UNIVERSIDAD TECNOLÓGICA ISRAEL

ESCUELA DE POSTGRADOS

MAESTRÍA EN TELEMÁTICA,

MENCIÓN: CALIDAD EN EL SERVICIO

(Aprobado por: RCP-SO-19-No.300-2016-CES)

TRABAJO DE TITULACIÓN EN OPCIÓN AL GRADO DE MAGISTER

Título:
Prototipo De Encriptación De Streaming De Video Mediante Python
Línea de investigación:
Telecomunicaciones y sistemas informáticos aplicados a la producción y la sociedad
Autor:
Ing. Juan Diego García Rivera
Tutor:
Ing. Fidel David Parra Balza PhD.

Quito-Ecuador

2020

APROBACIÓN DEL TUTOR

Yo, Fidel David Parra Balza portador de la C.I. 175746995-0 en mi calidad de Tutor del trabajo de investigación titulado: Prototipo De Encriptación De Streaming De Video Mediante Python, elaborado por Juan Diego García Rivera, estudiante de la Maestría en Telemática, mención Calidad en el Servicio de la UNIVERSIDAD TECNOLÓGICA ISRAEL (UISRAEL), para obtener el Título de Magister, me permito declarar que luego de haber orientado, estudiado y revisado la tesis de titulación de grado, la apruebo en todas sus partes.

Quito, 18 de febrero de 2020

Firma

DEDICATORIA

A mis abuelitos, Lupe y Fabian, Victoria y Antonio; por estar siempre pendientes de mi persona, y quienes sin lugar a dudas están inmersos en mis pensamientos.

Y a mis padres, Ximena y Manolo, que, gracias a su apoyo incondicional, me han permitido seguir adelante, aun frente a las dificultades que la vida nos pone en el día a día.

Juan

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABLAS

RESUMEN

La investigación realizada se basó en el estudio de la tecnología de streaming, la cual permite reproducir contenidos de audio o video, sin la necesidad de descargar el contenido en el equipo que se empleé para la reproducción. El problema que se pudo identificar, está dirigido al streaming de video propiamente, el cual no cuenta con un sistema de seguridad que evite el ser intervenido, pudiendo capturarse y vulnerar la privacidad de su contenido. En consideración a lo comentado se plantea como objetivo general, desarrollar un prototipo de encriptación para streaming de video en directo con Python, empleando cámaras digitales brindando un grado de seguridad a este tipo de tecnología. El proyecto se estructuro en base a un enfoque investigativo cuantitativo, evaluando el desempeño del prototipo a través de factores como tiempos de respuesta, rendimiento, seguridad, entre otros. Desde el punto de vista social, el aporte que tendrá el proyecto está dirigido hacia los grupos que manejen la tecnología de *streaming* de video en directo particularmente, dándoles a conocer sobre este criterio de seguridad para sus contenidos, para evitar la filtración o robo de sus contenidos por parte de terceras personas. El proyecto no conto con un gran aporte en cuanto a estudios, documentación y referencias tecnológicas como base para el diseño de este prototipo, y muchos menos relacionados con el lenguaje de programación de *Python*, sin embargo, la investigación en base a sus resultados, contribuye de gran manera a futuros proyectos como fuente de información tecnológica.

Palabras claves: encriptación, streaming, seguridad, privacidad, cámaras digitales.

ABSTRACT

The research was based on the study of streaming technology, which allows you to play audio or video content, without the need to download the content on the equipment used for playback. The problem that could be identified is that the video itself doesn't have a security system that avoids being intervened, able to capture and violate your privacy of its content. In consideration of the above, the general objective is to develop an encryption prototype for live video streaming with Python, using digital cameras providing a degree of security to this type of technology. The project was structured based on a quantitative research approach, evaluating the performance of the prototype through factors such as response times, performance, safety, among others. From the social point of view, the contribution that the project will have is directed towards the groups that handle live video streaming technology in particular, sharing about this security criterion for their contents, to prevent the filtration or theft of its contents by third parties. The project didn't have a great contribution in terms of studies, documentation and technological references as a basis for the design of this prototype, and much less related to the Python programming language, however, research based on its results, contributes greatly to future projects as sources of information.

Keywords: encryption, streaming, security, privacy, digital cameras.

INTRODUCCIÓN

En la actualidad la seguridad informática se ha tornado un factor indispensable al momento que se desea compartir información, a través de los medios tecnológicos que se emplean para este fin. Para lo cual se requiere que la red presente tres factores determinantes como son: la integridad, la disponibilidad y confidencialidad de los datos. Haciendo hincapié en la confiabilidad, las nuevas tecnologías se orientan para proteger la información contra personas ajenas. Si bien existen métodos de protección como antivirus, *firewalls*, *antispam*, etc. constituyen aplicaciones que requieren actualizaciones constantes debido a los diversos ataques que enfrentan diariamente.

Es por aquello que las empresas tecnológicas y desarrolladoras de *software* buscan en las alternativas de seguridad emplear métodos de cifrado de información, sin dejar de lado los sistemas de protección antes mencionados. Los nuevos desarrollos están fundamentados en todo un análisis matemático, que incluyen algoritmos probabilísticos y números primos para la generación de llaves de encriptación y que se ha establecido como una tecnología bastante recurrente para sistemas de protección hoy en día.

Cabe destacar que las técnicas de encriptación datan desde tiempos antiguos, es así como los jeroglíficos del antiguo Egipto son una muestra de aquello. Constituyen los primeros ejemplos de escritura oculta y donde era necesario de una piedra de roseta para descifrar los mismo. Si bien estos métodos eran considerados débiles desde el punto de vista de robustez, hoy día se puede garantizar en gran medida la confidencialidad de la información.

En la actualidad la criptografía es un factor determinante para la seguridad informática, debido a como se mencionó anteriormente emplea algoritmos matemáticos para asegurar la información. Así las aplicaciones que dispongan de información vital como: bases de datos, mensajes de texto, documentos, archivos, entre otros, están seguros.

Por otra parte, y gracias a los avances de la tecnología específicamente con la optimización del ancho de banda, la tecnología de *streaming* está en auge hoy en día. Es así como varias empresas entre las que destacan *Netflix*, *YouTube*, *Spotify*, *Deezer*, emplean *streaming* para reproducir sus contenidos de audio o video directamente sin la necesidad de descargar dicho contenido. Es así como con la tecnología de *streaming* es viable transmitir contenidos grabados previamente o transmitirlos en vivo y a su vez poder visualizarlo en un

Smart tv, en computador, tableta o un *smartphone*. Considerando que la información no se almacena en los equipos, disminuye la probabilidad de plagio o captura de las transmisiones.

Sin embargo, existen equipos como son las cámaras digitales, cuya fuente de video es posible manejarlo a través del *streaming*. Entre estos están cámaras de video vigilancia, cámaras web, la cámara de un smartphone, entre otros, cuyas transmisiones emplea *streaming* en directo. La transmisión se podría equiparar a una video llamada, es decir que tiene un origen y un destino de transmisión.

El problema que se presenta con estas plataformas es que el video aún puede ser intervenido, pudiendo capturarse el mismo y vulnerar la privacidad. Es por aquello que surge la necesidad de crear un prototipo de encriptación para la transmisión de *streaming* en directo, con la finalidad de proteger la información transmitida.

Para poder llevar a cabo esta investigación se hizo una revisión exhaustiva de trabajos previos relacionados, con la finalidad de contextualizar el problema y platear una solución más acertada. Se indago entre diferentes fuentes de información como la *web*, revistas científicas (*IEEEXplorer*, *Redalyc*, *Google Academic*, entre otras), bibliotecas y repositorios de las universidades del Ecuador (UISRAEL, EPN, ESPE, ESPOCH, entre otras).

De esta forma se han podido recabar una serie de trabajos como el desarrollado por Rogel (2016), quien implementa un sistema de *streaming* de video en tiempo real. Mediante su metodología analítica, obtiene como resultado un sistema de *streaming* de video que permite a los usuarios visualizar en tiempo real la funcionalidad de las aplicaciones. Donde concluye que la visualización de los contenidos depende mayormente de la plataforma de uso (exploradores web), la conexión de red, velocidad, latencia y la calidad de servicio del *streaming*, para garantizar la recepción de los contenidos. Así, el estudio permite entender el funcionamiento de un sistema de *streaming* de video.

De igual manera Rodríguez (2001), en su tesis, implementa un simulador de encriptación de audio en Matlab, que, a través de la metodología analítica aplicada, el simulador permitirá estudiar los efectos de la encriptación en un sistema digital, y cuyo alcance es recomendar la aplicación de los algoritmos criptográficos a las tecnologías de tipo digital. Finalmente es importante destacar que el estudio permite comprender que es posible encriptar un sistema, en este caso audio digital con la programación orientada a objetos.

En base a los avances realizados en los estudios mencionados, la presente investigación pretende desarrollar el prototipo de encriptación de *streaming* de video en directo por medio del lenguaje de programación *Python.* El cual constituye uno de los lenguajes más populares

entre los programadores, por su versatilidad y programación orientada a objetos permitiendo desarrollar infinidad de aplicaciones.

Para el desarrollo de esta investigación se plantea como objetivo general:

Desarrollar un prototipo de encriptación para *streaming* de video en directo con *Python* brindando un grado de seguridad a este tipo de tecnología.

Para dar cumplimento al objetivo general se precisan como objetivos específicos:

- Determinar que algoritmos de encriptación actuales se aplican al *streaming* de video.
- Diseñar el prototipo de tal forma que se optimice los recursos tecnológicos en donde se ejecute el mismo.
- Crear el sistema mediante la programación en el lenguaje *Python*.
- Validar mediante pruebas el funcionamiento del prototipo.

El diseño de este prototipo dispondrá de una pertinencia científica al constituirse como una fuente de información, para proyectos relacionados a la encriptación de *streaming* de video, que a su vez será un aporte tecnológico dentro de los protocolos de seguridad y privacidad de datos que se compartan a través de la red. Esta investigación está orientada a aquellas sociedades que dispongan o tenga como proyecto implementar la tecnología de *streaming*, particularmente de video en directo como visión de negocios, considerándolo como un enfoque social y evitar la filtración o robo de sus contenidos por terceras personas.

El presente estudio se estructuro de la siguiente manera:

Capítulo I, donde se describen los fundamentos que conforman el sustento teórico de este estudio y las variables que conforman el mismo. Seguidamente el Capítulo II, que constituye la descripción del proceso investigativo para cumplir con el objetivo propuesto. Así mismo, el Capítulo III muestra el análisis de las pruebas realizadas y la reflexión de los resultados. Y finalmente las conclusiones y recomendaciones de la presente investigación.

CAPITULO I

MARCO TEÓRICO

1.1 SEGURIDAD INFORMÁTICA

En la actualidad la seguridad informática tiene un papel destacado, cuando se establece comunicación entre distintos ordenadores. La gran cantidad de posibilidades que existe de interconectarse a través de la red con la infinidad de plataformas disponibles, hace que las condiciones cambien muy rápidamente.

Para la mayoría de los expertos de esta rama tecnológica, opinan que asegurar un sistema al 100% no es posible, sin embargo, se han considerado ciertos criterios para la protección de los datos en la medida posible. La seguridad informática por definición constituye un conjunto de herramientas, procedimientos y estrategias con el único objetivo de garantizar:

- La integridad de los datos: Donde cualquier cambio o modificación en la información debe ser autorizado y realizado por personas autorizadas de la entidad.
- La disponibilidad del sistema: Se debe mantener una operación continua para mantener la productividad de la entidad.
- La confidencialidad: la divulgación de los datos debe ser autorizada (PandaAncha, 2017).

1.1.1 TIPOS DE SEGURIDAD INFORMÁTICA

La seguridad informática es la rama de la tecnología de la información, cuyo propósito es la protección de los datos en la red. Hoy en día toda organización es dependiente de una u otra forma de la informática, así las tecnologías relacionadas con la seguridad deben mantener un desarrollo constante. La seguridad informática se divide en 3 tipos.

- Seguridad de *Hardware*

La seguridad en *hardware* se orienta a la protección de los ordenadores o dispositivos frente a las amenazas, es decir implica la protección física a los equipos que se emplean en las actividades diaria (Viewnext, 2018).

Entre los métodos más empleados se puede mencionar firewalls de *hardware* y servidores proxy como los de mayor uso, existiendo otros métodos que emplean módulos con claves criptográficas, para cifrar y autenticar los sistemas (Univ. de Valencia, 2019).

- Seguridad de *Software*

La seguridad de *software* tiene como objetivo proteger las aplicaciones y el *software* de amenazas externas, como pueden ser virus o ataques malintencionados. Entre los métodos más empleados se encuentran programas como antivirus, cortafuegos, filtros antispam y *softwares* contra publicidad no deseada, entre otros (Viewnext, 2018).

Los defectos de *software* pueden ser de varios tipos, como errores de implementación, defectos de diseño, mal manejo de errores, etc. Las aplicaciones actuales cuyo diseño es orientado hacia su uso a través de la red, deben disponer de su protección contra ataques, dado que cualquier momento, intrusos pueden acceder mediante la explotación de algún defecto (Univ. de Valencia, 2019).

- Seguridad de la red

La seguridad en la red es aplicada por medio del *hardware* y el *software* del sistema para proteger la información. Así este tipo de seguridad debe enfrentar las amenazas en la red prohibiendo el ingreso a personas no deseadas. Entre las amenazas se encuentran virus, troyanos, *phishing*, suplantación de identidades, entre otros.

Los mecanismos que se emplean son *antispyware*, cortafuegos, redes privadas virtuales (VPN) para garantizar un acceso seguro a la red. Es por eso la importancia de mantener los *softwares* actualizados para mantener un control contra las amenazas (Univ. de Valencia, 2019).

1.2 ENCRIPTACIÓN

La encriptación es un proceso que consiste en asegurar la privacidad de un texto sin formato (texto plano como también se lo denominada) en una secuencia de caracteres ilegibles mediante una clave. El procedimiento consiste en tomar los datos, como pueden ser archivos o documentos y alterarlos mediante un algoritmo, evitando que sean legibles para personas no autorizadas.

La encriptación informática, emplea el termino encriptar o cifrar para codificar la información y protegerla frente a terceros. Constituye un recurso muy empleado en la

actualidad para garantizar la transferencia de datos. Solamente y a través de un software de decodificación que únicamente conoce el autor de la encriptación, los documentos y/o archivos podrán ser decodificados, empleando la llave o clave con la que el autor inicio el proceso de encriptación.

A modo de ejemplo se puede citar a los sistemas de mensajería, correos electrónicos, bancos y muchas entidades privadas y públicas que disponen de bases con datos personales. En definitiva, la forma de encriptan los datos es empleando dos elementos, un algoritmo criptográfico y una llave secreta.

1.2.1 METODOLOGÍAS DE ENCRIPTACIÓN

1.2.1.1 TRANSPOSICIÓN

Consisten en alternar de posición(permutar) las letras de un mensaje, y donde es común emplear una matriz (matriz de transposición) para la reconstrucción del mensaje original. Resulta un método muy simple, y por lo tanto no muy efectivo, considerando que siempre se mantiene el texto plano.

1.2.1.2 SUSTITUCIÓN

Es un método bastante simple, consiste en que cada letra del alfabeto le corresponde otra. Dicha correspondencia se encuentra en la clave, por lo cual aquí es vital la seguridad de la misma. Constituye uno de los primeros métodos, donde considerando la época, resultaban muy útiles, pero no eran muy robustos por lo que el grado de dificultad para descifrar era bajo.

1.2.2 TIPOS DE CIFRADO

La criptografía moderna emplea algoritmos matemáticos, como métodos para modificar los datos con el objetivo de añadir la característica de seguridad requerida. El nivel de robustez de cada algoritmo es difícil de evaluar, dado que las vulnerabilidades que podrían mostrarse son ante diferentes técnicas o ataques muy impredecibles.

Pero la robustez es una las características para evaluar los algoritmos, también está presente el tiempo en el proceso de cifrado, el tiempo en el proceso de descifrado, el tamaño del archivo original y el algoritmo de cifrado propiamente, entre otros.

De forma general los tipos de algoritmos se ven reflejados en la siguiente clasificación de cifrados:

- Criptografía Simétrica
- Criptografía Asimétrica
- Cifrado hibrido

A continuación, se brindará detalles de los tipos de cifrado antes mencionado, donde es necesario recalcar que, si bien existen otros tipos, estas corresponden a los métodos modernos de criptografía.

1.2.2.1 CRIPTOGRAFÍA SIMÉTRICA

Este sistema de cifrado emplea la misma clave o llave para cifrar y descifrar un archivo o documento, lo que implica que tanto el emisor y el receptor tengan conocimiento de la misma. Obviamente en este algoritmo la definición de la llave o clave tendrá una gran complejidad, para evitar que sea descifrada por los ordenadores modernos, por lo que su fortaleza dependerá de su complejidad y longitud.

Sin embargo, se puede apreciar un punto débil por decirlo de alguna manera, en esta encriptación, el hecho de compartir la llave o clave de forma insegura es la principal desventaja, razón por la cual es necesario buscar la forma más adecuada y segura para transmitir dicha clave. Para superar esta desventaja se desarrolló la criptografía asimétrica., pero la criptografía simétrica no deja de ser una herramienta competitiva.

Entre los algoritmos simétricos más conocidos estas: DES, 3DES, RC4, Blowfish y AES. Un esquema de este tipo de criptografía se muestra en la figura 1 (Real, 2019).

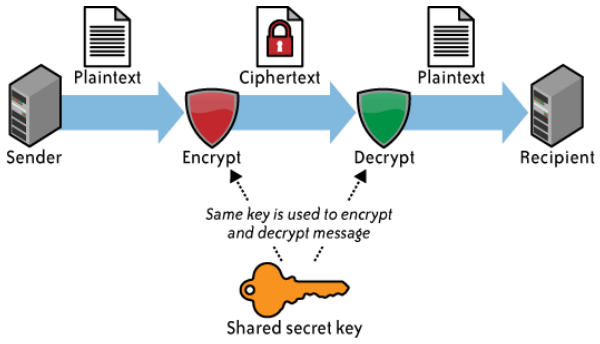

Figura 1. Criptografía simétrica. Copyright 2020 por PandaAncha.

1.2.2.2 CRIPTOGRAFÍA ASIMÉTRICA

También conocida como la encriptación de llave pública. Es un sistema de cifrado que usa dos claves diferentes. Emplea un conjunto de algoritmos criptográficos, donde la llave publica, la cual se puede difundir sin ningún inconveniente de seguridad y privacidad, y la llave privada que en lo posible nadie debe tener acceso, más que el personal autorizado.

La forma en que se maneja este algoritmo, consiste en encriptar el documento mediante la llave publica para que pueda ser enviado. El receptor, en cambio deberá ejecutar el descifrado con la llave privada. La criptografía asimétrica resuelve una desventaja de la criptografía simétrica. Por lo que no es necesario disponer de un canal seguro para distribuir la llave o clave publica, sin embargo, la llave privada si debe disponer de un canal seguro.

Al mismo tiempo surge la desventaja de esta criptografía, y resulta que no es muy eficiente, debido a que las claves que se generan deben ser largas y tarda un tiempo considerable en aplicarlas. Entre los algoritmos asimétricos conocidos están RSA, DSA. (Real, 2019). Un esquema de este tipo de criptografía se muestra en la figura 2.

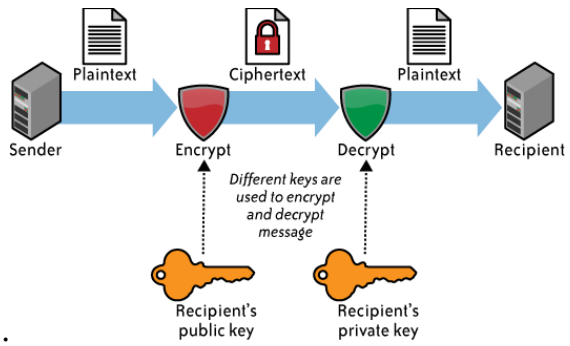

Figura 2. Criptografía asimétrica. Copyright 2020 por PandaAncha.

1.2.2.3 CRIPTOGRAFÍA HIBRIDA

Este tipo de criptografía tiene como principal característica los atributos de los modelos asimétrico y simétrico. Sin embargo, no significa que sea más o menos seguro que los métodos anteriores.

Aquí el cifrado de clave publica se emplea para compartir la clave de cifrado simétrico. Es decir, se cifra el archivo o documento con la llave privada. La gran mejora que se tiene aquí es que cada clave simétrica será diferente, y solo se usará en una sesión por vez. Al final para desencriptar el archivo o documento, se emplea la clave privada para descifrar la clave

simétrica y posteriormente descifrar el archivo. En la figura 3 se puede apreciar un esquema de esta criptografía.

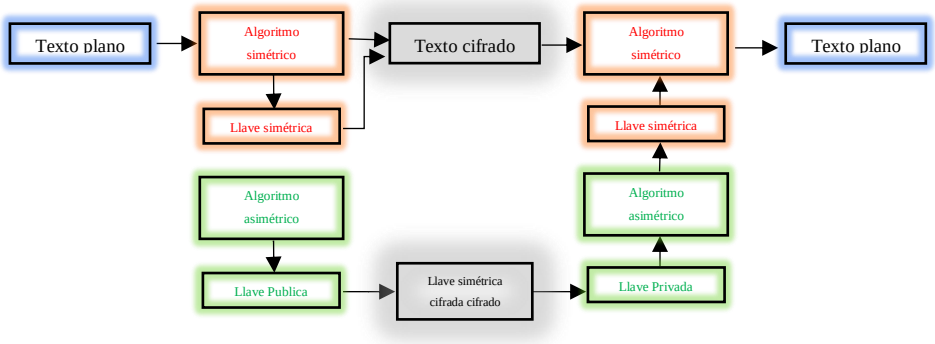

Figura 3. *Criptografía hibrida.*

1.2.3 CLAVE DIGITAL

Los métodos actuales de codificación emplean sus claves digitales en base a una secuencia de bits, específicamente en la longitud. Así, los nuevos métodos como AES permiten escoger la longitud de la clave de entre 128, 192 o 256 bits.

Los cifrados de 128 bits por ejemplo equivale a 2^{128} combinaciones de claves, lo que corresponde aproximadamente 340 millones de quintillones. Es decir que prácticamente tratar de descifrar estas combinaciones tomaría años, incluso con la tecnología actual.

1.2.4 ENCRIPTACIÓN AES

Por sus siglas Estándar de Encriptación Avanzada *(Advanced Encryption Standard*), constituye una técnica de cifrado simétrica, la cual se ha optado por su fiabilidad, rendimiento y seguridad. Entre las principales aplicaciones donde se aplica están a entidades financieras, gubernamentales y militares.

Otra característica de este método, es que no requiere de licencias ni de realizar pagos por su adquisición respecto a las patentes. Los requisitos de *hardware* y almacenamiento son relativamente bajos.

AES permite un bloque de cifrado de tamaño variable de 128 bits, 192 bits y 256 bits asociado a claves de los mismos tamaño o longitud. La variación del tamaño de cada clave de cifrado hace que el algoritmo cambie, aumentando la complejidad del algoritmo de cifrado en sí. Los datos cifrados pueden ser descifrados y editados en distintas plataformas con cualquier aplicación compatible AES (Real, 2019).

1.2.5 ENCRIPTACIÓN RSA

Este sistema criptográfico también es conocido como el sistema de clave publica, y sus siglas se deben a sus creadores RSA (Rivest, Shamir, Adleman). RSA es uno de los métodos más conocidos y más usados dentro de los sistemas de llave pública. Una de las principales ventajas es que no conlleva complicaciones en la implementación lo que a su vez hace que sea un sistema muy rápido.

Entre las aplicaciones más importantes se encuentran las páginas *web*, donde se tiene SSL/TLS (*Secure Socket Layer* o en español, capa de puertos seguros). Algoritmos que se encuentra funcionando proporcionando privacidad. En lo que respecta a las claves, lo más recomendable es que las mismas tengan una longitud de al menos 1024 bits.

1.3 PROTOCOLOS DE TRANSPORTE

1.4 PROTOCOLO TCP

TCP/IP son las siglas de Protocolo de Control de Transmisión/Protocolo de Internet. TCP constituye un protocolo diseñado para el transporte de datos entre ordenadores. La entrega de datos está garantizada en estas transmisiones por lo que los paquetes serán entregados en el mismo orden en el cual fueron enviados.

Constituye un protocolo orientado a conexión, es decir que no solo se encarga de transmitir la información de un punto a otro, sino también verifica la correcta recepción de la información. La característica de fiabilidad, lo convierte en uno de los protocolos más habituales, que incluso soporta a los protocolos HTTP, SMTP, SSH y FTP.

1.4.1 CAPAS DEL MODELO TCP/IP

En este modelo se dispone de las siguientes capas:

Capa de aplicación: es la capa que proporcionan la interfaz entre las aplicaciones que se emplean para comunicar y la red en la cual se transmiten los mensajes. Entre algunas de las aplicaciones que se utilizan para intercambiar datos se encuentran Telnet, SMTP, FTP, HTTP, etc.

Capa de transporte: garantiza que los paquetes lleguen en secuencia y sin errores. Permite conocer el estado de transmisión, los datos de enrutamiento, y los puertos que se

asocian a la aplicación. Entre los protocolos se tiene TCP y UDP. Cabe recalcar que los protocolos TCP proporcionan un servicio completo y fiable, mientras que UDP constituye un servicio poco fiable.

Capa de internet: este nivel se encarga de direccionar y guiar los datos desde el origen hacia el destino independientemente de la ruta. El protocolo que rige esta capa es Protocolo Internet (IP), proporcionando los servicios.

Capa de acceso a la red: este nivel tiene como objetivo intercambiar los datos entre el sistema final y la red de por medio. Y constituye la primera capa del modelo, como el acceso físico a la red, determinando la topología a través de *routers*, *switch*, etc (*OpenWebinars*, 2019).

1.5 PROTOCOLO UDP

UDP son las siglas de *User Datagram Protocol, c*onstituye un protocolo no orientado a conexión. Al igual que otros protocolos, pertenece a la familia de protocolos de internet, por lo que puede clasificarse en la capa de trasporte del modelo TCP/IP.

Constituye un protocolo simple, dado que no verifica la recepción de los datos transmitidos entre un dispositivo y otro, se puede expresar también como que el destinatario no conocerá al emisor de datos excepto su dirección IP.

Sin embargo, la principal ventaja de UDP consiste en su velocidad. Al omitir la verificación, lo hace el protocolo idóneo para la transmisión de servicio de audio y video en streaming, donde la transmisión es más importante que una posible pérdida puntual.

1.6 STREAMING

El *streaming* es el término que se utiliza para hablar de toda aquella emisión de contenidos audiovisuales realizada a través de Internet. Las aplicaciones más comunes son escuchar música o ver videos mientras se transmitan en vivo o se encuentre alojados en un servidor, sin la necesidad de realizar la descarga previa en dispositivos como computadores, teléfonos inteligentes, entre otros.

La arquitectura de un servicio de *streaming* se puede visualizar en la figura 4:

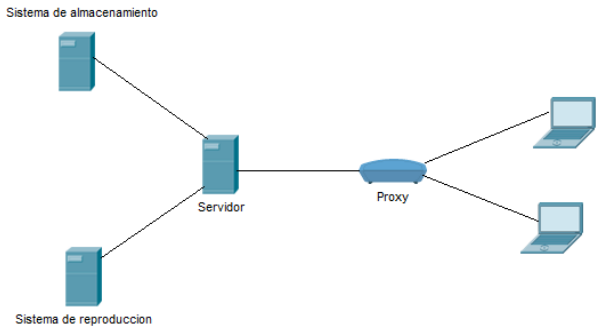

Figura 4. Arquitectura tradicional de streaming.

La captura de contenido audio o video, se lo realiza de elementos como cámaras y micrófonos. Y mediante un software para la codificación se obtiene el formato para la transmisión. El servidor de *streaming* procesa las peticiones de los clientes, cuando este último comienza a recibir el archivo, construye un buffer para almacenar. Y cuando este buffer se ha llenado con una fracción inicial del archivo, el reproductor comienza a mostrarlo, mientras en segundo plano se continua con el resto de la descarga.

1.6.1 TECNOLOGÍAS DE *STREAMING*

1.6.1.1 EN DIRECTO O EN VIVO

Evento que se da cuando la transmisión es en vivo, y se transmite en el mismo instante que sucede. En este tipo de tecnología, la orientación está definida a la multidifusión, y no existe interactividad alguna. De acuerdo al tipo de transmisión puede ser:

- *Unicast*: se envía un flujo a cada usuario
- *Multicast*: se envía un solo flujo único de información, tal y como se puede apreciar en la figura 5.

Figura 5. Tipos de transmisión de streaming. Tomado de Tecnologías de Streaming por Suárez (2010/2011).

1.6.1.2 VIDEO BAJA DEMANDA

En esta clasificación, los usuarios pueden demandar las transmisiones en cualquier instante. Es decir, se envía un flujo a cada usuario solicitante. Entre las interacciones es posible hacer pausas o saltos hacia adelante o atrás sobre la transmisión, la cual es de tipo *unicast*.

1.7 PROTOCOLOS DE *STREAMING*

Los protocolos de *streaming* de igual manera se clasifican en base a los tipos que existen:

1.7.1 PROTOCOLOS PARA *STREAMING* EN DIRECTO

Los protocolos para la transmisión en tiempo real no pueden estar basados en TCP. Debido a su funcionamiento, si en algún momento de la conexión se produce un error o una falla, obligatoriamente la transmisión deberá volver a iniciar. Sin embargo, en ciertos casos es posible emplear TCP. Entre los protocolos basados en UDP, se tiene a RTP (*Real-time Transport Protocol*), que como su nombre lo indica proporciona una transmisión de extremo a extremo y en tiempo real.

RTSP (*Real-time Streaming Protocol*), constituye otro protocolo que establece y controla la sesión entre el cliente y servidor de datos en tiempo real. En las sesiones de RTSP es posible realizar las acciones de pausar, retroceder o avanzar (Rogel, 2016).

1.7.2 PROTOCOLOS PARA *STREAMING* BAJO DEMANDA

Dado que la transmisión de audio y video no es en tiempo real, es posible emplear protocolos basados en TCP, como puede ser HTTP y FTP. De esta forma se asegura que los datos lleguen a su destino.

HTTP Streaming, transmite audio y video a través de HTTP desde un servidor *web* en respuesta a un evento. Entre las aplicaciones más comunes para este protocolo están, sistemas de mensajería, chat en vivo, juegos entre otros. Cabe recalcar que HTTP funciona normalmente en el puerto 80 o 8080 (American Dominios, 2020).

1.8 PYTHON

Es un lenguaje de programación moderno, orientado a varios estilos como objetos, programación estructurada, programación funcional, entre otros. Tiene un gran potencial gracias a su simplicidad y es de código abierto. Guido Van Rossum fue su creador. Cuyo objetivo era evitar aquellos lenguajes que empleaba C, tornándolo de una manera más accesible para todo el mundo. *Python* es un lenguaje de programación de *script*, los cuales usan un intérprete en vez de empleados directamente en un compilador (OpenWebinars, 2019).

Considerado en la actualidad uno de los más populares, tiene la ventaja de disponer de una gran cantidad de librerías, funciones, módulos, *frameworks*, plantillas, etc. Entre sus aplicaciones están:

- Cálculos de ingeniería
- Desarrollo *web*
- Juegos
- *Big Data*
- *Machine Learning.*

1.9 OPENCV-PYTHON

Constituye una biblioteca de *Python*, diseñada para resolver problemas de visión por computadora. Adicional hace uso de *Numpy*, una librería para realizar operaciones matemáticas. Es decir que todas las operaciones de *OpenCV* se convierten en matrices *Numpy*, lo que a su vez facilitará que estas operaciones se acoplen a otras liberarías para su uso.

Otra característica de *OpenCV*, es que al ser de libre distribución es posible modificar, o realizar contribuciones para mejorar esta librería. Entre otra de sus ventajas, es que es multiplataforma al igual que Python, por lo que es posible disponer de sus versiones para *GNU/Linux*, *Mac OS*, *Windows* y *Android* (*OpenCV-Python*, 2013).

CAPITULO II

DESCRIPCIÓN DEL PROCESO INVESTIGATIVO

2.1 MARCO METODOLÓGICO

La presente investigación se estructuró en base a un enfoque investigativo cuantitativo, manteniendo en consideración que se podrá evaluar el desempeño tecnológico del prototipo, a través de factores como tiempos de respuesta, rendimiento, facilidad de operación, seguridad, entre otros.

Acorde al enfoque investigativo, es indispensable adoptar una metodología que se acople a la naturaleza del proyecto, es así como se precisa usar el método experimental, cuyo objetivo es tratar de predecir eventos mediante la formulación de pruebas o planteamiento de hipotesis. El prototipo de encriptación de streaming de video mediante Python, manejara como variable independiente la fuente de video y como variable dependiente el algoritmo de encriptación (González, 2018).

La técnica que permitirá poner en práctica la metodología seleccionada es el análisis de documentos. La interpretación y el análisis, permitirán revelar la importancia de los mismos acordes a los propósitos del investigador. Desde el punto de vista social, el aporte que tendrá el proyecto está dirigido hacia los grupos que manejen la tecnología de *streaming*, dándoles a conocer sobre este criterio de seguridad para sus contenidos.

2.2 DISEÑO DEL PROTOTIPO

En la figura 6 se muestra el esquema del diseño del prototipo, el cual se estructuro en base a una revisión exhaustiva de trabajos en relacionados al tema de investigación y en la lógica del proceso. La forma de este esquema permitirá identificar de mejor manera el tratamiento o el procesamiento de la información proveniente de la fuente de video, sin alterar su contenido como resolución, imágenes por segundo, etc. Adicionalmente la estructura permitirá identificar de forma ágil la presencia de algún error o inconveniente en el funcionamiento del prototipo.

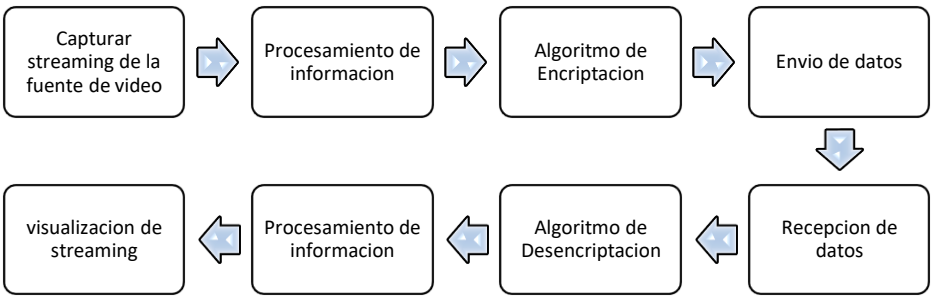

Figura 6. Esquema del prototipo de encriptación

2.3 ENTORNO DE PROGRAMACIÓN

El desarrollo de este prototipo se llevará a cabo en el lenguaje de programación de *Python*. El objetivo es diseñar un programa que, mediante la programación orientada a objetos, se pueda capturar el video *streaming* y procesar esta información en los algoritmos de encriptación que existen en la actualidad, y cuya integración con *Python* sea totalmente confiable.

Dentro de este diseño será necesario implementar un sistema de envío y recepción de datos, con el fin de cubrir la transmisión por medio de la red, empleando la figura cliente-servidor. Para de esta forma garantizar que solo personal autorizado tenga acceso a la información.

2.4 FUENTE DE VIDEO *STREAMING*

Para este diseño se ha considerado el uso de cámaras de video digitales, a través del protocolo HTTP *streaming*. Con lo cual se podrá capturar el video en tiempo real y realizar el procesamiento de encriptación.

Uno de los equipos que se empleará será un *Smartphone*, tal y como se visualiza en la figura 7. Dado que dispone de una cámara es posible y mediante el uso aplicaciones obtener el *streaming* de video, con lo cual constituye una de las posibles fuentes de video que puede aplicarse en este prototipo.

Constituye una opción viable considerando que es un equipo del diario vivir de las personas, donde mayormente es empleado para fotografías y videos, sin considerar su potencial.

Figura 7. Cámara de smartphone.

Una segunda alternativa, constituye las cámaras integradas en las *laptops*, tal y como se puede apreciar en la figura 8. En la actualidad muchos de estos equipos cuentan con este dispositivo comúnmente para realizar video conferencias, es ahí donde se puede aplicar el concepto de la encriptación de video, considerando que son transmisiones en tiempo real.

Figura 8. Cámara de laptop.

Sin embargo, otra alternativa existente y que ha perdido popularidad, son las cámaras *web* USB, tal y como se aprecia en la figura 9. Que de igual manera dependiendo de la aplicación en las que sean empleadas pueden transmitir video en tiempo real. Lo cual constituye una opción para el este diseño.

Figura 9. Cámara USB.

2.4.1 CAPTURA DE *STREAMING*

Empleando los protocolos de HHTP *streaming* o RSTP es posible acceder al video de los equipos digitales, tales como teléfonos inteligentes, cámaras de video vigilancia, cámaras

incorporadas en *laptops*, entre otros. Es decir, que este prototipo teóricamente podría acceder a cualquier equipo que disponga de estas características y que se encuentre en la red.

Para la captura de video es necesario identificar las URL que servirá como origen para la librería *OpenCV* para la adquisición propiamente del video.
Entre varios ejemplos se pueden citar los siguientes:

- http://192.168.1.2:4747/video
- http://134.1.213.225:4747/video
- rtsp://192.168.1.88
- rtsp://194.218.96.92:554
- rtsp://admin:Usuario@192.168.1.35:554/h264/ch1/main/av_stream

Como se puede apreciar existe varios formatos, los cuales se definen por cada fabricante o modelo del equipo. La manera más apropiada de conocer el formato adecuado, es indagar dentro de la *web* de cada fabricante y ubicar dicha URL.

Un método muy efectivo para visualizar los *streaming* de video independientemente del equipo a modo de prueba, es empleando el reproductor VLC *player*. El cual dispone de la opción "Abrir Ubicación de red" donde es posible tipiar las URL mencionadas y obtener el video. Tal y como se puede apreciar en la figura 10.

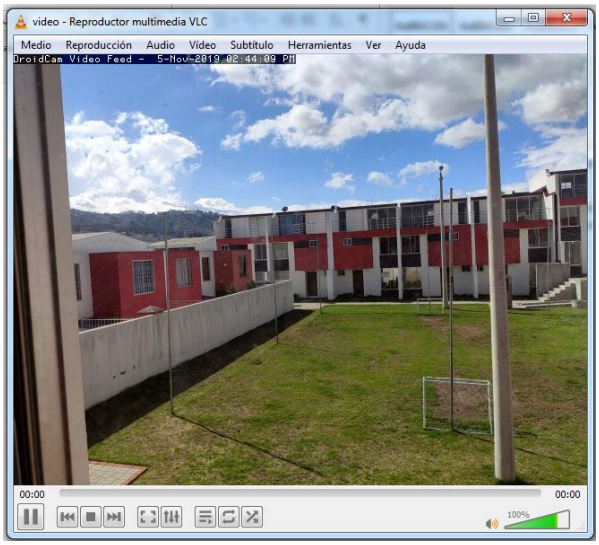

Figura 10. Demostración de video streaming por *VLC player*.

2.5 ENCRIPTACIÓN Y DESENCRIPTACIÓN DE DATOS

Para el proceso de encriptación de la información se procederá con los métodos criptográficos modernos, es decir, se analizará cuál de los métodos simétrico o asimétrico se acopla de mejor manera al diseño del prototipo.

Python dispone de varias librerías, y dentro de ellas se encuentra "*pycrypto*". Una herramienta de criptografía con varios algoritmos de cifrados como son AES, DES, RSA, ELGAMAL, etc (Pycrypto, 2013). La ventaja de emplear esta herramienta es que es posible experimentar con algoritmos los cuales han sido probados y que cuentan con un alto grado de seguridad.

Al interior de la librería se cuenta con una herramienta para generar números aleatorios(*random*), cuya aplicación está destinada mayormente para la generación de las llaves, las cuales son vitales para la encriptación y desencriptación.

2.6 ENVIÓ Y RECEPCIÓN DE DATOS

Parte de la transmisión de archivos, documentos, imágenes, videos, entre otros, a través de la red involucra un transmisor y un receptor. De igual manera este diseño contempla la figura de un cliente y un servidor. Dada la facilidad de *Python*, es posible crear *scripts* para la ejecución de este prototipo.

El diseño será de la siguiente manera, ce dispondrá de un código fuente para la captura, encriptación y envío de datos, archivo que en otras palabras será denominado "cliente". Y por otro lado se dispondrá de un código fuente para la recepción, desencriptación y visualización del video, es decir será el "servidor". La transmisión de datos será a través de uno de los protocolos de red UDP o TCP.

Al interior de *Python* una de las librerías que permite ejecutar la comunicación entre equipos de una red, es mediante "*sockets*". Un *socket* está definido por la dirección IP, el puerto y el protocolo. Los *sockets* pueden ser implementados por medio de diferentes canales, como TCP, UDP, o de dominio de UNIX. La librería en mención dispone de una gran documentación que facilita entender su funcionamiento, mediante la ejecución de comandos. De esta forma se estructura un código fuente para el cliente y para el servidor.

2.7 VISUALIZACIÓN DEL *STREAMING*

Como se comentó anteriormente el reproductor VLC *player* permite visualizar el *streaming* se video, de manera externa. Sin embargo, para el diseño del prototipo se plantea emplear el visualizador que dispone la librería de *OpenCV*, que se acopla al sistema operativo en el cual se esté ejecutando el programa. A manera de ejemplo se puede visualizar en la figura 11 el visualizador de *OpenCV*. De esta manera el objetivo es optimizar los recursos de las librerías de *Python*, sin agregar otras aplicaciones externas.

Figura 11. Visualizador de OpenCV.

2.8 DESARROLLO DE PROGRAMAS

Este diseño emplea dos *scripts* desarrollados en *Python*. Uno con la figura de "cliente" y el otro con la de "servidor".

El código del cliente, ejecutara las funciones de captura de video *streaming*, encriptación de la información y envió de datos hacia el servidor. Desde el lado del servidor, se ejecutará las funciones de recepción de datos, desencriptación y visualización del video propiamente dicho.

2.8.1 PROGRAMA SERVIDOR

Se inicia con la importación de las librerías, la definición de variables, configuración de puertos y demás elementos. Al ser considerado el servidor, como es lógico siempre deber estar en ejecución, para que de esta forma los equipos clientes, puedan ubicarlo de primera mano dentro de la red.

En este diseño, el servidor es el encargado de generar la llave criptográfica, la cual deberá ser compartida solo con el personal autorizado. Si bien el ambiente puede ser una red

privada, los factores de seguridad deben mantenerse en cuanto a privacidad se refiere. En la figura 12 se muestra el diagrama de flujo desarrollado para el código fuente del servidor.

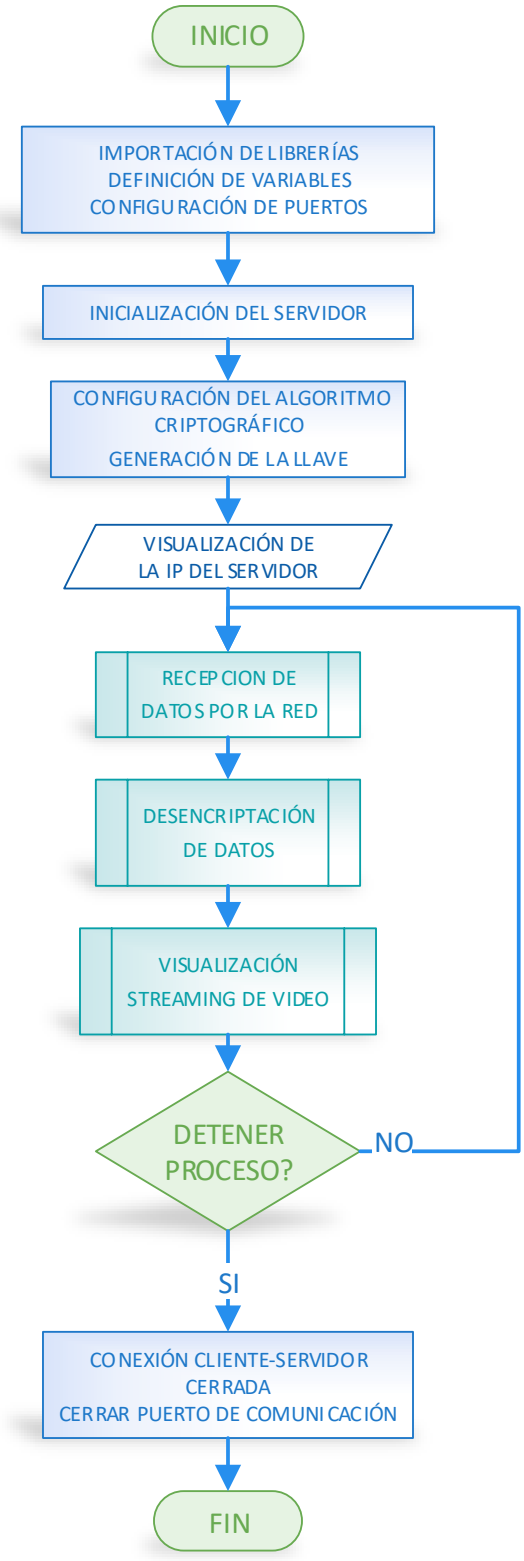

Figura 12. Diagrama de flujo del servidor.

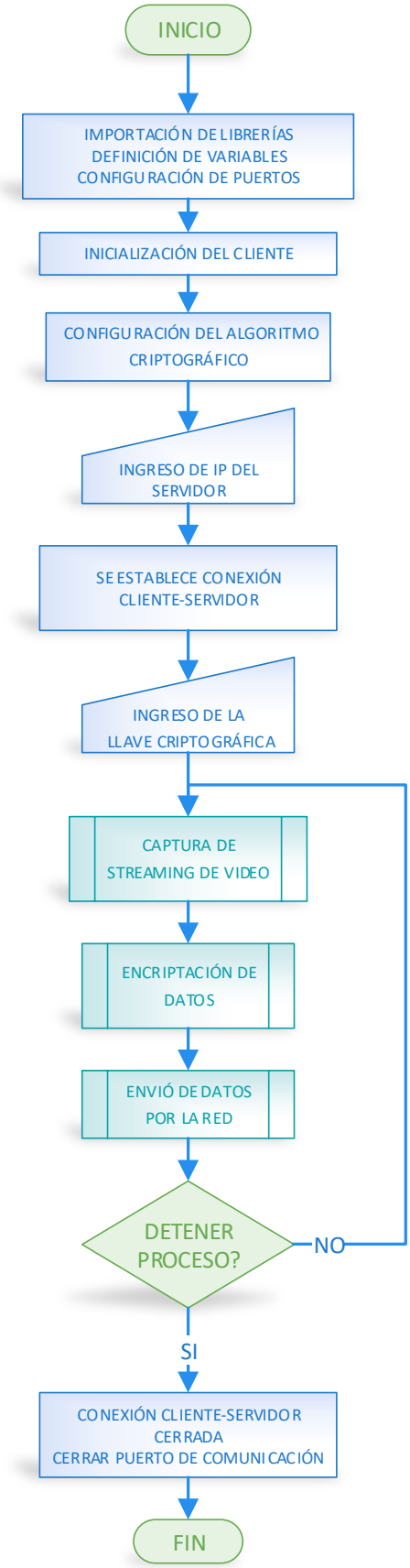

Figura 13. Diagrama de flujo del cliente.

2.8.2 PROGRAMA CLIENTE

De igual manera se inicia con la importación de las librerías, definición de variables, configuración de puertos y demás elementos. Considerando que el programa se encuentra en un lazo repetitivo, la captura y procesamiento de la información será constante.

Dentro de la programación se incluye una sección para ingresar la llave criptográfica provista desde el servidor. Lógicamente si la llave ingresada no es la adecuada, todo el proceso no se realizará correctamente. En la figura 13 se muestra el diagrama de flujo desarrollado para el código fuente del cliente.

2.8.3 SUBRUTINA CAPTURA/VISUALIZACIÓN DE STREAMING

Esta subrutina fue diseñada en un solo contexto, con la diferencia que es necesario identificar que secciones se acoplan al código fuente del cliente o del servidor. En la figura 14 se muestra el diagrama de flujo desarrollado para esta subrutina.

Figura 14. Diagrama de flujo de la subrutina captura/visualización del video.

2.8.4 SUBRUTINA ENCRIPTACIÓN/DESENCRIPTACIÓN DE DATOS

Esta subrutina se ha elaborado en un solo contexto, por lo que solo es necesario identificar que secciones se acoplan al código fuente del cliente o del servidor. En la figura 15 se muestra el diagrama de flujo desarrollado para esta subrutina.

Figura 15. Diagrama de flujo de la subrutina encriptación/desencriptación de datos.

2.8.5 SUBRUTINA ENVIÓ/RECEPCIÓN DE DATOS

Esta subrutina se ha elaborado en un solo contexto, por lo que solo es necesario identificar que secciones se acoplan al código fuente del cliente o del servidor. En la figura 16 se muestra el diagrama de flujo desarrollado para esta subrutina.

Figura 16. Diagrama de flujo de la subrutina envió/recepción de datos.

CAPITULO III

PROPUESTA

3.1 ESTUDIO DE FACTIBILIDAD

El desarrollo del proyecto se estructuro en base a un estudio de factibilidad, el cual permite definir la viabilidad de un proyecto para su ejecución y poder convertir el mismo en un bien o servicio de forma general. A continuación, se detalla los componentes del estudio en mención.

El proyecto plantea como objetivo implementar una técnica de encriptación para los sistemas de streaming de video. Resumiendo, el proyecto trata sobre encontrar un método para brindar seguridad y privacidad al streaming de video en directo, problema que se identificó considerando que dichas transmisiones pueden ser intervenida y vulneradas. El streaming de análisis se lo puede equiparar al funcionamiento de una video llamada, es decir, no requiere de la descarga de contenido adicional en el dispositivo que se empleé para su funcionamiento, más que la transmisión propiamente entre 2 usuarios. En consideración de lo mencionado se realiza el estudio en los siguientes factores:

Mercado potencial:

Indudablemente existe un mercado donde se puede presentar esta propuesta, y son aquellos grupos que emplean el streaming para fines comerciales, y dependido del tipo acorde a la aplicación pueden ser streaming en directo, empresas como Microsoft, Google, Apple, entre otros, o streaming bajo demanda, empresas como Netflix, HBO, Amazon Prime y otros más.

Factibilidad técnica:

Efectuando una indagación sobre el tema de intereses, lamentablemente no existe documentación en sí, sobre proyectos similares que puedan aportar al desarrollo de este proyecto. En tal razón, se plantea definir que plataformas de programación se puedan acoplar de mejor manera a este desarrollo. A continuación, se muestra la tabla 1 con el análisis correspondiente:

Tabla 1. Estudio de factibilidad técnica

Lenguaje de programación	Matlab	C++	Python	Java
Multiplataforma	Si	Si	Si	Si
Librerías de visión artificial	Si	Si	Si	Si
Comunicación TCP/UDP	Si	Si	Si	Si
Creación de scripts	Si	Si	Si	Si
Licenciamiento	Pago	Pago	Libre	Pago
Recursos mínimos de funcionamiento	Procesador: Intel o AMD x86-64 RAM: 1GB mínimo, 4 GB recomendado. Tarjeta gráfica: Soporte para OpenGL 3.3 recomendado con 1 GB en GPU.	RAM: 1 GB de RAM para x86. 2 GB de RAM para x64. Tarjeta gráfica: vídeo compatible con DirectX 9, resolución 1024 x 768 o superior. Se requiere Microsoft Visual C++	Procesador: Intel o AMD x86-64 RAM: 2 GB o más superior.	Procesador: x64. RAM: 128 MB o superior Se requiere: Internet Explorer 9 o superior, Firefox

Las opciones presentadas, cumplen con los requisitos fundamentales para el proyecto como se puede apreciar, sin embargo, y considerando que se trata de la creación de un prototipo, la opción del lenguaje de programación de *Python*, es una muy buena alternativa, razón por la cual se opta por esta.

Factibilidad Operativa:

En esta sección se analiza el recurso humano para el desarrollo del proyecto. Considerando que el diseño está orientado a la figura de un trasmisor y de un receptor, el personal idóneo seria 2 programadores, para que su interacción sea complementaria, y tanto en la fase de auditoria del código, así como en la ejecución de pruebas permitiendo alcanzando una optimización del personal.

Factibilidad económica

Sin lugar a dudas uno de los temas con mayor discusión es el factor económico en todo proyecto, más aún cuando se trata de definir sobre el desarrollo de un programa o un software. Normalmente estos estudios inicialmente no reflejan una valoración en cuanto a la iniciativa, que se da por parte de los programadores cuando se propone proyectos tecnológicos. En consecuencia, de aquello se consideran otro tipo de factores que en combinación con los seleccionados para este estudio se presentan a continuación:

Tabla 2. Análisis de factibilidad económica

Factores	Cantidad
Recurso humano	2 programadores con fundamentos en programación Python.
Tiempo de ejecución	3 meses
Recursos de Hardware y Software	2 computadores, con sistemas Operativo (Windows, Mac o Linux) con licencia (en caso que aplique) y editor de texto. Características mínimas técnicas del equipo (seleccionado en base a los requisitos recomendados del sistema operativo Windows 10) Procesador: Procesador a 1 GHz o superior. RAM: 1 GB para 32 bits o 2 GB para 64 bits. Espacio en disco duro: 16 GB para un SO de 32 bits o 32 GB para un SO de 64 bits. Tarjeta gráfica: DirectX 9 o posterior.
	1 Router con capacidad 300mbps y accesorios de conexión

Como se puede apreciar en la tabla 2, la viabilidad de este proyecto, económicamente es positiva, considerando que el factor más destacable es el recurso humano, en comparación con otras.

3.2 FUNDAMENTOS DE LA PROPUESTA

En este capítulo se expone la propuesta en base a los lineamientos técnicos y teóricos, para el desarrollo del "Prototipo de encriptación de streaming de video en Python", el cual se fundamenta en la necesidad de brindar seguridad y privacidad a este tipo de tecnología evitando su intervención o filtración de sus contenidos.

La presente investigación tiene como principales beneficiarios aquellos desarrolladores y programadores que empleen la tecnología de *streaming* para la transmisión de contenidos de video en este caso.

El propósito de la propuesta es incentivar a los beneficiarios a la implementación de algoritmos de seguridad basados en la criptografía para las transmisiones de *streaming* en directo, teniendo como base el desarrollo del presente proyecto.

El prototipo desde una perspectiva técnica, cuenta con una gran factibilidad, dado que se emplea como principal recurso, el lenguaje de programación *Python*, el cual dispone de una gran acogida a nivel mundial para el desarrollo de múltiples aplicaciones.

Por otro lado, es necesario indicar la perspectiva económica de esta propuesta, la cual no involucra la inversión de ningún hardware adicional para la ejecución del prototipo. Indudablemente el producto en este caso requiere mejoras, dado que, si se habla de una versión comercialmente, la misma aún no está disponible por temas de desarrollo, y es por aquello que el estudio económico se enfoca directamente en los costó del programador, el cual este sujeto a las competencias laborales.

Entre las principales ventajas que se describen en esta propuesta sobre el prototipo se tiene las siguientes:

1. El algoritmo desarrollado permitirá manejar diferentes equipos que tengan acceso al *streaming*, entre ellos se pude nombrar, cámaras web, cámaras de video vigilancia, incluso las cámaras del smartphone, así se podrá visualizar el potencial del prototipo.
2. Al ejecutarse sobre el lenguaje de programación de *Python*, no es necesario adquirir licencias, ni claves de activación, ya que es de código abierto, y donde incluso existe una gran comunidad dispuesta a brindar ayuda en los desarrollos.

   De igual manera no es indispensable contar con un hardware extremadamente moderno para la ejecución de este software, ya que el mismo se encuentra optimizado.

3.3 PRUEBAS Y ANÁLISIS DE RESULTADOS

Una vez que se cumplió con la parte del diseño y la programación del prototipo, es indispensable exponer su funcionamiento, así como los diferentes inconvenientes encontrados durante su etapa de desarrollo, que son necesario mencionarlos. Para las pruebas que se describirán a continuación, se empleó las siguientes versiones de los *softwares*:

Python 3.7.3 *OpenCV* 4.1.1 *Crypto* 3.9.1

3.4 CAPTURA DE *STREAMING* DE VIDEO

Para el diseño es indispensable contar con algún aplicativo que permita capturar video y a su vez que se acople a lenguaje de programación *Python*. Razón por la cual se ha seleccionado la librería de *OpenCV*. La cual no es exclusiva de *Python*, sin embargo, se acopla mediante librerías. Entre las primeras pruebas se diseñó un código que permita capturar el video de varias fuentes, tal y como se puede apreciar a continuación.

Captura 1: en la figura 17, se puede apreciar la captura del video en tiempo real de una cámara IP 720p, de la marca *Hikvision*, por medio del protocolo RTSP. La reproducción de la imagen se logra a través del visualizador de la librería *OpenCV*.

Figura 17. Video de una cámara IP.

Captura 2: en la figura 18, se puede apreciar la captura de video en tiempo real de la cámara integrada en una *laptop*.

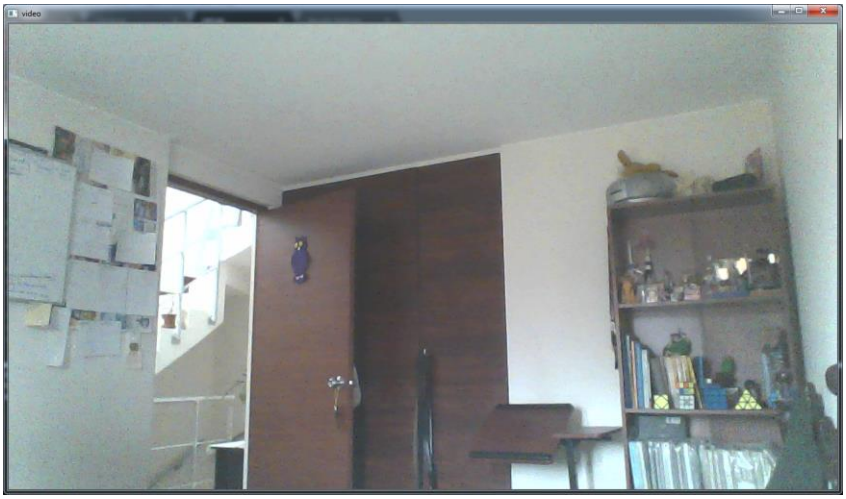

Figura 18. Video de una cámara de laptop.

Captura 3: por medio de una aplicación instalada en un smartphone de sistema operativo Android, conocida como *DroidCam*, se puede captar el video en directo. En la figura 19 se puede apreciar la captura del video en tiempo real por medio de HTTP streaming.

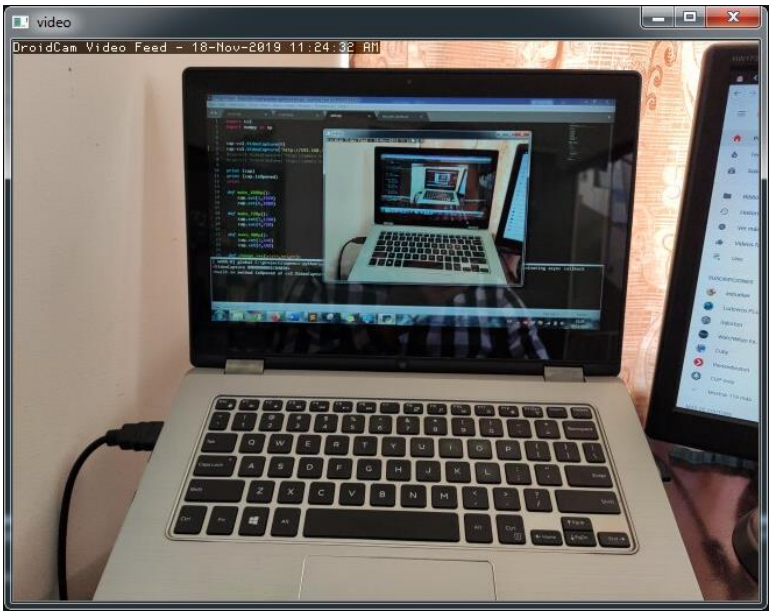

Figura 19. Video de la cámara de un smartphone.

Tal y como se puede apreciar el programa realizado para la captura de *streaming* está funcionando acorde al diseño preliminar, con buenos resultados.

3.5 PROTOCOLOS DE COMUNICACIÓN

En el lenguaje de programación de *Python* se dispone de una librería para ejecutar protocolos de comunicación entre un cliente y un servidor. Entre ellos están TCP y UDP, a través de la librería *socket.* Las pruebas realizadas se ejecutaron dentro de una misma red, empleando *Wi-Fi.*

3.5.1 PRUEBAS CON UDP

Prueba 1: para esta prueba el escenario se planteó de la siguiente forma:

Hardware del servidor:
Procesador: Intel Core i7-5500 2.4GHz,
RAM: 8 GB de RAM
SO: Windows 7 64 bits

Hardware del cliente
Procesador: AMD Turion 64 x2 1.9GHz,
RAM: 2 GB de RAM
SO: Windows 7 32 bits

Figura 20. Prueba 1 comunicación UDP.

Como se aprecia en la figura 20 la ejecución de los programas con este protocolo se inicializa, pero surgen inconvenientes. El equipo de la derecha que constituye el cliente, si bien la cámara *web* se enciende, desde el lado del servidor no se visualiza el video. Los programas no revelan ningún error ya sea en la captura el video o en la transmisión. Se presume que un cambio de *hardware* podría resolver estas fallas.

Prueba 2: para esta prueba el escenario se planteó de la siguiente forma:

Hardware del servidor:
Procesador: Intel Core i7-5500 2.4GHz,
RAM: 8 GB de RAM
SO: Windows 7 64 bits

Hardware del cliente
Procesador: Intel Core i5-7200 2.7GHz,
RAM: 8 GB de RAM
SO: Windows 10 Home 64 bits

Figura 21. Prueba 2 comunicación UDP.

En la figura 21 se pude apreciar el cambio de *hardware*, se emplea una *laptop* con mejores prestaciones. Y nuevamente se ejecutan los programas, donde lamentablemente se puede observar que la cámara *web* se enciende, pero aún no se recibe el video desde el lado del servidor y tampoco se visualizan errores o advertencias de ninguna índole. Considerando que UDP no regula los paquetes y no hay confirmación de entrega, la tendencia orienta a que posiblemente este protocolo no se acople a las necesidades del proyecto. Por lo que se efectúa una prueba más para analizar este comportamiento.

Prueba 3: para esta prueba el escenario se planteó de la siguiente forma:

Hardware del servidor y el cliente:
Procesador: Intel Core i7-5500 2.4GHz,
RAM: 8 GB de RAM
SO: Windows 7 64 bits

Como se aprecia en la figura 22, al emplear el mismo equipo como servidor y cliente, se demostró que la programación realizada cumple las funciones de captura de video, envió y recepción de paquetes de datos y la visualización del video, considerando los efectos descritos en la prueba 2.

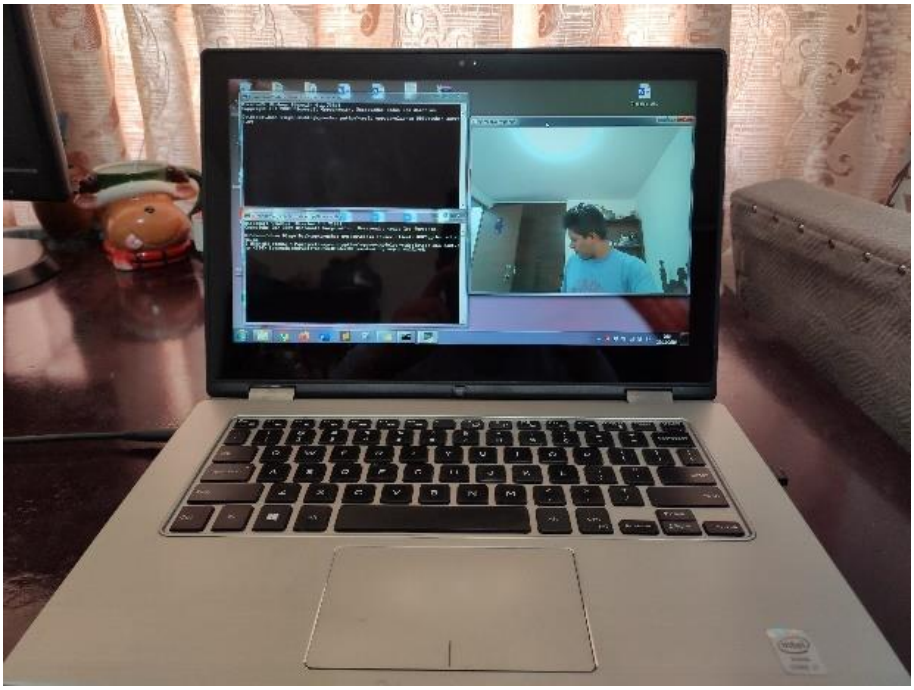

Figura 22. Prueba 3 comunicación UDP.

Adicionalmente se puede apreciar que, al momento de tomar la fotografía, el video no se transmite en vivo, y es debido a que el mismo se actualizaba en cualquier momento. Tema que no es recomendable para este diseño.

Con estas pruebas se concluye en, cambiar a otro tipo de protocolo para efectuar la comunicación entre servidor y cliente. Como nota puntual es necesario destacar que, hasta el momento, en ninguna prueba se ha introducido en la programación los algoritmos de encriptación y desencriptación.

3.5.2 PRUEBAS CON TCP

Una vez establecidas las configuraciones tanto en el script del servidor y del cliente, acorde al protocolo en mención, se realizaron las siguientes pruebas.

Prueba 1: para esta prueba el escenario se planteó de la siguiente forma:

Hardware del servidor:
Procesador: Intel Core i7-5500 2.4GHz,
RAM: 8 GB de RAM
SO: Windows 7 64 bits

Hardware del cliente
Procesador: AMD Turion 64 x2 1.9GHz,
RAM: 2 GB de RAM
SO: Windows 7 32 bits

Figura 23. Prueba 1 comunicación TCP.

Como se aprecia en la figura 23 la programación realizada fue exitosa. El equipo cliente(derecho) toma la imagen en este caso de la cámara *web* por USB, envía la información hacia el servidor y este último la captura y la muestra. La cámara *web* empleada es de la marca *Logitech*. Perdida de video no existe, sin embargo, se presenta lentitud, posiblemente al tipo de protocolo empleado, el cual usa comprobación de datos al momento de enviar y recibir los datos, o al *hardware* del cliente.

Prueba 2: para esta prueba el escenario se planteó de la siguiente forma:

Hardware del servidor:
Procesador: Intel Core i7-5500 2.4GHz,
RAM: 8 GB de RAM
SO: Windows 7 64 bits

Hardware del cliente
Procesador: Intel Core i5-7200 2.7GHz,
RAM: 8 GB de RAM
SO: Windows 10 Home 64 bits

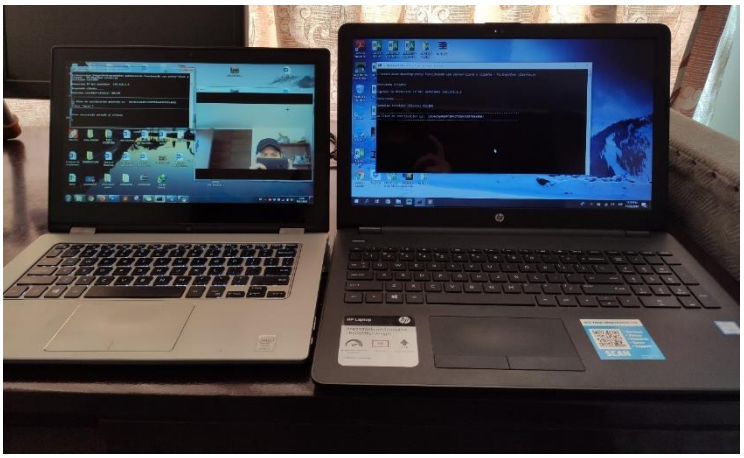

Figura 24 Prueba 2 comunicación TCP.

Para esta prueba se cambia el *hardware* del cliente, el cual toma la imagen en este caso de la cámara *web* integrada en la *laptop,* y la envía hacia el servidor. No existe perdida de video o algún mensaje de error, sin embargo, se presenta lentitud, se asume el mismo factor que en la prueba 1.

Prueba 3: para esta prueba el escenario se planteó de la siguiente forma:

Hardware del servidor:
Procesador: Intel Core i7-5500 2.4GHz,
RAM: 8 GB de RAM
SO: Windows 7 64 bits

Hardware del cliente
Procesador: Intel Core i5-7200 2.7GHz,
RAM: 8 GB de RAM
SO: Windows 10 Home 64 bits

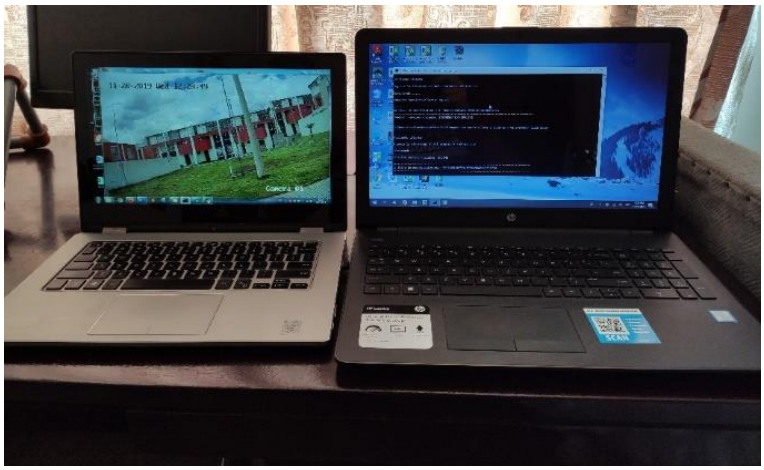

Figura 25. Prueba 3 comunicación TCP.

Con las pruebas anteriores se constata que la programación realizada cumple con la parte del diseño hasta esta sección, obviamente con ciertas limitaciones las cuales atienden mayormente al protocolo de comunicación. Y que en *Python* resulta ser el método más eficiente hasta el momento para este diseño. Tal es así, que se procede a realizar una prueba con una cámara IP de la marca *Hikvision*. El resultado se muestra en la figura 25. Entre los resultados obtenidos se destacan los siguientes:

1. Momentáneamente se visualizó el video en directo y seguidamente la programación mostro un error el cual se aprecia en la figura 26. El inconveniente muestra que la librería *OpenCV* mostro su primera limitante en este diseño. La condición en este caso apunta a que la cámara IP seleccionada requiere desarrollar un driver para poder acoplarse a *OpenCV,* y es por aquello que se muestra el error en reiteradas ocasiones. La alternativa en este caso sería esperar a la siguiente actualización de la librería o buscar otra librería que permita manejar el video.

```
cv2.error: OpenCV(4.1.1) C:\projects\opencv-python\opencv\modules\highgui\src\wi
ndow.cpp:352: error: (-215:Assertion failed) size.width>0 && size.height>0 in fu
nction 'cv::imshow'
```

Figura 26. Error de *OpenCV*.

2. Otra característica que se pudo apreciar es que *OpenCV* permite modificar la resolución del video para visualizarlo, sin embargo, con la cámara IP no fue posible realizar esta acción. Aun cuando se realizaron las modificaciones en la interfaz *web* de la cámara, la resolución siempre se mantuvo en HD. Se podría asumir que se trata de una característica propia de la tecnología del fabricante.

Prueba 4: para esta prueba el escenario se planteó de la siguiente forma:

Hardware del servidor:
Procesador: Intel Core i7-5500 2.4GHz,
RAM: 8 GB de RAM
SO: Windows 7 64 bits

Hardware del cliente
Procesador: Intel Core i5-7200 2.7GHz,
RAM: 8 GB de RAM
SO: Windows 10 Home 64 bits

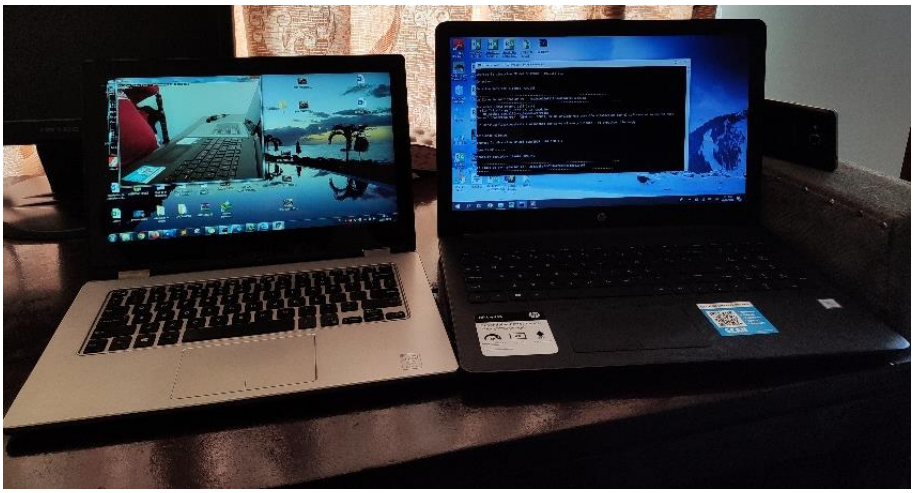

Figura 27. Prueba 4 comunicación TCP con smartphone.

Esta prueba se la realiza con la cámara de un smartphone, tal y como se puede apreciar en la figura 27. El establecimiento de la conexión se efectuó sin ningún problema, la latencia sigue siendo un factor a pesar del cambio de equipos. De esta forma se mantiene la programación realizada, considerando que se tiene mejores resultados que con el protocolo UDP.

Con estas pruebas de igual manera se concluye en: mantener la comunicación TCP para la transmisión del streaming. Es necesario destacar que, hasta el momento, en ninguna

prueba se ha introducido en la programación los algoritmos de encriptación y desencriptación, tema que se desarrolla a continuación.

3.6 PRUEBAS PROTOCOLOS DE ENCRIPTACIÓN

En *Python* mediante su gran cantidad de librerías es posible desarrollar cualquier tipo de programación orientada a objetos. En esta ocasión se empleará la librería 'Crypto' para el cifrado de la información.

3.6.1 CRIPTOGRAFÍA ASIMÉTRICA

El algoritmo seleccionado para cifrar dentro de este grupo se trata de RSA (*Rivest, Shamir y Adleman*). El cual como es de conocimiento emplea 2 llaves, una publica, que se usa para la encriptación propiamente y la llave privada para la desencriptación.

Mediante *Python* se crean las llaves a través de un generador *random*, donde es necesario especificar la longitud de la llave, y entre los ejemplos típicos estan 1024, 2048, 4096 bits. En la figura 28 se puede apreciar un ejemplo de las llaves generadas para este algoritmo.

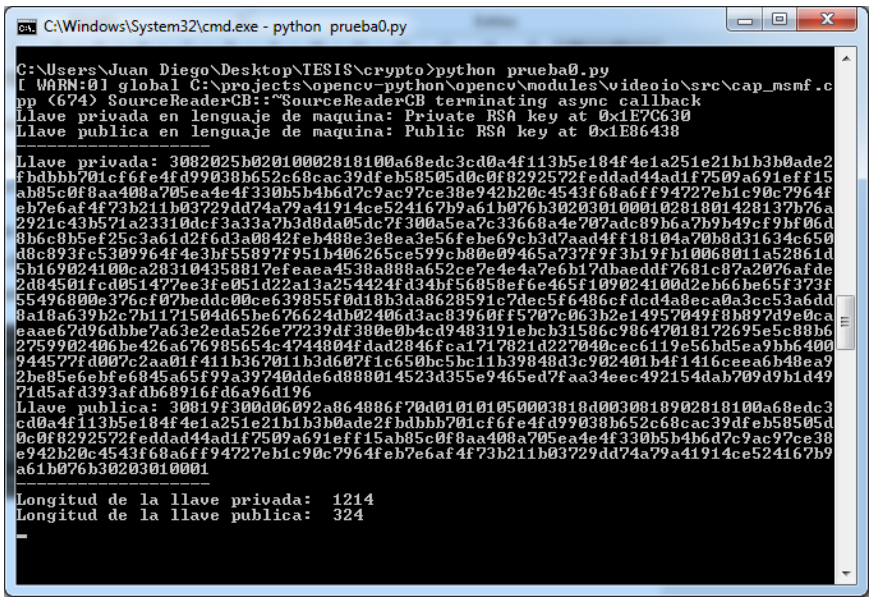

Figura 28. Llaves criptográficas asimétricas con RSA.

Al combinar el algoritmo de encriptación al programa ya probado hasta el momento, se presentaron las observaciones que se detallan a continuación. Cabe mencionar que para esta prueba se está empleando la cámara *web* integrada en la *laptop* como fuente de video directo.

- Las llaves de encriptación se generaron normalmente. En este algoritmo siempre la llave privada tendrá una longitud mayor a la llave pública.
- La captura del video también se efectuó con tranquilidad.
- El programa determino un error proveniente del algoritmo de encriptación, que se muestra en la figura 29. Y tal y como lo indica la descripción, la captura del video, los *frames* por segundos que se realiza en tiempo real y que se encuentran en texto plano son demasiado largos para este algoritmo. Todo indica que el algoritmo RSA si bien maneja archivos de imagen y video, son con un determinado límite de bits.

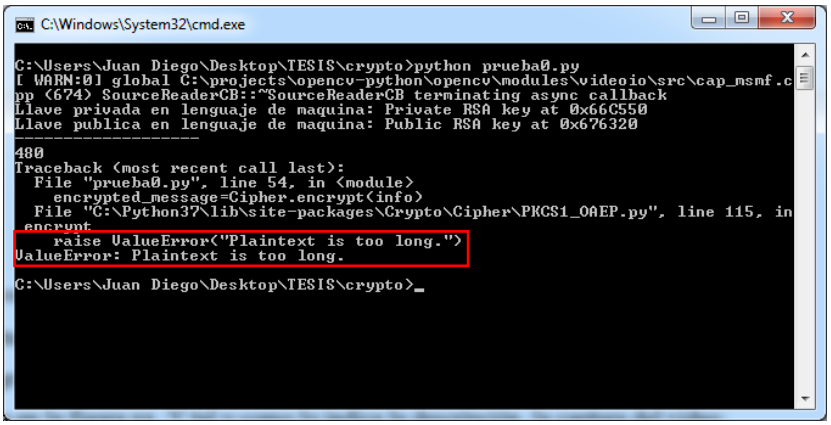

Figura 29. Inconveniente con el algoritmo RSA *Python.*

Con esta prueba es indispensable buscar otro algoritmo alternativo que permita manejar un mayor número de bits en texto plano y que se acople de igual manera a este diseño.

3.6.2 CRIPTOGRAFÍA SIMÉTRICA

El algoritmo seleccionado para cifrar dentro de este grupo se trata de AES (*Advanced Encryption Standard*). El cual emplea una llave única para la encriptación y desencriptación de información. De igual manera en el algoritmo previo la llave se crea a partir de un generador *random*, donde solo es posible trabajar con llaves de 128, 192 y 256 bits, que son correspondientes a los algoritmos AES-128, AES-192, AES-256 de criptografía. En la figura 30 se aprecia un ejemplo de las llaves generadas para este algoritmo.

Al combinar el algoritmo de encriptación con la rutina de adquisición del video se presentaron las siguientes observaciones, cabe mencionar que para esta prueba se emplea la cámara *web* integrada en la *laptop*.

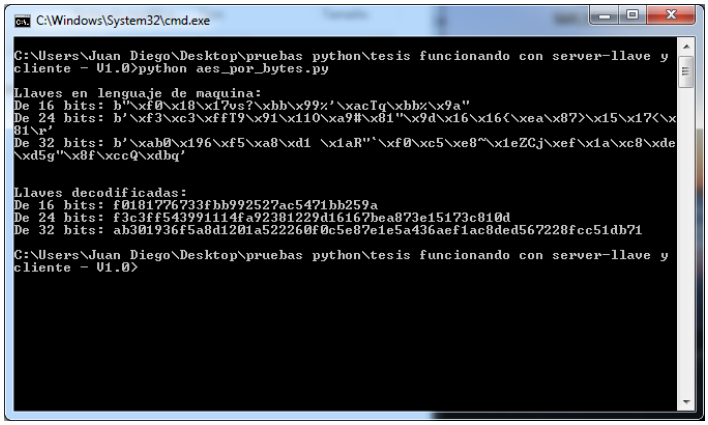

Figura 30. Llaves criptográficas simétricas con AES.

- La ejecución del programa fue exitosa, tal y como se puede apreciar en la figura 3, donde no se produjo errores por la cantidad de datos en texto plano.
- En esta oportunidad se programó para trabajar con una llave de 16 bits, lo que introduce al algoritmo de 128 bytes respectivamente.
- Con esta prueba se concluyó que el algoritmo AES es el más apropiado para el diseño del prototipo.

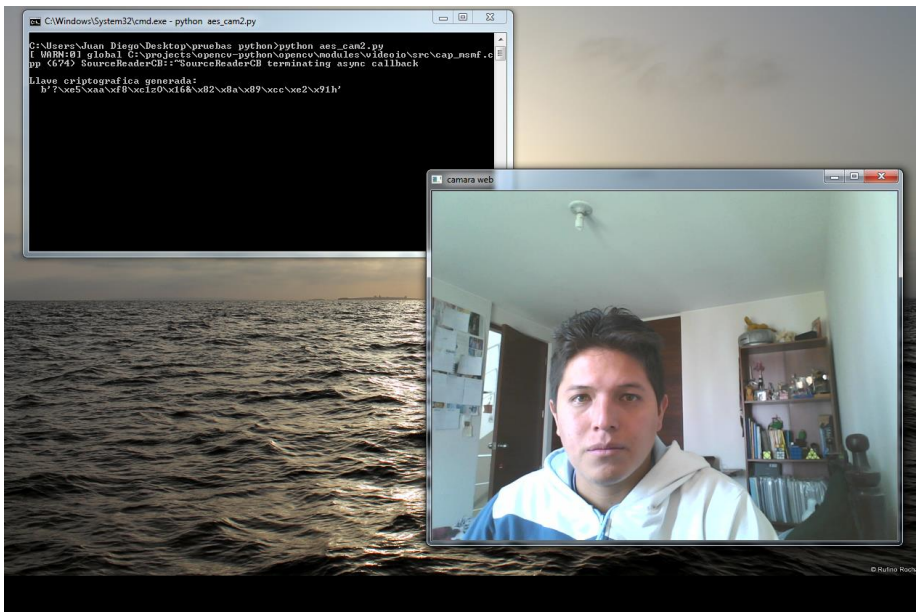

Figura 31. Ejecución del algoritmo AES y captura de video.

3.7 PRUEBAS FINALES

Una vez definido todas las características para este diseño se procede a estructurar el programa final y para ello se ejecutaron las siguientes pruebas:

Prueba 1: datos en texto plano y cifrado

El objetivo en esta prueba es mostrar los datos en texto plano y los cifrados a través del programa realizado. En la figura 32 se puede observar el programa ejecutándose, donde se muestra el video en vivo y los datos en texto sin cifrado.

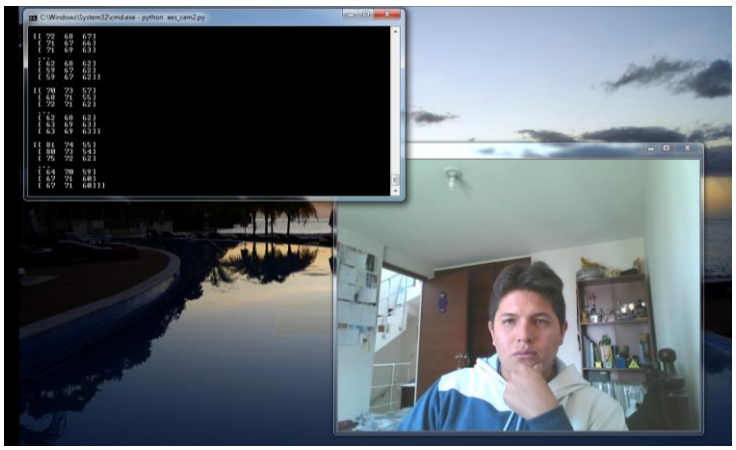

Figura 32. Datos sin cifrar.

Como se puede apreciar en la figura 32, los datos son un arreglo de matrices que se obtienen directamente de la fuente de video. Considerando que se trata de una captura en tiempo real, en las figuras 33 y 34, se muestran los datos ante diferentes exposiciones del lente de la cámara.

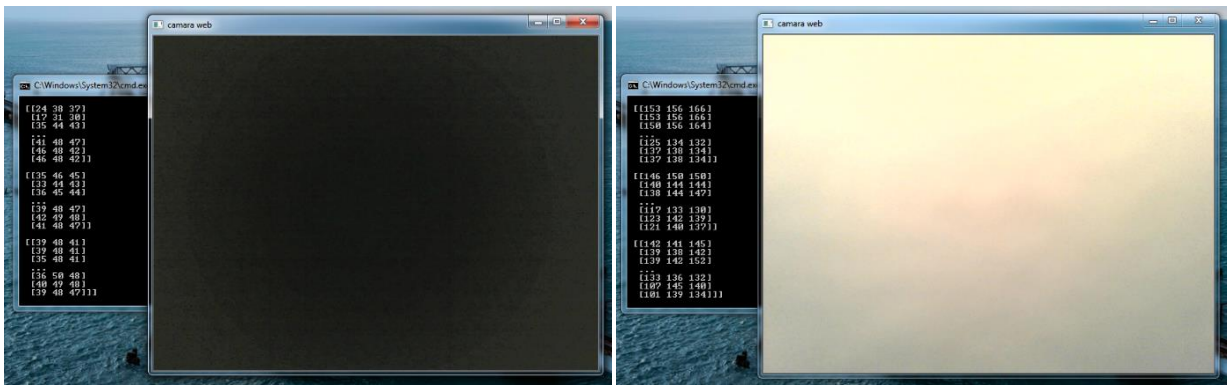

Figura 33. Datos sin cifrar ante diferentes exposiciones del lente 1.

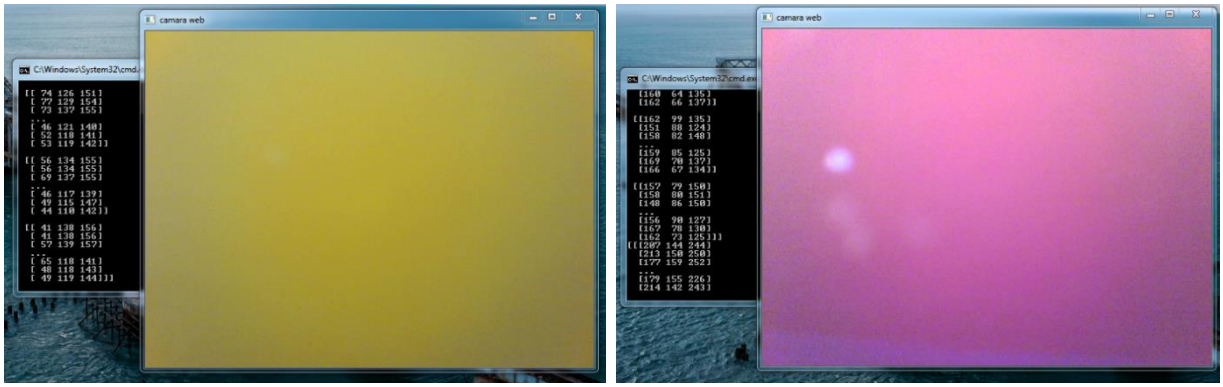

Figura 34. Datos sin cifrar ante diferentes exposiciones del lente 2.

A continuación, mediante programación se efectúa la visualización de los datos cifrados, mientras la visualización permanece en línea, tal y como se aprecia en la figura 35.

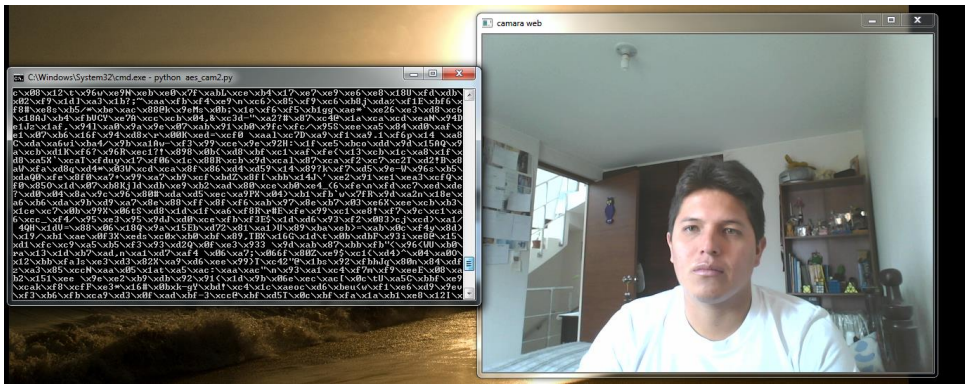

Figura 35. Datos cifrados

A continuación, en las figuras 36, 37, 38 se presenta algunas capturas donde se cifran los datos y se visualiza el video al mismo tiempo, con variaciones de exposición en el lente.

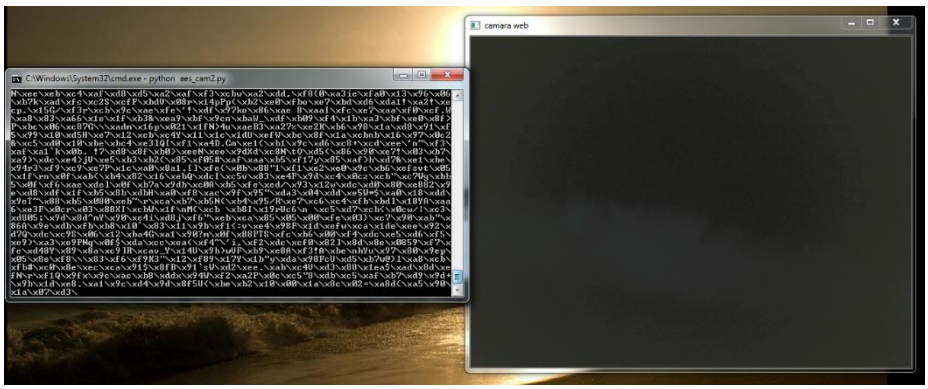

Figura 36. Datos cifrados ante diferentes exposiciones del lente 1.

Figura 37. Datos cifrados ante diferentes exposiciones del lente 2.

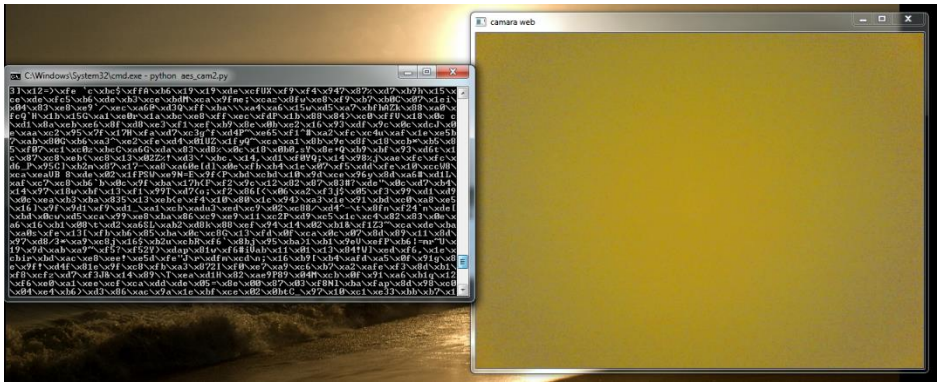

Figura 38. Datos cifrados ante diferentes exposiciones del lente 3.

Con las pruebas realizadas, se han obtenido los resultados esperados en cuanto a lo relacionado al cifrado de los datos, por lo que únicamente falta incluir la subrutina de comunicación entre el cliente y el servidor.

Prueba 2: Comunicación y cifrado de video

En estas pruebas se procede a integrar ya todas las subrutinas desarrolladas como parte del prototipo final, conociendo las conclusiones de las pruebas anteriores.

Prueba 2.1: Para esta prueba el escenario se planteó de la siguiente forma:

Hardware del servidor:
Procesador: Intel Core i7-5500 2.4GHz,
RAM: 8 GB de RAM
SO: Windows 7 64 bits

Hardware del cliente
Procesador: AMD Turion 64 x2 1.9GHz,
RAM: 2 GB de RAM
SO: Windows 7 32 bits

Figura 39. Prueba 2.1 Encriptación de un streaming de video en directo

En la figura 39 se puede apreciar la funcionalidad del programa final, correspondiente a la versión 1.0. A través de la cual se puede visualizar que el equipo cliente(derecha) captura el streaming de video en directo de la cámara *web*, lo encripta y lo envía hacia el servidor, el cual recibe los datos, desencripta y los visualiza. Sin embargo, existe la presencia de latencia en la transmisión por el tipo de protocolo de comunicación empleado. En la figura 40, se aprecia la captura de video, encriptación, transmisión y visualización al mismo tiempo, condición que no se recomienda debido al incremento de los recursos en los equipos.

Figura 40. Encriptación con video en directo.

Prueba 2.2: Para esta prueba el escenario se planteó de la siguiente forma:

Hardware del servidor:
Procesador: Intel Core i7-5500 2.4GHz,
RAM: 8 GB de RAM
SO: Windows 7 64 bits

Hardware del cliente
Procesador: Intel Core i5-7200 2.7GHz,
RAM: 8 GB de RAM
SO: Windows 10 Home 64 bits

En la figura 41, se aprecia una segunda prueba, en esta ocasión se emplea la cámara de un smartphone. Como se indicó en capítulos anteriores es necesario emplear un aplicativo adicional para los sistemas operativos *Android*.

Figura 41. Prueba 2.2 de encriptación con video en directo.

La laptop de la derecha captura el video del smartphone colocado horizontalmente junto a su pantalla, lo encripta y lo envía hacia el servidor

En la figura 42, nuevamente se presenta la funcionalidad de los programas, mostrando el cifrado y la transmisión. Como se indicó anteriormente la combinación de estas operaciones ralentiza el funcionamiento.

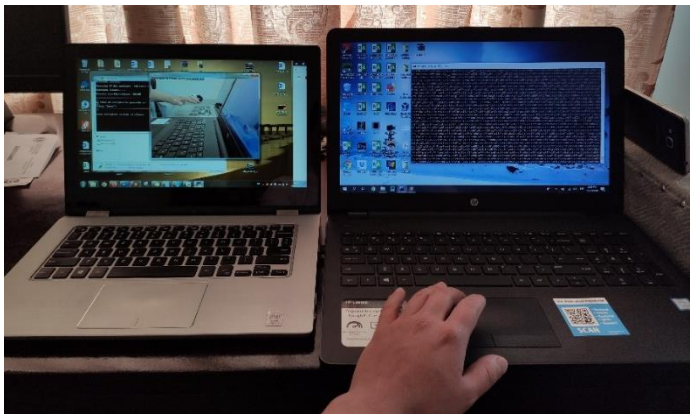

Figura 42. Encriptación con video en directo a través de un smartphone.

3.8 PRUEBA ENTRE DIFERENTES SISTEMAS OPERATIVOS

Entre las ventajas del prototipo, está la facilidad de ejecutar los scripts en diferentes sistemas operativos. Para esta prueba el escenario se planteó de la siguiente forma:

Hardware del cliente:
Procesador: Intel Core i7-5500 2.4GHz,
RAM: 8 GB de RAM
SO: Ubuntu 16.04 LTS 64 bits

Hardware del servidor
Procesador: Intel Core i5-7200 2.7GHz,
RAM: 8 GB e RAM
SO: Windows 10 Home 64 bits

En la figura 43 se puede apreciar la operatividad del prototipo, sobre los sistemas operativos mencionados. La limitante de acuerdo a esta prueba es el hardware de cada PC, bata con disponer de una infraestructura moderna para evitar inconvenientes en la ejecución del programa.

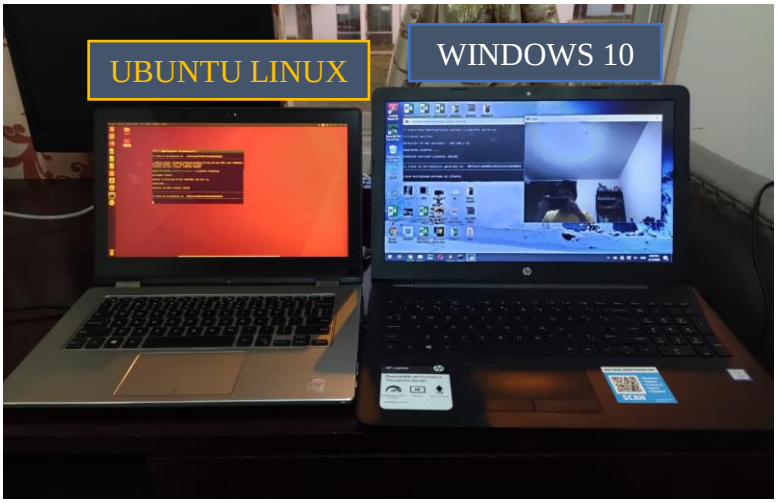

Figura 43. Ejecución del prototipo entre un S.O. Linux y Windows

3.9 TIEMPOS DE TRANSMISIÓN

Se realiza este análisis debido a que la parte fundamental de este diseño es la transmisión del streaming de video, el cual como es de conocimiento conlleva grandes paquetes de datos que, al estar inmersos en una red indudablemente introducirá tráfico, y es importante conocer cómo se ve afectado este prototipo.

La manera de cuantificar los tiempos de transmisión, es verificando el intervalo de tiempo que existe desde que se captura y se transmite la señal de video hasta cuando se visualiza en el receptor.

En las figuras 44, 45, 46 a continuación se podrá apreciar el intervalo de tiempo que existe entre la figura de la izquierda que constituye el video receptado y la figura de la derecha que corresponde al video emitido. Para esta oportunidad se emplea como fuente de video, el generado a través de un lente en un smartphone.

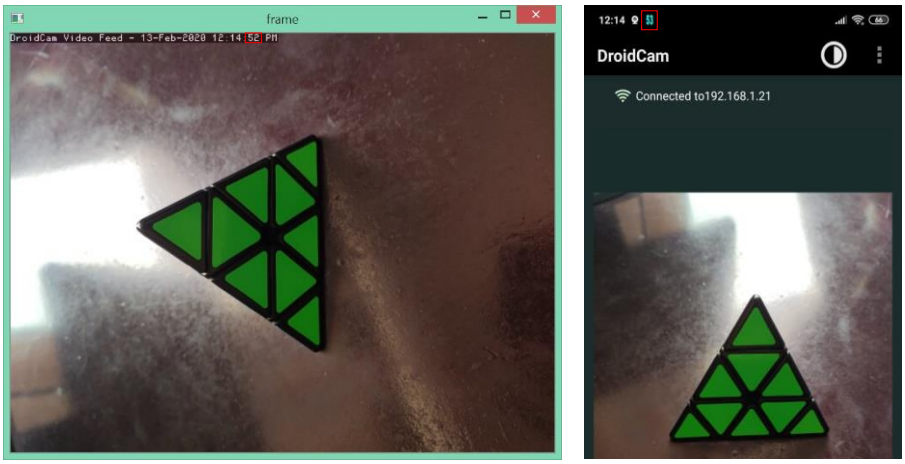

Figura 44. Prueba #1 de transmisión en tiempo entre el equipo transmisor y el receptor

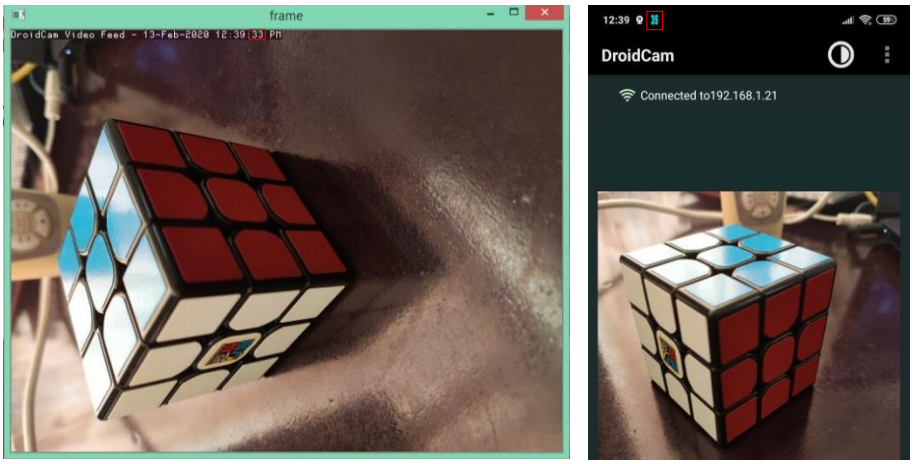

Figura 45. Prueba #2 de transmisión en tiempo entre el equipo transmisor y el receptor

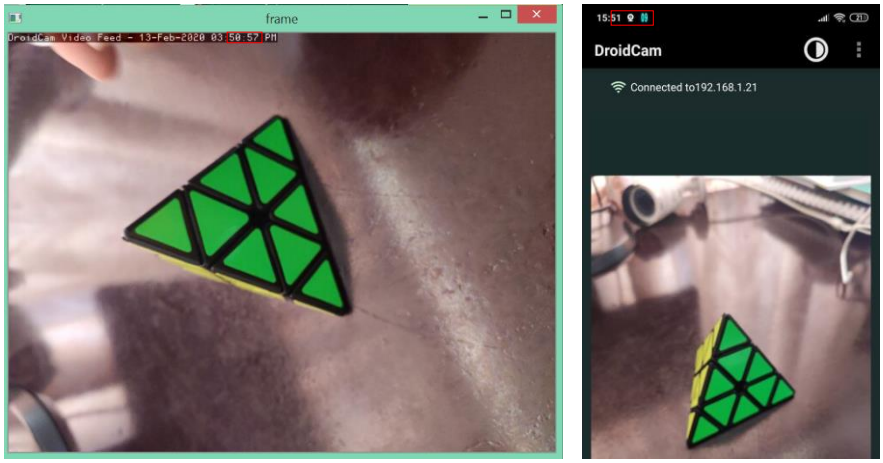

Figura 46. Prueba #3 de transmisión en tiempo entre el equipo transmisor y el receptor

A continuación, se muestra la tabla 3, con un resumen de las mediciones realizadas:

Tabla 3. Pruebas de tiempo del prototipo

Item	Tiempo en el receptor (h:m:s)	Tiempo en el emisor (h:m:s)	Intervalo de tiempo (seg)
1	12:14:52	12:14:53	1
2	12:17:09	12:17:09	0
3	12:18:05	12:18:06	1
4	12:20:09	12:20:09	0
5	12:22:11	12:22:11	0
6	12:39:33	12:39:35	2
7	14:55:03	14:55:05	2
8	15:37:02	15:37:12	10
9	15:39:55	15:40:59	4
10	15:50:57	15:51:09	12
11	15:58:30	15:58:35	5
12	11:58:29	11:58:33	4
13	11:59:02	11:59:06	4
14	12:00:02	12:00:07	5
15	12:00:22	12:00:26	4
16	14:22:39	14:22:39	0
17	14:23:08	14:23:09	1
18	14:23:35	14:25:35	0
19	14:24:08	14:24:08	0
20	14:24:30	14:24:30	0

En la figura 47 se podrá apreciar una interpretación grafica de la tabla 3, en la cual se puede visualizar la variación de los tiempos, y el tiempo promedio que quedaría establecido de esta muestra de datos.

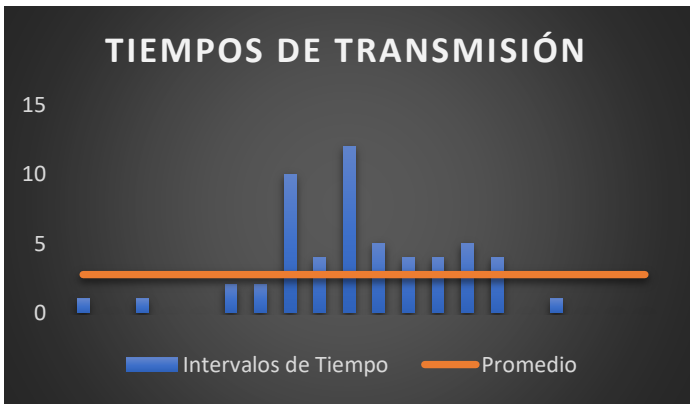

Figura 47. Interpretación grafica de los tiempos de transmisión

El tiempo promedio que se obtuvo para este prototipo acorde a las pruebas es de 2,75 seg. Es decir que, desde la captura, encriptación y envió del streaming de video hasta la recepción, desencriptación y visualización del video va desde 0 a 2,75 seg.

3.10 EVALUACIÓN DE CÓDIGO

Como parte del presente proyecto es indispensable cumplir con lo relacionado a la calidad de servicio que se va a proveer a través del desarrollo de este prototipo. La opción más adecuada para este diseño constituye la evaluación del código fuente, considerando la programación ejecutada. El método por el cual se realizará esta evaluación es empleando el software de *SonarQube*. El cual constituye un software libre y dispone de múltiples herramientas para el análisis del código fuente.

La evaluación se ejecutó a los scripts, cliente y servidor, y a continuación se presentan los resultados. En la figura 48 se puede apreciar la ejecución de *SonarQube* sobre el *script* servidor, el cual como lo indica ahí, se ejecutó correctamente. De la misma forma en la figura 49 se aprecia la ejecución de *SonarQube* sobre el script cliente, tal cual lo indica ahí se ejecutó correctamente. Finalmente, la evaluación propiamente dicha se visualiza el *dashboard* de *SonarQube*, el cual se muestra en la figura 50.

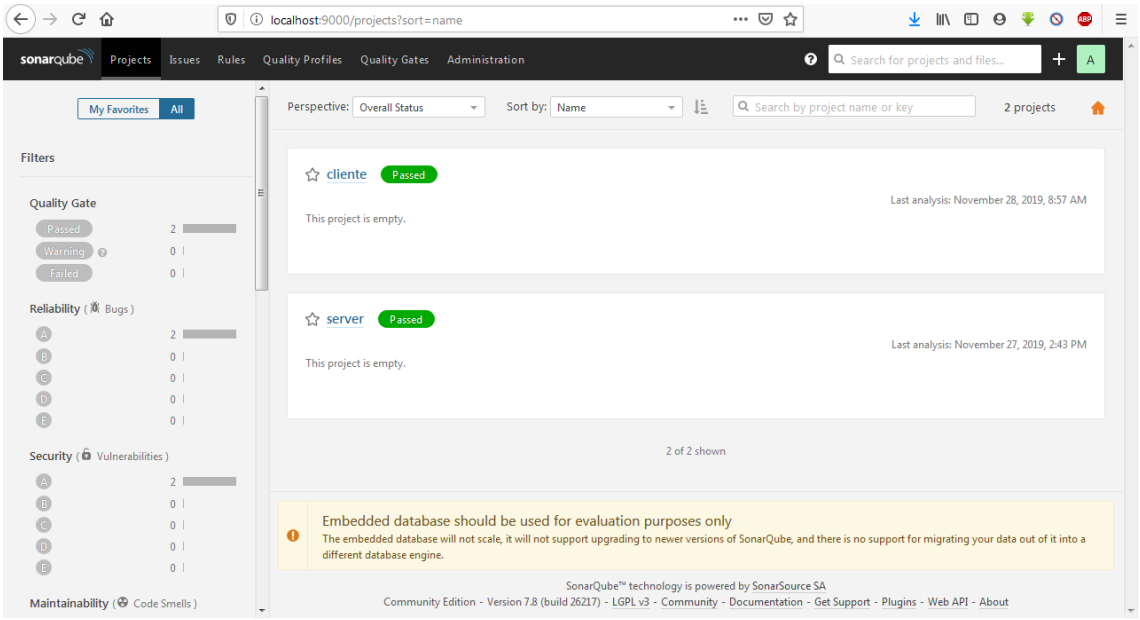

Figura 48. Dashboard de *SonarQube*-resultados.

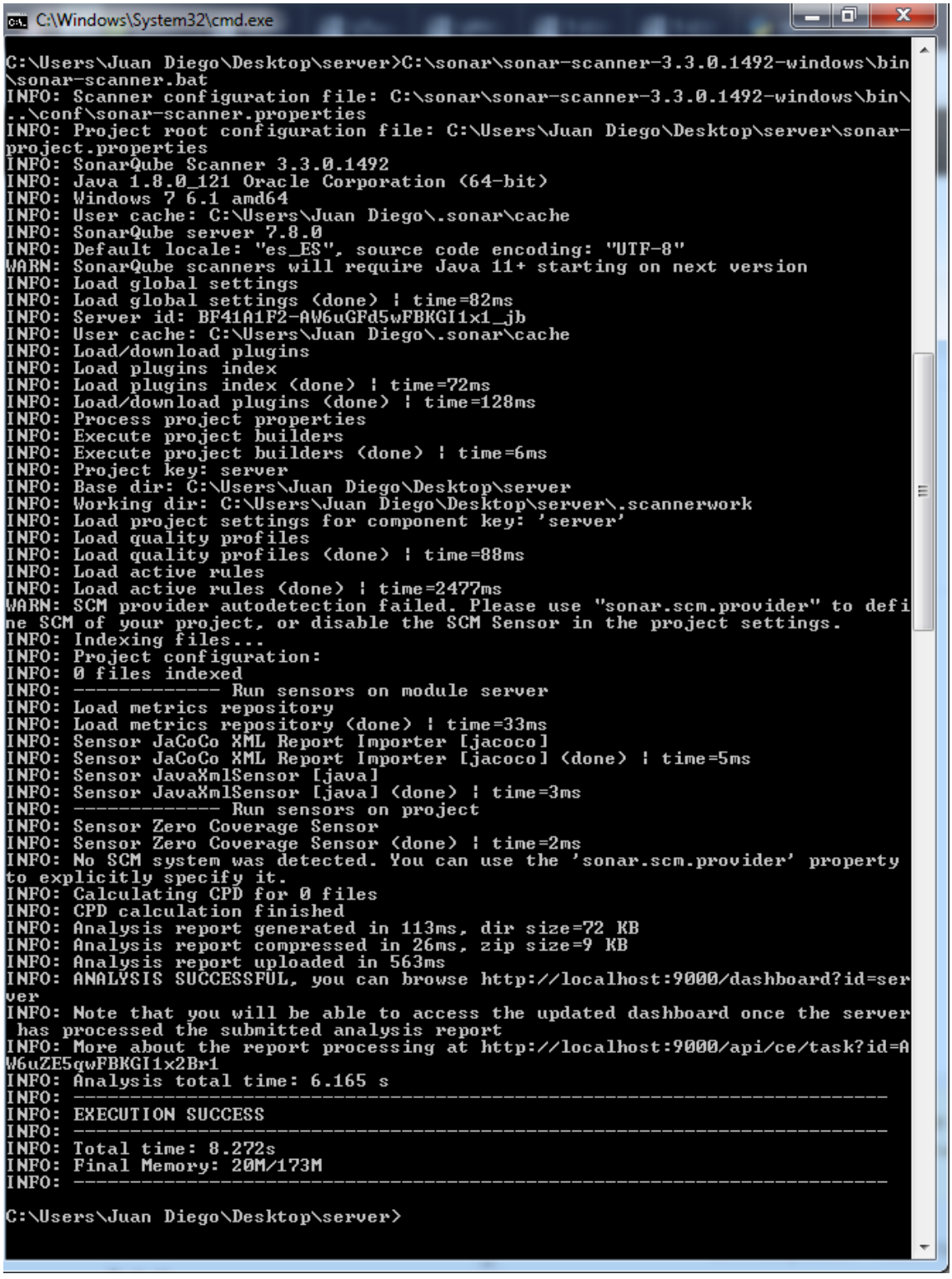

Figura 49. Evaluación del script servidor.

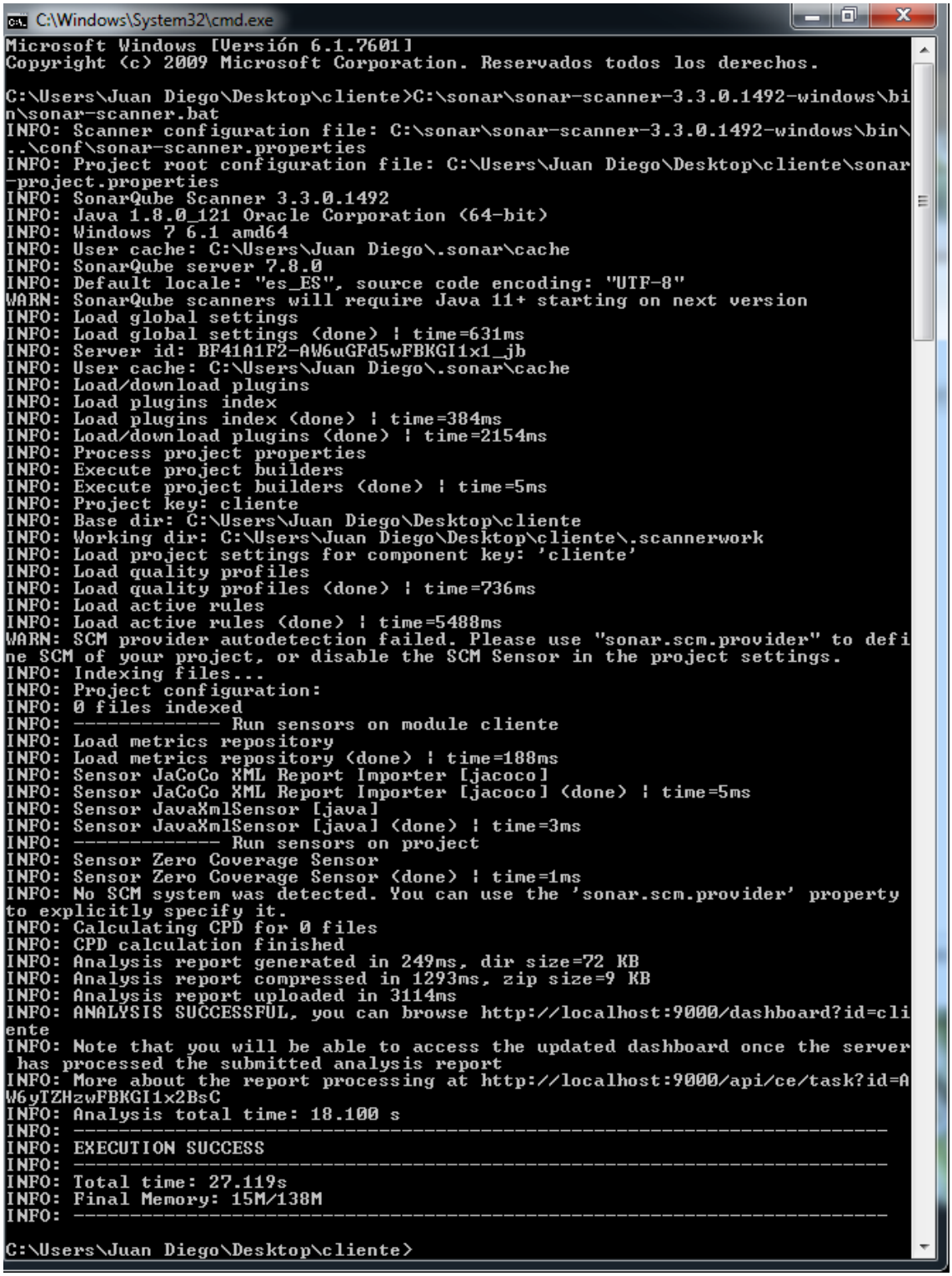

Figura 50. Evaluación del *script* cliente.

De la figura 50 se obtienen los siguientes resultados:

Previo a la ejecución de la evaluación, se eliminó todo código incensario como es comentarios, ejecuciones omitidas, entre otros.

- Los 2 scripts son aprobados por *SonarQube*, es decir que no presentan vulnerabilidades y son fiables.
- Como se aprecia también de forma general no se tiene la presencia de los llamados "*bugs*" o "*code smells*", esto debido a que en si la programación de *Python* ha sido estructurada omitiendo los factores antes mencionados.
- Otro factor a considerar es que el número de líneas de programación se reduce considerablemente debido a *Python*, razón por la cual tampoco existe la presencia de *warning*.
- De esta forma se puede asegurar que los *scripts*, conllevan un gran factor de seguridad y calidad para su ejecución.

CONCLUSIONES

Analizando los diferentes algoritmos de encriptación, se determinó que la criptografía simétrica es la opción más idónea para este proyecto, considerando la cantidad de paquetes que se emplean al momento de capturar el video. En comparación con las criptográfica asimétrica y sin desmerecer sus capacidades, para el tema de streaming mostro una limitante en cuanto a los paquetes en texto plano, sin embargo, es ideal para otros proyectos.

La optimización del prototipo se alcanzó empleando el protocolo de comunicación TCP, mediante la figura cliente-servidor. Si bien el protocolo TCP emplea mayores recursos para su funcionamiento es la opción más adecuada para el streaming de video, en comparación a UDP que al ser un protocolo no orientado a comunicación es propenso a la perdida de información.

Python a través de sus librerías, las cuales fueron seleccionadas en base a una exhaustiva investigación permitieron desarrollar y poner en funcionamiento este prototipo. Indudablemente todo desarrollo de software conlleva mejoras, por lo que siempre será importante buscar alternativas para cumplir con el objetivo del proyecto, y considerando la orientación del proyecto, no cabe duda que existirán nuevas propuestas tecnológicas.

El funcionamiento del prototipo cumple con su objetivo, sin embargo, existe la presencia de latencia (tiempo que tarde en trasmitirse un paquete dentro de una red) al momento de la ejecución. Se puede concluir que los factores que generan la latencia en este proyecto son los recursos tecnológicos que dispongan los computadores, el sistema operativo empleado y el router a través del cual se estableció la red. La validación se estructuro en base a una serie de pruebas donde se determinó que el tiempo promedio de latencia es de 2,75 seg.

Inicialmente el proyecto no conto con un gran soporte respecto a estudios, documentación, aportes, referencias tecnológicas, etc como base para el diseño de este prototipo. Posiblemente si existan aplicaciones similares orientadas al objetivo del proyecto, pero que por cuestiones ajenas no sea de libre distribución. Sin embargo, existe la posibilidad que este proyecto sirva de fuente de estudio para proyectos futuros relacionados al tema.

RECOMENDACIONES

- Compartir la llave de encriptación o desencriptación independientemente de cuál sea el método de criptografía que se emplea, siempre se deberá asegurar que la misma sea compartida por medios seguros y confiables. Caso contrario no tendría ningún sentido elaborar algoritmos criptográficos si la llave no está segura.
- El prototipo se orienta al streaming de video en directo, sin embargo, al no contar con un servidor de streaming propio, se puede realizar la ejecución del mismo sobre los streaming bajo demanda y analizar su comportamiento.
- Una alternativa de mejora puede ser la implementación de este diseño sobre lenguajes de programación diferentes de Python, como puede ser C,C++, Java, etc. Y poder validar la capacidad de este prototipo en diferentes plataformas y que de igual forma operen sobre distintos sistemas operativos.

REFERENCIAS

Rodríguez, H. (2001). *Implementación de un sistema simulador de encriptación de audio utilizando entorno de programación Matlab*. (Tesis de pregrado). Universidad Politécnica Nacional, Quito, Ecuador. Recuperado de https://bibdigital.epn.edu.ec/bitstream/15000/5440/1/T1708.pdf

Real, F. (2019). *Desarrollo de una herramienta de uso didáctico que permita esconder información encriptada dentro de una imagen con formato jpeg empleando esteganografía*. (Tesis de pregrado). Universidad Politécnica Nacional, Quito, Ecuador. Recuperado de https://bibdigital.epn.edu.ec/handle/15000/20405.

Rogel, L. (2016). *Implementación de video streaming para la visualización en tiempo real de la funcionalidad de una aplicación interactiva transmitida en bts (broadcast transport stream)*. (Tesis de pregrado). Universidad de las Fuerzas Armadas, Sangolqui Ecuador Recuperado de https://repositorio.espe.edu.ec/bitstream/21000/11708/1/T-ESPE-053206.pdf

González A. & Gallardo T. & Del Pozo F. (2018). *Metodología de la investigación*. Quito, Ecuador, Editorial Jurídica del Ecuador.

PandaAncha. (2017). ¿Qué es encriptar? Tipos de cifrado y sus características. Recuperado de https://www.pandaancha.mx/noticias/que-es-encriptar-tipos-de-cifrado-y-sus-caracteristicas.html

Viewnext. (2018). Tipos de Seguridad informática. Recuperado de https://www.viewnext.com/tipos-de-seguridad-informática/

Universidad Internacional de Valencia. (2018). Tres tipos de seguridad informática que debes conocer. Recuperado de https://www.universidadviu.com/tres-tipos-seguridad-informatica-debes-conocer/

OpenWebinars. (2019). Que es TCP/IP. Recuperado de https://openwebinars.net/blog/que-es-tcpip/

Suárez, F. (2010-2011). Tecnologías de streaming. Recuperado de http://www.atc.uniovi.es/teleco/5tm/archives/8streaming.pdf

American Dominios. (2020). Que son protocolos RTSPU, RTSPT, MMSU y MMST. Recuperado de https://americandominios.com/conta/knowledgebase/233/Que-son-los-protocolos-RTSPU-RTSPT-MMSU-y-MMST.html

Librería Pycrypto (2013), Python. Recuperado de https://pypi.org/project/pycrypto/

OpenCV-Python (2013), Introduction to OpenCV-Python Tutorials. Recuperado de https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_setup/py_intro/py_intro.html