IMPLEMENTACIÓN DEL SISTEMA HANUMAN EN MENSAJERÍA INSTANTÁNEA

HANUMAN SYSTEM IMPLEMENTATION ON INSTANT MESSENGER

Jaime David Ríos Arrañaga
Universidad de Guadalajara, México
jaime.rios@alumnos.udg.mx

José Luis Magaña Chávez
Universidad de Guadalajara, México
jluis.magana@alumnos.udg.mx

Sergio Ramos Jaramillo
Universidad de Guadalajara, México
sergio.rj@alumnos.udg.mx

Juan José Raygoza Panduro
Universidad de Guadalajara, México
juan.raygoza@cucei.udg.mx

Sandra Eloísa Balderas Mata
Universidad de Guadalajara, México
sandra.balderas@cucei.udg.mx

Edwin Christian Becerra Álvarez
Universidad de Guadalajara, México
edwin.becerra@cucei.udg.mx

Recepción: 28/octubre/2020 **Aceptación:** 2/diciembre/2020

Resumen

El uso de los servicios de mensajería instantánea en la comunicación personal, y de negocios entre miembros de distintas organizaciones y sus clientes, pone a los mensajeros instantáneos a la mira de agentes mal intencionados. En este trabajo se presenta una exploración al uso de la Encriptación Autenticada con Dato Asociado, como una alternativa que permita la identificación del usuario y la integridad del mensaje recibido. El sistema implementado se basa en el algoritmo HANUMAN de la familia de algoritmos de encriptación PRIMATE. Para probarlo, se creó un servicio de mensajería instantánea en Python v3.8 que realiza la

encriptación del mensaje previo a su envio. Las propiedades del tipo de encriptación permitieron verificar la integridad del mensaje.

Palabras Clave: Cifrado HANUMAN, Criptografía Autenticada, Mensajería Instantánea, Seguridad Informática.

Abstract

The use of instant messaging services in personal and business communication between staff of different organizations and their clients put instant messengers in the sights of malicious agents. This paper presents an exploration of Authenticated Encryption with Associated Data, as an alternative that allows user authentication and the integrity of the received message. The system used was the HANUMAN algorithm from the PRIMATE family of encryption algorithms. To test it, an instant messaging service was created in Python v3.8 that performs the encryption of the message before to its delivery. The encryption type properties allow us to identify the integrity of the message.

***Keywords*:** *Authenticated Cryptography, Computer Security, HANUMAN encryption, Instant Messenger.*

1. Introducción

La mensajería instantánea (IM, por sus siglas en inglés de *Instant Messenger*) es en la actualidad una importante y popular herramienta de comunicación digital, utilizada en la comunicación entre personas en distintas partes del mundo. Estadísticas propias de cada servicio de mensajería como *WhastApp, Facebook Instant Messenger y Skype Messenger*, por mencionar algunos, así como información de fuentes externas, cuentan con miles de millones de usuarios para estos servicios en el mundo [Statista, 2020].

Este uso no es únicamente con fines personales, gran parte de la comunicación que involucra un sistema de IM se lleva a cabo entre personal de distintas empresas, que realizan conversaciones de trabajo y negocio, uso que lo convierte en un blanco atractivo para el robo de identidades y de información, así como cualquier problema resultado de estos robos [Kusof, 2011].

Diseñar un sistema seguro de IM requiere de considerar el nivel de importancia de la información que se transmite, y garantice los principios de autenticidad e integridad; los recursos de Hardware disponibles, en las plataformas móviles, computadoras de escritorio, dentro de una red privada o pública, entre otros; y en última instancia la interfaz de trabajo Humano-Computadora.

Actualmente, existen diferentes propuestas para blindar con seguridad un sistema de IM, entre las que se encuentran sistemas multi-protocolo con registro de cuenta de usuario [Bala, 2017], protocolos de encriptación *end-to-end* [Whatsapp, 2017] [Zhang, 2017], [Ali, 2020] y sistemas de encriptación del texto, utilizados previamente en la transmisión, que emplean esquemas de cifrado Simétricos y Asimétricos aprobados por el NIST (del inglés, *National Institute of Standards and Tecnology*), de los que resaltan el uso de AES (del inglés, *Advanced Encryption Standard*) [Zhang, 2017], Algoritmos Hash [Yusof, 2011], así como sistemas de clave pública (PKC*, Public Key Cryptography) como RSA* [Ali, 2020] y de Curva Elíptica [Wanda, 2014], [Zhang, 2017].

Sin embargo, las soluciones con PKC propuestas para dar autenticación de usuario en los mensajes que no muestran resultados satisfactorios, al implementarse en un modelo cliente-servidor con múltiples usuarios, requieren mayor uso de memoria y tiempo de cómputo [Wanda, 2020]. Estudios en los que se comparan sistemas mixtos (clave publica - clave privada) como RSA-AES, contra sistemas AEAD (*Authenticated Encryption with Associated Data*), concluyen que los algoritmos AEAD tiene un mejor desempeño utilizando el mismo hardware y plataforma de software [Wang, 2011].

En este trabajo se presenta una propuesta al uso del esquema AEAD, como una alternativa que permita la identificación del usuario y la integridad del mensaje recibido, basado en la propuesta del trabajo de Wang y Yu [Wang, 2011].

Un esquema de cifrado autenticado (AE, por sus siglas del inglés: *Authenticated Encryption*), es un esquema de cifrado simétrico, también llamado de clave privada, cuyo objetivo es proporcionar privacidad, integridad y autenticidad de los datos encapsulados [Bellare, 2008]. En dicho esquema, el proceso de encriptación aplicado por el remitente utiliza la clave y un texto plano o mensaje (*Plaintext*), para

devolver un texto cifrado (*Ciphertext*), mientras que el proceso de descifrado aplicado por el receptor utiliza la misma clave y el *Ciphertext* para devolver, ya sea el *Plaintext*, o un símbolo especial que evalúa la autenticidad del mensaje [Bellare, 2008].

AEAD, es una variante de los esquemas de AE que enlaza datos asociados al Ciphertex y el contexto a donde pertenecen, de tal manera que los intentos de "cortar y pegar" un texto cifrado válido en un contexto diferente son detectados y rechazados.

2. Métodos

La metodología de este trabajo se centra en dos puntos principales: 1) La selección e implementación del sistema de cifrado, 2) La implementación del servicio de mensajería instantánea.

El sistema de cifrado

El sistema utilizado para el cifrado fue el algoritmo HANUMAN, de la familia de encriptadores PRIMATE, uno de los candidatos finalistas de la segunda ronda en la competencia CAESAR (*Competition for Authenticated Encryption: Security, Applicability, and Robustness*) [Vizár, 2016].

Primate es un grupo de encriptadores autenticados con dato asociado, conformado por los modos de operación APE, recomendado cuando se requiere mayor seguridad; HANUMAN, para aplicaciones ligeras y GIBBON, para aplicaciones donde la velocidad es fundamental [Andreeva, 2014], por lo que en esta aplicación se optó por el uso de HANUMAN, Debido al hecho de que PRIMATE puede usarse para aplicaciones que requieren encriptación y/o autenticación de mensajes muy cortos [Šijačić, 2017].

El nivel de seguridad de cada uno de los modos puede ser 80 o 120 bits que es determinado por: el subconjunto de permutaciones de PRIMATEs, el tamaño de la llave y de la etiqueta. La longitud del dato asociado y del texto pueden ser variables [Andreeva, 2014].

Algunas características específicas de HANUMAN son:

- El *nonce* para este modo debe ser único y por lo tanto no se debe de repetirse para que sea un sistema seguro.
- HANUMAN es secuencial y ofrece cifrado en línea, esto significa que puede crear bloques de texto cifrado sin la necesidad de tener conocimiento de longitud del texto o de los siguientes bloques del texto.
- Para este algoritmo es importante el uso de las permutaciones P1 y P4. Estas permutaciones consisten en el uso cíclico de las funciones: CA → MC → SR → SE. Estas funciones se iteran en una cantidad de rondas determinadas, que para el caso de HANUMAN corresponden a 12 iteraciones para P1 y P4.
- *ConstantAddition (CA):* Consiste en aplicar un XOR a un elemento determinado con un constante de ronda (rc) que se genera previamente con un LFSR Fibonacci 5-bit (*Linear Feedback Shift Register*) con un valor inicial de 1 para P1, y de 24 para P4.
- *MixColumns (MC):* Son multiplicaciones de columnas entre la matriz de entrada por una matriz, definido por ecuación 1.

$$F_{2^5} \cong \frac{F_2[x]}{(x^5 + x^2 + 1)} \tag{1}$$

- *ShiftRows (SR):* En esta función se realizan desplazamientos cíclicos a la izquierda en los renglones de acuerdo con s_i = {0,1,2,4,7} para un nivel de seguridad de 80.
- SubElements (SE): Es una transformación no lineal que consiste en una *S-box* con elementos de 5 bits, en el cual cada elemento *x* es sustituido por un elemento $S(x)$ de acuerdo con la tabla 1.

Tabla 1 S-Box utilizada en la familia PRIMATE.

x	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
$S(x)$	1	0	25	26	17	29	21	27	20	5	4	23	14	18	2	28
x	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
$S(x)$	15	8	6	3	13	7	24	16	30	9	31	10	22	12	11	19

Para este trabajo se eligió específicamente operar con HANUMAN80 el cual tiene un nivel de seguridad de 80 bits y al que le corresponde los parámetros de la tabla 2 [Andreeva, 2014].

Tabla 2 Parámetros de HANUMAN.

Notación	***Significado***	***Tamaño (bits)***
R	La tasa	40
C	La capacidad	160
$K \in C^{1/2}$	Llave	80
$N \in C^{1/2}$	*Nonce*	80
$A \in \{0,1\}^*$	Dato asociado	Variable
$M \in \{0,1\}^*$	Texto	Variable
$C \in \{0,1\}^*$	Texto cifrado	Variable de acuerdo con M
$T \in C^{1/2}$	Etiqueta	80
0^r	40 ceros	40
$\|\|$	Concatenación	-
$\oplus$	Operación XOR	-
$\neq$	Desigualdad	-
$\perp$	Indica que las etiquetas no coindicen.	-
$M[1]M[2] \ldots M[w] \leftarrow M$	Divide el texto en bloques.	Variable
$M[w] \,\|\|\, 10^*$	Rellena el último bloque de M comenzando con un bit en uno y seguido de ceros hasta que sea un múltiplo de r.	Variable
$J = \|M[w]\|$	Mide el número de bits en el último bloque de M.	Variable
$\lfloor M[w] \rfloor_n$	Toma los n bits más significativos de último bloque de M.	Variable
V	Estado (*state*)	200
V_r	Parte de la tasa (*rate part*)	40
V_c	Parte de la capacidad (*capacity part*)	160

En figuras 1 y 2, se muestran los diagramas de flujo para encriptación y desencriptación del algoritmo HANUMAN, respectivamente.

El Servidor y la aplicación

El servidor y la aplicación del usuario fueron programados en *Python* v3.8 para su mejor integración con el encriptador programado en ese lenguaje.

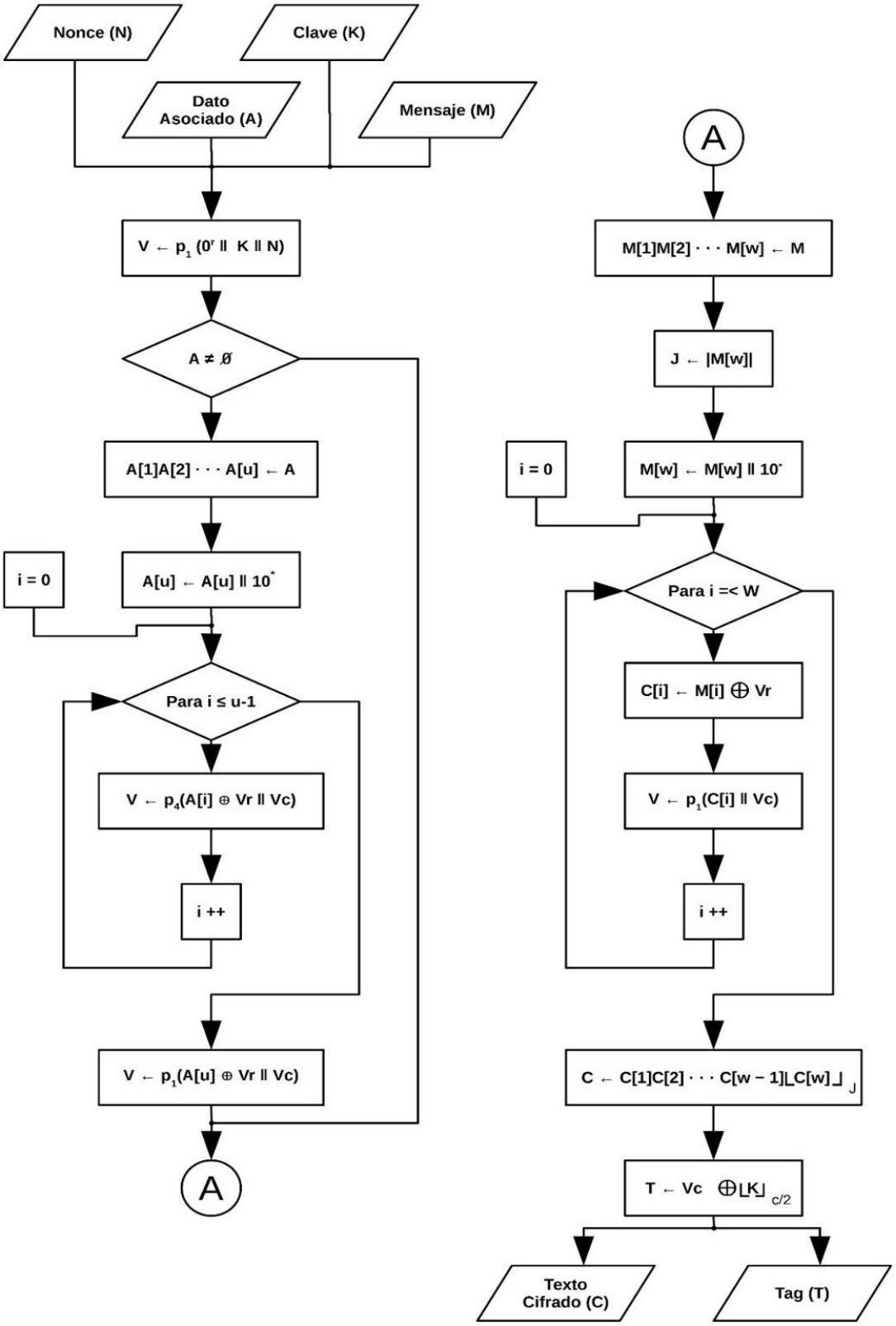

Figura 1 Diagrama de flujo del algoritmo de encriptacion HANUMAN.

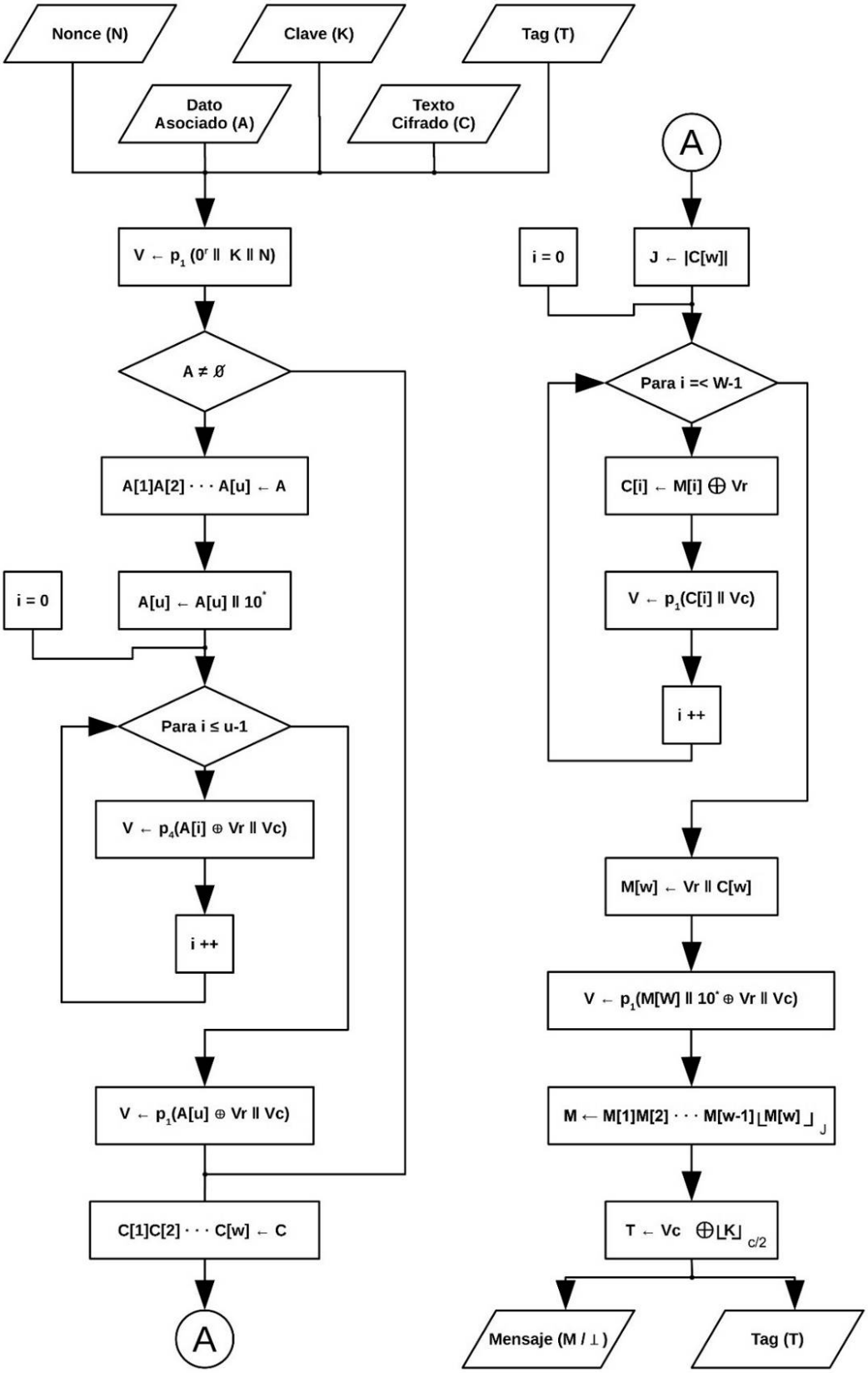

Figura 2 Diagrama de flujo del algoritmo de desencriptacion HANUMAN.

Primero se procedió a programar el servidor que concentra las conexiones, así como el flujo de mensajes. Se utilizaron *sockets udp* para crear la *interfaz* local al *host* que es controlado por el Sistema Operativo (OS, del inglés: *Operative System*); estos *sockets* ofrecen dos tipos de servicios de transporte [Rhodes, 2014]:

- *Stream:* Flujo de datos orientado a conexiones.
- *Dgram:* Datagramas.

El *socket* es una puerta entre el proceso de aplicación y el protocolo de transporte. Además, se utilizó *AF_INET*, una familia de direcciones que se usa para designar el tipo de conexión con el cual el *socket* se puede comunicar (en este caso con el protocolo de direcciones IPv4).

Se agregó la librería *threading* que sirve para separar el flujo de ejecución en "hilos", con el propósito de hacer que el servidor le asigne un hilo de ejecución a cada usuario que se conecta.

Se definieron las constantes y variables:

- *clients*: Identifica al cliente.
- *addresses*: Guarda la dirección *IP* del cliente.
- *Host:* Introduce la dirección *IP* del servidor de chat.
- *Port:* Asigna el puerto por el cual se conectará al servidor.
- *Bufsiz:* Determina el tamaño máximo de memoria asignada a los caracteres que pueden escribirse en la *interface*.
- *Addr:* Almacena valores de *clients* y *addresses*.
- *Server:* Establece el tipo de direcciones *IP* que se utilizarán, así como el flujo de datos que maneja el servidor.
- *Server.bind:* Enlaza al servidor con los clientes.
- Para conexiones entrantes se utilizó una función que al conectarse un cliente al servidor toma la *IP* y el nombre de usuario que el cliente introduzca, y asigna esta conexión a un hilo de proceso, como se ilustra en la figura 3.

Una vez conectado el usuario al servidor lo siguiente es la introducción del cliente a la sala del chat, y el control de su salida con la función presente en la figura 4. Para

la transmisión de los mensajes se definió la función *broadcast*, presente en la figura 5, la cual toma el nombre del cliente que hace la petición de transmisión a la sala de chat y lo transmite junto con el mensaje.

```
30 def accept_incoming_connections():
31     """Sets up handling for incoming clients."""
32     while True:
33         client, client_address = SERVER.accept()
34         print("%s:%s has connected." % client_address)
35         client.send(bytes("Now type your name and press enter!", "utf8"))
36         addresses[client] = client_address
37         Thread(target=handle_client, args=(client,)).start()
```

Figura 3 Definición de función a conexiones entrantes.

```
40 def handle_client(client):  # Takes client socket as argument.
41     """Handles a single client connection."""
42 
43     name = client.recv(BUFSIZ).decode("utf8")
44     welcome = 'Welcome %s! If you ever want to quit, type {quit} to exit.' % name
45     client.send(bytes(welcome, "utf8"))
46     msg = "%s has joined the chat!" % name
47     broadcast(bytes(msg, "utf8"))
48     clients[client] = name
49 
50     while True:
51         msg = client.recv(BUFSIZ)
52         if msg != bytes("{quit}", "utf8"):
53             broadcast(msg, name+": ")
54         else:
55             client.send(bytes("{quit}", "utf8"))
56             client.close()
57             del clients[client]
58             broadcast(bytes("%s has left the chat." % name, "utf8"))
59             break
```

Figura 4 Función manejo de clientes.

```
62 def broadcast(msg, prefix=""):  # prefix is for name identification.
63     """Broadcasts a message to all the clients."""
64 
65     for sock in clients:
66         sock.send(bytes(prefix, "utf8")+msg)
```

Figura 5 Función de transmisión.

El código principal del servidor, descrito en la figura 6, contiene las funciones:

- *SERVER.listen():* Función que establece las conexiones máximas permitidas para la sala de chat.
- *ACCEPT_THREAD = Thread():* Llama la función de conexiones de entrada y asigna un hilo de proceso para cada conexión entrante.

- *ACCEPT_THREAD.start():* Función de inicio del servidor.
- *ACCEPT_THREAD.join():* Función de enlace a nuevos clientes, y su salida de la transmisión utilizando el comando *{quit}*.
- *SERVER.close():* Función que cierra el servidor y todas sus conexiones.
- Para la interfaz gráfica de la aplicación que utiliza el cliente para conectarse al servidor, se utiliza *Tkinter v8.6*, que es una librería de *Python* v3.8.

```
79 if __name__ == "__main__":
80     SERVER.listen(5)
81     print("Waiting for connection...")
82     ACCEPT_THREAD = Thread(target=accept_incoming_connections)
83     ACCEPT_THREAD.start()
84     ACCEPT_THREAD.join()
85     SERVER.close()
```

Figura 6 Código principal del servidor.

Codificación del cliente

Además de *socket, threading* y *tkinter*, se utilizan las librerías *HanumanEncriptador*; encargada de encriptar la información, y *HanumanDesencriptador*; para el descifrado, previamente codificadas en *Python v3.8* utilizando los algoritmos de las figuras 1 y 2. Los parámetros de entrada para la encriptación es el mensaje, no mayor a 25 caracteres como lo establece el esquema de encriptación HANUMAN, lo mismo sucede con los parámetros de salida en la desencriptación. La codificación de la función para la recepción de los mensajes provenientes del servidor utiliza la función descrita en la figura 7 donde:

- *msgc:* Es la variable que contiene el mensaje encriptado.
- *msg = client_socket.recv(BUFSIZ).decode("utf8"):* Recibe el mensaje proveniente del servidor utilizando la función *client _socket.recv.*
- *BUFSIZ*: Es el tamaño del buffer del mensaje.
- *.decode("utf8")*: Decodifica el mensaje a valores *utf8* para su empleo como lo requiere la librería *socket.*

Se resalta que no todos los mensajes que se reciben del servidor son de otros usuarios, algunos son información útil del servidor como la conexión y desconexión de usuarios.

```
37 def receive():
38     """Handles receiving of messages."""
39     while True:
40         try:
41             global b
42             msgc = ""
43             msg = client_socket.recv(BUFSIZ).decode("utf8")
44             if (": " in msg):
45                b = msg.find(": ")
46                b += 2
47                for i in range(b, 30):
48                    msgc += msg[i]
49                msgc = Desencriptacion(msgc)
50             msg_list.insert(tkinter.END, msg)
51             msg_list.insert(tkinter.END, msgc)
52             msgc = ""
53         except OSError:  # Possibly client has left the chat.
54             break
```

Figura 7 Función *receive*.

Para identificar sí se trata de un *ciphertext* se buscan los caracteres *": "*. De encontrarlos, se le asigna una bandera, denotada con la variable "*b*" para indicar el inicio del mensaje a descifrar.

En la figura 5 las líneas 50 y 51 presentan el *ciphertext* y el *plaintext*, respectivamente. Además de la información del usuario que lo envía.

La función *"send"* que se presenta en la figura 8, es habilitada mediante el evento "*msg = my_msg.get()"* que ocurre cuando se presiona el botón "*send"* de la *interface*, esta función toma los valores escritos de la entrada de caracteres y los guarda en la variable "*msg"*, enseguida se activa una bandera para denotar que el primer mensaje que se envía es el nombre del usuario y así no cifrarlo, la siguiente línea limpia el campo de entrada de caracteres. A partir de la línea 63 en la figura 8 se evalúa que el mensaje sea distinto al comando de cierre "{*quit*}", de no cumplirse evalúa y asigna la variable *"a"* sí el mensaje es el nombre del usuario, de lo contrario se procederá a evaluar que el tamaño de la cadena de caracteres no sea mayor al permitido y a rellenarlo con caracteres nulos de ser menor; al ser codificados en *utf8* se puede presentar un cambio de caracteres de no estar completo.

En la línea 70 la función de encriptación manda el mensaje en la variable "*msg"* para encriptar. La línea 72 muestra un mensaje de error en caso de que el mensaje exceda el número máximo de caracteres. La función "*client_socket.send()*", una vez codificada la información en *utf8* la manda al servidor. De cumplirse la condición

correspondiente al comando *"{quit}"* en la línea 75 se cierran las conexiones con el servidor y termina la ejecución del programa.

La función de cierre de la interfaz, y el envío de término de la conexión al servidor es definido como se muestra en la figura 9. El código presente en la figura 10 establece la IP y el puerto para la conexión con la sala de chat del servidor, que son definidas en la terminal de ejecución del programa.

```
57 def send(event=None):  # event is passed by binders.
58     """Handles sending of messages."""
59     global a
60     msg = my_msg.get()
61     a += 1
62     my_msg.set("")  # Clears input field.
63     if msg != "{quit}":
64        if a > 1:
65           if len(msg) < 26:
66              j = len(msg)
67              for i in range(j, 25):
68                  msg += " "
69
70              msg = Encriptacion(msg)
71           else:
72              msg = "Mensaje Con Mas De 25 Caracteres No se Enviara"
73              msg = ""
74     client_socket.send(bytes(msg, "utf8"))
75     if msg == "{quit}":
76         client_socket.close()
77         top.quit()
```

Figura 8 Función *send*.

```
94 def on_closing(event=None):
95     """This function is to be called when the window is closed."""
96     my_msg.set("{quit}")
97     send()
```

Figura 9 Función cierre de aplicación.

3. Resultados

Se tienen cuatro códigos en *Python*, siendo éstos los códigos del sistema de cifrado: *HanumanEncriptador y HanumanDesencriptador;* el servidor y el cliente. El servidor fue verificado en condiciones de laboratorio mediante una red local. Por otro lado, dicho servidor se ejecuta mediante el comando: *python3 chatServer.py.* Además, el cliente se ejecuta desde otro equipo situado en la misma red, utilizando el comando: *python3 guiChatClient.py.*

```
#----Now comes the sockets part----
HOST = input('Enter host: ')
PORT = input('Enter port: ')
my_msg.set("")
if not PORT:
    PORT = 33000
else:
    PORT = int(PORT)

BUFSIZ = 1024
ADDR = (HOST, PORT)

client_socket = socket(AF_INET, SOCK_STREAM)
client_socket.connect(ADDR)

receive_thread = Thread(target=receive)
receive_thread.start()
tkinter.mainloop()  # Starts GUI execution.
```

Figura 10 Configuración de IP y puerto del servidor.

Al ejecutarse el sistema, como se muestra en la figura 11, se solicita la dirección IP del servidor en el campo "*Enter host"* y el número de puerto para la conexión en la sección "*Enter port*". Por otro lado, cuando se realiza la conexión del cliente con el servidor, se presenta una interfaz de usuario como se observa en la figura 12.

```
luis@luis-pc:~/EncriptadorHanumanChat$ python3 guiChatClient.py
Enter host: 192.168.0.41
Enter port: 33000
```

Figura 11 Conexión del cliente.

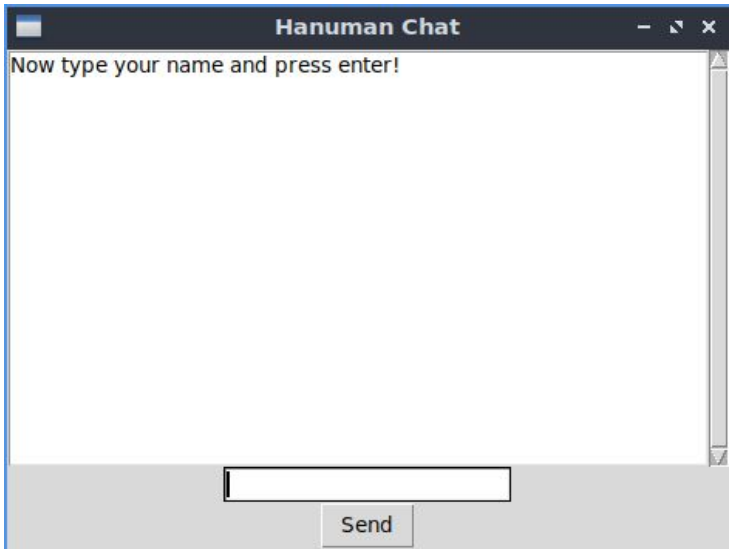

Figura 12 Interfaz cliente.

Al conectar el primer usuario al servidor, en la terminal aparece la información correspondiente al usuario, como es la dirección IP, seguida del número de puerto de entrada asignado por el servidor. En el momento en que más usuarios se conecten al servidor se presentará la información de los nuevos clientes en la terminal como se muestra en figura 13.

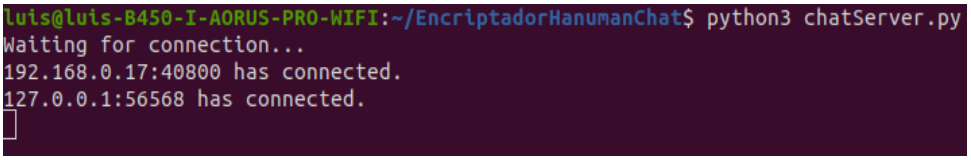

Figura 13 Conexión de varios clientes al servidor.

En la figura 14 se presenta un ejemplo de conversación entre dos usuarios conectados, "*Jose*" y "*Luis".* Lo primero que se muestra después de la línea de bienvenida es el usuario nuevo que se unió a la sala de chat. La línea siguiente muestra el mensaje encriptado que envió el usuario "*Jose*" (Cabe destacar que ese es el mismo mensaje que el servidor recibe ya que el mensaje solo se desencripta al momento de recibirse), la siguiente línea es el mensaje desencriptado. De igual manera las dos siguientes líneas son el mensaje que envió el usuario "*Luis*" la primera es el mensaje encriptado y la segunda es el mensaje desencriptado.

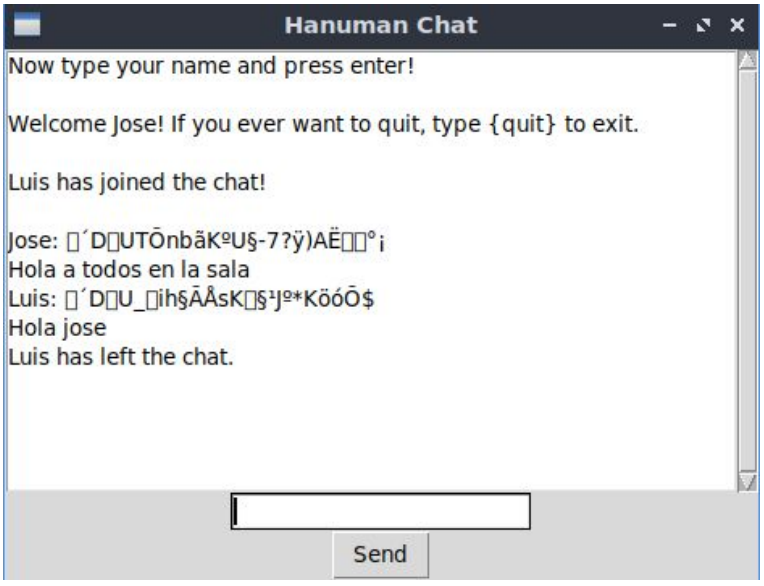

Figura 14 Ejemplo de conversación de dos usuarios.

Para salir de la sala de chat se utiliza el comando "*{quit}*" que hará la desconexión al servidor, y cierra la ventana de chat terminando la ejecución del programa.

4. Discusión

El objetivo en este trabajo fue la utilización de esquemas de AEAD como una alternativa que brinde de seguridad, una manera de autenticación del usuario y verificación de la integridad del mensaje recibido. Para lo cual se usó el algoritmo HANUMAN, de la familia de encriptadores PRIMATEs, por su ligereza ya que esto nos permite utilizarlo en sistemas de bajo recursos.

El sistema de encriptación y el mensajero instantáneo fueron programados en *Python v3.8* por la sencillez y versatilidad del lenguaje, sin embargo, la programación de ambos sistemas en otros lenguajes es posible, incluso llegando a mejorar el tiempo de respuesta, por ejemplo, del encriptador, sobre un lenguaje de programación compilado.

La seguridad del sistema recae en el esquema de cifrado y los protocolos de comunicación, y este trabajo solo presenta la utilización del encriptador sobre los mensajes, dejando como trabajo siguiente verificar la seguridad total del sistema, y desarrollar las pruebas necesarias contra ataques a los clientes.

Cabe destacar que en la aplicación real de la sala de chat no se presentarían los mensajes encriptados y solo aparecen como prueba del cifrado.

Una ventaja de esta implementación de mensajería es la separación del servidor y el cliente puesto que, sí se ve comprometido el nivel de seguridad de alguno de ellos, los usuarios no tendrían problema con sus conversaciones porque tanto el servidor como los clientes no almacenan ni pueden revisar ninguno de los textos ya que están encriptados desde la aplicación de cada uno de los usuarios, y una vez que se cierra se borra el historial del chat. Los autores analizan la posibilidad de utilizar este sistema de cifrado en la comunicación de proyectos *IoT* (del inglés: *Internet of Things*). Mejorar la implementación del mensajero instantáneo para su uso en grupos de trabajo a distancia y su compatibilidad con sistemas de seguridad más sofisticados, de certificación, envió de claves, e incluso con sistemas *PKI* (por sus siglas en inglés: *Public Key Interface*)**.**

5. Conclusiones

Este trabajo presenta una propuesta de implementación experimental en el uso de los algoritmos de encriptación autenticada para mensajería instantánea, abriendo una posibilidad de mejora y optimización. Se describe detalladamente la codificación en *Python* v3.8 del sistema de IM con cifrado basado en AE. Se caracterizó bajo condiciones de laboratorio mediante una red local.

El sistema implementado se basa en el algoritmo HANUMAN, de la familia PRIMATE, este grupo de encriptadores está conformado por los modos de operación APE, HANUMAN y GIBBON; que pertenecen al tipo de encriptación autenticado con datos asociados, de entre los tres fue elegido HANUMAN por ser el recomendado por los mismos autores para aplicaciones con bajo consumo de recursos.

Las características del sistema son portabilidad y bajo consumo de recursos, éstas permiten que pueda ser considerado para su posible implementación dentro del sistema de comunicaciones en proyectos de *IoT*. Por otro lado, la modularidad de la arquitectura permite el cambio del sistema de cifrado por otros esquemas con prestaciones similares, según la aplicación. Además, es posible incluir más de un esquema de cifrado, con el objetivo de incrementar las prestaciones en el envió de información, como son imágenes y audio.

6. Bibliografía y Referencias

[1] Ali R. M. and Alsaad S. N., Instant Messaging Security and privacy secure instant messenger design. IOP Conf. Ser.: Mater. Sci. Eng. Vol. 881, doi:10.1088/1757-899X/881/1/012117, April 2020.

[2] Andreeva, E., Bilgin, B., Bogdanov, A., Luykx, A., Mendel, F., Mennink, B., Mouha, N., Wang, Q., Yasuda, K., PRIMATEs v1.02 Submission to the CAESAR Competition. http://primates.ae/, 2014.

[3] Bala S. and Wasilczyk T., Secure integration of multiprotocol instant messenger, 2017 IEEE International Conference on Innovations in Intelligent SysTems and Applications (INISTA), Gdynia, pp. 495-500, doi: 10.1109/INISTA.2017.8001210, 2017.

[4] Bellare, M., Namprempre, C. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. J Cryptol 21, pp. 469–491. https://doi.org/10.1007/s00145-008-9026-x, 2008.

[5] Moore A. D., Python GUI Programming with TKinter: Develop Responsive and Powerful GUI Applications with Tkinter, Packt Publishing, ISBN: 1788835883, May 2018.

[6] Rhodes B., Goerzen J., Foundations of Python Network Programming, Third Edition, eBook ISBN: 978-1-4302-5855-1, doi: 10.1007/978-1-4302-5855-1, 2014.

[7] Šijačić D., Kidmose A. B., Yang B., Banik S., Bilgin B., Bogdanov A. and Verbauwhede I. Hold Your Breath, PRIMATEs Are Lightweight. In: Avanzi R., Heys H. (eds) Selected Areas in Cryptography, SAC 2016. Lecture Notes in Computer Science, vol 10532 pp. 197–216, doi:10.1007/978-3-319-69453-5_11, 2017.

[8] Statista. (2020). Aplicaciones de mensajería más populares según el número de usuarios mensuales activos a nivel mundial a enero de 2020, Url: https://es.statista.com/estadisticas/599043/aplicaciones-de-mensajeria-mas-populares-a-nivel-mundial-de/. Consultado en mayo del 2020.

[9] Vizár D., Ciphertext forgery on HANUMAN. Cryptology ePrint Archive, Report 2016/697, 2016.

[10] Wanda P. & Jie J. H., Efficient Data Security for Mobile Instant Messenger. TELKOMNIKA (Telecommunication Computing Electronics and Control), vol.16, no.3, pp. 1426-1435, ISSN: 1693-6930, doi: 10.12928/TELKOMNIKA.v16i3.4045. June 2018.

[11] Wanda P., Selo and Hantono B. S., Efficient message security based Hyper Elliptic Curve Cryptosystem (HECC) for Mobile Instant Messenger, 2014 The 1st International Conference on Information Technology, Computer, and Electrical Engineering, Semarang, pp. 245-249, doi: 10.1109/ICITACEE.2014.7065750, 2014.

[12] Wang, Hao & Yu., Comparison between PKI (RSA-AES) and AEAD (AES-EAX PSK) cryptography systems for use in SMS-based secure transmissions.

Communications in Computer and Information Science. Vol. 136. Ed. 2, pp. 1-12, Doi:10.1007/978-3-642-22185-9_1, 2011.

[13] Whatsapp., WhatsApp Encryption Overview Technical white paper: https://scontent.whatsapp.net/v/t61.22868-34/68135620_760356657751682621299752885183355 9_n.pdf/WhatsApp-Security-Whitepaper.pdf?_nc_sid=41cc27&_nc_ohc=JlPsBK-B7v4AX8x_-t1&_nc_ht=scontent.Whatsapp.net&oh=a50263e0fe1d6ea56419d80f2eb2a378&oe=5F632793, 2017.

[14] Yusof M. K. and Abidin A. F. A., A secure private instant messenger, The 17th Asia Pacific Conference on Communications, pp. 821-825, doi: 10.1109/APCC.2011.6152921, Sabah, 2011.

[15] Zhang L., Ji Q., and Yu F., The Security Analysis of Popular Instant Messaging Applications, 2017 International Conference on Computer Systems, Electronics and Control (ICCSEC), pp. 1324-1328, doi: 10.1109/ICCSEC.2017.8446863, Dalian, 2017.