Recibido: 10 enero 2022 Aceptado: 28 de febrero 2022

Estudio comparativo de los algoritmos de criptografía simétrica AES, 3DES y ChaCha20

Comparative study of the symmetric cryptography algorithms AES, 3DES and ChaCha20

Leonardo Sergio Centellas Claros[1], Leticia Blanco Coca[1], Juan Pablo Sandoval Alcocer[2]

[1] Carrera de Licenciatura en Ingeniería Informática, Universidad Mayor de San Simón, Cochabamba, Bolivia.

[2] Departamento en Ciencias de la Computación, Escuela de Ingeniería, Pontificia Universidad Católica de Chile, Santiago, Chile

201800122@est.umss.edu

Resumen: El constante avance de la tecnología incrementa cada vez más la dependencia del internet y el envío de información a través de la misma, aumenta también la necesidad de seguridad en la red y los dispositivos conectados a ella. Para mantener la seguridad de toda esta información se crearon los algoritmos de encriptación, los cuales están encargados de cifrar los datos de modo que sólo quienes estén autorizados puedan entenderlos. Las crecientes amenazas de seguridad han forzado la creación continua de distintos algoritmos de encriptación, recientemente se introdujo el algoritmo ChaCha20 que trata de sustituir a su predecesor el Salsa20, el cual era considerablemente rápido y bastante confiable respecto a su seguridad. El ChaCha20 promete aumentar la difusión respecto a su predecesor para mejorar su seguridad sin comprometer su rendimiento. Sin embargo, poco se sabe de su eficiencia en comparación a los otros algoritmos, lo cual limita una decisión racional al momento de elegir si utilizarlo o no.

Este artículo presenta un estudio empírico en el cual comparamos el rendimiento de ChaCha20 con otros dos algoritmos de encriptación AES, y 3DES. En este artículo evaluamos el tiempo de ejecución, consumo de CPU y consumo de energía que toma cada uno de los algoritmos para encriptar y des-encriptar información. Se desarrolló un experimento con 150 archivos de distintos formatos y tamaños para probar cada algoritmo. Se determinó que el algoritmo ChaCha20 es el más rápido, AES es el que menos CPU consume y no existe diferencia significativa en el consumo de energía de ninguno de los algoritmos.

Palabras clave: encriptación simétrica, criptografía simétrica, AES, 3DES, ChaCha20, rendimiento, eficiencia.

Abstract: The constant advancement of technology increases the dependence on the Internet and the sending of information through it, it also increases the need for security in the network and the devices connected to it. To maintain the security

of all this information, encryption algorithms were created, which are in charge of encrypting the data so that only those who are authorized can understand it. The increasing security threats have forced the continuous creation of different encryption algorithms, recently the ChaCha20 algorithm was introduced, which tries to replace its predecessor, Salsa20, which was considerably fast and quite reliable regarding its security. The ChaCha20 promises to increase the spread compared to its predecessor to improve its security without compromising its performance. However, little is known about its efficiency compared to other algorithms, which limits a rational decision when choosing whether to use it or not.

This article presents an empirical study in which we compared the performance of ChaCha20 with two other encryption algorithms AES, and 3DES. In this article we evaluate the execution time, CPU consumption and energy consumption that each of the algorithms take to encrypt and decrypt information. An experiment with 150 files of different formats and sizes was developed to test each algorithm. It was determined that the ChaCha20 algorithm is the fastest, AES is the one that consumes the least CPU and there is no significant difference in the power consumption of any of the algorithms.

Keywords: symmetric encryption, symmetric cryptography, AES, 3DES, ChaCha20, performance, efficiency.

1 Introducción

Los algoritmos de encriptación son los encargados de cifrar la información, de modo que aunque alguien tenga acceso a ella no pueda entenderla sin conocer el método con el que fue encriptado. Actualmente existe una gran cantidad de algoritmos para este propósito, y el número sigue creciendo. Aunque la encriptación ha existido desde antes de las computadoras y la red, esta ha cobrado mayor importancia conforme ha crecido la necesidad de seguridad digital.

Un sistema criptográfico consiste de varios elementos, los principales son: texto plano, cifrador, texto cifrado y llave. El texto plano es aquel mensaje o información que se desea enviar. Los cifradores son los algoritmos matemáticos que se encargan de cifrar o descifrar. El texto cifrado es el resultado de aplicar el algoritmo de cifrado al texto plano. La llave o clave es un arreglo de bits que usa un algoritmo para cifrar o descifrar un mensaje (Migga, 2015).

Los algoritmos de encriptación se pueden clasificar en dos grupos, simétricos y asimétricos. Los algoritmos de encriptación simétricos utilizan una sola llave tanto para encriptar como para desencriptar. Los asimétricos utilizan una llave para cifrar y otra llave para descifrar. Cada uno de estos grupos tiene sus ventajas y desventajas. En el caso de los algoritmos simétricos existe mayor riesgo de seguridad al compartir la clave, mientras que en los asimétricos la cantidad de llaves existentes escala exponencialmente con cada equipo que se agrega a la red.

En la actualidad uno de los algoritmos más usados es el AES, que actualmente es usado para cifrar las conversaciones en la conocida aplicación WhatsApp (WhatsApp, 2021). El algoritmo 3DES es el antecesor del AES, pero aún es bastante usado y sujeto de estudio a pesar de que este y el AES surgieron en el siglo pasado. El tercer algoritmo usado es el ChaCha20, el más nuevo de los tres, siendo publicado en 2008 (Bernstein, 2008). Este último algoritmo se escogió por ser relativamente nuevo y no contar con tantos estudios formales de rendimiento como los anteriores.

En este documento se desarrolla un estudio empírico, el cual compara el rendimiento de los tres algoritmos de encriptación ya mencionados. El propósito es determinar si los algoritmos AES y 3DES siguen siendo relevantes o debería considerarse usar el algoritmo ChaCha20. Para determinar esto se midió el tiempo, consumo de energía y el porcentaje GPU empleado por cada uno de los algoritmos para encriptar y desencriptar con 150 archivos con diferentes formatos y tamaños.

2 Trabajos relacionados

En 2006 Al Tamimi realizó una comparación de 4 algoritmos de cifrado: DES, 3DES, AES y Blowfish. La comparación fue realizada usando distintos modos de operación en bloques. La conclusión a la que se llegó fue que el algoritmo Blowfish tiene el mejor rendimiento y hasta la fecha en la que se realizó el estudio no se conocían brechas de seguridad en dicho algoritmo. Por el otro lado se concluyó que el algoritmo AES tiene el peor rendimiento debido a la gran cantidad de poder de procesamiento que requiere (Al Tamimi, 2006).

En 2010 Elminaam hizo un estudio del rendimiento de los algoritmos de encriptación simétrica: AES, DES, 3DES, RC2, Blowfish y RC6. Se compararon los algoritmos tomando en cuenta aspectos como el tiempo de encriptación y desencriptación, múltiples formatos de datos, variación en el tamaño de llave, etc. Se llegó a la conclusión de que el algoritmo Blowfish tiene el mayor rendimiento cuando se varía el tamaño de los paquetes y que el algoritmo AES tiene mejor rendimiento que DES, 3DES y RC2 (Abd Elminaam et al., 2010).

En 2020 Alenezi realizó un estudio comparando los algoritmos de encriptación simétrica: AES, BlowFish, DES, 3DES, SEED, IDEA, RC2, RC4, RC6, SEED, y XTEA. Este estudio utilizó los factores de tiempo de encriptación y uso de CPU. Se determinó que el algoritmo AES es el mejor candidato debido a su rendimiento. También se recalcó que para escoger un algoritmo de encriptación se deberían tomar en cuenta otros factores además del rendimiento (Alenezi et al., 2020).

3 Criptografía

La criptografía es un proceso complejo, pero no solo por los algoritmos de encriptación, sino también por los otros elementos que de los que se compone.

- Texto plano.- El texto plano es aquel mensaje que se desea enviar. En el estado en el que está cualquiera puede leerlo y por ende no es seguro.
- Llave.- La llave o clave es la cadena de bits que utilizan los algoritmos para cifrar o descifrar los mensajes.
- Cifradores.- Los cifradores son los algoritmos matemáticos que usan la llave para cifrar y descifrar. Suelen funcionar en pares para que cada uno se encargue de una tarea, es decir uno para encriptar y otro para desencriptar.
- Texto cifrado.- Es el resultado de cifrar el texto plano. Este texto no puede ser leído a menos que se sepa el medio por el cual fue cifrado y la llave, por lo cual es seguro enviarlo a través de la red.
- Nonce.- Un nonce es un número aleatorio usado en la criptografía como algo similar a la llave. Está pensado para usarse solo una vez por comunicación, de esta manera no se podrá replicar, ya que para cuando se descifre ya habrá cambiado.
- Flujo de llaves.- Un flujo de llaves es un conjunto de bits generado para tener el mismo tamaño que el texto plano o texto cifrado. Este flujo se aplica al texto con alguna operación matemática como suma o XOR. Los cifradores de flujo suelen usar este método.

Como ya se pudo ver, la criptografía tiene muchos componentes. En términos simples, un cifrador recibe un texto plano y una llave y devuelve un texto cifrado que se puede enviar por la red, una vez llegado al destino el algoritmo contraparte del cifrador utiliza la llave y el texto cifrado para dar como resultado el texto plano original. Esta es una versión resumida del proceso que se realiza, ya que diferentes algoritmos pueden presentar modificaciones o utilizar más elementos.

3.1 Criptografía simétrica

La criptografía simétrica se caracteriza por usar solamente una llave, tanto para encriptar como para desencriptar. Existen muchos algoritmos de este tipo, estos pueden clasificarse en 2 tipos: cifradores por bloques y cifradores de flujo.

Los cifradores por bloques se caracterizan por separar los datos a cifrar/descifrar en bloques de tamaño definido. El proceso que realizan es encriptar/desencriptar cada bloque individualmente y unir los resultados en el mismo

orden que los tomaron. Por otro lado, los cifradores de flujo generan un flujo de claves del mismo tamaño que el texto plano. Este flujo se combina con el texto plano para generar el texto cifrado o con el texto cifrado para recuperar el texto plano. Esta combinación se realiza usualmente con la operación XOR.

3.2 Modos de operación

Los algoritmos de cifrado por bloques como AES y 3DES están diseñados para cifrar información de un tamaño de bytes específico, pero por lo general la información no cumple con esta condición. Este problema se resuelve dividiendo la información en bloques del tamaño requerido para aplicar el algoritmo de cifrado en cada uno. En caso de que el último bloque no tenga los bytes suficientes se lo rellena con una cadena de bytes determinada, la cual se elimina al momento de desencriptar.

Es a nivel de estos bloques que se pueden realizar modificaciones al algoritmo llamadas modos de operación. Una definición es "Un algoritmo para la transformación criptográfica de datos que presenta un algoritmo de cifrado por bloques de clave simétrica" (Dworkin, 2001). Entre los distintos modos de operación que existen los 2 más comunes son ECB y CBC, pero también existen otros como CFB, OFB y CTR.

El libro de códigos electrónico (ECB) es el modo más simple que hay, no realiza ninguna modificación al funcionamiento del algoritmo. En otras palabras, en este modo cada bloque se cifra de manera independiente con el mismo algoritmo y llave (Tanenbaum y Wetherall, 2012).

El encadenamiento de bloques de cifrado (CBC), como su nombre lo indica, encadena los bloques uno a continuación del otro. El primer bloque es juntado con un bloque del mismo tamaño llamado Vector de Inicialización, IV por sus siglas en inglés, con la operación XOR antes de ser encriptado, esto aporta cierto nivel de aleatoriedad al resultado. A excepción del primero, antes de encriptar a cada bloque se le aplica la operación matemática XOR con el bloque encriptado anterior. Al momento de descifrar, después de desencriptar un bloque se le aplica la operación XOR con el bloque encriptado anterior, nuevamente omitiendo el primero, ya que este se junta con el bloque IV (Tanenbaum y Wetherall, 2012).

4 Descripción de los algoritmos

Aunque los algoritmos seleccionados pertenecen todos a la criptografía simétrica, cada uno tiene sus respectivos procedimientos. Los algoritmos AES y 3DES son algoritmos por bloques, mientras que el ChaCha20 es un cifrador de flujo.

4.1 AES

El algoritmo AES fue estandarizado por el Instituto Nacional de Estándares y Tecnología en la publicación de la FIPS 197 (Dworkin et al., 2001). El estándar admite tres versiones distintas del mismo algoritmo, estas varían esencialmente en el tamaño de llave que reciben. Los tamaños de llave permitidos son 128, 192 y 256 bits. En dicho documento, donde se describe el estándar del algoritmo, también se encuentran datos y ejemplos de prueba para verificar el correcto funcionamiento de cualquier implementación.

El algoritmo trabaja sobre bloques de 128 bits, es decir que separa los datos del texto plano en bloques de 128 bits, encripta uno, devuelve un bloque de 128 bits y realiza el mismo procedimiento sobre cada uno de los demás bloques. El proceso es el mismo para todos los bloques.

4.1.1 Cifrado

El proceso de cifrado de un bloque se realiza por rondas, durante las cuales se realizan las siguientes operaciones: SubBytes, ShiftRows, MixColumns y AddRoundKey.

- SubBytes.- Recibe un arreglo de 16 bytes y cambia cada elemento por el resultado de una transformación matemática. del mismo. El proceso se puede simplificar usando una tabla de valores precalculados llamada S-box.
- ShiftRows.- Recibe un arreglo de bytes {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15} y devuelve el arreglo ordenado como {0,5,10,15,4,9,14,3,8,13,2,7,12,1,6,11}.
- MixColumns.- Recibe un arreglo de 16 bytes expresado como una matriz de 4x4. Reemplaza cada columna de la matriz con el resultado de la multiplicación matricial de la misma columna y una matriz de tamaño 4x4 propia de la función.
- AddRoundKey.- Recibe un arreglo de 16 bytes y le aplica la operación XOR con el bloque de la expansión de llave correspondiente a la ronda de encriptación actual.

La cantidad de rondas que realiza el algoritmo son 10, 12 o 14 según si la clave es de 128, 192 o 256 bits respectivamente, a este número de rondas se le denomina *nr*.

Las distintas rondas realizan las operaciones de la siguiente manera:

Ronda 0

AddRoundKey

Rondas 1 a nr -1

SubBytes

ShiftRows

MixColumns

AddRoundKey

Ronda nr

SubBytes

ShiftRows

AddRoundKey

4.1.2 Descifrado

Al momento de descifrar, la cantidad de rondas que se realizan también depende del tamaño de la llave. Las operaciones que se efectúan en esas rondas son: InvSubBytes, InvShiftRows, InvMixColumns y AddRoundKey. Las primeras 3 operaciones son, como indican sus nombres, las inversas de las efectuadas al momento de encriptar.

- InvSubBytes.- Recibe un arreglo de 16 bytes y aplica a cada elemento la transformación matemática opuesta que la función SubBytes. Esta función también se puede simplificar con el uso de un tabla llamada S-box inversa.
- InvShiftRows.- Recibe un arreglo de bytes {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15} y devuelve el arreglo ordenado como {0,13,10,7,4,1,14,11,8,5,2,15,12,9,6,3}.
- InvMixColumns.- Al igual que su contraparte esta función recibe un arreglo de 16 bytes expresado como una matriz de 4x4. Reemplaza cada columna de la matriz con el resultado de la multiplicación matricial de la misma columna y una matriz de tamaño 4x4 propia de la función.

Las distintas rondas realizan las operaciones de la siguiente manera:

Ronda 0

AddRoundKey

Rondas 1 a nr – 1

InvShiftRows

InvSubBytes

AddRoundKey

InvMixColumns

Ronda nr

InvShiftRows

InvSubBytes

AddRoundKey

4.1.3 Expansión de llave

Para realizar las múltiples rondas de la encriptación se va utilizando la llave de cifrado, pero al utilizar la misma una y otra vez, le quitaría el propósito a las múltiples rondas. Es por eso que la clave original se expande lo suficiente para que cada ronda se realice con un bloque de la clave diferente. Esta expansión se logra agregando bytes a la llave original generados de manera que sean aparentemente aleatorios.

Para cada ronda se necesita un bloque de la llave extendida de 16 bytes. Por este motivo al expandir la llave se toma en cuenta el tamaño original para abastecer de suficientes bloques a cada ronda.

4.2 3DES

Las recomendaciones para la implementación de este algoritmo se encuentran descritas en la publicación especial del NIST 800-67, en su segunda revisión (Baker y Mouha, 2017). Este algoritmo recibe como entrada 3 llaves de 64 bits cada una o también una sola de 192 bits. En algunos casos se menciona el tamaño de estas llaves como 56 bits, esto debido a que el algoritmo sólo utiliza esa cantidad de bits de las llaves, pero recibe como entrada 64. Ejemplos con valores intermedios del funcionamiento de este algoritmo pueden ser encontrados en la página oficial del NIST.

El algoritmo 3DES basa su funcionamiento en su predecesor DES, ya que se puede resumir en que el algoritmo 3DES es básicamente usar DES 3 veces. Al momento de cifrar, el algoritmo 3DES encripta datos usando DES con una clave *k1* de 64 bits, desencripta el resultado usando DES con otra clave *k2* de 64 bits y por último encripta nuevamente usando DES con una tercera clave *k3* de 64 bits. Para desencriptar se realiza la operación inversa, se desencripta con k3, se encripta con k2 y se desencripta con k1.

DES trabaja por bloques de 64 bits, divide el texto plano en bloques de este tamaño y realiza las mismas operaciones sobre cada uno, tanto al encriptar como al desencriptar y al finalizar une los resultados. El algoritmo DES, y por tanto también 3DES, trabaja a nivel de bits, es decir, sus operaciones más básicas se realizan entre los bits de los bloques.

4.2.1 Cifrado DES

El proceso de encriptado de DES es realizado en 16 rondas. Al bloque inicial de 64 bits se le aplica la permutación inicial (IP), un proceso propio de DES, y el resultado se divide en 2 bloques L_0y R_0. L_0 tiene los primeros 32 bits del bloque permutado y R_0 los últimos 32. A partir de estos valores se aplican los mismos pasos en cualquier ronda n para encontrar L_n y R_n hasta llegar a L_{16} y R_{16}. El cálculo de L_n y R_n es:

$$L_n = R_{n-1}$$

$$R_n = L_{n-1} \oplus F(R_{n-1}, K_n)$$

Ecuación 1: Cálculo de L_n y R_n

Donde F es una función propia de DES, K_n es el resultado de la función KS, igualmente propia de DES, y $\oplus$ representa la operación XOR. La función F se encarga de realizar transformaciones a los bits de los elementos que recibe como entrada para devolver un único arreglo de 32 bits. La función KS se encarga de la expansión de llave explicada más adelante.

Finalmente se concatenan R_{16} y L_{16} en ese orden y se aplica la función IP^{-1}, la función opuesta de IP, dando como resultado el bloque cifrado.

4.2.2 Descifrado DES

El proceso de desencriptación DES es realizar los pasos para el cifrado, pero en orden inverso. A un bloque cifrado de 64 bits se le aplica la función IP. Luego se divide el resultado en 2 bloques R_{16} y L_{16}, teniendo R_{16} los primeros 32 bits y L_{16} los últimos. Posteriormente se realizan las 16 rondas en orden inverso hasta obtener L_0 y R_0. El cálculo de cualquier L_{n-1} y R_{n-1} es el siguiente:

$$R_{n-1} = L_n$$

$$L_{n-1} = R_n \oplus F(L_n, K_n)$$

Ecuación 2: Cálculo de R_{n-1} y L_{n-1}

Una vez obtenidos L_0 y R_0 se los concatena en ese orden y se aplica la función IP^{-1} obteniendo así el texto plano original.

4.2.3 Expansión de llave

La función KS o Programación de Llaves se encarga de expandir la llave que recibe el algoritmo DES generando distintas llaves para cada ronda de encriptación y desencriptación. Cada ronda del algoritmo necesita su propia llave, por lo que la función KS realiza sus operaciones en 16 rondas, después de cada ronda se devuelve un bloque de llaves de 48 bits. La función recibe como entrada la llave original de 64 bits y sus operaciones se limitan a hacer reordenamientos o permutaciones de sus elementos.

4.3 ChaCha20

El algoritmo ChaCha20 fue descrito por primera vez en 2008 como reemplazo del Salsa20 (Bernstein, 2008). Este algoritmo no se encuentra estandarizado y no hay ninguna recomendación para él por parte del NIST. Esto lleva a que existan algunas variantes entre el algoritmo, principalmente en el tamaño del nonce que utiliza. Para este experimento se utilizó la versión descrita por Nir y Langley en 2018, ya que es la más detallada. En el mismo también se describen ejemplos de los resultados esperados del algoritmo.

El algoritmo recibe como entrada una llave de 256 bits y un nonce de 96 bits. Al ser un cifrador de flujo utiliza estos datos para generar un flujo de llaves del mismo tamaño que los datos a encriptar/desencriptar. Una vez generado el flujo lo une con los datos mediante la operación XOR para obtener el resultado. Como la operación XOR es reversible el proceso de encriptación y desencriptación es exactamente el mismo.

El algoritmo trabaja con unidades de 32 bits o palabras. Esto quiere decir que cada 32 bits se considera un solo elemento y se realizan las operaciones tomándose como uno.

4.3.1 Flujo de llaves

El flujo de llaves se genera por bloques de 512 bits, o 16 palabras de 32 bits, hasta alcanzar el tamaño adecuado. Para cada bloque que se genera se cambia un valor inicial, el contador, para que los bloques sean distintos entre sí.

Es importante notar que el algoritmo hace gran uso de la notación little-endian. Esta notación se caracteriza por invertir el bytes en cada conjunto de bytes, pero no el orden del conjunto como tal. Por ejemplo, el conjunto en notación hexadecimal 0x123456, 0x7890ab, 0xcdef12 se expresaría de la siguiente manera en notación little-endian: 0x563412, 0xab9078, 0x12efcd.

Los datos que se necesitan para inicializar un bloque son:

- Una constante de 128 bits, cuyo valor es el código ASCII de la cadena 'expand 32-byte k', separada en 4 palabras en formato little-endian.
- La llave de 256 bits dividida en 8 palabras en formato little-endian cada una
- Un contador de bloque de 32 bits que va aumentando con cada bloque, inicia en 1
- Un nonce de 96 bits dividido en 3 palabras en formato little-endian cada una

Una vez unidos estos valores se tiene el bloque inicial, que debe pasar por 20 rondas de encriptación para obtener un bloque. Por último después de que el bloque ha pasado por las 20 rondas se le suma el bloque original. Esta operación se realiza con módulo 2^32 para conservar los elementos de tamaño 32 bits. Esto da como resultado uno de los bloques del flujo de llaves.

4.3.2 Quarter round

Cada ronda del algoritmo consiste en aplicar la función Quarter round 4 veces al bloque inicial. La función Quarter round solo modifica 4 valores indicados del bloque de 16 palabras, por eso se realiza 4 veces por ronda. Los valores seleccionados para la función se intercalan entre rondas, primero una ronda de columnas, luego una ronda de diagonales y se repite hasta haber realizado 20 rondas. Los valores que recibe la función en cada ronda son:

Ronda columnas

Quarter round(0, 4, 8, 12)

Quarter round(1, 5, 9, 13)

Quarter round(2, 6, 10, 14)

Quarter round(3, 7, 11, 15)

Ronda diagonales

Quarter round(0, 5, 10, 15)

Quarter round(1, 6, 11, 12)

Quarter round(2, 7, 8, 13)

Quarter round(3, 4, 9, 14)

Suponiendo que los elementos del bloque están numerados del 0 al 15.

5 Configuración del experimento

Esta sección describe el conjunto de datos utilizado, la implementación de los algoritmos a evaluar, las métricas recolectadas y el equipo que se utilizó para el experimento.

5.1 Conjunto de datos

Para los datos del experimento se utilizaron tres tipos de archivos: imagen, audio y texto. Cada uno de estos tipos se obtuvo de manera que cumplieran con una distribución de tamaño lo más uniforme posible para poder observar la variación de las mediciones a los algoritmos según variaba el tamaño del archivo usado.

- Textos. Se generaron 50 textos en formato texto plano usando la página lipsum.com. Esta página genera textos de ejemplo en latin de manera semi aleatoria. Los archivos varían su tamaño de manera uniforme entre 20 y 1000 bytes.
- Audios. Se descargó la aplicación gratuita Audacity, la cual permite grabar y editar audios, para generar 50 archivos mp3. Usando este programa se grabaron y modificaron los archivos para que varien de la manera más uniforme posible entre 10000 y 500000 bytes.
- Imágenes. Para recolectar las imágenes se utilizó el repositorio de fotografías gratuito sp.depositphotos.com. De esta página se consiguieron 50 imágenes jpg. Estos archivos varían entre 100000 y 300000 bytes.

5.2 Implementación

Se implementaron los algoritmos acorde a sus especificaciones. Se decidió implementar cada algoritmo para tener un mejor control de las variables del experimento, por ejemplo, cerciorarse de que el algoritmo AES haga uso de las tablas precalculadas S-box en lugar de realizar las operaciones matemáticas manualmente. Otro ejemplo es controlar que solo se encripte un bloque a la vez para los algoritmos AES y 3DES, ya que esa es parte de las condiciones determinadas para el experimento. Todo lo mencionado anteriormente resulta difícil de controlar o determinar con las librerías existentes que implementan estos algoritmos.

Para la implementación de los distintos algoritmos se utilizó el lenguaje Python en su versión 3.10.0. Cada algoritmo se implementó siguiendo sus respectivas especificaciones. Para verificar el correcto funcionamiento de las implementaciones se realizaron pruebas con los datos de muestra mencionados en la descripción de cada algoritmo. Los algoritmos AES y 3DES se programaron para trabajar con el

modo de operación ECB. Se hizo uso en gran medida de la biblioteca NumPy, en su versión 1.21.5, para facilitar el manejo de arreglos de datos sin comprometer la eficiencia del código. Se implementó el algoritmo AES para trabajar con sus 3 versiones para poder medirlas todas.

El código desarrollado se encuentra disponible en el siguiente enlace (https://github.com/leoncente/Estudio-comparativo-de-los-algoritmos-de-criptografia-simetrica-AES-3DES-y-ChaCha20)

5.3 Medición de resultados

Para facilitar la realización del experimento y la medición de datos se implementó una interfaz de consola simple en lenguaje Python. Dicha interfaz permite seleccionar el tipo de medición que se desea y ejecuta todas las pruebas correspondientes a todos los archivos con cada algoritmo.

- Tiempo. Para medir el tiempo que le tomaba a cada algoritmo encriptar o desencriptar se utilizó la biblioteca Time que permite guardar marcas de tiempo. En cada experimento se sacaban 2 marcas de tiempo, una al iniciar y otra al finalizar, restando estos 2 valores se obtuvo la medición de tiempo en segundos.
- Velocidad. Esta unidad se obtuvo en función del tiempo y el peso de cada archivo. En este caso se entiende por velocidad lo siguiente:

$$Velocidad = \frac{Tamaño\ del\ archivo\ (bytes)}{Tiempo\ de\ encriptación/desencriptación\ (s)} \qquad \text{Ec 3: Velocidad}$$

- CPU. Usando la biblioteca threading y Psutil se creaba un hilo de programa que se ejecuta a la par que el proceso de encriptación, cuyo único propósito era medir el porcentaje de consumo de CPU cada cierto tiempo. La máxima medición obtenida durante el proceso era tomada como el resultado.
- Energía. Para medir el consumo de energía se utilizó un wattímetro digital KEWEISI modelo KWS-AC300-100A. Con el wattímetro conectado al cargador de la laptop, se sacó una potencia media de 16.00 Watts en estado normal y durante la encriptación/desencriptación se media el mayor valor alcanzado. Se restaban ambos resultados y se obtenía el consumo del algoritmo en Watts. Todo este cálculo se realizó de manera manual ya que el wattímetro solo devuelve los resultados en su pantalla digital.

Se realizaron múltiples pruebas sobre cada uno de los archivos del conjunto de datos y se realizó un promedio de los resultados obtenidos.

5.4 Equipo de pruebas

Para el experimento se utilizó una computadora portátil con las siguientes características:

- Modelo:Alienware 15 R2
- S.O.: Windows 10 home (64 bits)
- Procesador: Intel® Core™ i7-6700HQ CPU @ 2.60GHz
- RAM: 16,0 GB

Para tener los resultados más fiables posible se aseguraron las siguientes condiciones en el equipo:

- Desconexión de todos los periféricos para minimizar la variación del consumo de energía
- Se cerraron todas las aplicaciones y apagaron las funciones que no fueran del sistema operativo para no alterar la medida del consumo de recursos
- Se mantuvo el cargador conectado y la batería cargada completamente en todo momento

6 Resultados

Después de realizar las pruebas los resultados obtenidos fueron los siguientes:

6.1 Tiempo - Velocidad

Los resultados de tiempo de encriptación y desencriptación se convirtieron a velocidad para poder sacar un promedio de todas las velocidades. El resultado se muestra en la figura 1.

Como se puede observar el algoritmo ChaCha20 es el más veloz por mucho, hasta 6 veces más rápido que el siguiente más rápido. También se obtuvo que AES en su versión de 128 bits es casi 1,4 veces más rápido que su versión de 256 bits tanto al encriptar como al desencriptar.

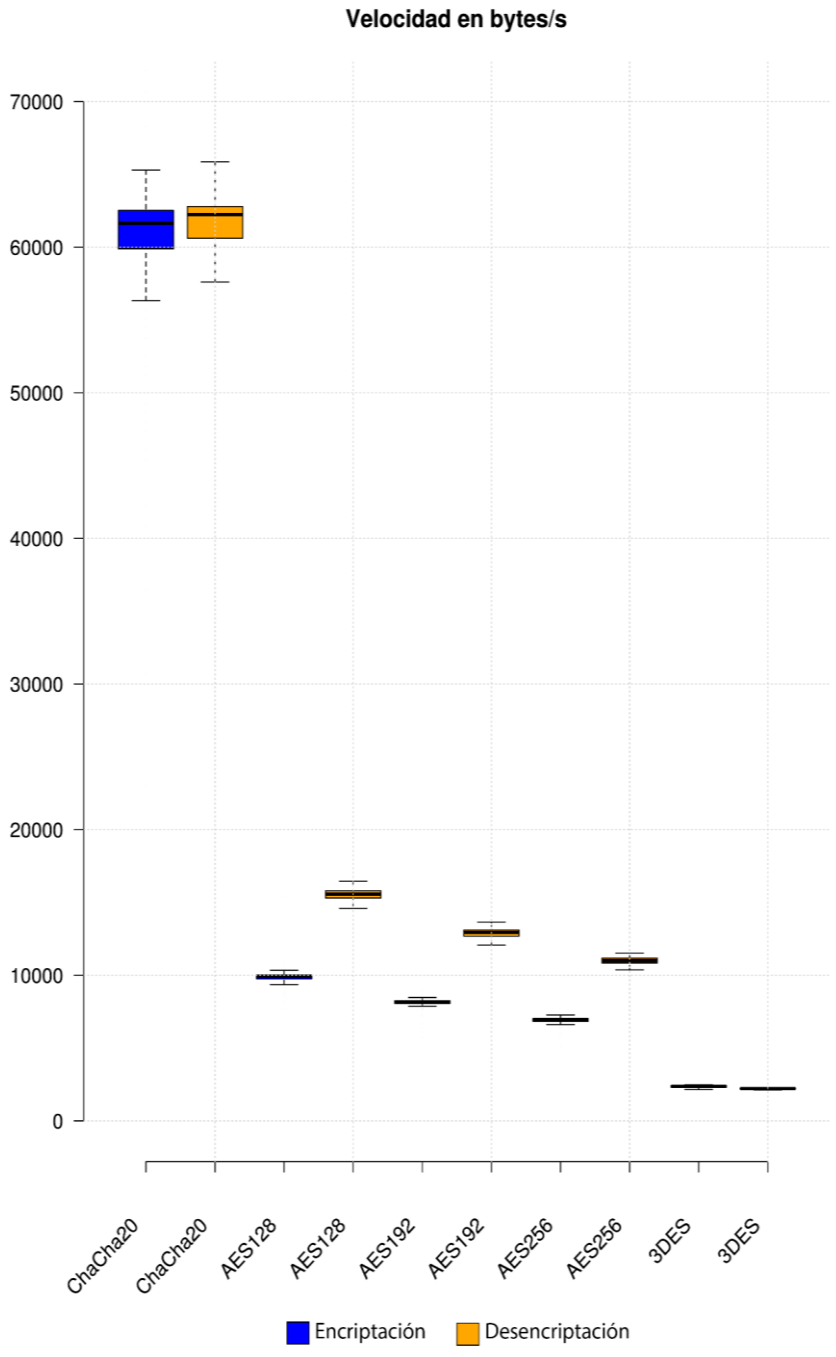

Figura 1: Velocidad de encriptación/desencriptación promedio en bytes/s

6.2 Consumo de CPU

Los resultados de la medición del porcentaje de consumo de CPU se muestran en las figuras 2 y 3.

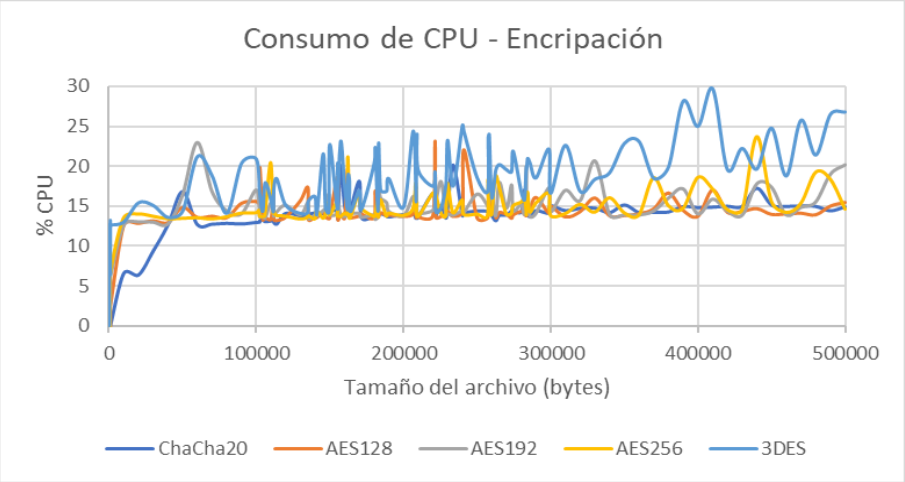

Figura 2: Consumo de CPU en función del peso durante encriptación.

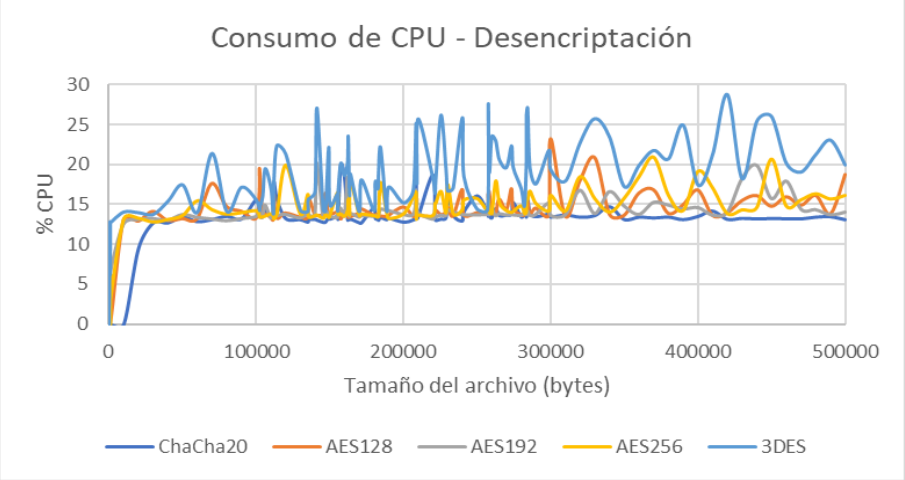

Figura 3: Consumo de CPU en función del peso durante desencriptación.

En ambos casos el algoritmo 3DES es el que consume más recursos computacionales. El algoritmo AES no muestra ninguna diferencia sustancial entre

sus distintas versiones, las cuales tienen un consumo muy similar al algoritmo ChaCha20, pero presentan mayor volatilidad o variación en sus valores..

6.3 Consumo de energía

Los resultados para el consumo de energía se muestran en las figuras 4 y 5.

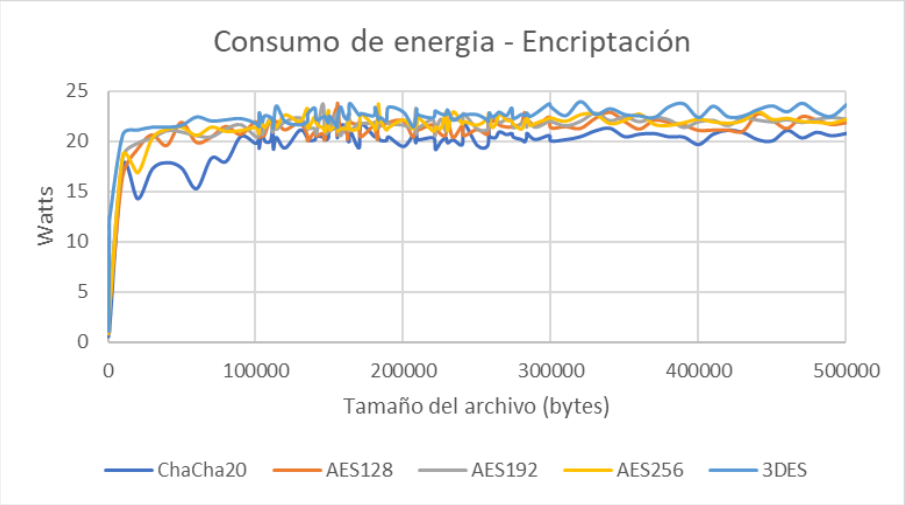

Figura 4: Watts en función del peso durante encriptación

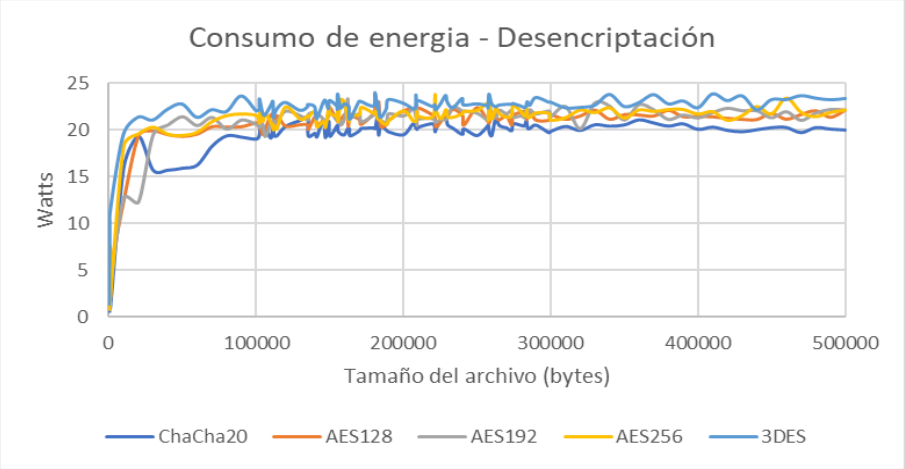

Figura 5: Watts en función del peso durante desencriptación

En el caso de la energía consumida el algoritmo ChaCha20 es el que presenta valores más bajos, pero no existe una diferencia sustancial entre todos los algoritmos, siendo las discrepancias menores a 5 Watts en la mayoría de los casos.

7 Discusión y trabajos futuros

Para el desarrollo de este experimento solo se implementaron los algoritmos AES y 3DES en modo ECB ya que es el modo que menos operaciones requiere. Todas las implementaciones son únicamente a nivel de software, pero debido al gran uso de AES se ha desarrollado hardware especializado para optimizar su proceso. Es importante notar estos puntos si se desea realizar estudios similares o extender este.

Es importante notar que los algoritmos se implementaron para encriptar/desencriptar un bloque a la vez, ya que requeriría otro estudio determinar cuál es la cantidad óptima de bloques para encriptar en paralelo. Esto podría afectar los resultados en comparación con estudios similares que implementen una configuración distinta.

Al momento de escoger un algoritmo de encriptación es importante tomar en cuenta tanto la eficiencia como la seguridad que provee el mismo. En este estudio se determinó el rendimiento de los algoritmos seleccionados, pero para determinar cuál de ellos es el más adecuado actualmente se debe considerar la seguridad de cada uno.

8 Conclusiones

El objetivo de este experimento era determinar el algoritmo con mejor rendimiento entre AES, 3DES y ChaCha20. Se recabaron 150 archivos de distintos formatos y tamaños para probar cada algoritmo. Se aseguraron ciertas condiciones para que los resultados obtenidos fueran lo más fiables posibles, como la desconección de periféricos o la terminación de programas no esenciales en el computador.

Comparando los resultados de tiempo, el algoritmo ChaCha20 es el más rápido. Se pudo observar que este algoritmo es cerca de 6 veces más rápido que AES en cualquiera de sus tres versiones. Esta diferencia de velocidades es congruente con la descripción de los algoritmos, ya que, aunque ChaCha20 no es un algoritmo por bloques, genera su flujo de llaves por bloques de 512 bits, esto es 4 veces más grande que los bloques de AES y 8 más que 3DES. Con esto se puede notar la ventaja de ChaCha20 al realizar su proceso de encriptación y desencriptación 4 veces menos que AES.

El algoritmo ChaCha20 es en promedio el que menor consumo de CPU tiene, pero es seguido muy de cerca por AES. Este último iguala al ChaCha20 en varios puntos, pero presenta mayor cantidad de picos o fluctuaciones, lo que sube su promedio de consumo.

Para el consumo eléctrico el algoritmo ChaCha20 nuevamente demostró tener los mejores resultados, pero en esta ocasión se pudo notar que los valores entre todos los algoritmos eran muy similares, por lo que la ventaja del ChaCha20 es en promedio menor a unos pocos Watts.

Considerando todo lo anterior mencionado se concluyó que el algoritmo más eficiente de los 3 escogidos en una implementación de software es el ChaCha20. Se llegó a esta respuesta basándose principalmente en la gran diferencia de velocidad que tiene este algoritmo respecto a los otros, notando que en las otras mediciones no presentó una ventaja tan abrumadora. También se concluyó que el algoritmo 3DES es el menos eficiente, ya que presenta los peores valores en todos los campos medidos.

9 Bibliografía

[1] Abd Elminaam, D. S., Abdual Kader, H. M., y Hadhoud, M. M. (2010). Evaluating The Performance of Symmetric Encryption Algorithms. International Journal of Information and Network Security (IJINS), 10(3), 213-219.

[2] Al Tamimi, A.K. (2006). Performance Analysis of Data Encryption Algorithms

[3] Alenezi, M. N., Alabdulrazzaq, H., y Mohammad, N. Q. (2020). Symmetric encryption algorithms: Review and evaluation study. International Journal of Communication Networks and Information Security, 12(2), 256-272.

[4] Baker, E. y Mouha, N. (2017) Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher. NIST Special Publication (SP) 800-67, Rev. 2.

[5] Bernstein, D. (2008). ChaCha, a variant of Salsa20. Workshop record of SASC (Vol. 8, No. 1, pp. 3-5)

[6] Dworkin, M., Barker, E., Nechvatal, J., Foti, J., Bassham, L., Roback, E. y Dray, J. (2001), Advanced Encryption Standard (AES), Federal Inf. Process. Stds. (NIST FIPS), National Institute of Standards and Technology, Gaithersburg, MD

[7] Dworkin, M. (2001), Recommendation for Block Cipher Modes of Operation: Methods and Techniques. NIST Special Publication (SP) 800-34A.

[8] Migga, J. (2015). Guide to Computer Network Security (Computer Communications and Networks) (3rd ed.).

[9] Nir, Y. y Langley, A. (2018). ChaCha20 and Poly1305 for IETF Protocols. RFC 8439, DOI 10.17487/RFC8439. Obtenido de: https://www.rfc-editor.org/info/rfc8439

[10] Tanenbaum, A. S. y Wetherall, D. J. (2012). Redes De Computadoras (5.a ed.). Pearson Educación

[11] Whatsapp, (2021). WhatsApp Encryption Overview: Technical white paper. Obtenido de: https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf