP_TX.py").read()) #Ejecuta la comunicación
18
19 exec(open("USRP.py").read()) #Ejecuta la comunicación
20
21 end = time.process_time()
22 delay = end - start
23 delay2 = end2 - start
24 print("Tiempo trancurrido: " + str(delay) + " segundos")
25 print("Tiempo trancurrido: " + str(delay2) + " segundos")
```

Fig 87. Script Master-TX.py. [autoría propia]

En la Fig 41, Se resalta en rojo la toma de los tiempos y en verde el cálculo del delay.

En la Fig 41, Se evidencia el Script que se ejecuta los demás scripts de encriptación y comunicación en el transmisor. Nótese que se toman los tiempos desde que acaba de ejecutar la interfaz gráfica y uno lo toma hasta la fase de encriptación y el otro lo toma hasta la ejecución completa.

En la Fig 42, Se obtiene el tiempo de ejecución tanto en el transmisor como en el receptor.

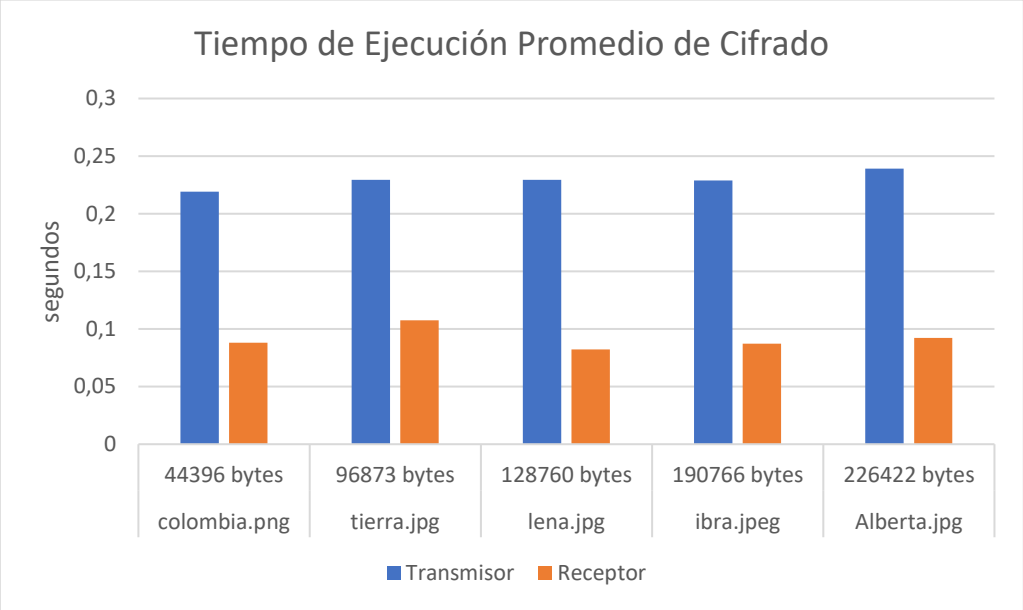


Fig 88. Tiempo de Ejecución de algoritmo de cifrado en el Transmisor y Receptor. [autoría propia]

Como se puede evidenciar en la Fig 43, Se observa como incide muy poco frente al tamaño de las muestras, ya que solo hay una relación entre el tamaño de la muestra y el tiempo de ejecución en la transmisión, pero no en la recepción. Otra cosa que se puede observar es que el tiempo de descifrado es menor en comparación al tiempo de cifrado.

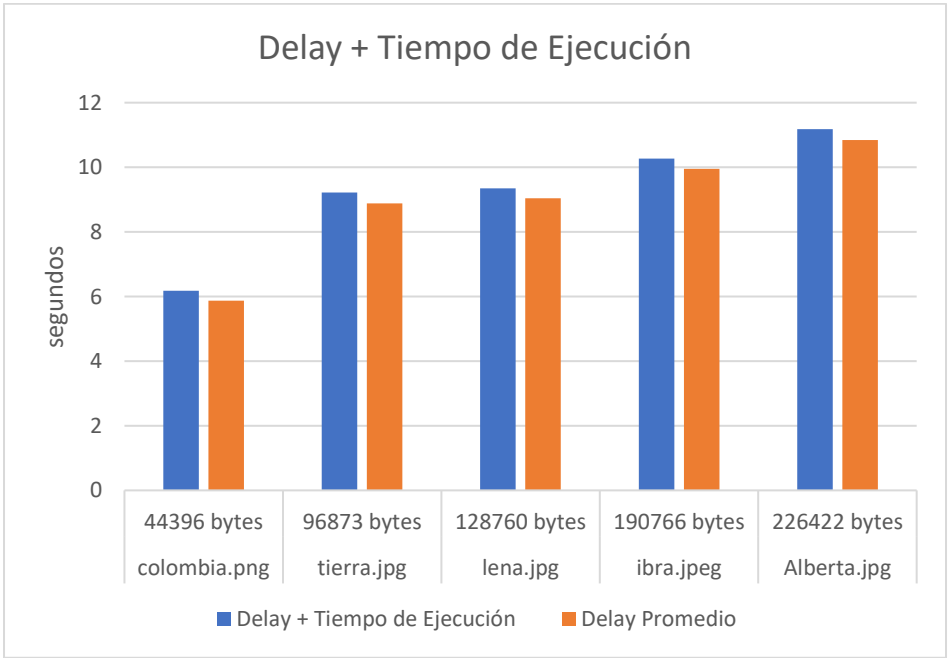


Fig 89. Delay + Tiempo de ejecución. [autoría propia]

En la Fig. 43. Se encuentra comparado el tiempo de ejecución con el Delay, se evidencia que no hay diferencia relevante en este factor, por lo cual, se puede inferir que el delay varía en función a la rata de símbolo programado tanto en los flujogramas del transmisor como en el receptor de GNU Radio, en este caso con la rata de bit de 195.312 kbps.

7.2 Cálculo del Throughput

Para el Throughput se basa en un principio matemático, cantidad de información enviada de manera exitosa sobre el tiempo total de transmisión, como se observa en la Ec. (5)

$$Throughput = \frac{Data}{t}\left[\frac{Bits}{segundos}\right]$$
$$Throughput = \frac{Bits}{segundos} \tag{5}$$
$$Throughput = \frac{Bytes}{segundos} * 8\left[\frac{bits}{s}\right]$$

Para la obtención del valor del **Throughput** se usó la herramienta Wireshark, en el menú **Statistics** y luego en el primer ítem **Capture File Properties,** tal y como se observa en la Fig 44.

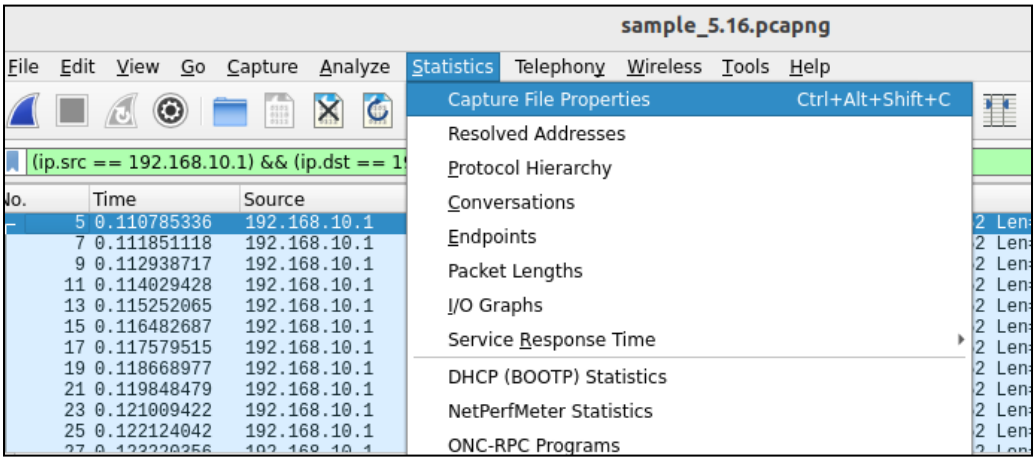


Fig 90. Menú de Toma de datos de Delay y Throughput. [autoría propia]

Luego se puede observar en la **Error! Reference source not found.** en **Average Bits/s** o **Average Bytes/s** se puede tomar el Throughput.

Interfaces

Interface	Dropped packets	Capture filter	Link type	Packet size limit (snaplen)
enp2s0	0 (0.0%)	none	Ethernet	262144 bytes

Statistics

Measurement	Captured	Displayed	Marked
Packets	45801	44449 (97.0%)	—
Time span, s	14.831	14.720	—
Average pps	3088.3	3019.7	—
Average packet size, B	1363	1402	—
Bytes	62408695	62303386 (99.8%)	0
Average bytes/s	4.208 k	4.232 k	—
Average bits/s	33 M	33 M	—

Fig 91. Toma del Throughput en Wireshark, en el apartado Average bytes/s ó Average bits/s. [autoría propia]

De las 10 muestras por cada archivo enviado se hace un promedio, como se observa en la Fig 45.

TIMEVALUE | =AVERAGE(I5:I14)

	A	B	G	H	I	
1			RX			
2			Python			
3			Cifrado	Wireshark RX		
4		Tamaño	Tiempo de Ejecución	Tiempo de último paquete	Throughput	Jitt
5	Muestra 1	44396	0.077865	7:59:54 PM	33056	0.0
6			0.08409	8:05:32 PM	32696	0.0
7			0.08807	8:08:56 PM	30600	0.0
8			0.07304	8:11:44 PM	35264	0.0
9			0.09727	8:37:06 PM	30392	0.
10			0.09543	9:36:34 PM	34296	0.0
11			0.09659	9:40:05 PM	29912	0.0
12			0.08733	9:45:43 PM	28956	0.0
13			0.09693	9:52:29 PM	37280	0.0
14			0.08425	10:05:34 PM	34472	0.0
15		Promedio	0.0880865		=AVERAGE(I5:I14)	

Fig 92. Toma del Throughput por muestra. [autoría propia]

Se tomó el Throughput promedio en el Transmisor en cada una de las muestras (archivos de imagen) y se obtuvo el resultado como se observa en la Fig 46.

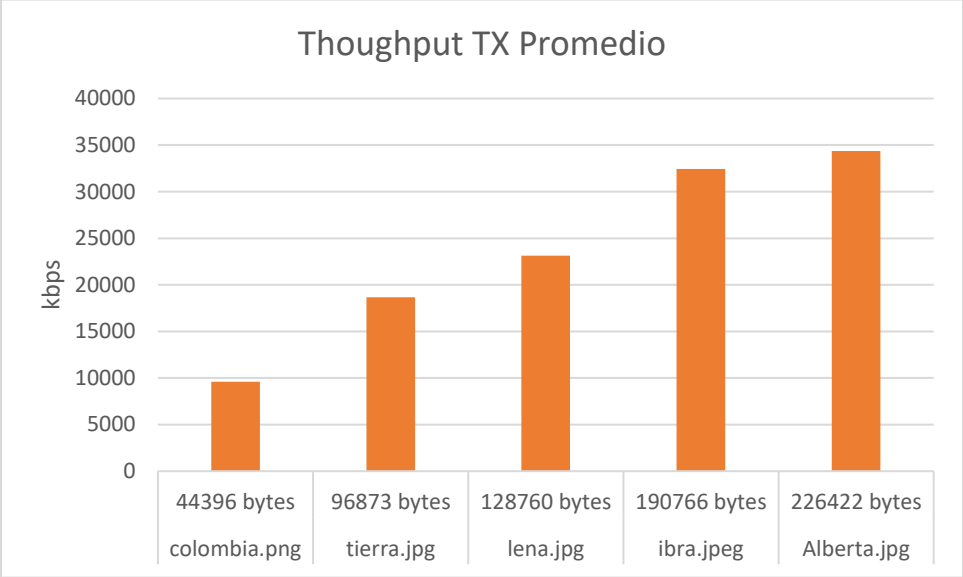


Fig 93. Throughput Promedio Transmisor. [Autoria propia]

En la Fig 47, Se evidencia el Throughput en el transmisor en cada una de las muestras. Nótese que varía en función del tamaño del archivo, en este caso el Delay es poco significativo para el tamaño del archivo.

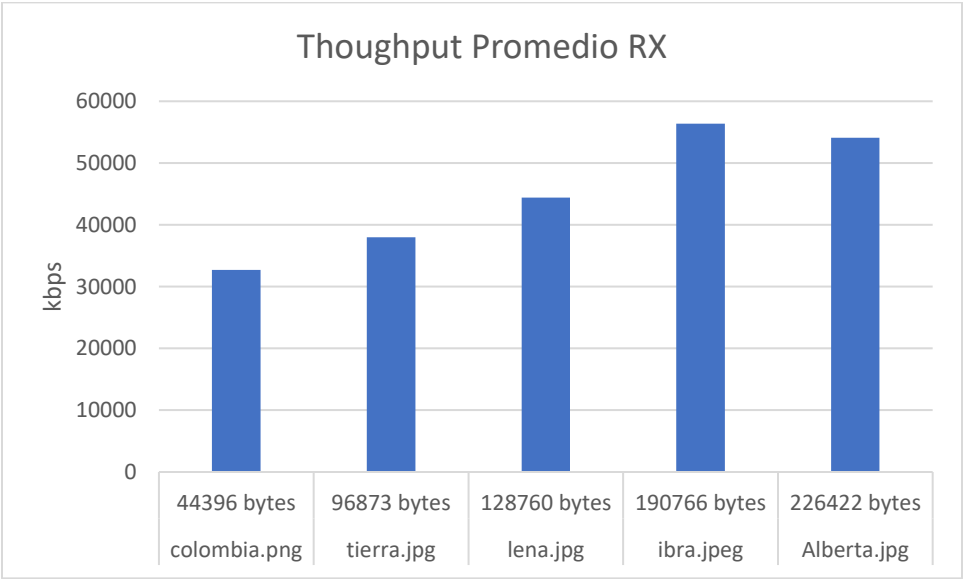


Fig 94. Throughput Promedio Receptor. [autoría propia]

En la Fig 48, Se evidencia el Throughput en el receptor en cada una de las muestras. Nótese que varía en función del tamaño del archivo, en este caso el Delay es poco significativo para el tamaño del archivo.

Como se puede observar en la Fig 49, Hay una amplia relación entre el Throughput y el tamaño del archivo, solo la muestra 5 de “tierra.jpg” es menor a la muestra 5 de “colombia.png”.

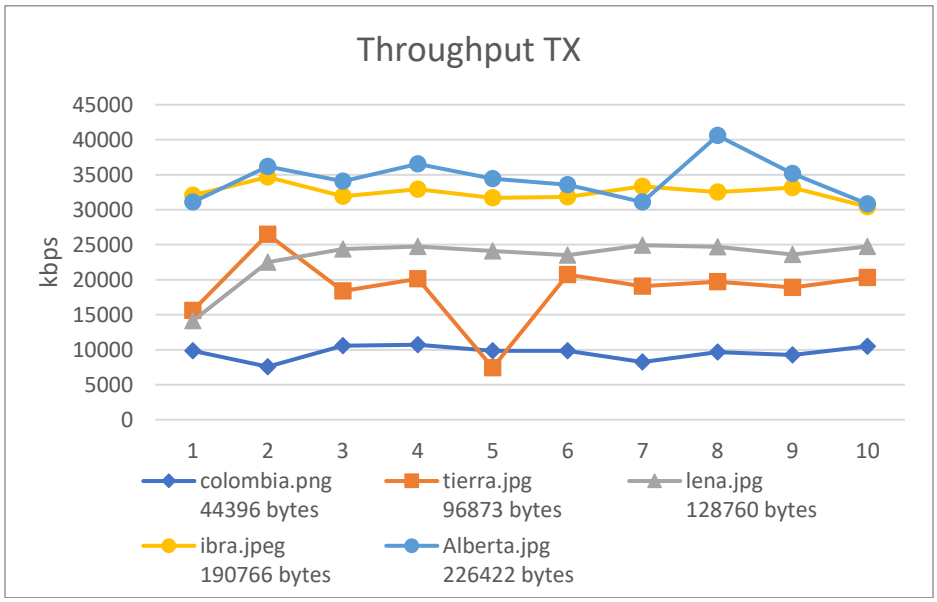


Fig 95. Throughput en el receptor en relación con las 10 muestras tomadas. [autoría propia]

Como se puede ver en la Fig 50, También hay una estrecha relación entre el throughput y el tamaño del archivo, donde las únicas diferencias notorias que se pueden observar es la muestra 5 de “tierra.jpg” en donde se ve muy bajo con respecto a todas las muestras y donde “Alberta.jpg” presenta grandes fluctuaciones en las tres últimas muestras.

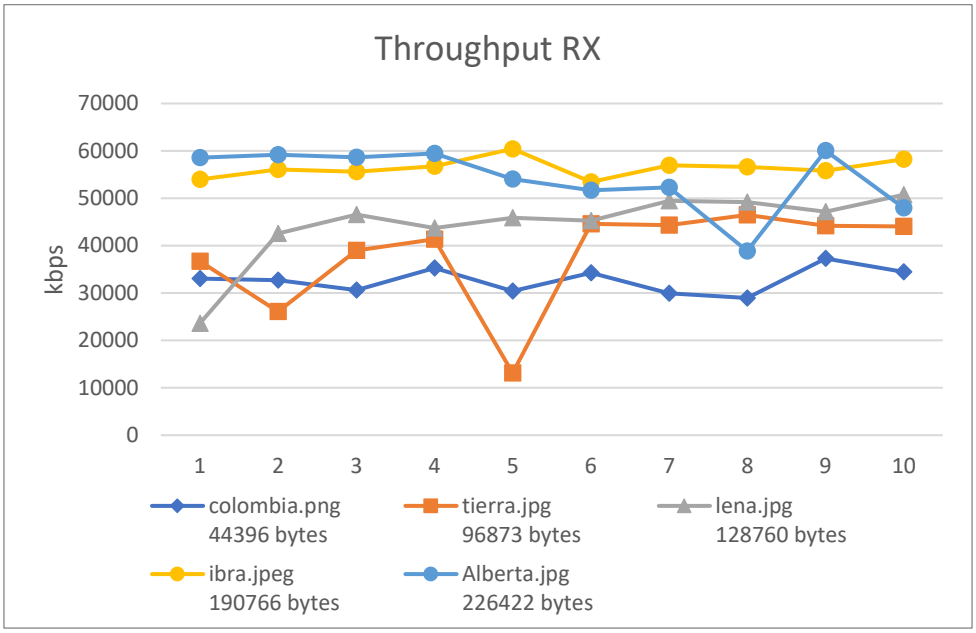


Fig 96. Throughput en el receptor en relación con las 10 muestras tomadas. [autoría propia]

Por lo tanto, se puede concluir que el delay no es muy significativo ya que el throughput no cambia significativamente en relación con el tamaño de los archivos.

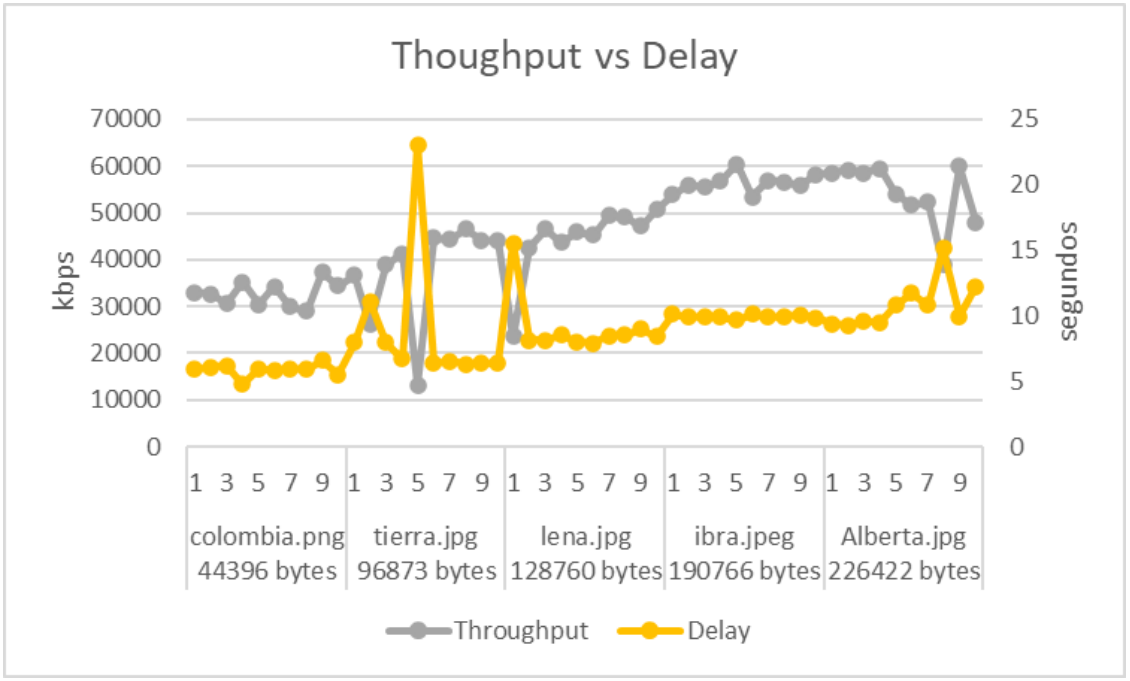
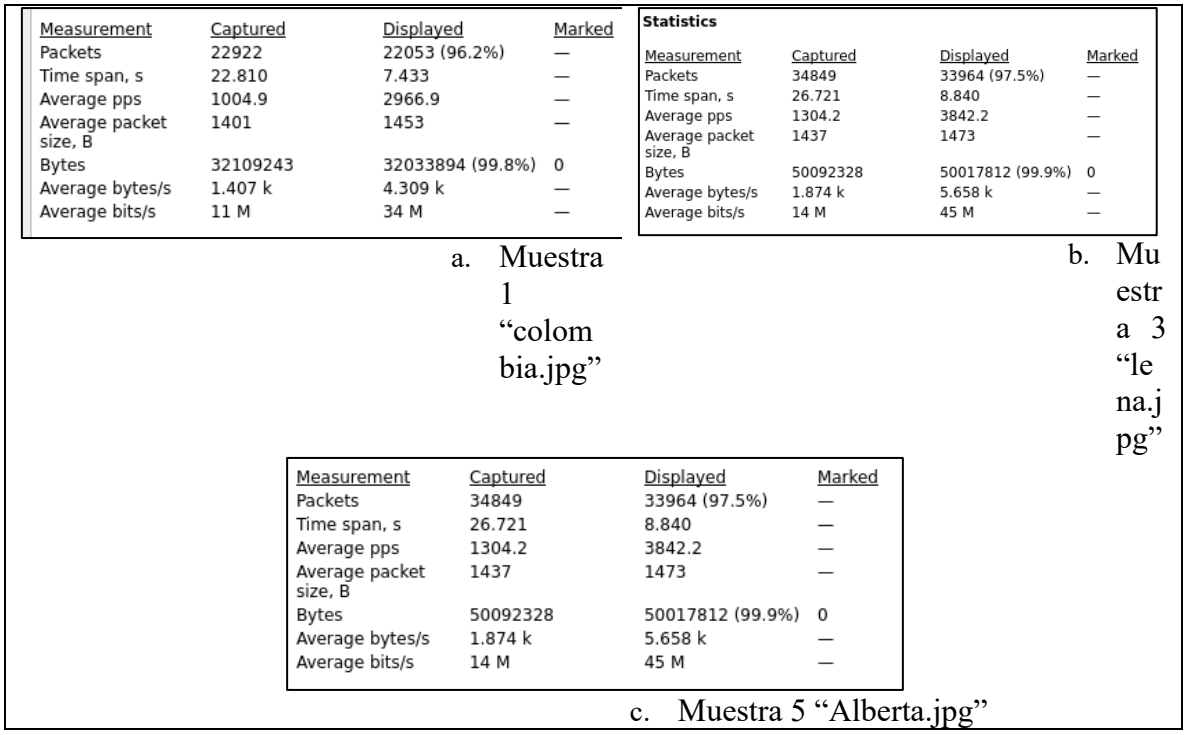


Fig 97. Throughput vs Delay en la recepción. [autoría propia]

Como se puede observar en la Fig 51, Aunque tanto el throughput como el delay son inversamente proporcionales aumentan de manera progresiva junto al tamaño del archivo, y

se puede ver como en un caso en donde el delay fue bastante alto en algunas muestras, así mismo el throughput a su vez fue bastante bajo. Por lo que se puede concluir, que el delay no es plenamente significativo en esta escala, ya que el throughput aumenta también. Esta razón es debido a la información de control que necesita el USRP para su pleno funcionamiento. Como se están repitiendo las transmisiones y como se vió en la **Error! Reference source not found.** en el capítulo anterior solo se toma en cuenta los paquetes involucrados en la comunicación, por lo que los paquetes de control y de acuse de recibo (ACK) están siendo ignorados, por lo que influye en el Throughput de manera poco significativa ya que el porcentaje de esta data es menor al 2%.

Se puede ver un ejemplo en la donde se puede ver la cantidad de paquetes que se tienen en cuenta, la data y el throughput general (izquierda) y el throughput efectivo (derecha)



Measurement	Captured	Displayed	Marked
Packets	22922	22053 (96.2%)	—
Time span, s	22.810	7.433	—
Average pps	1004.9	2966.9	—
Average packet size, B	1401	1453	—
Bytes	32109243	32033894 (99.8%)	0
Average bytes/s	1.407 k	4.309 k	—
Average bits/s	11 M	34 M	—

a. Muestra 1 “colombia.jpg”

Statistics

Measurement	Captured	Displayed	Marked
Packets	34849	33964 (97.5%)	—
Time span, s	26.721	8.840	—
Average pps	1304.2	3842.2	—
Average packet size, B	1437	1473	—
Bytes	50092328	50017812 (99.9%)	0
Average bytes/s	1.874 k	5.658 k	—
Average bits/s	14 M	45 M	—

b. Muestra 3 “lena.jpg”

Measurement	Captured	Displayed	Marked
Packets	34849	33964 (97.5%)	—
Time span, s	26.721	8.840	—
Average pps	1304.2	3842.2	—
Average packet size, B	1437	1473	—
Bytes	50092328	50017812 (99.9%)	0
Average bytes/s	1.874 k	5.658 k	—
Average bits/s	14 M	45 M	—

c. Muestra 5 “Alberta.jpg”

Fig 98. Comparación del Throughput. [Autoria propia]

En la Fig 52, Se observa como con la información efectiva de la comunicación (filtro aplicado) vs. Throughput general (información de control). Nótese que inclusive y poco perceptible el throughput general se ve afectado, esto tiene que ver con la información de cabecera en comparación con la información útil

Inclusive, el Throughput general se ve afectado de manera más leve, debido a la información de cabecera de los paquetes con respecto a la carga útil. Aquí se puede concluir que la información de control afecta de manera muy poco significativa al Throughput y que entre

más grande sea el archivo esta medición tenderá a un valor fijo con una gráfica con una asíntota horizontal que tienda a ese valor

7.3 Cálculo del Jitter

El Jitter, se define como la fluctuación del retardo en el tiempo y se mide de acuerdo con la Ec. (6).[56].

$$Jitter = \frac{\sum_{i=0}^{n}(R_i - R_{i-1})}{n} \quad (6)$$

Donde:

R_i es el tiempo de recepción del paquete actual

R_{i-1} es el tiempo de recepción del paquete anterior

n es la cantidad de paquetes

De acuerdo con la formula dada, se obtuvo la Fig 53.

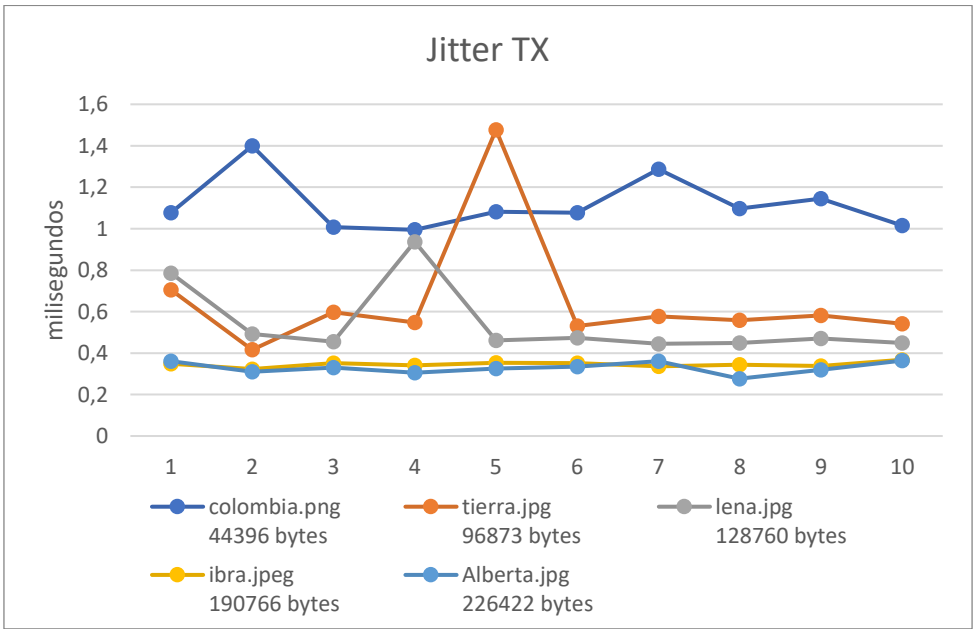


Fig 99. Jitter en el Transmisor. [autoría propia]

En la Fig 53, Se evidencia el resultado de la medición del Jitter en el Transmisor durante él envió de cada una de las cinco muestras ejecutadas en las pruebas.

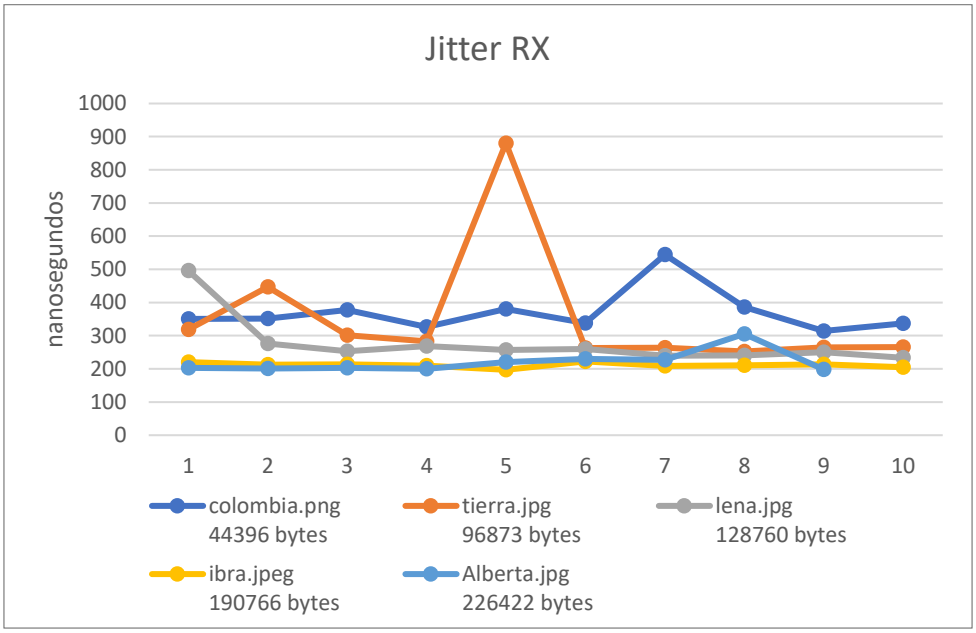


Fig 100. Jitter en el Receptor. [autoría propia]

En la Fig 54, se evidencia el resultado de la medición del Jitter en el Receptor durante la recepción de cada una de las cinco muestras ejecutadas en las pruebas.

Como se puede observar en la Fig. 54. el jitter en el transmisor es constante salvo por la muestra 5 en “tierra.jpg” donde su valor es casi 1,5 ms, mientras que en “colombia.png” todas las muestras van por encima del milisegundo. En la recepción, el jitter oscila en valores menores a los 500 ηs, salvo en la muestra 5 de la imagen “tierra.jpg” que apenas llega a los 900 ηs. La imagen “colombia.png” tiene el mayor jitter en promedio como se puede observar en la Fig 54.

Se hace un comparativo del Jitter con el Throughput como se puede ver en la recepción en la Fig 55.

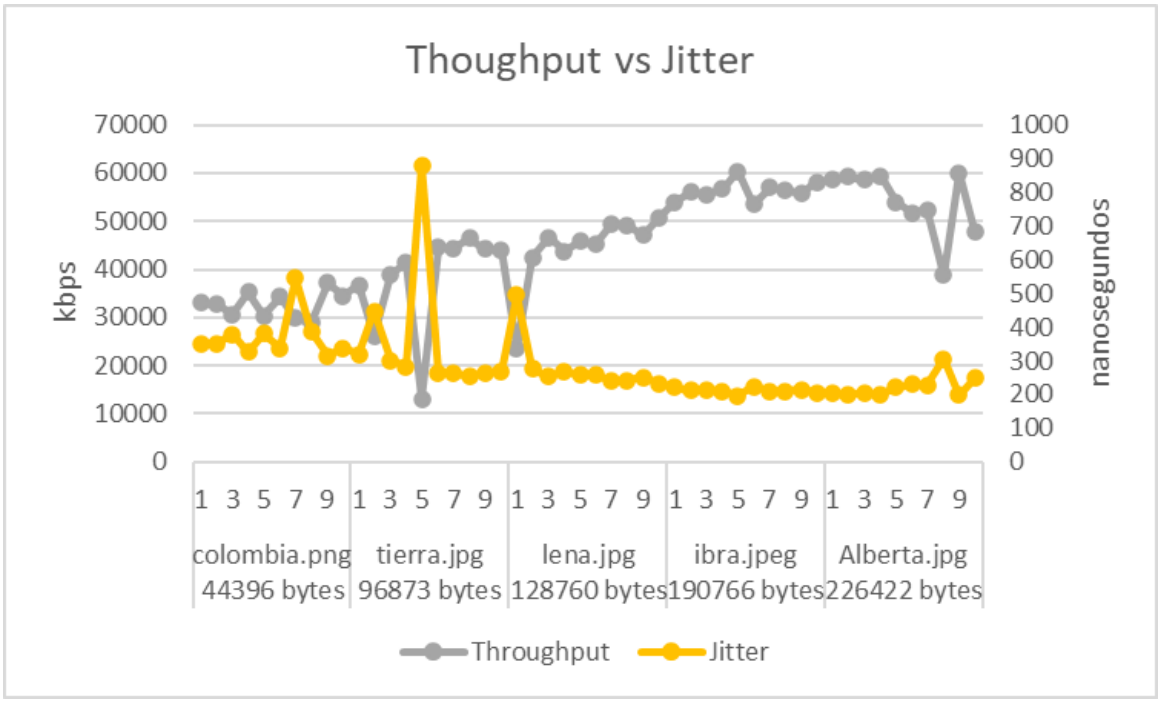


Fig 55. Comparación de Jitter y Throughput en la recepción

Como se puede ver en la Fig 55, el Jitter resulta inversamente proporcional al throughput, por lo cual entre más se retarde el paquete degrada el throughput en el resultado final.

7.4. Análisis de Resultados

Después de realizar las muestras se puede dar cuenta de que el Throughput varía inversamente con el delay y el jitter. La tasa de bit configurada en GNU Radio es la que principalmente deriva los tiempos de ejecución de la plataforma. El consumo de los recursos de las máquinas es algo excedido, por lo cual se aconseja usar un sistema de modulación más eficiente como FSK, PSK ó QAM, que mejore la detección de errores, ya que la no llegada de los datos, sea por un bit no descifra

8. CONCLUSIONES Y RECOMENDACIONES

- El sistema de cifrado híbrido garantiza la integridad de la información como la eficiencia del algoritmo que sólo dependerá de la capa de comunicaciones de GNU Radio, ya que la llave de sesión de cifrado simétrico que está encriptando la información, está encriptada a su vez con una técnica de cifrado asimétrico de llave pública y privada, y esta vez protegida por contraseña, el tiempo de procesamiento es poco perceptible frente al Delay de la comunicación.

- Se desarrolló en modulación ASK, ya que es necesario que el archivo de texto cifrado llegue igual desde la transmisión hasta la recepción con su estructura binaria para que el algoritmo de descifrado lo pueda desencriptar, si hay un cambio de un solo bit el algoritmo no reconoce el archivo de texto cifrado y no logra desencriptar. Puesto que esta modulación fue la más fácil de interpretar e implementar en GNU Radio fue la que se desarrolló, ya que el proceso de la capa de comunicaciones fue desarrollado a prueba y error.
- El Delay varía de acuerdo con el tamaño del archivo, pero de una manera poco apreciable a pequeña escala.
- El Delay en este proyecto varía de acuerdo con la rata de bit ajustado en el flujograma de GNU Radio para tal fin.
- Se evidencia que es necesario que el archivo cifrado debe llegar exactamente igual desde la transmisión hasta la recepción, puesto que, si hay al menos un carácter de más, un carácter menos o un carácter cambiado en dicho archivo no desencripta en la recepción.
- Cualquier modulación digital que garantice la integridad de los bits puede ser usada para este fin. Por lo tanto, este algoritmo se puede optimizar con una modulación digital más robusta como el PSK o el QAM, modulación OFDM que mejoraría el throughput, el Delay y así la eficiencia de la comunicación.
- Es recomendable mejorar este algoritmo con una técnica de detección de errores robustas en la capa de comunicaciones de GNU Radio, para garantizar que el archivo llegue completo en la recepción y pueda ser desencriptado.
- La interfaz gráfica proporciona la posibilidad de poder ingresar la contraseña de sesión como la posibilidad de ingresar el archivo que se va a enviar por el canal cifrado.
- En este proyecto, el algoritmo garantiza una alta eficiencia en cuanto a los tiempos de envío y recepción de archivos de imagen y de audio comprimida que se podrían usar en tiempo real, si se usará una modulación que garantice mayor capacidad se podría manejar video en tiempo real.
- Para este trabajo, debido a que la modulación ASK es poco eficiente en cuanto al rata bit vs procesamiento, se recomienda que se corran en computadores robustos con procesadores Intel i5, similares o superiores; y con una memoria RAM mayor a 8 GB.

9. BIBLIOGRAFIA

[1] J. Areitio Bertolín, "Exploración del sistema de red TETRA desde la óptica de la ciberseguridad y privacidad - Electrónica," *Canales Sectoriales, Interempresas*, 2017. https://www.interempresas.net/Electronica/Articulos/205039-Exploracion-del-sistema-de-red-TETRA-desde-la-optica-de-la-ciberseguridad-y-privacidad.html (accessed Jun. 21, 2021).

[2] WIKIPEDIA, "Ataque de REPLAY - Wikipedia, la enciclopedia libre," 2019. https://es.wikipedia.org/wiki/Ataque_de_REPLAY (accessed Jun. 14, 2021).

[3] D. Piscitello, "¿Qué es un ataque de intermediarios?," 2015. https://www.icann.org/es/blogs/details/what-is-a-man-in-the-middle-attack-2-11-2015-es (accessed Jun. 14, 2021).

[4] http://www.scielo.org.co/pdf/dyna/v85n204/0012-7353-dyna-85-204-00319.pdf

[5]https://programmerclick.com/article/1313116474/#:~:text=Las%20ventajas%20son%20principalmente%20que,este%20m%C3%A9todo%20no%20es%20bueno.

[6] http://repositorio.espe.edu.ec/bitstream/21000/15538/1/T-ESPE-040614.pdf

[7] H. Jara and F. G. Pacheco, "Seguridad de la Información," in *Ethical Hacking 2.0*, 2012th ed., redUSERS, Ed. Buenos Aires, Argentina, 2012, p. 15.

[8] Hytera, "Estándares mision crítica | Hytera Latinoamérica," 2021. https://www.hytera.la/recursos/p25-vs-tetra (accessed Jun. 21, 2021).

[9] K. P. F. Briones and O. A. B. Córdova, "Diseño E Implementación De Un Sistema De Defensa Contra Ataques De Denegación De Servicio Distribuido En La Red Para Una Empresa De Servicios," p. 258, 2015.

[10] F. A. Rolón Abreo, "ESTUDIO Y ANÁLISIS PARA LA SELECCIÓN DE UNA TECNOLOGÍA A IMPLEMENTAR," Universidad Santo Tomás, 2018.

[11] diarioti.com, "La Policía Nacional de Colombia actualiza el sistema de comunicaciones con tecnología de Motorola Solutions | Diario TI," 2011. https://diarioti.com/la-policia-nacional-de-colombia-actualiza-el-sistema-de-comunicaciones-con-tecnologia-de-motorola-solutions/30485 (accessed Jun. 19, 2021).

[12] H. Bakht, *Aplicaciones de redes móviles Ad-hoc*. 2018.

[13] E. I. Amaya Barrera, C. A. Suárez Parra, R. E. Huérfano Ortiz, J. D. Moreno Posada, and F. A. Parra Fuentes, "Modelo de encriptación simétrica basada en atractores caóticos," *Ingeniería*, vol. 21, no. 3, pp. 378–390, Oct. 2016, doi: 10.14483/udistrital.jour.reving.2016.3.a08.

[14] A. Ray, A. Potnis, S. Kumar, P. Dwivedy, and S. Soofi, "Hybrid model for Image Encryption and Decryption using RSA, AES, and Affine with XOR Operation," *J. Eng. Res. Appl. www.ijera.com*, vol. 7, p. 90, 2017, doi: 10.9790/9622-0708019094.

[15] E. Salim and I. Harba, "www.etasr.com Harba: Secure Data Encryption Through a Combination of AES," 2017. [Online]. Available: www.etasr.com.

[16] M. Laghate, S. Chaudhari, and D. Cabric, "USRP N210 demonstration of wideband sensing and blind hierarchical modulation classification," May 2017, doi: 10.1109/DySPAN.2017.7920748.

[17] A. M. M. Chandran and M. Zawodniok, "Transmitter leakage analysis when operating USRP (N210) in duplex mode," in *Conference Record - IEEE*

Instrumentation and Measurement Technology Conference, Jul. 2015, vol. 2015-July, pp. 340–345, doi: 10.1109/I2MTC.2015.7151291.
[18] K. D, "Criptografía homomórfica: un paradigma de cifrado cada vez más cercano," *Welivesecurity*, 2019. https://www.welivesecurity.com/la-es/2019/09/04/criptografia-homomorfica-paradigma-cifrado-cercano/ (accessed Jun. 15, 2021).
[19] S. Miriyala and M. S. Sairam, "Improving privacy in SDN based MANET using hybrid encryption and decryption algorithm," *Microprocess. Microsyst.*, no. November, p. 103501, 2020, doi: 10.1016/j.micpro.2020.103501.
[20] M. A. Albahar, O. Olawumi, K. Haataja, and P. Toivanen, "Novel Hybrid Encryption Algorithm Based on Aes, RSA, and Twofish for Bluetooth Encryption," *J. Inf. Secur.*, vol. 09, no. 02, pp. 168–176, 2018, doi: 10.4236/jis.2018.92012.
[21] Y. Alemami, M. Afendee Mohamed, S. Atiewi, M. Mamat, and M. A. Mohamed, "Speech encryption by multiple chaotic maps with fast fourier transform," *Int. J. Electr. Comput. Eng.*, vol. 10, no. 6, pp. 5658–5664, 2020, doi: 10.11591/ijece.v10i6.pp5658-5664
[22] O. M. Al-Hazaimeh, "A new dynamic speech encryption algorithm based on Lorenz chaotic map over internet protocol," *Int. J. Electr. Comput. Eng.*, vol. 10, no. 5, pp. 4824–4834, 2020, doi: 10.11591/ijece.v10i5.pp4824-4834.
[23] A. Ali and A. Ware, "Effective Performance Metrics for Multimedia Mission-critical Communication Systems," *Ann. Emerg. Technol. Comput.*, vol. 5, no. 2, pp. 1–14, Apr. 2021, doi: 10.33166/aetic.2021.02.001.
[24] S. Mobayen, C. K. Volos, Çavuşoğlu, and S. Kaçar, "A simple chaotic flow with hyperbolic sinusoidal function and its application to voice encryption," *Symmetry (Basel).*, vol. 12, no. 12, 2020, doi: 10.3390/sym12122047.
[25] D. K. Kilcoyne, S. Bendelac, J. M. Ernst, and A. J. Michaels, "Tire Pressure Monitoring System encryption to improve vehicular security," in *Proceedings - IEEE Military Communications Conference MILCOM*, Dec. 2016, pp. 1219–1224, doi: 10.1109/MILCOM.2016.7795497.
[26]https://www.scielo.br/j/jmoea/a/TCg4wdgNMmHZVK9TGZKZS8C/?format=pdf&lang=en
[27] http://www.scielo.org.co/pdf/itec/v17n1/1692-1798-itec-17-01-19.pdf
[28] Digital Guide ICONOS, "El encriptado informático: así se protege la comunicación," *https://www.ionos.es/digitalguide/servidores/seguridad/todo-sobre-los-metodos-de-encriptado/*, Aug. 19, 2019.
[29] https://pythondiario.com/2020/07/criptografia-en-python-aes.html#google_vignette
[30] J. C. Mendoza T., "Julio César Mendoza T. Ingeniería de Sistemas – Quito," *INGENIUS*, pp. 46–53, 2008, [Online]. Available: https://dspace.ups.edu.ec/bitstream/123456789/8185/1/Demostración de cifrado simétrico y asimétrico.pdf.
[31] R. Gónzalez Duque, *Python Para todos*. España.
[32] https://pycryptodome.readthedocs.io/en/latest/src/introduction.html#
[33] https://pycryptodome.readthedocs.io/en/latest/src/api.html
[34] S. Romero, D. F. Zamora, and J. Andres, "ESTUDIO DE LA TECNOLOGÍA RADIO DEFINIDO POR SOFTWARE."
[35] DOCPLAYER, "Radio Definido por Software," https://docplayer.es/9195547-1-radio-definido-por-software.html. .

[36] C. Manuel García Algora Tutor and V. Alfonso Reguera, “Universidad Central ‘Marta Abreu’ de Las Villas TRABAJO DE DIPLOMA Radio Definido por Software usando MATLAB ‘Año 53 de la Revolución,’” 2011.
[37] https://medium.com/modulaciones-digitales-ask-psk-fsk/ask-amplitude-shift-keying-d658d5128168
[38] Ettus Research, “USRP N210,” *07495_Ettus_N200-210_DS_Flyer_HR_1*. .
[39] inter export, “Sistemas de Radiocomunicaciones para Misión Crítica ,” *https://www.interexport.cl/wp-content/uploads/2015/07/Soluciones-mision-critica.pdf*. .
[40] BGH Tech Parther, “Sistemas de comunicaciones de radio para misión crítica,” *https://www.bghtechpartner.com/2020/05/06/sistemas-comunicaciones-radio-mision-critica/*, May 06, 2020.
[41]https://repository.udistrital.edu.co/bitstream/handle/11349/4539/An%C3%A1lisis%20de%20un%20sistema%20de%20telecomunicaciones%20para%20una%20microrred%20aislada%20en%20un%20entorno%20rural.pdf?sequence=1&isAllowed=y
[42]https://repository.udistrital.edu.co/bitstream/handle/11349/22369/CarlosJulianSabogalUribe2019.pdf?sequence=1&isAllowed=y
[43]IEEE, “Los estándares IEEE para Software,” IEEE. Los estándares IEEE para Software, disponible en www.ieee.org.
[44] Mintic, “Espectro Radioeléctrico,” Gobierno de Colombia. [Online]. Available: https://www.mintic.gov.co/portal/inicio/Atencion-al-Publico/VUTIC-Ventanilla-Unica-de-Tramites/Espectro-Radioelectrico/
[45] Gobierno de Colombia, “Constitución política,”. [Online]. Available: http://cms-static.colombiaaprende.edu.co/cache/binaries/articles-186370_constitucion_politica.pdf?binary_rand=1416
[46] MinTic, “Subdirección de Radiodifusión Sonora,” Gobierno Nacional de Colombia. [Online]. Available: https://www.mintic.gov.co/portal/inicio/2291:Informacion-General-del-Servicio
[47] “Leyes desde 1992 - Vigencia expresa y control de constitucionalidad [LEY_1273_2009],” Jan. 05, 2009. http://www.secretariasenado.gov.co/senado/basedoc/ley_1273_2009.html (accessed Jun. 08, 2021).
[48] “Ley 1266 de 2008 - EVA - Función Pública. 2008”. Accessed: Jun. 08, 2021. [Online]. Available: https://www.funcionpublica.gov.co/eva/gestornormativo/norma.php?i=34488
[49] “Ley 1581 de 2012 - EVA - Función Pública,” Oct. 17, 2012. https://www.funcionpublica.gov.co/eva/gestornormativo/norma.php?i=49981 (accessed Jun. 08, 2021).
[50] P. de Ley N, “POR MEDIO DE LA CUAL SE APRUEBA EL «CONVENIO SOBRE LA CIBERDELINCUENCIA», . ADOPTADO EL 23 DE NOVIEMBRE DE 2001, EN BUDAPEST.”
[51] “Leyes desde 1992 - Vigencia expresa y control de constitucionalidad [LEY_0527_1999],” Aug. 17, 1999. http://www.secretariasenado.gov.co/senado/basedoc/ley_0527_1999.html (accessed Jun. 08, 2021).

10. ANEXOS

Repositorio en GitHub: https://github.com/Jorgealza9206/ProgrammingProject
Vídeo en YouTube: https://youtu.be/IurODxLxX9c