ENCRIPTACIÓN Y COMPRESIÓN DE ARCHIVOS EN UN MODELO CLIENTE-SERVIDOR.

Víctor Tomás Tomás Mariano[1], Felipe de Jesús Núñez Cardenas[1], Raúl Hernández Palacios[1]

[1]Lic. En Sistemas Computacionales, Universidad Autónoma del Estado de Hidalgo - Escuela Superior Huejutla. Corredor Industrial S/N, Parque de Poblamiento, Huejutla de Reyes, Hidalgo. CP. 43000. Tel. 771-7172000, ext. 5880,5881, e-mail: victor_tomasm@hotmail.com, felipe.huejutla@hotmail.com

RESUMEN

Se presenta un sistema modelo cliente-servidor, que implementa el algoritmo RSA para encriptación de clave pública y el algoritmo de Huffman para la compresión de archivos de texto plano, imágenes, video, y archivos generados por aplicaciones: Word, Excel, Power Point. El sistema se desarrolla completamente en java, se hace uso del protocolo de comunicación TCP/IP para el envío de la los paquetes, sockets y procesamiento multihilo o threads.

Palabras Clave: Compresión, Encriptación, Sockets, TCP/IP, Threads.

INTRODUCCION.

Con el incremento en el "*tamaño de bytes*" en información digital como videos, música, archivos de texto o binarios, y el surgimiento de dispositivos de almacenamiento cada vez con más capacidad, es necesario diseñar sistemas capaces de ofrecer seguridad e integridad en los datos, la conservación de la confidencialidad de estos es muy importante para las organizaciones, más cuando se comparten con otros.

Resultado de esta necesidad, los algoritmos de encriptación otorgan la posibilidad de alterar la información y ocultar a terceras personas los datos originales, es por ello que el sistema RSA se implementa con tal fin. Así mismo, el incremento en el tamaño de los archivos, requiere de la necesidad de reducción de los contenidos digitales, más cuando estos se transmiten en canales inseguros, de ahí que se implementa un sistema de compresión de datos sin pérdida. Otro factor a considerar es que el intercambio de información debe ser lo más seguro posible y el envío de los paquetes de información lleguen a su destino de una forma completa e integra [1].

ROC&C'2012 / CP-06 PONENCIA RECOMENDADA POR EL COMITÉ DE COMPUTACIÓN DEL IEEE SECCIÓN MÉXICO Y PRESENTADA EN LA REUNIÓN INTERNACIONAL DE OTOÑO, ROC&C'2012, ACAPULCO, GRO., DEL 11 AL 15 DE NOVIEMBRE DEL 2012.

DESARROLLO.

RSA es un algoritmo de clave pública creado en 1978 por Rivest, Shamir y Adlman, y es uno de los algoritmos criptográficos asimétricos más conocido y usado [7] [8].

RSA se basa en el hecho matemático de tener que realizar la factorización de números muy grandes. Para factorizar un número lo más lógico consiste en empezar a dividir sucesivamente éste entre 2, entre 3, entre 5, entre 7, y así sucesivamente, buscando que el resultado de la división sea exacto, es decir, el residuo igual a 0, con lo que se encuentra un divisor del número.

Ahora bien, si el número considerado es un número primo (el que sólo es divisible por 1 y por él mismo), se tiene, que para factorizarlo hay que empezar por 2, 3,... hasta llegar a él mismo, ya que por ser primo ninguno de los números anteriores es divisor suyo. Y si el número primo es lo suficientemente grande, el proceso de factorización es complicado y consume mucho tiempo.

El Algoritmo RSA, para su ejecución es dividido en 3 etapas:

Etapa 1: Generación de claves

Basado en la exponenciación modular de exponente y módulo fijos, el sistema RSA crea sus claves de la siguiente forma:

1. Se buscan dos números primos lo suficientemente grandes: **p** y **q** (de entre 100 y 300 dígitos).
2. Se obtienen los números **n** = p * q y **Ø** = (p-1) * (q-1).
3. Se busca un número **e** tal que no tenga múltiplos comunes con Ø.
4. Se calcula **d** = e^{-1} mod Ø, con mod = resto de la división de números enteros.

De los valores obtenidos, **n es la clave pública** y **d es la clave privada**. Los números p, q y Ø se destruyen. También se hace público el número **e**, el cual es necesario para alimentar el algoritmo.

El cálculo de estas claves se realiza en secreto en la máquina en la que se va a guardar la clave privada, y una vez generada ésta conviene protegerla mediante un algoritmo criptográfico simétrico.

En cuanto a las longitudes de claves, el sistema RSA permite longitudes variables, siendo aconsejable actualmente el uso de claves de no menos de 1024 bits [7].

RSA basa su confiabilidad en ser una función computacionalmente segura, ya que si bien realizar la exponenciación modular es fácil, su operación inversa, la extracción de raíces de módulo **Ø** no es factible a menos que se conozca la factorización de **e**, clave privada del sistema[7][8].

Etapa 2: Encriptación del mensaje

Una vez generadas las claves se procede a la encriptación del mensaje:

1. La entidad emisora toma la clave pública (n, e).
2. El texto plano ***m*** se representa como un entero.
3. Cada entero del texto plano es cifrado con la ecuación (1):
   $c = m^e \pmod n$ (1)
4. Posteriormente se almacena cada valor cifrado del texto plano m, obteniendo así el mensaje final cifrado.

Etapa 3: Decriptación del mensaje

Cuando la entidad receptora tiene en su poder el mensaje m cifrado sigue los siguientes pasos para obtener el mensaje original:

1. Esta entidad toma la clave privada d y la clave pública n.
2. Cada entero del texto cifrado es desencriptado con la ecuación (2):
   $m = c^d \pmod n$ (2)

RSA es el más conocido y usado de los sistemas de clave pública, y también el más rápido de ellos. Presenta todas las ventajas de los sistemas asimétricos, incluyendo la firma digital, aunque resulta más útil a la hora de implementar la confidencialidad el uso de sistemas simétricos, por ser más rápidos. Se suele usar también en los sistemas mixtos para encriptar y enviar la clave simétrica que se usa en la comunicación cifrada [2].

COMPRESION DE DATOS.

En los últimos años se ha visto un aumento tanto de la capacidad de almacenamiento de los ordenadores como de la velocidad de procesamiento de éstos. Aunado a esto una baja en los precios de memoria principal y secundaria así como también un aumento de velocidad de estos dispositivos. Esto nos hace preguntarnos ¿para qué la compresión?. Sin embargo, el auge que últimamente han tenido las redes de ordenadores hace que cada vez más usuarios pidan más prestaciones a la red sobre la que están conectados. Prestaciones que, como siempre, están por encima de las posibilidades reales. Cuando hablamos de posibilidades nos referimos principalmente a la velocidad de transferencia de datos.

La compresión de datos es un conjunto de operaciones que se realizan sobre la información de una fuente con la finalidad de conseguir que ocupe menos espacio de almacenamiento, lo cual se logra eliminando la redundancia de información que existe en los datos fuente[3][6].

Los métodos de compresión se valoran por:

- Tiempo de ejecución .
- Espacio requerido durante el proceso.
- Factor de compresión.

Existen 2 tipos de compresión de datos:

1. Compresión de datos SIN pérdida (*lossless*), en la cual los datos a comprimir pasan por un algoritmo de compresión- descompresión y al final se obtienen los datos intactos, tal y como se tenían antes de comprimirlos [3].

   Ejemplos de algoritmos para comprimir datos sin pérdidas son el RLE, LZ77, algoritmos estadísticos como lo es el algoritmo Shannon-Fano y el algoritmo de Huffman [2] [5] [6], que es el objeto de estudio en este proyecto.

2. Compresión de datos con perdida (losy), donde los datos que se desean comprimir sufren alguna pérdida de información la cual no es muy perceptible. Este tipo de compresión es aplicado sobre todo a imágenes y sonido donde las pérdidas de datos son muy imperceptibles para el ojo y el oído humano respectivamente [2][5]. Además de que ésta pérdida es insignificante en comparación con la ganancia en compresión.

Como ejemplos de esta categoría tenemos los algoritmos estándar JPEG y estándar MPEG.

ALGORITMO HUFFMAN.

Este algoritmo se basa en asignar códigos de distinta longitud de bits a cada uno de los caracteres de un archivo. Si se asignan códigos más cortos a los caracteres que aparecen más a menudo se consigue una compresión del archivo[5].

La compresión es mayor cuando la variedad de caracteres diferentes que aparecen es menor. Por ejemplo: si el texto se compone únicamente de números o mayúsculas, se conseguirá una compresión mayor.

Para recuperar el archivo original es necesario conocer el código asignado a cada carácter, así como su longitud en bits, si ésta información se emite, y el receptor del fichero la conoce, podrá recuperar la información original[5][6]. De este modo es posible utilizar el algoritmo para encriptar archivos. De lo contrario no podrá descomprimir la información.

Para realizar el algoritmo de Huffman se siguen los pasos enumerados a continuación:

1. Calcular las frecuencias de aparición de los símbolos involucrados.
2. En un árbol binario alinear los símbolos por frecuencias descendentes en nodos.
3. Ligar 2 nodos de menor frecuencia en un nuevo nodo de tal manera que la frecuencia sea una suma de frecuencias de los 2 símbolos.
4. Ir al paso 2 hasta generar un solo nodo de tal manera que la frecuencia sea 1.
5. Trazar el árbol de la codificación desde una raíz (el nodo generado con la probabilidad 1) a los nodos del origen y asigna a cada rama derecha 1 y a cada rama izquierda 0

Supongamos que tenemos la siguiente cadena de caracteres:

"badaaaaabaadaadaabacbcaaaaaababdbbcbcbcd"

Calculando las probabilidades de aparición tenemos que:

a =0.5 b =0.25 c =0.125 d =0.125

Se alinean los símbolos por frecuencias descendentes en nodos como se muestra en la figura 1(a). En un árbol procedemos a ligar los nodos de menor frecuencia en un nuevo nodo y sumar sus frecuencias, ya que esa suma será el valor del nuevo nodo. Observemos en la figura 1(a) que los nodos c, d son de frecuencia menor. Sumando ambas frecuencias obtenemos una suma la cual representa el nodo padre, ver figura 1(b). En 1(c), el nodo generado en 1(b) se convierte en hijo al sumar el valor del nodo con el otro nodo que sigue en probabilidad menor, se obtiene otro nodo padre. En 1(c) se observa que se ha construido una ramificación en la cual unicamente falta por unir el nodo "a" y la ramificación formada anteriormente, que es realizado en 1(d). Se verifica que todos los nodos suman 1. Una vez terminado el procedimiento se tiene una ramificación completa y se procede a asignar 0 y 1 a las ramas izquierda y derecha respectivamente.

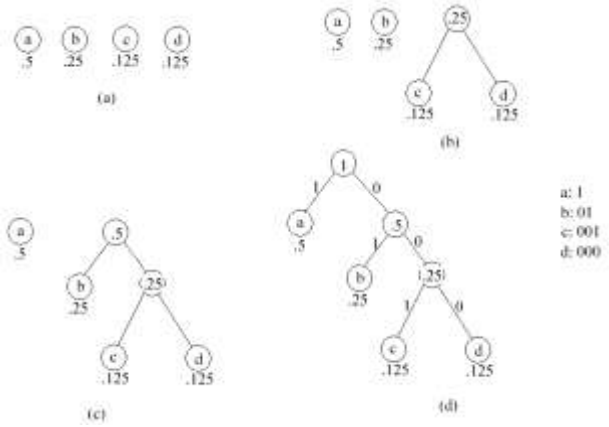

Figura 1: Unión de nodos con menor frecuencia y generación del proceso de codificación de los caracteres.

SERVIDOR-CLIENTE: SOCKETS.

Los sockets son puntos o mecanismos de comunicación entre procesos que permiten que un proceso emita o reciba información con otro proceso, incluso si estos se están ejecutando en distintas máquinas[1][4].

La comunicación entre procesos a través de sockets se basa en la filosofía cliente/servidor: un proceso en esta comunicación actuará de proceso servidor creando un socket cuyas características (dirección IP, puerto) debe conocer el proceso cliente, el cual podrá comunicarse con el proceso servidor a través de la conexión con dicho socket. A continuación se mencionan los tipos de sockets a considerar:

Sockets Stream: Utilizan el protocolo de comunicación TCP (Transport Control Protocol, protocolo de control de transporte), por lo que reciben el

nombre de sockets orientados a conexión, ya que para establecer una comunicación utilizando el protocolo TCP, hay que establecer en primer lugar una conexión entre un par de sockets. Mientras uno de los sockets atiende peticiones de conexión (servidor), el otro solicita una conexión (cliente). Una vez que los sockets estén conectados, se pueden utilizar para transmitir datos en ambas direcciones[1][4]. Este tipo de sockets responde exactamente a los dos esquemas gráficos que se muestran en las figuras 2 y 3.

Sockets Datagrama: Utilizan el protocolo UDP (User Datagram Protocol, protocolo de datagrama de usuario). Son más eficientes que TCP, pero está no garantiza la fiabilidad. Los datos se envían y reciben en paquetes (datagramas), cuya entrega no está garantizada. Los paquetes pueden ser duplicados, perdidos o llegar en un orden diferente al que se envió. Proporcionan un servicio de transporte sin conexión, esto quiere decir que cada vez que se envían datagramas es necesario enviar el descriptor del socket local y la dirección del socket que debe recibir el datagrama[3][7].

Sockets Stream y Datagrama: El tipo de sockets que se debe usar, UDP o TCP depende del tipo de aplicación cliente/servidor que se quiera escribir. Por las características y para el propósito de este proyecto, se usa los Sockets Stream.

El mecanismo de comunicación vía sockets tiene los siguientes pasos :

1. El proceso servidor crea un socket y espera a que se conecte un cliente.
2. El proceso cliente crea un socket con los datos del servidor (IP y puerto).
3. El proceso cliente realiza una petición al servidor.
4. El proceso servidor recibe la petición del cliente y responde.
5. El proceso cliente lee la respuesta de servidor y termina o vuelve al paso 3.

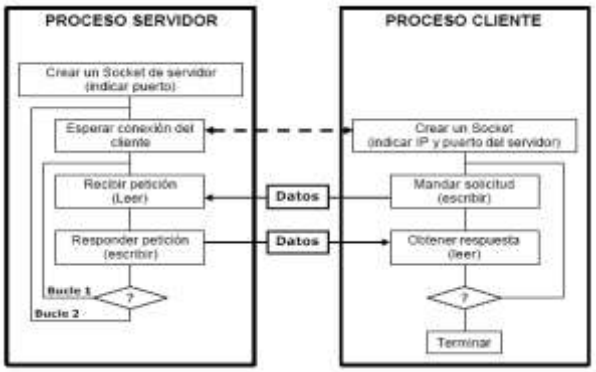

Figura 2. Comunicación Servidor-Cliente vía sockets

De forma gráfica, una comunicación básica vía sockets entre un cliente y un servidor, gráficamente, se podría representar como en la figura 2.

En la figura 2 se muestra el servidor, al establecer una conexión con el cliente, entra en un bucle (bucle 1) en el que empieza a mandar y recibir datos a través del socket. Mientras el servidor se encuentre en este bucle no podrá atender peticiones de nuevos clientes, sólo cuando el cliente al que se está atendiendo actualmente decida terminar la comunicación, el servidor saldrá del bucle 1 y volverá al paso en el que queda a la espera de iniciar una conexión con otro cliente (bucle 2). Es decir, este servidor no es capaz de atender a dos clientes simultáneamente.

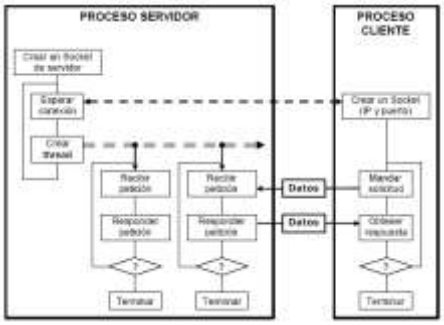

Figura 3. Comunicación Servidor-Cliente con threads.

SERVIDOR-CLIENTE: THREADS.

Para diseñar un servidor que pueda atender a varios clientes a la vez hay que introducir el concepto de hilo de ejecución o 'thread'. Un thread es la unidad básica de ejecución. Cualquier programa que se ejecute consta de, al menos un thread. Ahora bien, es posible diseñar programas con varios threads independientes, es decir, con varias unidades de ejecución, que podrían estar ejecutando tareas relacionadas o no entre si [1][4].

Un servidor capaz de atender peticiones de varios clientes deberá ser capaz de crear un thread por cada cliente que solicite servicio. De esta forma, el esquema de la figura 2, se modifica para obtener tal como se muestra en la figura 3.

Como el protocolo TCP está orientado a conexión, se establece esta conexión entre los dos sockets, lo que implica un cierto tiempo empleado en el establecimiento de la conexión, que no existe en UDP.

RESULTADOS.

Las pruebas se realizaron con archivos de distintos tamaños, entre estos están: Word, Power Point, Excel, Música y Video, obteniendo resultados satisfactorios en un 100% en la recuperación de la información original.

El fichero encriptado aumentó en ocasiones hasta en un 300% del tamaño original del archivo, esto se debe al cálculo de valores enteros muy grandes al aplicar el RSA.

En la tabla 1 se visualizan los tamaños de los archivos obtenidos, la compresión se realiza con respecto al archivo encriptado, los archivos resultantes al aplicar el algoritmo de Huffman: tabla de frecuentas y el archivo comprimido con los símbolos, influyen en la cantidad de datos que son enviados por el canal.

Tabla 1: Tamaños de archivos encriptados y comprimidos, Powert Point.

Archivo Original (hash.ppt)	131KB
Archivo encriptado (hash.pptenc)	525KB
Archivo Comprimido tabla de frecuencias (hash.pptenc.frec)	2KB
Archivo Comprimido Huffman (hash.pptenc.huffman)	346 KB
Ratio de compresión con respecto al archivo encriptado	34%

En archivos planos o de solo texto, se refleja un mayor porcentaje de compresión, a diferencia de aquellos que contienen imágenes, graficas en su contenido, e incluso los que almacenan música o video. Sin embargo, el propósito fue validar el proceso de recuperación e integridad de los datos al enviarlos por un canal inseguro o en red.

La recepción y envío de información con el protocolo TCP y sockets en java, en un 100% se cumplió en la integridad de la información.

CONCLUSIÓNES

El envió y recepción de archivos no es una tarea segura. Existen varios intrusos que pueden hacerse pasar por la entidad receptora a través de un canal inseguro, lo cual puede causar temor al enviar información valiosa en una red local o peor aun: Internet. Sin embargo en este trabajo se ha visto una manera de solucionar este problema al encriptar los datos de interés y posteriormente comprimirlos utilizando algoritmos muy elaborados como lo son RSA y Huffman respectivamente.

El algoritmo RSA como resultado de la encriptación de los datos estos tienden a incrementar hasta en un 300% el tamaño original del archivo, para enfrentar este problema, usamos el algoritmo de Huffman para comprimir el archivo resultante de la encriptación, lo que reduce el tamaño del archivo para su envió.

TCP es un protocolo de comunicación en red que ofrece buenos resultados para el envío y recepción de la información.

BIBLIOGRAFÍA

[1] Ahuja J.J, Client Server Application in Java, Thesis Master of Science in Computer Science, PACE UNIVERSYTI, 1997.

[2] Bruce S., Applied Cryptography: Protocols, Algorithms, and Source Code in C, Second Edition, Wiley, 2nd edition, 1996.

[3] David S., Data Compression: The Complete Reference, Fourth Edition, Springer, 2008.

[4] Deitel and Deitel, Cómo Programar en Java, Pearson Prentice Hall, 2010.

[5] Jeffrey S, V., Dynamic Huffman Coding, Brown University, ACM Transaction on Mathematical Software. Vol. 15, No. 2, June 1989. Pages 158-167.

[6] Khalid Sayood, Introduction to Data Compression, Third Edition. Elsierver, 2006.

[7] Niels F., Bruce S., Tadayoshi K., Cryptography Engineering: Design Principles and Practical Applications, Wiley, 1 edition, 2010.

[8] Standar Specifications for Public-Key Cryptography, IEEE Std 1363-2000, 2000.

CURRICULUM

Víctor Tomas Tomás Mariano es Maestro en Ciencias Computacionales egresado de la Universidad Autónoma del Estado de Hidalgo (UAEH). Profesor investigador titular, con perfil PROMEP, incorporado a la Licenciatura en Sistemas Computacionales de la Escuela Superior Huejutla UAEH, en Huejutla de Reyes, Hidalgo. Publicaciones recientes: *Algoritmo para la Generación de Laberintos de Conexión Múltiple en 2D*.

Vigesimosegunda Reunión Internacional de Otoño, de Comunicaciones, Computación, Electrónica, Automatización, Robótica y Exposición Industrial ROC&C2011. La Conjunción de las Tecnologías Digitales en las Redes Inteligentes. IEEE Sección México. Acapulco, Guerrero, México. *Propuesta para la generación de laberintos ampliados en 2D.* Simposio Iberoamericano Multidisciplinario de Ciencias e Ingeniería SIMCI 2011. Zempoala, Hidalgo.

Felipe de Jesús Núñez Cárdenas es Maestro en Ciencias de la Administración con Especialidad en Informática egresado del Centro de posgrado en Administración e Informática A. C. (CPAI). Profesor Investigador Asociado, incorporado a la Licenciatura en Sistemas Computacionales de la Escuela Superior Huejutla UAEH, en Huejutla de Reyes, Hidalgo, desde el año 2011. Su interés en áreas de investigación incluye: Ingeniería de Software, Sistemas de Información, Base de Datos, Minería de Datos, ha publicado:
Portal Integral de Educación Superior en el XIV Congreso Internacional Sobre educación electrónica, virtual y a distancia. TELEEDU 2007 (Bogotá Colombia).

Raúl Hernández Palacios, Maestro en Ingeniería de Computadores y Redes, egresado de la Universidad de Granada, España en 2007. Actualmente como Profesor Investigador y Perfil Promep de la Escuela Superior de Huejutla de la Universidad Autónoma del Estado de Hidalgo, incorporado a la Licenciatura de Sistemas Computacionales de la misma Universidad. Interés en área de investigación de Sistemas Distribuidos; sistemas de archivos en red; optimización de comunicaciones eficientes en cluster de computadoras con mecanismos como multicast, Socket Direct Protocol (SDP), Remote Direct Memory Access (RDMA); alta disponibilidad en sistemas cluster a nivel de red, servidor y almacenamiento (iSCSI).