UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE ESTUDIOS SUPERIORES ACATLAN

TECNICAS CRIPTOGRAFICAS

T E S I S

QUE PARA OBTENER EL TITULO DE:

LIC. MATEMATICAS APLICADAS Y COMPUTACION

P R E S E N T A :

GUILLERMO GOMEZ VILLA

ASESOR: **LIC. ANDRES BALDERAS HERNANDEZ**

OCTUBRE - 2004

A MI MAMA:

Sra. Alicia Villa Martínez

Por todo tu amor, siempre vivirás en mi corazón.

A MI PAPA:

Sr. Juan Gómez Tabares

Por tu cariño y apoyo, por ti soy todo lo que soy.

A MIS HERMANOS:

Alicia, Juan, Jorge

Por siempre brindarme un gran apoyo.

A MIS SOBRINOS:

Juan, Jorge, Alicia

Por darme siempre un motivo para ser mejor cada día.

A MI ASESOR:

Por su paciencia y sabios consejos.

A MIS MAESTROS:

Por compartirme sus conocimientos.

A MIS AMIGOS:

Por hacerme reír en los momentos difíciles.

A TODAS LAS PERSONAS ESPECIALES DE MI VIDA

SINCERAMENTE

Guillermo Gómez Villa

ÍNDICE

INTRODUCCIÓN

En el mundo actual la revolución tecnológica y comercial ha tomado velocidades vertiginosas, es indudable que en esta revolución uno de los factores más importante es la información, Internet en este aspecto a tomado un papel fundamental ya que ha roto las fronteras de la comunicación electrónica, cada vez es mas común realizar compras electrónicas sin importar las distancias entre comprador y vendedor, establecer contratos comerciales de manera electrónica, realizar transferencias bancarias en tan solo un par de minutos o simplemente enviar correos electrónicos.

Así mismo, la seguridad y privacidad en las comunicaciones es fundamental en el intercambio de información, ya que, de no existir seguridad en las comunicaciones el comercio electrónico y el sistema financiero mundial se verían fuertemente vulnerados. Muchos aspectos de la vida moderna como celulares, televisión satelital, tarjetas de crédito, etc. se encontrarían muy limitados.

En la actualidad todas las comunicaciones se basan en sistemas computacionales e informáticos, por lo que no se concibe ningún sistema informático serio sin al menos algún mecanismo de seguridad que salvaguarde la integridad de la información procesada. De ahí, la enorme importancia para el profesional diseñador y desarrollador de software el conocimientos de los métodos de seguridad para fortalecer la integridad y privacidad de la información.

La criptografía es una de las partes más importantes de la seguridad de la información, ya que, por medio de técnicas y algoritmos matemáticos conformados por bases sólidas y probadas, transforma la información de manera ininteligible a entidades no autorizadas. El diseñador y desarrollador de software debe de tener los conocimientos necesarios para implementar de manera correcta las técnicas criptográficas adecuadas a sus necesidades.

El presente trabajo tiene como objetivo realizar un análisis detallado de las técnicas criptográficas simétricas y asimétricas más usadas en la actualidad, determinar los puntos débiles y fuertes de los métodos analizados para su correcta selección e implementación en el mundo informático, también tiene como objetivo mostrar las aplicaciones más comunes de la criptografía en la actualidad.

El capítulo 1 se abordará las bases matemáticas necesarias para la comprensión de los algoritmos criptográficos clásicos y modernos, la criptografía es la combinación de distintas áreas matemáticas muy diversas, el capitulo inicia con una sencilla introducción histórica de la criptografía para, así, dar paso a las definiciones básicas para el entendimiento de los algoritmos criptográficos, también abordaremos de manera sencilla la Teoría de la Información, Teoría de Números y Teoría de la complejidad computacional. Al final el lector contará con las herramientas básicas necesarias para el análisis de las ventajas y debilidades de los algoritmos criptográficos actuales.

El capítulo 2 inicia con el estudio de los métodos clásicos de cifrado, estos métodos utilizados hasta antes de la llegada de las computadoras es la base de los algoritmos actuales de cifrados

simétricos, entre estos métodos simétricos actuales destaca por su enorme difusión y uso el algoritmo criptográfico DES del cual se realiza un estudio detallado de su estructura, por último se realiza un estudio aun mas profundo al algoritmo criptográfico RIJNDAEL el cual se convierte desde 2002 como el nuevo algoritmo criptográfico seguro estándar en las comunicaciones modernas. Al finalizar del capitulo el lector será capaz con base a las ventajas y desventajas de los algoritmos simétricos analizados de implementar el algoritmo criptográfico simétrico más adecuado a sus necesidades.

En el capítulo 3 se realiza el estudio de los algoritmos criptográficos asimétricos también conocidos como de clave pública, los algoritmos asimétricos de cifrado son enormemente importantes en las comunicaciones por canales no seguros (Internet) de ahí su gran importancia en el comercio electrónico, se realiza principalmente el estudio al algoritmo criptográfico RSA el cual se ha definido como el algoritmo criptográfico asimétrico estándar en la actualidad, posteriormente se realiza el estudio a los algoritmos Rabin y ElGamal que en conjunto con RSA basan su fuerza en la resolución de problemas matemáticos complejos. Al finalizar el capitulo el lector conocerá las desventajas y ventajas de los algoritmos criptográficos asimétricos frente a los simétricos, las dificultades de su implementación y su utilidad.

En el capítulo 4 se estudiarán las aplicaciones criptográficas en conceptos como métodos de autenticación, funciones resumen, firma digital y certificados digitales, los cuales se conforman por una combinación de algoritmos criptográficos simétricos y asimétricos. Se realizará el estudio a la infraestructura de claves publicas que son la base del comercio electrónico y financiero de la actualidad, para este fin se estudiaran los estándares SSL, X.509, S/MIME, IPSec y LDAP, se definirán los conceptos de Autoridad Certificadora, Autoridad de Registro y Anillos de confianza. Al finalizar el lector podrá implementar las aplicaciones criptográficas en los sistemas informáticos de comercio electrónico actuales.

Por ultimo se agrega un pequeño apartado acerca de las implicaciones sociales y políticas de la criptografía en el mundo moderno, la criptografía es una herramienta básica en el comercio electrónico, pero también es usada como medio de control de los gobiernos y peligrosa herramienta de los criminales, así mismo se discute el derecho de todo ciudadano de proteger su privacidad.

CAPÍTULO 1

Bases Criptográficas

1.- Introducción Histórica.

Del Antiguo Egipto a la era digital, los mensajes cifrados han jugado un papel destacado en la historia. Arma de militares, diplomáticos y espías, son la mejor defensa de las comunicaciones y datos que viajan por Internet.

Desde la antigüedad, el hombre ha hecho gala de su ingenio para garantizar la confidencialidad de sus comunicaciones. La criptografía (del griego kryptos, "escondido", y graphein, "escribir"), el arte de enmascarar los mensajes con signos convencionales, que sólo cobran sentido a la luz de una clave secreta, nació con la escritura. Su rastro se encuentra ya en las tablas cuneiformes, y los papiros demuestran que los primeros egipcios, hebreos, babilonios y asirios conocieron y aplicaron sus inescrutables técnicas, que alcanzan hoy su máxima expresión gracias al desarrollo de los sistemas informáticos y de las redes mundiales de comunicación.

Entre el antiguo Egipto e Internet, los criptogramas han protagonizado buena parte de los grandes episodios históricos. Existen mensajes cifrados en todos lados, abundan en los textos diplomáticos, pueblan las órdenes militares en tiempos de guerra, muy importante en las transacciones comerciales, en los secretos industriales y, por supuesto, son la esencia de la actividad de los espías.

1.1.- Los Métodos Antiguos de Cifrado

Los espartanos utilizaron, en el año 400 A.C., la Scitala, que puede considerarse el primer sistema de criptografía por transposición, es decir, que se caracteriza por enmascarar el significado real de un texto alterando el orden de los signos que lo conforman. Los militares de la ciudad griega escribían sus mensajes sobre una tela que envolvía una vara. El mensaje sólo podía leerse cuando se enrollaba sobre un bastón del mismo grosor, que poseía el destinatario lícito.

El método de la scitala era extremadamente sencillo, como también lo era el que instituyó Julio César, basado en la sustitución de cada letra por la que ocupa tres puestos más allá en el alfabeto.

En los escritos medievales sorprenden términos como Xilef. Para esconder sus nombres, los copistas empleaban el alfabeto zodiacal, formaban anagramas alterando el orden de las letras (es el

caso de Xilef, anagrama de Félix) o recurrían a un método denominado fuga de vocales, en el que éstas se sustituían por puntos o por consonantes arbitrarias.

La criptografía resurgió en la Europa de la Edad Media, impulsada por las intrigas del papado y las ciudades-estado italianas. Fue un servidor del Papa Clemente VII, Grabiele de Lavinde, quien escribió el primer manual sobre la materia en el viejo continente.

En 1466, León Battista Alberti, músico, pintor, escritor y arquitecto, concibió el sistema polialfabético que emplea varios abecedarios, saltando de uno a otro cada tres o cuatro palabras. El emisor y el destinatario han de ponerse de acuerdo para fijar la posición relativa de dos círculos concéntricos, que determinará la correspondencia de los signos.

Un siglo después, Giovan Battista Belaso de Brescia instituyó una nueva técnica. La clave, formada por una palabra o una frase, debe transcribirse letra a letra sobre el texto original. Cada letra del texto se cambia por la correspondiente en el alfabeto que comienza en la letra clave.

El siglo XX ha revolucionado la criptografía. Retomando el concepto de las ruedas concéntricas de Alberti, a principios de la centuria se diseñaron teletipos equipados con una secuencia de rotores móviles. Éstos giraban con cada tecla que se pulsaba. De esta forma, en lugar de la letra elegida, aparecía un signo escogido por la máquina según diferentes reglas en un código polialfabético complejo. Estos aparatos, se llamaron traductores mecánicos. Una de sus predecesoras fue la rueda de Jefferson, el aparato mecánico criptográfico más antiguo que se conserva.

La primera patente data de 1919, y es obra del holandés Alexander Koch, que comparte honores con el alemán Arthur Scherbius, el inventor de Enigma una máquina criptográfica que los nazis creyeron inviolable, sin saber que a partir de 1942, propiciaría su derrota. En el desenlace de la contienda, hubo un factor decisivo y apenas conocido: los aliados eran capaces de descifrar todos los mensajes secretos alemanes.

Una organización secreta, en la que participó Alan Turing, uno de los padres de la informática y de la inteligencia artificial, había logrado descubrir las claves de Enigma, desarrollando más de una docena de artilugios -las bombas- que desvelaban los mensajes cifrados. La máquina alemana se convertía así en el talón de Aquiles del régimen, un topo en el que confiaban y que en definitiva, trabajaba para el enemigo.

Mientras los nazis diseñaron Enigma para actuar en el campo de batalla, los estadounidenses utilizaron un modelo llamado Sigaba y apodado por los alemanes como "la gran máquina". Este modelo, funcionó en estaciones fijas y fue el único artefacto criptográfico que conservó intactos todos sus secretos durante la guerra.

Finalizada la contienda, las nuevas tecnologías electrónicas y digitales se adaptaron a las máquinas criptográficas. Se dieron así los primeros pasos hacia los sistemas criptográficos más modernos, mucho más fiables que la sustitución y transposición clásicas. Hoy por hoy, se utilizan

métodos que combinan los dígitos del mensaje con otros, o bien algoritmos de gran complejidad. Una computadora moderna tardaría 200 millones de años en interpretar las claves más largas, de 128 bits.

Con la expansión de la red Internet se ha acelerado el desarrollo de las técnicas de ocultación, ya que, al mismo ritmo que crece la libertad de comunicarse, se multiplican los riesgos para la privacidad. La duda persiste. ¿Son capaces las complejas claves actuales de garantizar el secreto? Muchas de las técnicas que se han considerado infalibles a lo largo de la historia han mostrado sus puntos débiles ante la habilidad de los enemigos, desde los misterios de Enigma, que cayeron en poder del enemigo, hasta el algoritmo criptográfico DES (Data Encryption Standard), desechado por el propio gobierno estadounidense por poco fiable.

Pero a pesar de los muchos rumores que hablan de la poca seguridad que garantizan las transmisiones vía Internet, es muy improbable que un estafador pueda interceptar los datos reservados de una transacción, por ejemplo, el número de una tarjeta de crédito, porque los formularios que hay que rellenar han sido diseñados con programas que cifran los datos. Los hay tan simples que sustituye cada letra por la situada 13 puestos más adelante, o extremadamente complicados como los algoritmos criptográficos Rijndael y RSA.

2.- Conceptos Criptográficos.

Las raíces etimológicas de la palabra criptografía son kriptos, que significa oculto, y graphos, que se traduce como escribir, lo que nos da su definición clásica: arte de escribir mensajes en clave secreta o enigmáticamente.

Fue considerada un arte, hasta que el matemático estadounidense Claude Shannon publicó en 1949 la "Teoría de las comunicaciones secretas". Entonces la criptografía empezó a ser considerada una ciencia aplicada, debido a su relación con otras ciencias, como la estadística, la teoría de números, la teoría de la información y la teoría de la complejidad computacional.

La criptografía corresponde sólo a una parte de la comunicación secreta. Si se requiere secreto para la comunicación, es porque existe desconfianza o peligro de que el mensaje transmitido sea interceptado por el enemigo. Este enemigo, si existe, utilizará todos los medios a su alcance para descifrar esos mensajes secretos mediante un conjunto de técnicas y métodos que constituyen una ciencia conocida como criptoanálisis. Al conjunto de ambas ciencias, criptografía y criptoanálisis, se le denomina criptología.

Como en muchas otras áreas científicas, el mayor desarrollo de la criptografía tuvo lugar durante las dos guerras mundiales. En este caso se debió a la necesidad de establecer comunicaciones secretas militares y diplomáticas utilizando nuevas tecnologías, como la telegrafía y la radiotecnia.

En la segunda mitad de este siglo, con el desarrollo de la cultura informática, han surgido nuevas aplicaciones de la criptografía, debido a fundamentalmente al manejo de gran cantidad de información. En algunos casos, como en las redes informáticas, dicha información esta disponible a muchos usuarios, lo cual plantea de que los datos estén protegidos durante su transmisión y durante su almacenamiento.

Al mismo tiempo, este desarrollo de la informática produjo un cambio radical en el concepto de seguridad de los sistemas criptográficos, pues aquellos que eran supuestamente seguros frente a procedimientos manuales sucumbieron ante la eficacia de las computadoras. De esta forma, la supuesta seguridad de los sistemas antiguos o clásicos ha tenido que ser sustituida por una seguridad matemática y computacionalmente demostrable en los sistemas modernos.

En los procesos de almacenamiento y transmisión de la información normalmente aparece el problema de su seguridad. En el almacenamiento, el peligro lo representa el robo del soporte del mensaje o simplemente el acceso no autorizado a esa información, mientras que en las transmisiones lo es la intervención del canal de comunicación.

Hay dos tipos básicos de amenazas contra el mensaje secreto:

- Pasiva: Es el robo o el acceso no autorizado a la información.
- Activa: Es la alteración de la información.

La protección de la información se lleva a cabo variando su forma. Se llama cifrado (o transformación criptográfica) a una transformación del texto original (llamado también texto inicial o texto en claro) que lo convierte en el llamado texto cifrado o criptograma. Análogamente, se llama descifrado a la transformación que permite recuperar el texto original a partir del texto cifrado.

Cada una de estas transformaciones está determinada por un parámetro llamado clave. El conjunto de sus posibles valores se denomina espacio de claves *K*. La familia de transformaciones criptográficas se llama sistema criptográfico.

$$T = \{T_k / k \in K\}$$

Donde T = Transformación Criptográfica K = Espacio de claves posibles dentro del alfabeto

Para cada transformación criptográfica T_k se definen las imágenes de cada una de las palabras de *n* letras. Es decir, el sistema criptográfico se puede describir como $T = \{T_k^{(n)}; 1 \le n < \infty\}$, siendo $T_k^{(n)}(x) = y$, donde *y* es la palabra cifrada que corresponde a la palabra original *x*.

Para evitar ambigüedades, se hacen las siguientes suposiciones:

- $T_k^{(n)}$ es biyectiva.
- Se usa el mismo alfabeto para ambos textos, original y cifrado.
- Se define el cifrado de todas las posibles palabras, independientemente de si existen o no.
- Cada n-palabra se cifra en una n-palabra, teniéndose así el cifrado no cambia de longitud del texto original.

En general, no es necesario imponer simultáneamente todas estas condiciones, aunque en algunos casos, como el procesamiento de información digital, si es recomendable, porque:

- Se trabaja únicamente con alfabeto binario.
- Debe existir el cifrado de todas las palabras posibles.
- Si el cifrado cambiara la longitud del texto, sería necesario usar un nuevo formato para el texto cifrado.

2.1.- Reglas de Kerckhoffs.

Auguste Kerckhoffs Von Nieuwenhof (1835-1903), en su trabajo titulado "La criptografía militar", recomendó que los sistemas criptográficos cumpliesen las siguientes reglas, que efectivamente han sido adoptadas por gran parte de la comunidad criptográfica:

1. No debe existir ninguna forma de recuperar mediante el criptograma el texto inicial o la clave.

Esta regla se considera cumplida siempre que la complejidad del proceso de recuperación del texto original sea suficiente para mantener la seguridad del sistema. No obstante, en muchos casos no es necesario maximizar dicha complejidad, ya que en general la seguridad máxima se paga a la hora del descifrado. Hay que tener en cuenta la relación existente entre la tecnología y la seguridad del sistema con el espacio de claves finito que se consideraban seguros cuando no existían las computadoras, resultan ahora completamente inseguros, debido a la posibilidad del examen exhaustivo de todas las claves.

2. Todo sistema criptográfico debe estar compuesto por dos tipos de información:

    - Pública, como es la familia de algoritmos que lo definen.
    - Privada, como es la clave que usa en cada cifrado particularmente.

En los sistemas de clave pública, que se tratarán en el capítulo 3, parte de la clave también es pública.

El principio de Kerckhoffs dice que la seguridad de un criptosistema se mide suponiendo que el enemigo conoce completamente ambos procesos de cifrado y descifrado.

3. La forma de escoger la clave debe ser fácil de recordar y de modificar.
4. Debe ser factible la comunicación del criptograma con los medios de transmisión habituales.
5. La complejidad del proceso de recuperación del texto original debe corresponderse con el beneficio obtenido.

2.2.- Tipos de Ataque.

Para el estudio de los sistemas criptográficos es conveniente conocer la situación del enemigo. Se tiene los siguientes ataques posibles:

1. Ataque sólo con texto cifrado: Ésta es la peor situación posible para el criptoanalista, ya que representan cuando sólo conoce el criptograma.
2. Ataque con texto original conocido: Consiste en que el criptoanalista tiene acceso a un parte de texto en claro y cifrado. Se da este caso, por ejemplo, cuando se conoce el tema del mensaje, pues solo proporciona una correspondencia entre las palabras más probables y las palabras cifradas más repetidas.
3. Ataque con texto original escogido: Este caso se da cuando el enemigo puede obtener, además del criptograma que trata de descifrar, el cifrado de cualquier texto que él elija, entendiéndose que no es que él sepa cifrarlo, sino que lo obtiene ya cifrado.
4. Ataque con texto cifrado escogido: Se presenta cuando el enemigo puede obtener el texto original correspondiente a determinados textos cifrados de su elección.

De acuerdo a la primera regla de Kerckhoffs, se pueden distinguir dos tipos de secreto: el secreto teórico o incondicional y el secreto práctico o computacional. El primero se basa en que la información disponible para el enemigo no es suficiente para romper el sistema. Por ejemplo, se da este caso cuando el enemigo solo conoce una cantidad de criptograma insuficiente para el criptoanálisis. Por el contrario, el secreto práctico se mide de acuerdo con la complejidad computacional del criptoanálisis. Según las necesidades actuales, y debido principalmente a la gran cantidad de información que se transmite habitualmente, los diseñadores de sistemas criptográficos

deben suponer que el enemigo puede hacer por lo menos un ataque del segundo tipo (con texto original conocido), luego deben de intentar conseguir al menos el secreto práctico.

Un criptosistema tiene secreto:

- Teórico: Si es seguro contra cualquier enemigo que tenga recursos y tiempo ilimitados.
- Práctico: Si es seguro contra aquellos enemigos que tenga menos de una cantidad de tiempo y/o recursos.

3.- Teoría de la Información.

Iniciamos el estudio de los fundamentos teóricos principales de la criptografía dando una serie de nociones básicas sobre Teoría de la Información, introducida por Claude Shannon a finales de los años cuarenta. Esta disciplina nos permitirá efectuar una aproximación teórica al estudio de la seguridad de cualquier algoritmo criptográfico.

3.1.- Cantidad de Información.

Introduciremos este concepto partiendo de su idea intuitiva. Para ello analizamos el siguiente ejemplo: supongamos que tenemos una bolsa con nueve bolas negras y una blanca. ¿Cuanta información obtenemos si nos dicen que han sacado una bola blanca de la bolsa? ¿Y cuanta obtenemos si después sacan otra y nos dicen que es negra?.

Obviamente, la respuesta a la primera pregunta es que nos aporta bastante información, puesto que estamos casi seguros de que la segunda bola tenía que salir negra. Análogamente si hubiera salida negra diríamos que ese suceso no nos extraña (nos aporta poca información). En cuanto a la segunda pregunta, claramente podemos contestar que no aporta ninguna información, ya que no al no quedar bolas blancas sabíamos que iba a salir negra.

Podemos fijarnos en la cantidad de información como una medida de la disminución de incertidumbre acerca de un suceso. Por ejemplo, si nos dicen que el número que ha salido en un dado es menor que dos, nos dan más información que si nos dicen que el número que ha salido es par.

Se puede decir que la cantidad de información que nos aporta conocer un hecho es directamente proporcional al número posible de estados que esta tenia a priori. Si inicialmente teníamos diez posibilidades, conocer el hecho nos proporciona más información que si inicialmente tuviéramos dos. Por ejemplo, supone mayor información conocer la combinación ganadora del próximo sorteo de la lotería, que saber si una moneda lanzada al aire va a salir águila o sol. Claramente es más fácil

acertar en el segundo caso, puesto que el numero de posibilidades a priori (y por lo tanto la incertidumbre, suponiendo sucesos equiprobables) es menor.

También la cantidad de información es proporcional a la probabilidad de un suceso. En el caso de las bolas tenemos dos sucesos: sacar bola negra, que es más probable, y sacar bola blanca, que es menos probable. Sacar una bola negra aumenta nuestro grado de certeza inicial de un 90 % a un 100%, proporcionándonos una ganancia del 10%. Sacar una bola blanca aumenta esa misma certeza en un 90%. Podemos considerar la disminución de incertidumbre proporcional al número de certeza, por lo cual podemos decir que el primer suceso (sacar bola negra) aporta menos información.

A partir de ahora, con objeto de simplificar la notación, vamos a emplear una variable aleatoria *V* para representar los posibles sucesos que nos podemos encontrar. Notaremos el suceso *i-ésimo* como x_i, $P(x_i)$ será la probabilidad asociada a dicho suceso, y n será el número de sucesos.

Supongamos ahora que sabemos con toda seguridad que el único valor que puede tomar *V* es x_i, saber el valor de *V* no nos va a aportar ninguna información (lo conocemos de antemano). Por lo contrario, si tenemos una certeza del 99% sobre la posible ocurrencia del valor x_i, obtener un x_i nos aportara bastante información, como ya hemos visto. Este concepto es cuantificable y se puede definir de la siguiente forma:

$$I_i = -\log_2\left(P\left(x_i\right)\right)$$

Donde I_i= Cantidad de información proporcionada por un estado x_i y $P(x_i)$ = probabilidad de que suceda el estado x_i

Siendo $P(x_i)$ la probabilidad del estado x_i. Obsérvese que si la probabilidad de un estado fuera 1 (máxima), la cantidad de información que nos aporta es igual a 0, mientras que si su probabilidad se acercara a 0, tendería a $+\infty$ (esto es lógico, un suceso que no puede suceder nos aportará una cantidad infinita de información si llegara a suceder).

3.2.- Entropía.

Sumando ponderadamente las cantidades de información de todos los posibles estados de una variable aleatoria *V*, obtenemos:

$$H\left(V\right) = -\sum_{i=1}^{n} P\left(x_i\right)\log_2\left[P\left(x_i\right)\right]$$

Donde $H(V)$= La entropía generada por todos los sucesos x_i de una variable aleatoria *V* y $P(x_i)$= Probabilidad del suceso x_i

Esta magnitud $H(V)$ se conoce como la entropía de la variable aleatoria *V*. Sus propiedades son:

i. $0 \leq H(V) \leq \log_2 N$

ii. $H(V) = 0 \Leftrightarrow \exists i$ tal que $P(x_i) = 1$ y $P(x_j) = 0 \forall i \neq j$

iii. $H(x_1, x_2, \ldots x_n) = H(x_1, x_2, \ldots x_n, x_{n+1})$ si $P(x_{n+1}) = 0$

Obsérvese que la entropía es proporcional a la longitud media de los mensajes necesaria para codificar una serie de valores de *V* de manera óptima dado un alfabeto cualquiera. Esto quiere decir que cuando más probable sea un valor individual, aportará menos información cuando aparezca, y podremos codificarlo empleado un mensaje más corto. Si $P(x_i)$ = 1 no necesitaríamos ningún mensaje, puesto que sabemos de antemano que *V* va a tomar el valor x_i, mientras que si $P(x_i)$ = 0.9 parece más lógico emplear mensajes cortos para representar el suceso x_i y largos para los x_j restantes, ya que el valor que más nos va a aparecer en una ocurrencia de sucesos es precisamente x_j. Veamos un ejemplo:

La entropía de la variable aleatoria asociada a lanzar una moneda es la siguiente:

$$H(M) = -(0.5 \log_2(0.5) + 0.5 \log_2(0.5)) = 1$$

Este suceso aporta exactamente una unidad de información.

Si la moneda está trucada (60% de probabilidades para cara, 40% para cruz), nos sale:

$$H(M) = -\ (0.6\ \log_2(0.6) + 0.4 \log_2(0.4)) = 0.970$$

La unidad de cantidad de información que vamos a emplear es la cantidad de información que aporta el suceso más simple (dos posibilidades equiprobables, como el caso de la moneda sin trucar), a esa unidad se le denomina *bit*. Esta es precisamente la razón por la que empleamos logaritmos base 2, para que la cantidad de información del suceso más simple sea igual a la unidad.

Podemos decir que la entropía de un suceso es el número medio de bits que necesitamos para codificar cada uno de los estados de la variable (codificamos cada suceso empleado un mensaje escrito en un alfabeto binario), suponemos que queremos codificar los diez dígitos decimales usando secuencias de bits. Con tres bits no tenemos suficiente, así que necesitamos más, pero ¿cuántos más? Si usamos cuatro bits para representar todos los dígitos tal vez nos estemos pasando.

Veamos cuánta entropía tiene diez sucesos equiprobables:

$$H = -\sum_{i=1}^{10} \frac{1}{10} \log_2\left(\frac{1}{10}\right) = -\log_2\left(\frac{1}{10}\right) = 3.32 bits$$

El valor que acabamos de calcular es el límite teórico, que normalmente no se puede alcanzar. Lo único que podemos decir es que no existe ninguna codificación que emplee longitudes promedio de mensaje inferiores al número que acabamos de calcular. Veamos la siguiente codificación: 000 para 0, 001 para 1, 010 para 2, 011 para 3, 100 para 4, 101 para 5, 1100 para 6, 1101 para 7,1110 para 8 y 1111 para 9. Con esta codificación empleamos como media

$$\frac{3\cdot 6+4\cdot 4}{10}=3.4bits$$

para codificar este mensaje. El denominado Método de Huffman, uno de los más utilizados en transmisión de datos, permite obtener codificaciones binarias que se aproximan bastante al óptimo teórico de una forma sencilla y eficiente.

3.3.- Entropía Condicionada.

Supongamos que tenemos ahora una variable aleatoria bidimensional *(X,Y)*. Recordemos las distribuciones de probabilidad más usuales que podemos definir sobre dicha variable, teniendo *n* posibles casos para *X* y *m* para *Y*:

1. Distribución conjunta de *(X,Y)*:

$$P(x_i,y_j)$$

2. Distribuciones marginales de *X* e *Y*:

$$P(x_i)=\sum_{j=1}^{m}P(x_i,y_j)\qquad P(y_j)=\sum_{i=1}^{n}P(x_i,y_j)$$

3. Distribuciones condicionales de *X* sobre *Y* y viceversa:

$$P(x_i/y_j)=\frac{P(x_i,y_j)}{P(y_j)}\qquad P(y_j/x_i)=\frac{P(x_i,y_j)}{P(x_j)}$$

Definiremos la entropía de las distribuciones que acabamos de referir:

$$H(X,Y)=-\sum_{i=1}^{n}\sum_{j=1}^{m}P(x_i,y_j)\log_2\left(P(x_i,y_j)\right)$$

$$H(X/Y=y_j)=-\sum_{i=1}^{n}P(x_i,y_j)\log_2\left(P(x_i/y_j)\right)$$

Haciendo la suma ponderada de los $H(X/Y) = y_i$ obtenemos la expresión de la Entropía condicionada de X sobre Y:

$$H(X/Y) = -\sum_{i=1}^{n}\sum_{j=1}^{m} P(y_j)P(x_i/y_j)\log_2\left(P(x_i,y_j)\right) =$$

$$= -\sum_{i=1}^{n}\sum_{j=1}^{m} P(x_i,y_j)\log_2\left(P(x_i,y_j)\right)$$

Donde: *H(X/Y)* es la entropía condicionada de la variable aleatoria bidimensional *V(X,Y)*

$P(y_j)$ es la probabilidad del suceso yi

$P(x_i/y_j)$ es la probabilidad de ocurrencia del suceso x*i* tal que el suceso y_j ya haya sucedido.

Así como existe una Ley de la Probabilidad Total, análogamente se define la Ley de Entropías Totales:

$$H(X,Y) = H(X) + H(Y/X)$$

Donde H(X,Y) Entropía total de la variable aleatoria bidimensional *V(X,Y)* donde *X* y *Y* no son independientes

H(Y/X) es la entropía condicionada de la variable aleatoria bidimensional *V(X,Y)* donde *X* y *Y* no son independientes

H(X) es la entropía de todos los sucesos x_i de la variable aleatoria *V*.

cumpliéndose además, si *X* e *Y* son variables independientes:

$$H(X,Y) = H(X) + H(Y)$$

Donde *H(X,Y)* Entropía total de la variable aleatoria bidimensional *V(X,Y)* donde *X* y *Y* son independientes

H(X) es entropía de los sucesos x_i de la variable aleatoria *V*.

H(Y) es entropía de los sucesos y_j de la variable aleatoria *V*.

Teorema de Disminución de la Entropía: La entropía de una variable *X* condicionada por otra *Y* es menor o igual a la entropía de *X*, alcanzándose la igualdad si y solo si las variables *X* e *Y* son independientes.

Este teorema representa una idea intuitiva clara: conocer algo acerca de la variable *Y* puede que nos ayude a saber más sobre X (tener menos entropía), pero en ningún caso hará aumentar nuestra incertidumbre.

3.4.- Cantidad de Información entre dos Variables.

Shannon propuso una medida para la cantidad de información que aporta sobre una variable el conocimiento de otra. Se definirá, pues, la cantidad de información de Shannon que la variable *X* contiene sobre *Y* como:

$$I(X,Y) = H(Y) - H(Y/X)$$

Donde $I(X,Y)$ es la cantidad de información entre dos variables X y Y.

Esto quiere decir que la cantidad de información que nos aporta el hecho de conocer X al medir la incertidumbre sobre Y es igual a la disminución de entropía que este conocimiento conlleva. Sus propiedades son:

1.- $I(X,Y) = I(Y,X)$

2.- $I(X,Y) \geq 0$

3.5.- Criptosistema Seguro de Shannon.

Diremos que un criptosistema es seguro si la cantidad de información que nos aporta el hecho de conocer el mensaje cifrado c sobre la entropía del texto plano m vale cero, es decir:

$$I(C,M) = 0$$

Donde $I(C,M)$ es la cantidad de información entre el mensaje cifrado C y el texto plano M.

Esto significa sencillamente que la distribución de probabilidad que nos inducen todos los mensajes no cifrados no cambia si conocemos el mensaje cifrado. Para entenderlo mejor supongamos que sí se modifica dicha distribución: El hecho de conocer el mensaje cifrado, al variar la distribución de probabilidad sobre M haría unos mensajes más probables que otros, y por consiguiente unas claves de cifrado (aquellas que me permiten llegar a los m más probables al c concreto que tenga en cada momento) más probables que los otros, permitiéndonos romper el criptosistema.

Si por el contrario el sistema cumpliera la condición, jamás podríamos romper el criptosistema, ni siquiera empleado una máquina con capacidad de proceso infinita. Por ello los criptosistemas que cumplen la condición de Shannon se denominan también criptosistemas ideales.

Se puede demostrar también que para que un sistema sea criptoseguro según el criterio de Shannon, la cardinalidad del espacio de claves ha de ser al menos igual que la del espacio de mensajes. En otras palabras, que la clave he de ser al menos tan larga como el mensaje que queramos cifrar. Esto vuelve inútiles a estos criptosistemas en la práctica, porque si la clave es tanto o más larga que el mensaje, a la hora de protegerla nos encontraremos con el mismo problema que teníamos para proteger el mensaje.

3.6.- Redundancia.

Si una persona lee un mensaje en el que faltan algunas letras, normalmente puede construirlo, esto ocurre porque casi todos los símbolos de un mensaje en lenguaje natural que contienen información que se puede extraer de los símbolos de alrededor (información que, en la práctica, se esta enviando dos veces). En otras palabras, porque el lenguaje natural es redundancia. Puesto que tenemos mecanismos para definir la cantidad de información que presenta un suceso, podemos intentar medir el exceso de información (redundancia) de un lenguaje. Para ello vamos a dar una serie de definiciones:

- Índice de un lenguaje: Definiremos el índice de un lenguaje para mensajes de longitud *k* como:

$$r_k = \frac{H_k(M)}{k}$$

Donde r_k es el índice del lenguaje, H_k es la entropía del mensaje M y k la longitud del mensaje

Siendo $H_k(X)$ la entropía de todos los posibles mensajes de longitud *k*. Estamos midiendo el número de bits de información que nos aporta cada carácter en mensajes de una longitud determinada. Para idiomas como el Ingles, r_k suele valer alrededor de 1.3 bits/letra para valores pequeños de *k*.

- Índice absoluto de un lenguaje: Es el máximo número de bits de información que pueden ser codificados en cada carácter, asumiendo que todas las combinaciones de caracteres son igualmente probables. Suponiendo *m* letras diferentes en nuestro alfabeto (27 en el caso del español), este índice vale:

$$R = \log_2(m)$$

Donde R es índice del lenguaje y m es la cardinalidad del alfabeto usado.

En el caso del español podríamos codificar 4.7bits por letra aproximadamente, luego parece que el nivel de redundancia (asumiendo que su índice r sea parecido al del inglés) es alto.

- Finalmente, la redundancia de un lenguaje se define como la diferencia entre las dos magnitudes anteriores:

$$D = R - r$$

También se define el índice de redundancia como el siguiente cociente:

$$I = \frac{D}{R}$$

Desgraciadamente, para medir la autentica redundancia de un lenguaje, hemos de tener en cuenta secuencias de cualquier número de caracteres, por lo que la expresión del índice del lenguaje debería calcularse en realidad como:

$$r_{\infty} = \lim_{n \to \infty} \frac{H_n(M)}{n}$$

Precisamente una de las aplicaciones de la Teoría de la Información es la compresión de datos, que simplemente trata de eliminar la redundancia dentro de un archivo (considerando cada byte como un mensaje elemental, y codificándolo con más o menos bits según su frecuencia de aparición).

Otra de las aplicaciones directas de la Teoría de la Información son los Códigos de Redundancia Cíclica (CRC), que permiten introducir un campo de longitud mínima en el mensaje, tal que éste proporcione la mayor redundancia posible. Así, si el mensaje original resultase alterado, la probabilidad de que el CRC añadido siga siendo correcto es mínima.

Conocidos los patrones de redundancia de un lenguaje, es posible dar forma automática una estimación de si una cadena de símbolos corresponde o no a dicho lenguaje. Esta característica es aprovechada para efectuar ataque por la fuerza bruta, ya que ha de asignarse una probabilidad a cada clave individual en función de las características del mensaje obtenido al decodificar el criptograma con dicha clave. El número de claves suele ser tan elevado que resulta imposible una inspección visual. Una estrategia bastante interesante para protegerse contra este tipo de ataques consistirá en comprimir los mensajes antes de codificarlos. De esa manera eliminamos la redundancia y hacemos más difícil a un atacante apoyarse en las características del mensaje original para recuperar la clave.

3.7.- Desinformación y Distancia de Unicidad.

Definiremos desinformación de un sistema criptográfico como la entropía condicionada de un mensaje *m* dado un texto cifrado *c*:

$$H(M/C) = -\sum_{k \in K} \sum_{c \in C} P(c) P(m/c) \log_2 (P(m/c))$$

Donde $H_c(m)$ es la desinformación de un sistema criptográfico.

Esta expresión nos dice la incertidumbre que nos queda sobre *m* si conocemos *c*. si esa incertidumbre fuera la misma que desconociendo *c*, no encontraríamos frente a un criptosistema seguro de Shannon. Lo habitual es que $H_c(m) < H(m)$

Adicionalmente, si el valor de $H_c(m)$ fuera muy pequeño con respecto a $H(m)$, significaría que el hecho de conocer *c* nos aporta mucha información sobre *m*, lo cual quiere decir que nuestro criptosistema es inseguro. El peor de los casos seria que $H_c(m) = 0$, puesto que entonces, conociendo *c* tendríamos absoluta certeza sobre *m*.

Esta magnitud se puede medir también en función de la clave *k*, y entonces nos dirá la incertidumbre que nos queda sobre *k* conocida *c*:

$$H(K/C) = -\sum_{m \in M} \sum_{c \in C} P(c)P(k/c)\log_2(P(k/c))$$

Definiremos finalmente la distancia de unicidad de un criptosistema como la longitud mínima de mensaje cifrado que aproxima el valor $H_c(k)$ a cero. En otras palabras, es la cantidad de texto cifrado que necesitamos para poder descubrir la clave. Los criptosistemas seguros de Shannon tienen distancia de unicidad infinita. Nuestro objetivo a la hora de diseñar un sistema criptográfico será que la distancia de unicidad sea lo más grande posible.

3.8.- Confusión y Difusión.

Según la Teoría de Shannon, las dos técnicas básicas para ocultar la redundancia en un texto plano son la confusión y la difusión, estos conceptos, a pesar de su antigüedad, poseen una importancia clave en la criptografía moderna.

- Confusión: Trata de ocultar la relación entre en texto plano y el texto cifrado. Recordemos que esa relación existe y se da a partir de la clave *k* empleada, puesto que si no existiera jamás podríamos descifrar los mensajes. El mecanismo más simple de confusión es la sustitución, que consiste en cambiar cada ocurrencia de un símbolo en el texto plano de otro. La sustitución puede ser tan simple o tan compleja como queramos.
- Difusión: Diluye la redundancia del texto plano repartiéndola a lo largo de todo el texto cifrado. El mecanismo más elemental para llevar a cabo una difusión es la transposición (que consiste en cambiar de sitio elementos individuales del texto plano).

4.- Principios de Aritmética Modular.

La fuerza de los algoritmos criptográficos asimétricos en gran medida se depositan en problemas matemáticos complejos, la aritmética modular es una de las áreas de las matemáticas que es fundamental en el estudio de los algoritmos asimétricos que veremos posteriormente en el capítulo 3.

4.1.- Aritmética Modular.

Propiedades:

Dados tres números $a,b,c \in \mathbb{N}$, decimos que *a* es congruente con *b* módulo *n*, y se escribe

$$a \equiv b \pmod{n}$$

Donde

si se cumple:

$$a = b + kn \qquad k \in \mathbb{Z}$$

El número natural *n* indicará *n* clases de equivalencia (números congruentes con 0, números congruentes con 1,..., números congruentes con *n*-1). Podemos definir ahora las operaciones suma y producto en ese conjunto de clases de equivalencia:

- $a+b \equiv c \pmod{n} \Leftrightarrow a+b=c+kn \qquad k \in \mathbb{Z}$
- $ab \equiv c \pmod{n} \Leftrightarrow ab=c+kn \qquad k \in \mathbb{Z}$

La operación suma en este conjunto cumple las propiedades asociativa y conmutativa y posee elementos neutro y simétrico, por lo que el conjunto tendrá estructura de grupo conmutativo. A partir de ahora llamaremos grupo finito inducido por *n* a dicho conjunto.

Con la operación producto se cumplen las propiedades asociativa y conmutativa, y tiene elemento neutro, pero no necesariamente simétrico. La estructura del conjunto con las operaciones suma y producto es de anillo conmutativo.

4.2.- Algoritmo de Euclides.

Quizá sea el algoritmo más antiguo que se conoce, y a la vez es uno de los más útiles. Permite calcular el máximo común divisor de dos números.

Sean *a* y *b* dos números enteros de los que queremos calcular su máximo común divisor *m*, el algoritmo de Euclides explota la siguiente propiedad:

$$m \mid a \wedge m \mid b \Rightarrow m \mid (a - kb) con\ k \in \mathbb{Z} \Rightarrow m \mid (a \bmod b)$$

a|*b* quiere decir que *a* divide a *b*, o en otras palabras, que *b* es múltiplo de *a*.

Si llamamos a *c* a *(a mod b)*, podemos aplicar de nuevo la propiedad y tenemos:

$$m \mid (b \bmod c)$$

Sabemos, pues, que *m* tiene que dividir a todos los restos que vayamos obteniendo. Es evidente que el último resto será cero, puesto que los restos siempre son inferiores al divisor. El penúltimo valor obtenido es el mayor número que divide tanto a *a* como a *b*. El algoritmo queda entonces como sigue:

```
int euclides(int a,int b)
{
    int i;
    g[0]=a;
    g[1]=b;
    i=1;
      while (g[i]!0)
       {
           g[i+1]=g[i-1]%g[i];
           i++;
       }
    return(g[i-1]);
}
```

El invariante del algoritmo es el siguiente:

$$g_{i+1} = g_{i-1} (\bmod g_i)$$

4.3.- Cálculo de Inversas en Aritmética Modular.

4.3.1.- Existencia de la Inversa.

Hemos comentado anteriormente que los elementos de un grupo finito no tienen por qué tener inversa (elemento simétrico para el producto). Veremos qué condiciones han de cumplirse para que exista la inversa de un número dentro de un grupo finito. *lema*: Dados $a,n \in \mathbb{N}$

$$\mathrm{mcd}(a,n)=1 \Rightarrow ai \neq aj \pmod{n} \qquad \forall i \neq j \quad 0<i,j<n$$

donde mcd(a,n) es el máximo común divisor de a y n.

Demostración: Supongamos que *mcd(a,n)* = 1, y que existen $i \neq j$ tales que $ai \equiv aj \pmod{n}$. Se cumple, pues:

$$(ai-aj) \mid n \Rightarrow a(i-j) \mid n$$

puesto que *a* y *n* son primos entre si, *a* no puede dividir a *n*, luego

$$(i-j) \mid n \Rightarrow i \equiv j \pmod{n}$$

con lo que hemos alcanzado una contradicción.

Ahora podemos hacer la siguiente reflexión: Si $ai \neq aj$ para cualesquiera $i \neq j$, multiplicar *a* por todos los elementos del grupo finito módulo *n* nos producirá una permutación de los elementos del grupo (exceptuando el cero), por lo que forzosamente ha de existir un valor tal que al multiplicarlo por *a* nos de 1.

Eso nos conduce al siguiente teorema:

Teorema: Si *mod(a,n)* = 1, a tiene inversa módulo *n*.

Corolario: Si *n* es primo, el grupo finito que genera tiene estructura de cuerpo (todos sus elementos tienen inversa para el producto excepto el cero).

4.4.- Función de Euler.

Llamaremos conjunto reducido de residuos módulo *n* al conjunto de números primos relativos con *n*. en otras palabras, todos los números que tienen inversa módulo *n*. Por ejemplo, si *n* fuera 12, su conjunto reducido de residuos seria:

{1,5,7,11}

El cardinal del conjunto reducido de residuos módulo n es igual a:

$$\Phi(n)=\prod_{i=1}^{n} p_i^{e_i-1}(p_i-1)$$

Donde $\Phi(n)$ es la función de Euler de *n*.

siendo p_i los factores primos de *n* y e_i su multiplicidad. Por ejemplo, si *n* fuera el producto de dos números primos *p* y *q*, $\Phi(n)=(p-1)(q-1)$.

$\Phi(n)$ se conoce como la función de Euler sobre *n*.

Teorema: Si *mcd (a,n)* = 1:

$$a^{\Phi(n)} \equiv 1 \pmod{n}$$

Demostración: Puesto que *a* y *n* son primos entre si, a multiplicarlo por cualquier elemento del conjunto reducido de residuos módulo *n* $\{r_1,...,r_{\Phi(n)}\}$ ha de ser también primo con *n*, por lo tanto el conjunto $\{ar_1,...,ar_{\Phi(n)}\}$ no es más que una permutación del conjunto anterior, lo cual no lleva a:

$$\prod_{i=1}^{\Phi(n)} r_i = \prod_{i=1}^{\Phi(n)} ar_i = a^{\Phi(n)} \prod_{i-1}^{\Phi(n)} r_i \Rightarrow a^{\Phi(n)} \equiv 1 \pmod{n}$$

Teorema de Fermat: Si *p* es primo, entonces

$$a^{p-1} \equiv 1 \pmod{n}$$

Podemos emplear, pues, la función de Euler para calcular inversas módulo *n*, puesto que:

$$a^{\Phi(n)} = aa^{\Phi(n)-1} \equiv 1 \pmod{n} \Rightarrow a^{a-1} \equiv a^{\Phi(n)-1} \pmod{n}$$

4.5.- Algoritmo Extendido de Euclides.

El Algoritmo Extendido de Euclides simplemente tiene en cuenta los cocientes en cada paso además de los restos. El invariante que mantiene es el siguiente, suponiendo que se le pase como parámetros *n* y *a*:

$$g_i = nu_i + av_i$$

El último valor de g_i será 1 si *a* y *n* son primos relativos, por lo que tendremos:

$$1 = nu_i + av_i$$

$(v_i \bmod n)$ será entonces la inversa de *a* módulo *n*.

Nuestra segunda alternativa para calcular inversas, cuando desconozcamos $\Phi(n)$, será pues el Algoritmo Extendido de Euclides:

```
int euclides_ext(int n, int a)
{
    int c,i;
    g[0]=n;
    g[1]=a;
    u[0]=1;
    u[1]=0;
    v[0]=0;
    v[1]=1;
    i=1;
    while (g[i]!=0
       {
           c=g[i-1]/g[i];
           g[i+1]=g[i-1]%g[i];
           u[i+1]=u[i-1]-c*u[i];
           v[i+1]=v[i-1]-c*v[i];
           i++;
       }
    return(v[i-1]%n);
}
```

5.- Exponenciación y Logaritmos Discretos.

Muchos de los algoritmos simétricos o de clave pública emplean exponenciaciones dentro de grupos finitos para codificar los mensajes. Tanto las bases como los exponentes en esos casos son números astronómicos, de incluso miles de bits de longitud. Efectuar las exponenciaciones mediante multiplicaciones reiteradas de la base sería inviable. Veremos mecanismos eficientes para llevar a cabo estas operaciones. También comentaremos brevemente el problema inverso, el cálculo de los logaritmos discretos, puesto que en él basan muchos algoritmos su resistencia.

5.1.- Algoritmo de Exponenciación Rápida.

Supongamos que tenemos dos números naturales *a* y *b*, y queremos calcular a^b, el mecanismo más sencillo sería multiplicar *a* por si mismo *b* veces. Sin embargo, para valores muy grandes de *b* este algoritmo no nos sirve.

Tomemos la representación binaria de *b*:

$$b = 2^0 b_0 + 2^1 b_1 + 2^2 b_2 + ... + 2^n b_n$$

Expresemos la potencia que vamos a calcular en función de dicha representación:

$$a^b = a^{2^0 b_0 + 2^1 b_1 + 2^2 b_2 + ... + 2^n b_n} = \prod_{i=0}^{n} a^{2^i b_i}$$

recordemos que los b_i sólo pueden valer 0 ó 1, por tanto para calcular a^b sólo hemos de multiplicar los a^{2^i} correspondientes a los dígitos binarios de b que valgan 1.

Nótese, además, que $a^{2^i} = \left(a^{2^{i-1}}\right)^2$, por lo que, partiendo de *a*, podemos calcular el siguiente valor de esta serie elevando al cuadrado el anterior. El Algoritmo de Exponenciación Rápida queda como sigue:

```
int exp_rapida(int a, int b)
   {
      int z,x,resul;
      z=b;
      x=a;
      resul=1;
      while (z%1=1)
      {
       if (z%1==1)
              resul=resul*x;
       x=x*x;
       z=z/2;
      }
      return(resul);
}
```

La variable *z* se inicializa con el valor de *b* y se va dividiendo por 2 en cada paso para tener siempre el i-ésimo bit de *b* en el menos significativo de *z*. En la variable *x* se almacenan los valores de a^{2^i}.

5.2.- El Problema de los Logaritmos Discretos.

El problema inverso de la exponenciación es él calculo de logaritmos discretos. Dados dos números *a,b* y el módulo *n*, se define el logaritmo discreto de *a* en base *b* módulo *n* como:

$$c = \log_b(a) \pmod n \Leftrightarrow a \equiv b^c \pmod n$$

En la actualidad no existen algoritmos eficientes que sean capaces de calcular en tiempo razonable logaritmos de esta naturaleza, y muchos esquemas criptográficos basan su resistencia en esta circunstancia. El problema de los logaritmos discretos está íntimamente relacionado con el de la factorización, de hecho está demostrado que si se puede calcular un logaritmo, entonces se puede factorizar fácilmente (el recíproco no se ha podido demostrar).

5.3.- Factorización.

5.3.1.- Prueba de Primalidad.

Para explotar la dificultad de cálculo de logaritmos discretos, muchos algoritmos criptográficos de clave pública se basan en operaciones de exponenciación en grupos finitos. Dichos conjuntos deben cumplir la propiedad de que su módulo *n* sea un número muy grande con pocos factores (usualmente dos). Estos algoritmos funcionan si se conoce *n* y sus factores se mantienen en secreto.

Habitualmente para obtener *n* se calculan primero dos números primos muy grandes, que posteriormente se multiplican. Necesitamos pues mecanismos para poder calcular esos números primos grandes

La factorización es el problema inverso a la multiplicación: dado *n*, se trata de buscar un conjunto de números tales que su producto valga *n*. Normalmente, y para que la solución sea única, se impone la condición de que los factores de *n* que obtengamos sean todos primos. Al igual que para el problema de los logaritmos discretos, no existen algoritmos eficientes para efectuar este tipo de cálculos, siempre y cuando los factores (como veremos más adelante) hayan sido escogidos correctamente. Esto nos permite confiar en que los factores de *n* serán imposibles de calcular aunque se conozca el propio *n*.

En cuanto al cálculo de primos grandes, bastaría con aplicar un algoritmo de factorización para saber si un número es primo o no, este mecanismo es inviable, puesto que acabamos de decir que no hay algoritmos eficientes de factorización. Por suerte, si existen algoritmos probabilísticos que permiten decir con un grado de certeza bastante elevado si un número cualquiera es primo o compuesto.

Cabría preguntarse, dado que para los algoritmos asimétricos de cifrado necesitaremos generar muchos números primos, si realmente hay suficientes. De hecho se puede pensar que, a fuerza de generar números, llegara un momento en el que repitamos un primo generado con anterioridad. Podemos estar tranquilos, porque si a cada átomo del universo le asignáramos mil millones de números primos cada microsegundo desde su origen, harían falta un total de 10^{109} números primos diferentes, mientras que el total estimado de números primos de 512 bits o menos es aproximadamente de 10^{151}.

También podríamos pensar en calcular indiscriminadamente números primos para poder emplearlos en algún algoritmo de factorización rápida. Por desgracia, si quisiéramos construir un disco duro que albergara diez mil GBytes por cada gramo de masa o milímetro cúbico para almacenar todos los primos de 512 bits o menos, ese disco duro pesaría tanto que colapsaría en un agujero negro.

5.3.2.- Método de Lehmann.

Es una de las pruebas más sencillas para saber si un número p es o no primo:

1. Escoger un número aleatorio a < p.
2. Calcular $b = a^{(p-1)/2} \pmod{p}$.
3. Si $b \neq 1 \pmod{p}$ y $b \neq -1 \pmod{p}$, p no es primo.
4. Si $b \equiv 1 \pmod{p}$ o $b \equiv -1 \pmod{p}$, la probabilidad de que p sea primo es igual o superior al 50%.

Repitiendo el algoritmo n veces, la probabilidad de que *p* supere la prueba y sea compuesto (es decir, no-primo) será de 1 contra 2^n.

5.3.3.- Método de Rabin-Miller.

Es el algoritmo más empleado, debido a su facilidad de implementación. Sea *p* el número que queremos saber si es primo. Se calcula *b*, siendo *b* el número de veces que 2 divide a *(p - 1)*, es decir, 2^b es la mayor potencia de 2 que divide a *(p – 1)*. Calculamos entonces *m*, tal que $p = 1 + 2^b * m$.

1. Escoger un número aleatorio $a < p$.
2. Sea $j = 0$ y $z = a^m \pmod p$.
3. Si $z = 1$, o $z = p - 1$, entonces p pasa la prueba y puede ser primo.
4. Si $j > 0$ y $z = 1$, *p* no es primo.
5. Sea $j = j + 1$. Si $j = b$ y $z \neq p - 1$, *p* no es primo.
6. Si $j < b$ y $z \neq p - 1$, $z = z^2 \pmod p$. Volver al paso (4).
7. Si $j < b$ y $z = p - 1$, entonces p pasa la prueba y puede ser primo.
8. p no es primo.

La probabilidad de que un número compuesto pase este algoritmo para un número *a* es del 25%. Esto quiere decir que necesitaremos menos pasos para llegar al mismo nivel de confianza que el obtenido con el Algoritmo de Lehmann.

5.3.4.- Consideraciones Prácticas

A efectos prácticos el algoritmo que se suele emplear para saber si un número es o no primo es el siguiente:

1. Generar un número aleatorio p de n bits.
2. Poner a uno el bit más significativo (garantizaríamos que el número es de n bits) y el menos significativo (debe ser impar para poder ser primo).
3. Intentar dividir p por una tabla de primos precalculados (usualmente se usan los primos menores que 2000). Esto elimina gran cantidad de números no primos de una forma muy rápida. Basta decir a titulo informativo que más del 99.8% de los números impares no primos es divisible por algún número primo menor que 2000.
4. Ejecutar la prueba de Rabin-Miller como mínimo cinco veces.

5.3.5.- Primos Fuertes

Aunque p y q sean primos grandes, existen algunos casos en los que es relativamente fácil factorizar el número $n = pq$. Se proponen entonces una serie de condiciones para p y q que dificultan la factorización de n. Se dice que p y q son números primos fuertes si cumplen:

- El máximo común divisor de $p-1$ y $q-1$ debe ser pequeño.
- $p-1$ y $q-1$ debe tener algún factor primo grande p' y q'.
- Tanto $p'-1$ como $q'-1$ deben tener factores primos grandes.
- Tanto $p'+1$ como $q+1$ deben tener factores primos grandes.

Las dos primeras condiciones se cumplen si tanto (p-1)/2 como (q-1)/2 son números primos.

6.- Aritmética Entera de Múltiple Precisión.

6.1.- Representación de Enteros Largos.

Un número largo es un número que posee muchos dígitos, normalmente más de los que los tipos de datos convencionales de los lenguajes de programación clásicos pueden soportar, vamos a indicar cómo representarlos usando tipos de dato de menor precisión.

Conocemos la representación tradicional en base 10 de los números, en la que cada cifra contiene únicamente valores de 0 a 9. Esta representación no es más que un caso particular (B=10) de la siguiente expresión general:

$$n = (-)\sum_{-\infty}^{\infty} a_i B^i$$

donde los términos con índice negativo corresponden a la parte no entera (decimal) del número real n. Sabemos que dicha representación es única, y que significa que n en base B se escribe:

$$(-)\, a_n a_{n-1} \ldots a_0 . a_{-1} \ldots$$

En cualquier caso, puesto que nuestro objetivo es representar únicamente números enteros positivos, prescindiremos del signo y de los términos con subíndice negativo.

Cualquier número vendrá representando por una serie única de coeficientes a_i (cifras), de las que importa tanto su valor como su posición dentro del número. Esta estructura corresponde claramente a la de un arreglo. Para representar de forma eficiente enteros largos emplearemos una base que se

potencia de dos (normalmente se escoge $B = 2^{16}$ ó $B = 2^{32}$ para que cada cifra de nuestro número se pueda almacenar en un dato del tipo *unsigned int* sin desperdiciar ningún bit). Para almacenar los resultados parciales de las operaciones aritméticas emplearemos un tipo de dato de doble precisión (*unsigned long int*, correspondiente a $B = 2^{32}$ o $B = 2^{64}$) de forma que no se nos desborde al multiplicar dos cifras. Normalmente se escoge una longitud que pueda manejar directamente la unidad aritmética lógica de la computadora, para que las operaciones elementales entre cifras sean rápidas.

Por todo esto, para nosotros un número entero largo será un arreglo (dinámico o no) de *unsigned int*. En cualquier caso, y a partir de ahora, nuestro objetivo será estudiar algoritmos eficientes para efectuar operaciones aritméticas sobre este tipo de números, independientemente de la base en la que se encuentren representados.

6.2.- Operaciones Aritméticas sobre Enteros largos.

Vamos a describir cómo realizar operaciones aritméticas (suma, resta, multiplicación y división).

6.2.1.- Suma.

La suma de $a = (a_0, a_1, ..., a_{n-1})$ y $b = (b_0, b_1, ..., b_{n-1})$ se puede definir como:

$$(a+b)_i \begin{cases} (a_i + b_i + c_i) \bmod B & \text{para } i = 0...n - 1 \\ c_i & \text{para } i = n \end{cases}$$

siendo

$$c_i = \begin{cases} 0 & \text{para } i = 0 \\ (a_{i-1} + b_{i-1} + c_{i-1}) \text{ div B} & \text{para } i = 1...n \end{cases}$$

c_i es el acarreo de la suma de los dígitos inmediatamente anteriores. Tenemos en cuenta el coeficiente n de la suma porque puede haber desbordamiento, en cuyo caso la suma tendría $n + 1$ dígitos y su cifra más significativa sería precisamente c_n.

El algoritmo para la suma quedaría como sigue:

```
suma(unsigned *a, unsigned *b, unsigned *s)
```

```
{
   unsigned long sum;
   unsigned acarreo;
   n = max(num. De dígitos de a, num. De dígitos de b)
   acarreo = 0;
   for (i=0;i<n;i++)
   {
     sum = acarreo + a[i] + b[i];
     s[i] = sum%BASE;
     acarreo = sum/BASE;
   }
   s[n]=acarreo;
}
```

El resultado se devuelve en s.

6.2.2.- Resta.

La resta es muy parecida a la suma, salvo que en este caso los acarreos se restan. Suponiendo que *a* > *b*:

$$(a-b)_i = (a_i - b_i - r_i) \bmod B \qquad \text{para } i = 0...n-1$$

siendo

$$r_i = \begin{cases} 0 & \text{para i} = 0 \\ 1-\left((a_{i-1} - b_{i-1} - r_{i-1})\, div\ B\right) & \text{para } i = 1...n \end{cases}$$

r_i representa el acarreo de la resta, que puede valer 0 o 1 según la resta parcial salga positiva o negativa. Nótese que, como a > b, el último acarreo siempre ha de valer 0.

```
Resta (unsigned *a, unsigned *b, unsigned *d)
    {
       unsigned long dif;
       unsigned acarreo;
       n=max(num. De dígitos de a, num. de dígitos de b)
       acarreo=0;
       for(i=0;i<n;i++)
       {
          dif=a[i]-b[i]-acarreo+BASE;
          d[i]=dif%BASE;
```

```
            acarreo=1-dif/BASE;
        }
    }
```

El resultado se devuelve en d. La razón por la que sumamos la base a *dif* es para que la resta parcial salga siempre positiva y poder hacer el módulo correctamente. En ese caso, si el valor era positivo, al sumarle *B* y dividir por *B* de nuevo acarreo el valor *i-dif/BASE*.

Nos queda comprobar cuál de los dos números es mayor para poder emplearlo como minuendo. Esta comprobación se puede realizar fácilmente definiendo una función que devuelva el número cuyo dígito más significativo tenga un número de orden mayor. En caso de igualdad iríamos comparando dígito a dígito, empezando por los más significativos hasta que encontremos alguno mayor o lleguemos al último dígito, situación que únicamente ocurrirá si los dos números son iguales.

6.2.3.- Multiplicación.

Para obtener el algoritmo del producto emplearemos la expresión general de un número entero positivo en base *B*. Si desarrollamos el producto de dos números cualesquiera *a* y *b* de longitudes *m* y *n* respectivamente nos queda:

$$ab = \sum_{i=0}^{m-1} a_i B^i b$$

A la vista de esto podemos descomponer el producto como *m* llamadas a una función que multiplica un entero largo por $a_i B^i$ (es decir, un entero largo con un único dígito significativo) y después sumar todos los resultados parciales.

Para poder implementar esto primero definiremos una función que multiplique *b* por $a_i B^i$ y el resultado se lo sume al vector *s*, que no tiene necesariamente que estar a cero:

```
summult(unsigned ai, int i, unsigned *b, int m, unsigned *s)
{
    int k;
    unsigned acarreo;
    unsigned long prod, sum;
    acarreo=0;
    for(k=0;k<m;k++)
    {
       prod=ai*b[k]+s[i+k]+acarreo;
       s[i+k]=prod%BASE;
```

```
            acarreo=prod/BASE;
        }
        k=m+1;
        while(acarreo!=0)
        {
            sum=s[k]+acarreo;
            s[k]=sum%BASE;
            acarreo=prod/BASE;
            k++;
        }
    }
```

La segunda parte de la función se encarga de acumular los posibles acarreos en el vector *s*. A partir de la función que acabamos de definir, queda entonces como sigue:

```
producto(unsigned *a,int m,unsigned *b, int n, unsigned *p)
{
   int k;
   for(k=0;k<=m+n;k++) p[k]=0;
   for(k=0;k<m;k++)    summult(a[k],k,b,n,p);
}
```

El resultado se devuelve en *p*.

Existe otra propiedad de la multiplicación de enteros que nos va a permitir efectuar estas operaciones de manera más eficiente. Tomemos un número entero que nos va a permitir efectuar estas operaciones de manera más eficiente. Tomemos un número entero cualquiera *a* de *k* dígitos en base *B*. Dicho número puede ser representado mediante la siguiente expresión:

$$a = a_l B^{\frac{k}{2}} + a_r$$

Es decir, *partimos a* en dos mitades. Por razones de comodidad, llamaremos B_k a $B^{\frac{k}{2}}$.

Veamos ahora, cómo queda el producto de dos números cualesquiera a y b, en función de sus respectivas *mitades*:

$$ab = a_l b_l B_k^2 + \left(a_l b_r + a_r b_l\right) B_k + a_r b_r$$

Hasta ahora no hemos aportado nada nuevo. El truco para que este desarrollo nos proporcione un aumento de eficiencia consiste en hacer uso de la siguiente propiedad:

$$a_l b_r + a_r b_l = a_l b_r + a_r b_l + a_l b_l - a_l b_l + a_r b_r - a_r b_r = (a_l + a_r)(b_l + b_r) - a_l b_l - a_r b_r$$

quedando finalmente, lo siguiente:

$$x = a_l b_l \qquad y = (a_l + a_r)(b_l + b_r) \qquad z = a_r b_r$$

$$ab = xB_k^2 + (y - x - z)B_k + z$$

Hemos reducido los cuatro productos y tres sumas del principio a tres productos y seis sumas. Como es lógico, esta técnica debe emplearse dentro de una estrategia *divide* y *vencerás*.

6.2.4.- División.

El algoritmo más simple para dividir dos números se consigue a partir de su representación binaria:

```
cociente_bin(unsigned *c, unsigned *d, unsigned *a, unsigned *b)
  {
          calcular a = c div d
                   b = c mod d
    Bits_significativos(x) => Siendo x un digito, devuelve el numero
                              de bits de los dígitos quitando los
                              ceros de la izquierda.
    pow(a,b) => Calcula el valor de a elevado a b.
    Poner_bit_a_1(a,x) => Pone a 1 el i-ésimo bit de a.
    Poner_bit_a_0(a,x) => Pone a 0 el i-ésimo bit de a.
    m=Bits_Significativos(c);
    n=Bits_Significativos(d);
    b=c;
    a=0;
    d1=d*pow(2,m-n);
    for(i=m-n;i>=0;i--)
          {
            if(b>d1)
            {
              poner_bit_a_1(a,1);
              b=b-d1;
            }
            else  poner_bit_a_0(a,i);
            d1=d1/2;
          }
    }
```

El funcionamiento del algoritmo es extremadamente simple: copiamos el dividendo en *b* y desplazamos a la izquierda el divisor hasta que su longitud coincida con la del dividendo. Si el valor resultante es menor que *b*, se los restamos y ponemos a 1 el bit correspondiente a *a*.

Repitiendo esta operación sucesivamente se obtiene el cociente en *a* y el resto en *b*, a modo de ejemplo, dividiremos 37 (100101) entre 7(111):

$b = 100101; \quad a = ----; \quad d1 = 111000; \quad b \not> d1 \rightarrow a = ----$

$b = 100101; \quad a = 0---; \quad d1 = 11100; \quad b > d1 \rightarrow a = 01--$

$b = 001001; \quad a = 01--; \quad d1 = 110; \quad b \not> d1 \rightarrow a = 010-$

$b = 001001; \quad a = 010-; \quad d1 = 111; \quad b > d1 \rightarrow a = 0101$

$b = 010$

Este algoritmo resulta muy lento, ya que opera a nivel de bit, por lo que intentaremos encontrar otro más rápido –aunque con el mismo orden de eficiencia--. Nos basaremos en el tradicional de la división. Suponiendo que queremos dividir *c* por *d*, obteniendo su cociente (*a*) y resto (*b*), iremos calculando cada dígito del cociente en base *B* de un solo golpe. Nuestro objetivo será estimar a la baja el valor de cada uno de los dígitos *a*, e incrementando hasta el valor correcto. Para que la estimación se quede más cerca posible del valor correcto efectuaremos una normalización de los números, de forma que el dígito más significativo *d* tenga su bit de mayor peso a 1, esto se consigue multiplicando *c* y *d* por 2^k, siendo *k* el número de ceros a la izquierda del bit más significativo del divisor *d*. Posteriormente habremos de tener en cuenta lo siguiente:

$$c = ad + b \Leftrightarrow 2^k c = a\left(2^k d\right) + 2^k b$$

luego el cociente será el mismo pero el resto habrá que dividirlo por el factor de normalización. Llamaremos $\overline{c}$ con $\overline{d}$ a los valores de *c* y *d* normalizados.

Para hacer la estimación a la baja de los a_i, dividiremos $c_j B + c_{j-1}$ por $\overline{d_m} + 1$ (c_j es el dígito más significativo de *c* en el paso i, y $\overline{d}_m$ es el dígito más significativo de $\overline{d}$). Luego actualizamos $\overline{c}$ con $\overline{c} - \overline{d} a_i B^i$ y vamos incrementando a_i (y actualizando $\overline{c}$) mientras nos quedamos cortos. Finalmente, habremos calculado el valor del cociente (*a*) y el valor del resto será

$$b = \frac{\overline{b}}{2^k}$$

El algoritmo podría quedar como sigue:

```
cociente(unsigned *c, unsigned *d, unsigned *a, unsigned *b)
{
  /* calcular a=c div d
     b=c mod d

   Digito_mas_Significativo(a) => Devuelve el valor del digito de mayor peso de a.
   Bits_significativos(x) => Siendo x un digito, devuelve el numero de bits de los
                             dígitos quitando los ceros de la izquierda.
   Pow(a,b)  => Calcula el valor de a elevado a b.
  */
   despl=Num_bits_digito -
            Bits_significativos(digitos_mas_significativos(d));
   factor=pow(2,despl.);
   dd=d*factor;
   cc=c*factor;
   if(digitos(cc)==digitos(c)) poner_un_cero_a_la_izquierda(cc);
   t=digitos_mas_significativos(dd);
   poner_a_cero(a);
   for(i=digitos(c)-digitos(dd);i>=0;i--)
   {
      if(t==B-1)       /* No podemos dividir por t+1 */
         ai=cc[i+digitos(dd)];
      else
         ai=(cc[i+digitos(dd)]*B+cc[i+digitos(dd)-1]/(t+1);
      cc=cc-a1*ss*por(B,i);
      while(cc[i+digitos(dd)] || mayor(cc,dd*pow(B,i)))
      {
        ai++;
        cc=cc-dd*pow(B,i);
      }
      a[i]=ai;
   }
   b=cc/factor; /* lo que nos queda en cc es el resto */
                /* dividimos por factor para deshacer */
                /* la normalización */
}
```

Aunque a primera vista pueda parecer un algoritmo complejo, vamos a ver que no es tan complicado siguiendo su funcionamiento para un ejemplo concreto, con B = 16, c = 3FBA2, y d = 47:

1. Normalización: multiplicamos por 2 y nos queda $\overline{c} = 7F744$, $\overline{d} = 8E$
2. $a_2 = 7F \ div \ 9 = E$; $\overline{c} = \overline{c} - a_2\overline{d}B^2 = 7F744 - 7C400 = 3344$

   Puesto que $\overline{c} < \overline{d}B^2 = 8E00$, no hay que incrementar a_2.
3. $a_1 = 33 \text{ div } 9 = 5$; $\overline{c} = \overline{c} - a_1\overline{d}B = 3344 - 2C60 = 6E4$

Puesto que $\overline{c} < \overline{dB} = 8E0$, no hay que incrementar a_i.

4. $a_0 = 6E \text{ div } 9 = C;\ \overline{c} = \overline{c} - a_0\overline{d} = 6E4 - 6A8 = 3C$

   Puesto que $\overline{c} < \overline{d} = 8E$, tampoco hay que incrementar a_0

5. $a = E5C;\ \text{b} = \frac{\overline{c}}{2} = 1E$.

6.3.- Aritmética Modular con Enteros Largos.

Los algoritmos criptográficos de llave pública más extendidos se basan en operaciones modulares sobre enteros muy largos. Empleado los algoritmos anteriores son inmediatas las operaciones de suma, resta y multiplicación modulo n. La división habremos de tratarla de manera diferente.

- Para sumar dos números módulo n basta con efectuar su suma entera y, si el resultado es mayor que n, restar el módulo.
- Para restar basta con sumar el módulo n al minuendo, restarle el substraendo, y si el resultado es mayor que n, restar el módulo.
- El producto se lleva a cabo multiplicando los factores y tomando el resto de dividir el resultado por el módulo.
- La división habremos de implementarla multiplicando el dividendo por la inversa del divisor. Para calcular la inversa de un número módulo n basta con emplear el Algoritmo Extendido de Euclides, substituyendo las operaciones elementales por llamadas a las operaciones con enteros largos descritas anteriormente.

7.- Principios de Complejidad Algorítmica.

Cuando diseñamos un algoritmo criptográfico pretendemos plantear a un posible atacante un problema que éste sea incapaz de resolver. Pero, ¿bajo qué circunstancia podemos considerar que un problema es *intratable*? Evidentemente, queremos que nuestro atacante se enfrente a unos requerimientos de computación que no pueda asumir. La cuestión es ¿como modelizar y cuantificar la capacidad de calculo necesaria para abordar un problema?. A continuación daremos un breve repaso de las herramientas formales que nos van a permitir dar respuestas a estos interrogantes.

7.1.- Concepto de Algoritmo.

En la actualidad, prácticamente a todas las aplicaciones criptográficas emplean computadoras en sus cálculos, y las computadoras convencionales están diseñadas para ejecutar algoritmos.

Definiremos algoritmo como una secuencia finita y ordenada de instrucciones elementales que, dados los valores de entrada de un problema, en algún momento finaliza y devuelve la solución.

En efecto, las computadoras actuales poseen una memoria, que les sirve para almacenar datos, unos dispositivos de entrada y salida que les permiten comunicarse con el exterior, una unidad capaz de hacer operaciones aritméticas y lógicas, y una de control, capaz de leer, interpretar y ejecutar un programa o secuencia de instrucciones. Habitualmente, las unidades aritmético-lógica y de control se suelen encapsular en un único circuito integrado, que se conoce por microprocesador o CPU.

Cuando nosotros diseñamos un algoritmo de cifrado, estamos expresando, de un modo más o menos formal, la estructura que ha de tener la secuencia de instrucciones concreta que permita implementar dicho algoritmo en cada computadora particular. Habrá computadoras con más o menos memoria, velocidad o incluso número de procesadores –capaces de ejecutar varios programas al mismo tiempo-, pero en esencia todas obedecerán al concepto de algoritmo.

La Teoría de Algoritmos es una ciencia que estudia como construir algoritmos para resolver diferentes problemas. En muchas ocasiones no basta con encontrar una forma de solucionar el problema: la solución ha de ser óptima. En este sentido la Teoría de Algoritmos también proporciona herramientas formales que nos van a permitir decidir que algoritmo es mejor en cada caso, independientemente de las características particulares de la computadora concreta en la que queramos implantarlo.

La criptografía depende en gran medida de la Teoría de Algoritmos, ya que por un lado hemos de asegurar que el usuario legítimo, que posee la clave, puede cifrar y descifrar la información de forma rápida y cómoda, mientras que por otro hemos de garantizar que un atacante no dispondrá de ningún algoritmo eficiente capaz de comprometer el sistema.

Cabria plantearnos ahora la siguiente cuestión: si un mismo algoritmo puede resultar más rápido en una computadora que en otra, ¿podría existir una computadora capaz de ejecutar de forma eficiente algoritmos que sabemos que no lo son?. Existen un principio fundamental en Teoría de Algoritmos, llamado principio de invarianza, que dice que si dos implementaciones del mismo algoritmo consumen $t_1(n)$ y $t_2(n)$ segundos respectivamente, siendo n el tamaño de los datos de entrada, entonces existe una constante positiva c tal que $t_i(n) \leq c \cdot t_2(n)$, siempre que n sea lo suficientemente grande. En otras palabras, que aunque podamos encontrar una computadora más rápida, o una implementación mejor, la evolución del tiempo de ejecución del algoritmo en función del tamaño del problema permanecerá constante, por lo tanto la respuesta a la pregunta anterior es,

afortunadamente, negativa. Eso nos permite centrarnos por completo en el algoritmo en si y olvidarnos de la implementación concreta.

En muchas ocasiones, el tiempo de ejecución de un algoritmo viene dado por las entradas concretas que le introduzcamos. Por ejemplo, se necesitan menos operaciones elementales para ordenar de menor a mayor la secuencia {1,2,3,4,6,5} que {6,5,3,2,1,4}. Eso nos llevará a distinguir entre tres alternativas:

- Mejor caso: es el número de operaciones necesario cuando los datos se encuentran distribuidos de la mejor forma posible para el algoritmo. Evidentemente este caso no es muy práctico, puesto que un algoritmo puede tener un mejor caso muy bueno y comportarse muy mal en el resto.
- Peor caso: Es el número de operaciones necesario para la distribución más pesimista de los datos de entrada. Nos permitirá obtener una cota superior del tiempo de ejecución necesario. Un algoritmo que se comporte bien en el peor caso, será siempre un buen algoritmo.
- Caso promedio: Muchas veces, hay algoritmos que en un peor caso no funcionan bien, pero en la mayoría de los casos que se presentan habitualmente tienen un comportamiento razonablemente eficiente. De hecho, algunos algoritmos típicos de ordenación necesitan el mismo número de operaciones en el peor caso, pero se diferencian considerablemente en el caso promedio.

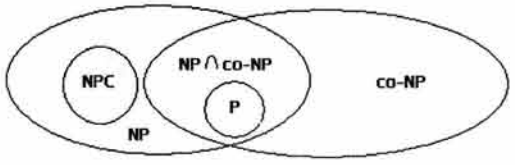
7.2.- Clases de Complejidad.

Para simplificar la notación, en muchas ocasiones se suele reducir el problema de la complejidad algorítmica a un simple problema de decisión, de forma que se considera un algoritmo como un mecanismo que permite obtener una respuesta *si* o *no* a un problema concreto.

- *La clase de complejidad P*: es el conjunto de todos los problemas de decisión que pueden ser resueltos en tiempo polinomial.
- *La clase de complejidad NP*: es el conjunto de todos los problemas para los cuales una respuesta afirmativa puede ser verificada en tiempo polinomial, empleando alguna información extra, denomina *certificado*.
- *La clase de complejidad co-NP*: es el conjunto de todos los problemas para los cuales una respuesta negativa puede ser verificada en tiempo polinomial, usando un certificado apropiado.

Nótese que el hecho de que un problema sea *NP*, no quiere decir necesariamente que el certificado correspondiente sea fácil de obtener, sino que, dado éste último, puede verificarse la respuesta afirmativa en tiempo polinomial. Una observación análoga puede llevarse a cabo sobre los problemas *co-NP*.

Sabemos que $P \subseteq NP$ y que $P \subseteq co\text{-}NP$. Sin embargo, aún no se sabe si $P = NP$, si $NP = co\text{-}NP$, o si $P = NP \cap co\text{-}NP$. Si bien muchos expertos consideran que ninguna de estas tres igualdades se cumple, este punto no ha podido ser demostrado matemáticamente.



Relación entre las clases de complejidad *P*, *NP*, *co-NP* y *NPC*.

Dentro de la clase NP, existe un subconjunto de problemas que se llaman NP-complejos, y cuya clase se nota como NPC. Estos problemas tienen la peculiaridad de que todos ellos son equivalentes, es decir, se pueden reducir unos en otros, y si lográramos resolver alguno de ellos en tiempo polinomial, los habríamos resuelto todos. También se puede decir que cualquier problema NP-complejo es al menos tan difícil de resolver como cualquier otro problema NP, lo cual hace a la clase NPC la de los problemas más difíciles de resolver computacionalmente.

Sea $A=\{a_1,a_2,...,a_n\}$ un conjunto de números enteros positivos, y es otro número entero positivo. El problema *NP*-complejo, y, como ya se ha dicho, todos los problemas de esta clase pueden ser reducidos a una instancia de este. Nótese que dado un subconjunto de A, es muy fácil verificar si suma *s*, y que dado un subconjunto de A que sume s –que desempeñaría el papel de certificado–, se puede verificar fácilmente que la respuesta al problema es afirmativa.

En la gráfica anterior se puede observarse gráficamente la relación existente entre las distintas clases de complejidad que acabamos de definir.

Finalmente, existe una clase de problemas, los denominados NP –duros esta clase se define sobre los problemas en general, no sólo sobre los de decisión–, y que contiene la versión computacional del problema definido anteriormente, que consistiría en encontrar el subconjunto de A cuyos elementos suman s.

7.3.- Algoritmos Probabilísticos.

Hasta ahora hemos estudiado la complejidad de algoritmos de tipo determinístico, que siempre siguen el mismo camino de ejecución y que siempre llegan –si lo hacen– a la misma solución. Sin embargo, existen problemas para los cuales pueden ser más interesante emplear algoritmos de tipo determinísticos, también llamados probabilísticos o aleatorizados. Este tipo de algoritmos maneja algún tipo de parámetro aleatorio, lo cual hace que dos ejecuciones diferentes con los mismos datos de entrada no tengan por qué ser idénticas. En algunos casos, métodos de este tipo permiten obtener soluciones en una cantidad de tiempo considerablemente inferior a los determinísticos.

Podemos clasificar los algoritmos no determinísticos según la probabilidad con la que devuelvan la solución correcta. Sea A un algoritmo aleatorizado para el problema de decisión L, y sea I una instancia arbitraria de L. Sea P1 la probabilidad de que A devuelva cierto cuando I es cierto, y P2 la probabilidad de que A devuelva cierto cuando I es falso.

- A es de tipo error nulo si P1 = 1 y P2 = 0.
- A es de tipo error simple si $P1 \geq c$, siendo c una constante positiva, y P2 = 0.
- A es de tipo error doble si $P1 \geq \frac{1}{2} + \varepsilon,\ y\ P2 \leq \frac{1}{2} - \varepsilon$.

Definiremos también el tiempo esperado de ejecución esperado para cada entrada, expresado en función del tamaño de la entrada. El tiempo de ejecución esperado para cada entrada será la media de los tiempos obtenidos para esa entrada y todas las posibles salidas del generador aleatorio.

Las clases de complejidad probabilística son las siguientes:

- Clase *ZPP*: conjunto de todos los problemas de decisión para los cuales existe un algoritmo de tipo *error nulo* que se ejecuta en un tiempo esperado de ejecución polinomial.
- Clase *RP*: conjunto de los problemas de decisión para los cuales existe un algoritmo de tipo *error simple* que se ejecuta en el peor caso en tiempo polinomial.
- Clase *BPP*: conjunto de los problemas de decisión para los cuales existe un algoritmo de tipo *error doble* que se ejecuta en el peor caso en tiempo polinomial.

Finalmente, diremos que $P \subseteq ZPP \subseteq RP \subseteq BPP$ y $RP \subseteq NP$.

7.4.- Complejidad Algorítmica y Criptografía.

Se ha contemplado únicamente aquellos problemas para los que existe una solución algorítmica –el programa finaliza siempre, aunque necesite un número astronómico de operaciones elementales– , hemos dejado a un lado deliberadamente aquellos problemas para los cuales no existen algoritmos cuya finalización esté garantizada (problemas no-decidibles y semidecidibles).

Se han repasado las clases genéricas de problemas que se pueden afrontar, en función del tipo de algoritmos que permiten resolverlos. Se ha puesto de manifiesto asimismo que un algoritmo ineficiente, cuando el tamaño de la entrada es lo suficientemente grande, es totalmente inabordable incluso para la más potente de las computadoras, al menos con la tecnología actual.

El hecho que no se conozca un algoritmo eficiente para resolver un problema no quiere decir que éste no exista, y por eso es tan importante la Teoría de Algoritmos para la Criptografía. Si, por ejemplo, se lograra descubrir un método eficiente capaz de resolver logaritmos discretos, algunos de los algoritmos asimétricos más populares en la actualidad dejarían de ser seguros. De hecho, la continua reducción del tiempo de ejecución necesario para resolver ciertos problemas, propiciada por la aparición de algoritmos más eficientes, junto con el avance de las prestaciones del hardware disponible, obliga con relativa frecuencia a actualizar las previsiones sobre la seguridad de muchos sistemas criptográficos.

8.- Criptografía y Números Aleatorios.

Los algoritmos de clave pública, debido a su mayor orden de complejidad, suelen ser empleados en conjunción de algoritmos simétricos de la siguiente forma: el mensaje primero se codifica empleando un algoritmo simétrico y la llamada clave de sesión, que será diferente cada vez. En la clave de sesión la que se codifica empleando criptografía asimétrica. La única manera de que estas claves sean seguras es que no exista ningún tipo de dependencia entre una clave y la siguiente, esto es, que sean aleatorias. De aquí surge el interés de los números aleatorios en Criptografía.

Los generadores tradicionales no nos permiten calcular secuencias realmente aleatorias, puesto que conociendo un número obtenido con el generador podemos determinar cualquiera de los posteriores – recordemos que cada elemento de la secuencia se emplea como semilla para el siguiente-. Si bien las series que producen estos generadores superan las pruebas estadísticas de aleatoriedad, son totalmente previsibles, y esa condición es inadmisible para aplicaciones criptográficas. Un famoso ejemplo ocurrió en una de las primeras versiones del navegador Web Netscape, que resulta insegura debido al uso de un generador pseudoaleatorio demasiado previsible.

Aquí vamos a caracterizar diferentes tipos de secuencias aleatorias, así como su interés en Criptografía. También veremos como implementar un buen generador aleatorio útil desde el punto de vista criptográfico.

8.1.- Tipos de Secuencias Aleatorias.

8.1.1.- Secuencias Pseudoaleatorias.

En realidad es casi del todo imposible generar secuencias auténticamente aleatorias en una computadora, puesto que estas máquinas son –al menos en teoría- completamente deterministas.

Todos los generadores pseudoaleatorios producen secuencias finitas y periódicas de números empleando operaciones aritméticas y lógicas. Lo único que podremos conseguir es que estas secuencias sean lo más largas posibles antes de comenzar a repetirse y que superen los pruebas estadísticas de aleatoriedad. En este sentido podríamos hablar de:

- Secuencias estadísticamente aleatorias: Secuencias que superan las pruebas estadísticas de aleatoriedad.

Un generador congruencia lineal cumple esta propiedad, pero en Criptografía será del todo inútil, debido a que cada valor de la secuencia se emplea como semilla para calcular el siguiente, lo cual nos permite conocer toda la serie a partir de un único valor. Supongamos que tenemos un sistema que se basa en emplear claves aleatorias para cada sesión y usamos un generador de este tipo.

Bastaría con que una de las claves quedara comprometida para que todas las comunicaciones –pasadas y futuras- pudieran ser descifradas sin problemas. Incluso se ha demostrado que conociendo únicamente un bit de cada valor de la secuencia ésta puede ser recuperada completamente con una cantidad relativamente pequeña de valores.

8.1.2.- Secuencias Criptográficamente Aleatorias.

El problema de las secuencias estadísticamente aleatorias, y lo que las haces poco útiles en criptografía, es que son completamente predecibles. Definiremos, por tanto:

- Secuencia criptográficamente aleatoria: Para que una secuencia psedoaleatoria sea criptográficamente aleatoria, ha de cumplir la propiedad de ser impredecible.

Esto quiere decir que debe ser computacionalmente intratable el problema de averiguar el siguiente número de la secuencia, teniendo total conocimiento acerca de todos los números anteriores y del algoritmo de generación empleado.

Existen generadores pseudoaleatorios criptográficamente resistentes que cumplen esta propiedad. Sin embargo no son suficientes para nuestros propósitos, debido a que se necesita una semilla para inicializar el generador. Si un atacante lograra averiguar la semilla que estamos empleando en un momento dado, podría de nuevo comprometer nuestro sistema. Necesitamos para ella valores realmente impredecibles, de forma que nuestro adversario no pueda averiguarlos ni tratar de simular el proceso de generación que nosotros hemos llevado a cabo. Necesitamos, pues, valores auténticamente aleatorios.

8.1.3.- Secuencias Totalmente Aleatorias.

Como ya se ha mencionado, no existe aleatoriedad cuando se habla de computadoras. En realidad se puede decir que no existen sucesos en el universo cien por cien aleatorios. En cualquier caso, y a efectos prácticos, consideraremos un tercer tipo de secuencias pseudoaleatorias.

Secuencias aleatorias: diremos que una secuencia totalmente aleatoria (o simplemente aleatoria) si no puede ser reproducida de manera fiable.

Llegados a este punto parece claro que nuestro objetivo no va a ser generar secuencias aleatorias puras, sino más bien secuencias impredecibles e irreproducibles. Será suficiente, pues, con emplear un generador criptográficamente aleatorio alimentado por una semilla totalmente aleatoria.

8.2.- Generación de Secuencias Aleatorias Criptográficamente Válidas.

Para poder obtener secuencias a la vez impredecibles e irreproducibles, haremos uso de generadores de secuencias criptográficamente aleatorias, en conjunción con algún mecanismo de recolección de bits aleatorios, que nos va a permitir inicializar la semilla del generador. Un esquema de este tipo será seguro siempre que se salvaguarde adecuadamente la semilla empleada. Primero veremos algunos mecanismos para obtener los bits de la semilla, y después nos centraremos en los generadores criptográficamente aleatorios propiamente dichos.

8.3.- Obtención de Bits Aleatorios.

Las operaciones aritméticas y lógicas que realiza una computadora son completamente deterministas. Sin embargo, las computadoras, como veremos a continuación poseen elementos menos deterministas que pueden ser útiles para nuestros propósitos.

Para obtener n bits aleatorios bastaría con que una persona lanzara una moneda al aire n veces y nos fuera diciendo el resultado. En la actualidad apenas hay computadores que incorporen hardware especifico para esta tarea, aunque existe y sería bastante barato y sencillo incorporarlo a la arquitectura de cualquier computadora.

Existen valores obtenidos del hardware de la computadora que suelen proporcionar algunos bits de aleatoriedad. Parece razonable que leer en un momento dado el valor de un reloj interno de alta precisión proporcione un resultado más o menos impredecible, por lo que podríamos emplearlo para recolectar valores aleatorios. Diferentes pruebas has demostrado sin embargo que mecanismos de este tipo, que pueden ser útiles en ciertas arquitecturas y sistemas operativos, dejan de servir en otros versiones del mismo sistema o en arquitecturas muy similares, por lo que hemos de tener mucho cuidado con esto.

Tampoco son útiles las fuentes públicas de información, como por ejemplo los bits de un CD de audio, puesto que nuestros atacantes pueden disponer de ellas, con lo que el único resto de aleatoriedad que nos va a quedar es la posición que acojamos dentro del CD para extraer los bits.

8.4.- Fuentes Adecuadas de Obtención de Bits Aleatorios.

Cuando no dispongamos de un elemento físico en la computadora específicamente diseñado para producir datos aleatorios, podemos emplear algunos dispositivos de hardware relativamente comunes en las computadoras actuales.

Tarjetas digitalizadoras de sonido o video. Un dispositivo digitalizador de audio (o video) sin ninguna entrada conectada, siempre que tenga ganancia suficiente, capta esencialmente ruido térmico, con una distribución aleatoria, y por lo tanto puede ser apto para nuestros propósitos.
Unidades de Disco. Las unidades de disco presentan pequeñas fluctuaciones en su velocidad de giro debido a turbulencias en el aire. Si se dispone de un método para medir el tiempo de acceso de la unidad con suficiente precisión, su pueden obtener bits aleatorios de la calidad necesaria.

Si no se dispone de una fuente fiable de bits aleatorios se puede efectuar la combinación de varias fuentes de información menos fiables. Por ejemplo, podríamos leer el reloj del sistema, algún identificador del hardware, la fecha y la hora local, el estado de los registros de interrupciones del

sistema. Esto garantizará que en total se ha recogido una cantidad suficiente de bits realmente aleatorios.

La mezcla de todas estas fuentes puede proporcionarnos suficiente aleatoriedad para nuestros propósitos. Teniendo en cuenta que el número de bits realmente aleatorios que se obtendrán como resultado final del proceso ha de ser necesariamente menor que el número de bits recogido inicialmente, hemos de buscar un mecanismo para llevar a cabo esa combinación. Emplearemos a tal efecto las denominadas funciones de mezcla fuertes.

Una función de mezcla es aquella que toma dos o más fuentes de información y produce una salida en que cada bit es una función compleja y no lineal de todos los bits de la entrada. Por término medio, modificar un bit en la entrada debería alterar aproximadamente la mitad de los bits de salida.

Podemos emplear diferentes algoritmos criptográficos para construir este tipo de funciones.

Un algoritmo simétrico de cifrado puede ser útil como función de mezcla de la siguiente forma: supongamos que usa una clave de *n* bits, y que tanto su entrada como su salida son bloques de *m* bits. Si disponemos de *n+m* bits inicialmente, podemos codificar *m* bits usando como clave los *n* restantes, y así obtener como salida un bloque de *m* bits con mejor aleatoriedad. Así, por ejemplo, si usamos DES (Data Encryption Standard), podemos reducir a 64 bits un bloque de 120.

Una función resumen puede ser empleada para obtener un número fijo de bits a partir de una cantidad arbitraria de bits de entrada.

8.4.1.- Eliminación del Sesgo.

En la mayoría de los casos, los bits obtenidos de las fuentes aleatorias están sesgados, es decir, que hay más unos que ceros o viceversa. Esta situación no es deseable, puesto que necesitamos una fuente aleatoria no sesgada, que presente igual probabilidad tanto para el 0 como para el 1. Como veremos a continuación, esta circunstancia no constituye un problema serio, ya que existen diversas técnicas para solucionarla.

8.4.2.- Bits de Paridad.

Si tenemos una secuencia de valores cero y uno, con un sesgo arbitrario, podemos emplear el bit de paridad de la secuencia para obtener una distribución con una desviación tan pequeña como queramos. Para comprobarlo, supongamos que *d* es el sesgo, luego las probabilidades que tenemos para los bits de la secuencia son:

$$p = 0.5 + d \qquad q = 0.5 - d$$

donde *p* es la probabilidad para el 1 y *q* es la probabilidad para el 0. Se puede comprobar que las probabilidades para el bit de paridad de los *n* primero bits valen

$$r=\frac{1}{2}((p+q)^n+(p-q)^n \qquad s=\frac{1}{2}((p+q)^n-(p-q)^n$$

donde *r* será la probabilidad de que el bit de paridad sea 0 ó 1 dependiendo de si *n* es par o impar. Puesto que $p+q=1$ y $p-q=2d$, tenemos:

$$r=\frac{1}{2}(1+(2d)^n) \qquad s=\frac{1}{2}(1-(2d)^n)$$

Siempre que $n>\frac{\log_2(2\varepsilon)}{\log_2(2d)}$ el sesgo de la paridad será menor que ε, por lo que bastará con tomar esos *n* bits. Por ejemplo, si una secuencia de bits tiene *p*=0.01 y *q* = 0.99 basta con tomar la paridad de cada 308 bits para obtener un bit de sesgo inferior a 0.001.

El bit de paridad de una secuencia vale 1 si el número de unos de dicha secuencia es par (paridad impar o impar (paridad par).

8.5.- Método de Von Neumann.

El método que propuso Von Neumann para eliminar el sesgo de una cadena de bits consiste simplemente en examinar la secuencia de dos en dos bits. Eliminamos los pares 00 y 11, e interpretamos 01 como 0 y 10 como 1. Por ejemplo, la serie 00.10.10.01.01.10.10.10.11 daría lugar a 1.1.0.0.1.1.1.

Es fácil comprobar que, siendo *d* el sesgo de la distribución inicial

$$P(01) = P(10) = (0.5+d)(0.5-d)$$

Por lo que la cadena de bits resultante presenta exactamente la misma probabilidad tanto para el 0 como para el 1. El problema de este método es que no sabemos a priori cuántos bits de información sesgada necesitamos para obtener cada bit de información no sesgada.

8.6.- Uso de funciones Resumen.

Si calculamos la entropía de una secuencia sesgada, obtendremos el número *n* de bits reales de información que transporta. Entonces podremos aplicar una función resumen y quedarnos exactamente con los *n* bits menos significativos del resultado obtenido.
Veamos el ejemplo, sea un secuencia de 300 bits con una probabilidad *P*(1) = 0.99. La entropía de cada bit será:

$$H = -0.99\log_2(0.99) - 0.01\log_2(0.01) = 0.08079 \text{ Bits}$$

Luego los 300 bits originales aportarán 300 x 0.08079+- 24 bits de información real. Podemos calcular la firma MD5 o SHA de dicha secuencia y considerar los 24 bits menos significativos del resultado como bits aleatorios válidos.

8.7.- Generadores Aleatorios Criptográficamente Seguros.

Suponiendo que ya tenemos una cantidad suficiente de bits auténticamente aleatorios (impredecibles e irreproducibles), vamos a ver un par de generadores pseudoaleatorios que permiten obtener secuencias lo suficientemente seguras como para se empleadas en aplicaciones criptográficas.

8.8.- Generador X9.17.

Propuesto en el Instituto Nacional de Estándares Norteamericano, permite a partir de una semilla inicial s_0 de 64 bits, obtener secuencias de valores también de 64 bits. El algoritmo para obtener cada uno de los valores g_n de la secuencia es la siguiente:

$$g_n = DES(k, DES(k,t) \oplus s_n)$$
$$s_{n+1} = DES(k, DES(k,t) \oplus g_n)$$

Donde DES es el algoritmo criptográfico de clave simétrica Data Encryption System

K es la clave aleatoria inicial que es usada por el algoritmo criptográfico.

$\oplus$ es la operación Xor realizada bit a bit.

Donde k es una clave aleatoria reservada para la generación de cada secuencia, y t es el tiempo en el que cada valor es generado –cuanta más resolución tenga (hasta 64 bits), mejor-. DES(K,M) representan la codificación de *M* mediante el algoritmo criptográfico DES que veremos en el capítulo

2, empleando la clave *K*, y $\oplus$ representa la función or-exclusivo. Nótese que el valor de *k* ha de ser mantenido en secreto para que la seguridad de este generador sea máxima.

8.9.- Generador Blum Blum Shub.

Es quizás el algoritmo que más pruebas de resistencia ha superado, con la ventaja adicional de su gran simplicidad –aunque es computacionalmente mucho más costoso que el algoritmo X9.17-. Consiste en escoger dos números primos grandes, *p* y *q*, que cumplan la siguiente propiedad:

$$p \equiv 3(\mathrm{mod}\,4) \qquad q \equiv 3(\mathrm{mod}\,4)$$

Sean entonces *n=pq* escogemos un número x aleatorio primo relativo con n, que será nuestra semilla inicial. Al contrario que *x*, que debe ser mantenido en secreto, *n* puede ser pública.

Calculamos los valores s_i de la serie de la siguiente forma:

$$\begin{aligned} s_0 &= (x^2)(\mathrm{mod}\,n) \\ s_{i+1} &= (s_i^2)(\mathrm{mod}\,n) \end{aligned}$$

Hay que tener cuidado de emplear únicamente como salida unos pocos de los bits menos significativos de cada s_i. De hecho, si tomamos no más de $\log_2(\log_2(s_i))$ bits en cada caso podemos asegurar que predecir el siguiente valor de la serie es al menos tan difícil como factorizar n.

CAPÍTULO 2

Criptografía Clásica

1.- Algoritmos Clásicos de Cifrado.

1.1.- Cifrados Monoalfabéticos.

Se engloban dentro de todos los algoritmos criptográficos que, sin desordenar los símbolos del lenguaje, establecen una correspondencia única para todos ellos en todo el mensaje. Es decir, si al símbolo A le corresponde el símbolo D, esta correspondencia se mantiene a lo largo de todo el mensaje.

1.2.- Algoritmo de César.

El algoritmo de César, llamado así porque es el que empleaba Julio César para enviar mensajes secretos, es uno de los algoritmos criptográficos más simples. Consiste en sumar 3 al número de orden de cada letra. De esta forma a la A le corresponde la D, ala B la E, y así sucesivamente. Si asignamos a cada letra un número (*A=0, B=1,...*), y consideramos un alfabeto de 26 letras, la transposición criptográfica sería:

$$C = (M + 3) \bmod 26$$

Obsérvese que este algoritmo no siquiera posee clave, la transformación siempre es la misma. Para descifrar basta con restar 3 al número de orden de las letras del criptograma.

1.2.1.- Sustitución Afín.

Es el caso general del algoritmo de César. Su transformación seria:

$$E_k(M) = (aM + b) \bmod 26$$

Siendo *a* y *b* dos números enteros menores que el cardinal *N* del alfabeto, y cumpliendo que *MCD(a,N)=1*. La clave *k* viene entonces dada por el par *(a,b)*. El algoritmo de Cesar será una transformación a fin con *k = (1,3)*.

1.3.- Cifrado Monoalfabético General.

Es el caso más general de cifrado monoalfabético. La sustitución ahora es arbitraria, siendo la clave *k* precisamente la tabla de sustitución de un símbolo por otro. En este caso tenemos *N!* posibles claves.

1.4.- Criptoanálisis de los Métodos de Cifrado Monoalfabéticos.

El cifrado monoalfabético constituye la familia de métodos más simple de criptoanalizar, puesto que las propiedades estadísticas del texto plano se conservan en el criptograma. Supongamos que, por ejemplo, la letra que más aparece en castellano es la A. Parece lógico que la letra más frecuente en el texto codificado sea aquella que corresponde con la A. Emparejando las frecuencias relativas de aparición de cada símbolo en el mensaje cifrado con el histograma de frecuencias del idioma en el que se supone está el texto plano, podremos averiguar fácilmente la clave.

En el peor de los casos, es decir, cuando tenemos un emparejamiento arbitrario, la distancia de unicidad de Shannon que obtenemos es

$$S = \frac{H(K)}{D} = \frac{\log_2(N!)}{D}$$

Donde D es la distancia de unicidad de Shannon

donde *D* es la redundancia del lenguaje empleado en el mensaje original, y *N* es el número de símbolos de dicho lenguaje. Como es lógico, suponemos que las *N!* claves diferentes son equiprobables en principio.

En casos más restringidos, como el afín, el criptoanálisis es aún más simple, puesto que el emparejamiento de todos los símbolos debe responder a alguna combinación de coeficientes *(a,b)*.

1.5.- Cifrados Polialfabéticos.

En los cifrados polialfabéticos la sustitución aplicada a cada carácter varía en función de la posición que ocupe éste dentro del texto plano. En realidad corresponde a la aplicación cíclica de *n* cifrados monoalfabéticos.

1.6.- Cifrado de Vigenère.

Es un ejemplo típico de cifrado polialfabético, cuya clave es una secuencia de símbolos $k = \{k_0, k_1, ..., k_{d-1}\}$ y que emplea la siguiente función de cifrado:

$$E_k(m_i) = m_i + k_{(i \bmod d)} (\bmod n)$$

Donde $E_k(m_i)$ es el cifrado Vigenère utilizando la clave k aplicado a cada elemento m_i del texto en claro.

siendo m_i el *i-ésimo* símbolo del texto plano y n el cardinal del alfabeto de entrada.

El sistema de cifrado de Vigenère (en honor al criptógrafo francés Blaise de Vigenère, siglo XVI) es un sistema polialfabético o de sustitución múltiple. Este tipo de criptosistemas aparecieron para sustituir a los monoalfabéticos o de sustitución simple, basados en el César, que presentaban ciertas debilidades frente al ataque de los criptoanalistas relativas a la frecuencia de aparición de elementos del alfabeto. El principal elemento de este sistema es la llamada Tabla de Vigenère, una matriz de caracteres cuadrada, con dimensión, como se muestra en el apéndice.

La clave del sistema de cifrado de Vigenère es una palabra de letras del alfabeto utilizado anteriormente; esta palabra es un elemento del producto cartesiano (veces), que es justamente el alfabeto del criptosistema de Vigenère. De esta forma, el mensaje a cifrar en texto claro ha de descomponerse en bloques de elementos - letras - y aplicar sucesivamente la clave empleada a cada uno de estos bloques, utilizando la tabla anteriormente proporcionada.

Veamos un ejemplo de aplicación del criptosistema de Vigenère: queremos codificar la frase "el día de la independencia" utilizando la clave "mexico". En primer lugar, nos fijamos en la longitud de la clave: es de seis caracteres, por lo que descomponemos la frase en bloques de longitud seis; aunque el último bloque es de longitud tres, esto no afecta para nada al proceso de cifrado:

```
eldiad elaind epende ncia
```

Ahora, aplicamos a cada bloque la clave mexico y buscamos los resultados como entradas de la tabla de Vigenère:

```
eldiad elaind epende ncia
mexico mexico mexico mexi
qplkoh bnomkl gdqram pque
```

Por ejemplo, la primera 'e' del texto en claro producto con la letra 'm' en la tabla de Vigenère le corresponde la letra 'q' y así sucesivamente. Finalmente, vemos que el texto cifrado ha quedado qplkoh bnomkl gdqram pque.

Este método de cifrado polialfabético se consideraba invulnerable hasta que en el siglo XIX se consiguieron descifrar algunos mensajes codificados con este sistema, mediante el estudio de la repetición de bloques de letras. La distancia entre un bloque y su repetición suele ser múltiplo de la palabra tomada como clave.

Una mejora sobre el cifrado de Vigenère fue introducida por el sistema de Vernam, utilizando una clave aleatoria de longitud igual a la del mensaje; la confianza en este nuevo criptosistema hizo que se utilizase en las comunicaciones confidenciales entre la Casa Blanca y el Kremlin, hasta, por lo menos, el año 1987.

1.6.1.- Criptoanálisis.

Para criptoanalizar este tipo de clave basta con efectuar *d* análisis estadísticos independientes agrupando los símbolos según la k_i empleada para codificarlos. Para estimar *d*, buscaremos la periodicidad de los patrones comunes que puedan aparecer en el texto cifrado. Obviamente, para el criptoanálisis necesitaremos al menos *d* veces más cantidad de texto que los métodos monoalfabéticos.

1.7.- Cifrado Vernam.

Dada una sucesión finita de variables aleatorias $\{K_i\}_{i=0,1,...,n-1}$ independientemente e idénticamente distribuidas según una distribución equiprobable sobre Z_m, el cifrado de Vernam o cinta aleatoria cifra el mensaje $X=\{X_0,...,X_{n-1}\}$ según la transformación $Y_i=T_i(X_i)=K_i+X_i(\mathrm{mod}\ m)\ 0\le i<n$.

Las variables aleatorias $Y_0,...,Y_{n-1}$ también son independientes e idénticamente distribuidas según una distribución equiprobable sobre Z_m.

Este sistema tiene secreto perfecto; es decir, es teóricamente irrompible. Ahora bien, a pesar de eso no resulta muy útil en la práctica, debido a que necesita una clave de igual longitud de texto a cifrar.

1.8.- Cifrados por Sustitución Homofónica.

Para corregir la sensibilidad frente a ataques basado en el estudio de las frecuencias de aparición de los símbolos, existe una familia de algoritmos que trata de ocultar las propiedades estadísticas del texto plano empleando un alfabeto de salida con más símbolos que el alfabeto de entrada.

Supongamos que nuestro alfabeto de entrada posee cuatro letras: {*a,b,c,d*}. Supongamos además que en nuestros textos la letra 'a' aparece con una probabilidad 0.4, y el resto con probabilidad 0.2. Podríamos emplear el siguiente alfabeto de salida $\{\alpha,\beta,\gamma,\delta,\varepsilon\}$ efectuando la siguiente asociación:

$$
\begin{aligned}
E(a) &= \begin{cases} \alpha & \text{con probabilidad } 1/2 \\ \beta & \text{con probabilidad } 1/2 \end{cases} \\
E(b) &= \gamma \\
E(c) &= \delta \\
E(d) &= \varepsilon
\end{aligned}
$$

En el texto cifrado ahora todos los símbolos aparecen con igual probabilidad, lo que imposibilita un ataque basado en frecuencias. A diferencia de los que se puede pensar en un principio este método presenta demasiados inconvenientes para ser útil en la práctica, además del problema de necesitar un alfabeto de salida mayor que el de entrada, para aplicarlo hace falta conocer la distribución estadística a priori de los símbolos en el texto plano, información de la que, por desgracia, no siempre se dispone.

1.9.- Cifrados de Transposición.

Este tipo de mecanismos de cifrado no sustituye unos símbolos por otros, sino que cambia su orden dentro del texto. Un mecanismo de transposición podría constituir en colocar el texto en una tabla de *n* columnas, y dar como texto cifrado los símbolos de una columna –ordenados de arriba a bajo- concatenados con los de otra, etc. La clave *k* se compondría del número *n* junto con el orden en el que deben leer las columnas.

Por ejemplo, supongamos que queremos cifra el texto "la información enviada es confidencial", con *n=5* y la permutación *{3,2,5,1,4}* como clave. Colocamos el texto en una tabla y obtenemos:

1	2	3	4	5
l	a		i	n
i	n	f	o	r
m	a	c	i	o
n		e	n	v
i	a	d	a	
e	s		c	o
n	f	i	d	e
n	c	i	a	l

Tendríamos como texto cifrado la concatenación de las columnas 3,2,5,1 y 4 respectivamente " *fced iiana asfcnrov oei limniennioinacda*". Nótese que hemos de conservar el espacio al principio del texto cifrado para que el mecanismo surta efecto.

1.9.1.- Criptoanálisis.

Este tipo de mecanismos de cifrado se pueden criptoanalizar efectuando un estudio estadístico sobre la frecuencia de aparición de pares y tripletas de símbolos en el lenguaje en que esté escrito el texto plano. Suponiendo que conocemos *n*, que el nuestro es igual a 5, tenemos 5!=120 posibles claves. Descifraríamos el texto empleado cada una de ellas y comprobaríamos si los pares y tripletas de símbolos consecutivos que vamos obteniendo se corresponden con los más frecuentes en castellano. De esa forma podremos asignarle una probabilidad automáticamente a cada una de las posibles claves.

Si, por el contrario, desconocemos *n*, basta con ir probando con $n = 2$, $n = 3$ y así sucesivamente. Este método es bastante complejo de llevar a cabo manualmente, a no ser que se empleen ciertos trucos, pero una computadora puede completarlo en un tiempo más que razonable sin demasiados problemas.

1.10.- Maquinas de Rotores.

1.10.1.- La Máquina Enigma.

En el año 1923, un ingeniero alemán llamado Arthur Sherbius patentó una máquina específicamente diseñada para facilitar las comunicaciones seguras, Se trataba de un instrumento de apariencia simple, parecida a una máquina de escribir. Quien deseara codificar un mensaje sólo tenía que teclearlo y las letras correspondientes al mensaje cifrado se irían iluminando en el panel. El destinatario copiaba dichas letras en su propia máquina y el mensaje original aparecía de nuevo. La clave la constituían las posiciones iniciales de tres tambores o rotores en que el invento poseía en su parte frontal.

En la maquina llamada Enigma los rotores no son más que tambores con contactos en su superficie y cableados en su interior, de forma que cuando se pulsa una tecla, la posición de éstos determina cuál es la letra que se ha de iluminar. Cada vez que se pulsa una tecla el primer rotor avanza una posición; el segundo avanza cuando el anterior ha dado una vuelta completa y así sucesivamente. El reflector no existía en los primeros modelos, se introdujo posteriormente para permitir que la misma maquina sirviera tanto para cifrar como para descifrar.

1.10.1.1.- La Teoría sobre la Maquina Enigma.

Observamos que un rotor no es más que una permutación dentro del alfabeto de entrada. El cableado hace que cada una de las letras se haga corresponder con otra. Todas las letras tiene imagen y no hay dos letras con la misma imagen. Si notamos una permutación como π, podemos escribir que la permutación resultante de combinar todos los rotores en un instante dado es:

$$\pi_{total} = \left\langle \pi_0, \pi_1, \pi_2, \pi_3, \pi_2^{-1}, \pi_1^{-1}, \pi_0^{-1} \right\rangle$$

Donde cada π_i es la posición de cada uno de los rotores (discos) que componen la maquina.

La permutación $\pi_3 = \pi_3^{-1}$, es decir, que aplicada dos veces nos dé lo mismo que teníamos al principio. De esta forma se cumple la propiedad de que, para una misma posición de los rotores, la codificación y la decodificación son simétricas.

La fuerza de la máquina Enigma radicaba en que tras codificar cada letra se giran los rotores, lo cual hace que la permutación que se aplica a cada letra sea diferente, y que esa permutación además no se repita hasta que los rotores recuperen su posición inicial, Tengamos en cuenta que hay 17576 posiciones iniciales de los rotores, y 60 combinaciones de tres rotores a partir de los cinco de entre los que se puede elegir. Puesto que el enchufe también presenta un número bastante alto de combinaciones, existe una cantidad enorme de posibles disposiciones iniciales de la maquina (al menos en aquella época). La potencia del método de criptoanálisis empleado radica en que se podía identificar un emparejamiento plausible entre un criptograma y el texto plano, de forma que sólo bastaba con rastrear dentro del espacio de posibles configuraciones para encontrar aquella que llevara a cabo la transformación esperada. No disponer de dicho emparejamiento hubiera complicado enormemente el criptoanálisis, tal vez hasta el punto de hacerlo fracasar.

2.- Algoritmos Simétricos de Cifrado.

La gran mayoría de los algoritmos de cifrado simétrico se apoyan en los conceptos de confusión y difusión inicialmente propuestos por Shannon, que se combinan para dar lugar a los denominados cifrados de producto.

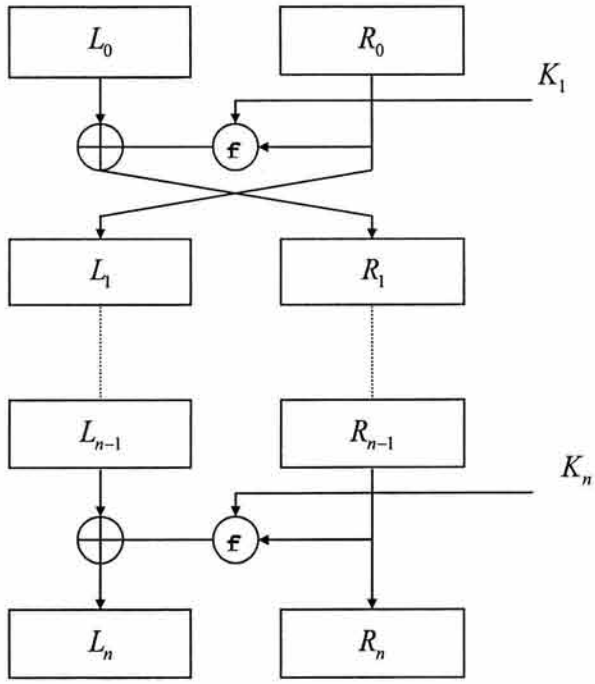
La confusión consiste en tratar de ocultar la relación que existe entre el texto plano, el texto cifrado y la clave. Un buen mecanismo de confusión hará demasiado complicado extraer relaciones estadísticas entre las tres cosas. Por su parte la difusión trata de repartir la influencia de cada bit de mensaje original lo más posible entre el mensaje cifrado.

La confusión por si sola sería suficiente, ya que si establecemos una tabla de sustitución completamente diferente para cada clave con todos los textos planos posibles tendremos un sistema extremadamente seguro. Sin embargo, dichas tablas ocuparían cantidades astronómicas de memoria, por lo que en la práctica serían inviables. Por ejemplo, un algoritmo que codificara bloques de 128 bits empleando una clave de 80 bits necesitaría una tabla de aproximadamente 10^{63} entradas.

Lo que en realidad se hace para conseguir algoritmos fuertes sin necesidad de almacenar tablas enormes es intercalar la confusión (sustituciones simples, con tablas pequeñas) y la difusión (permutaciones). Esta combinación se conoce como cifrado de producto. La mayoría de los algoritmos se basan en diferentes capas de sustituciones y permutaciones, estructura que denominaremos Red de Sustitución-Permutación. En muchos casos el criptosistema no es más que un paso simple de sustitución-permutación repetido n veces, como ocurre con el popular DES.

2.1.- Redes de Feistel.

Muchos de los cifrados de producto tienen en común que dividen un bloque de longitud n en dos mitades, L y R. Se define entonces un cifrado de producto interactivo en el que la salida de cada ronda se usa como entrada para la siguiente según la relación.



$$\left.\begin{array}{rcl} L_i & = & R_{i-1} \\ R_i & = & L_{i-1} \oplus f(R_{i-1}, K_i) \end{array}\right\} Si\ i < n$$
$$\begin{array}{rcl} L_n & = & L_{n-1} \oplus f(R_{n-1}, K_n) \\ R_n & = & R_{n-1} \end{array}$$

Donde L_i es la mitad izquierda de bits del bloque de entrada en la iteración i.

R_i es la mitad derecha de bits del bloque de entrada en la iteración i,

K_i es el bloque de bits de la clave en la iteración i,

$F(R_{i-1},K_i)$ es la función f de la iteración i que recibe la mitad izquierda de la iteración derecha y la clave k de la iteración i

n es la ultima iteración de la red de Feistel.

$\oplus$ es la operación Xor entre dos bloques de bits.

Este tipo de estructura se denomina Red de Feistel, y es empleada en multitud de algoritmos, como DES, Lucifer, FEAL, CAST, Blowfish, etc. Tiene la interesante propiedad de ser reversible, independientemente de cómo sea la función f, para ello basta con aplicar de nuevo el algoritmo al resultado, pero empleando las K_i en orden inverso. Esto nos va a permitir emplear el mismo mecanismo tanto para cifrar como para descifrar.

2.2.- Cifrados con Estructura de Grupo.

Otra de las cosas que hay que tener en cuenta en los cifrados de producto es la posibilidad de que posean estructura de grupo. Se dice que un cifrado tiene estructura de grupo si se cumple la siguiente propiedad:

$$\forall k_1, k_2 \ \exists\ k_3 \ \text{ tal que } \mathrm{E}_{k_2}(E_{k_1}(M)) = E_{k_3}(M)$$

Estos es, si hacemos dos cifrados encadenados con k_i y k_2, existe una clave k_3 que realiza la transformación equivalente.

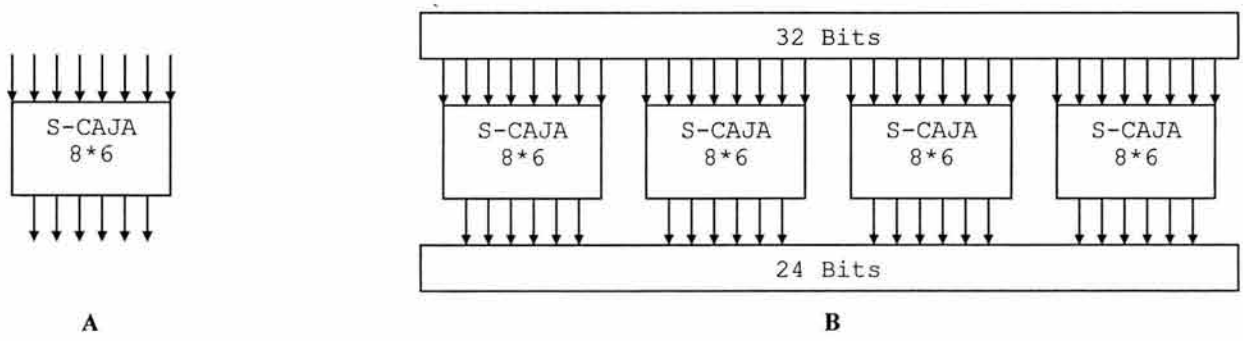
Es interesante que un algoritmo criptográfico no posea estructura de grupo, ya que si ciframos un mensaje primero con la clave k_2, es como si hubiéramos empleado una clave de longitud doble, aumentando la seguridad del sistema. Si, por el contrario, la transformación criptográfica presentara estructura de grupo, esto hubiera sido equivalente a cifrar el mensaje una única vez con una tercera clave, con lo que no habríamos ganado nada.

2.3.- S-Cajas.

Hemos dicho que para poder construir buenos algoritmos de producto, intercalaremos sustituciones sencillas (confusión), con las tablas pequeñas, y permutaciones (difusión), estas tablas pequeñas de sustitución se denominan de forma genérica S-Cajas (ver apéndice).

Una S-Caja de *m*n* bits es una tabla de sustitución que toma como entrada cadenas de *m* bits y da como salida cadenas de *n* bits, haciendo uso de la S-Caja correspondiente. Normalmente, cuando más grande sean las S-Cajas, más resistentes será el algoritmo resultante, aunque la elección de los valores de salida para que den lugar a un buen algoritmo no es en absoluto trivial.

El algoritmo llamado CAST, que emplea seis S-Cajas de 8*32 bits, CAST codifica bloques de 64 bits empleando claves de 64 bits, consta de ocho rondas y deposita prácticamente toda su fuerza en las S-Cajas. De hecho, existen muchas variantes de CAST, cada una con sus S-Cajas correspondientes –algunas de ellas secretas-. Este algoritmo se ha demostrado resistente a las técnicas habituales de criptoanálisis, y sólo se conoce la fuerza bruta como mecanismo para atacarlo.



A: S-Caja individual, B: combinación de cuatro cajas S-Cajas.

2.4.- El Algoritmo DES (Data Encryption Standard).

2.4.1.- Introducción.

En los sistemas vistos hasta el momento, es decir, en la criptografía tradicional se ha tratado de dificultar el criptoanálisis sobre todo mediante el empleo de claves muy largas, en lugar de complicar los algoritmos.

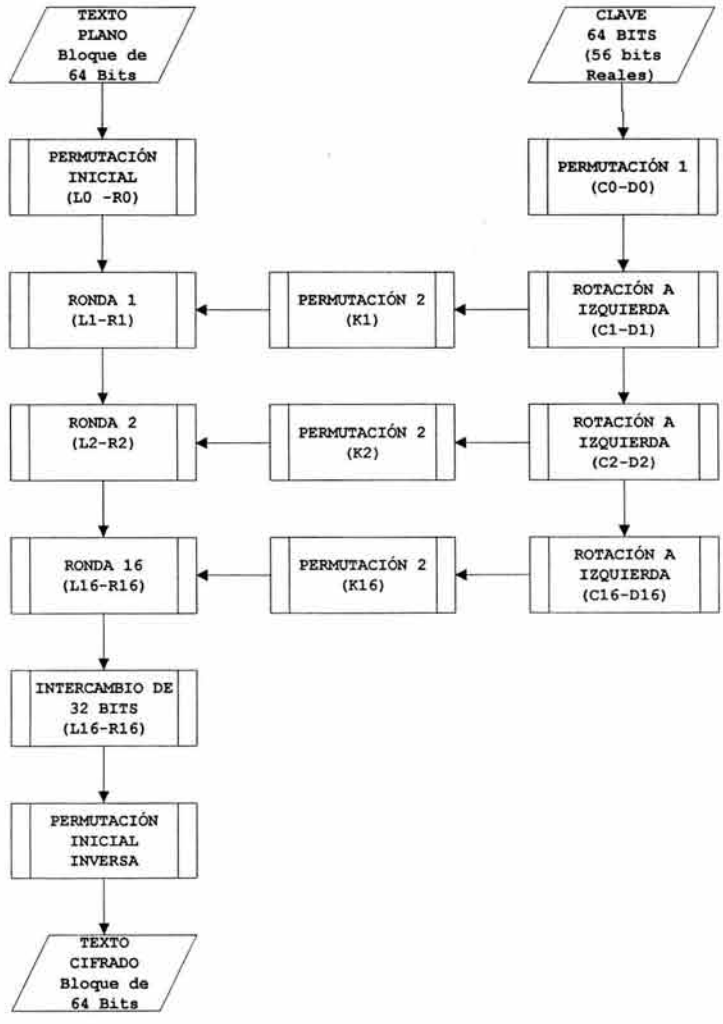
La razón es que antes del uso de las computadoras, un operador debía realizar las transformaciones manualmente, o con ayuda de un aparato concebido específicamente para la tarea. En el primer caso, un algoritmo complicado podía comprometer muy seriamente la velocidad de

cifrado o descifrado, y provocaba errores. En el segundo, además de las limitaciones técnicas, existía el problema de que fuera necesario cambiar de algoritmo.

Con la entrada en escena de las computadoras, los criptógrafos disponen de una herramienta mediante la cual puede complicar los algoritmos sin las limitaciones anteriores, pero deben preocuparse también de ataques basados en una mayor capacidad de cálculo.

El algoritmo DES está basado en el algoritmo LUCIFER creado por Horst Feistel quien trabajaba en IBM y diseñado por IBM a principios de los setenta. Después de algunas modificaciones de la NSA (National Security Agency), fue publicado en 1977 por el NBS (National Bureau of Standards), una sección del departamento de defensa de los EEUU.

Esquema General del Algoritmo DES.



La Oficina Nacional de Normas de los EEUU publicó en enero de 1977 una norma oficial en la que se describe el algoritmo de cifrado que las Agencias Federales deben utilizar para la protección

de información no clasificada. El objetivo de la Norma de Cifrado de Datos (DES) era que los sistemas de protección de información de los distintos estados fueran compatibles.

El criptosistema LUCIFER trabajaba sobre bloques de 128 bits, teniendo la clave de igual longitud. Se basaba en operaciones lógicas booleanas y podía ser implementado fácilmente, tanto en software como en hardware.

Tras las modificaciones introducidas por el NBS, consistentes básicamente en la reducción de la longitud de clave y de los bloques, DES cifra bloques de 64 bits, mediante permutación y sustitución y usando una clave de 64 bits, de los que 8 son de paridad (esto es, en realidad usa 56 bits), produciendo así 64 bits cifrados.

DES utiliza claves de 56 bits y un cifrado de bloques de 64 bits. Este algoritmo cumple con los principios de confusión y difusión de Shannon.

2.4.2.- Principios de Confusión y Difusión de Shannon.

Shannon propuso dos técnicas que deben utilizar los criptosistemas de bloque para evitar ataques basados en métodos estadísticos.

- Confusión: Sustituciones para que la relación entre el texto en claro y la clave sea lo más compleja posible.
- Difusión: Transformaciones para disipar las propiedades estadísticas entre el texto en claro y el texto cifrado.

2.4.3.- Descripción del algoritmo DES.

El algoritmo DES cifra la información por bloques, es decir, el texto en claro debe ser dividido en bloques de 64 bits que serán encriptados uno tras otro.

DES utiliza claves de 56 bits, aunque estas suelen distribuirse en forma de números de 64 bits. De estos 64 bits, uno de cada ocho, es utilizado como bit de paridad (64-8=56).

El algoritmo DES codifica bloques de 64 bits empleando claves de 56 bits. Es una Red de Feistel de 16 rondas, más dos permutaciones, una que se aplica al principio (P_i) y otra que se aplica al final (P_f), tales que $P_i = P_{f-1}$.

La función f se compone de una permutación de expansión (E), que convierte el bloque de 32 bits correspondiente en uno de 48. Después realiza un or-exclusivo con el valor K_i, también de 48 bits, aplica ocho S-Cajas de 64 bits, y efectúa una nueva permutación P.

Se calcula un total de 16 valores de K_i, uno para cada ronda, efectuando primero una permutación inicial *EP1* sobre la clave de 64 bits, llevando a cabo desplazamientos a la izquierda de

cada una de las dos mitades -de 28 bits- resultantes, y realizando finalmente una elección permutada (*EP2*) de 48 bits en cada ronda, que sería la K_i. Los desplazamientos a la izquierda son de dos bits, salvo para las rondas 1, 2, 9 y 16, en las que se desplaza sólo un bit.

Para descifrar basta con usar el mismo algoritmo (ya que $P_i = P_{t-1}$) empleando las K_i en orden inverso.

2.4.3.1.- La Tabla de Permutación A.

Después de recibir un bloque de entrada de 64 bits, el primer paso consiste en aplicar al bloque de entrada una permutación A mediante la tabla siguiente:

Tabla de permutación A
58 50 42 34 26 18 10 2
60 52 44 36 28 20 12 4
62 54 46 38 30 22 14 6
64 56 48 40 32 24 16 8
57 49 41 33 25 17 9 1
59 51 43 35 27 19 11 3
61 53 45 37 29 21 13 5
63 55 47 39 31 23 15 7

El primer bit de la entrada será colocado en la posición 58, el segundo bit en la posición 50.

2.4.3.2.- La Tabla de Permutación B.

De la misma manera, después de las 16 iteraciones se realiza otra permutación mediante la tabla B

Tabla de permutación B
40 8 48 16 56 24 64 32
39 7 47 15 55 23 63 31
38 6 46 14 54 22 62 30
37 5 45 13 53 21 61 29
36 4 44 12 52 20 60 28
35 3 43 11 51 19 59 27
34 2 42 10 50 18 58 26
33 1 41 9 49 17 57 25

El primer bit de la entrada será colocado en la posición 40, el segundo bit en la posición 8.

2.4.3.3.- Las 16 iteraciones.

Después de la permutación A y antes de la B, el algoritmo DES realiza 16 iteraciones. Cada iteración realiza lo siguiente:

1. Los 64 bits de entrada se dividen en dos partes de 32 bits.
2. La mitad de la derecha (R) se introduce en una función (f) que se describirá más adelante.
3. Se realiza un XOR entre la salida de la función y la parte izquierda (L).
4. El resultado (32 bits) pasará a ser la parte derecha de la siguiente iteración, mientras que la parte derecha actual pasará a ser la parte izquierda.

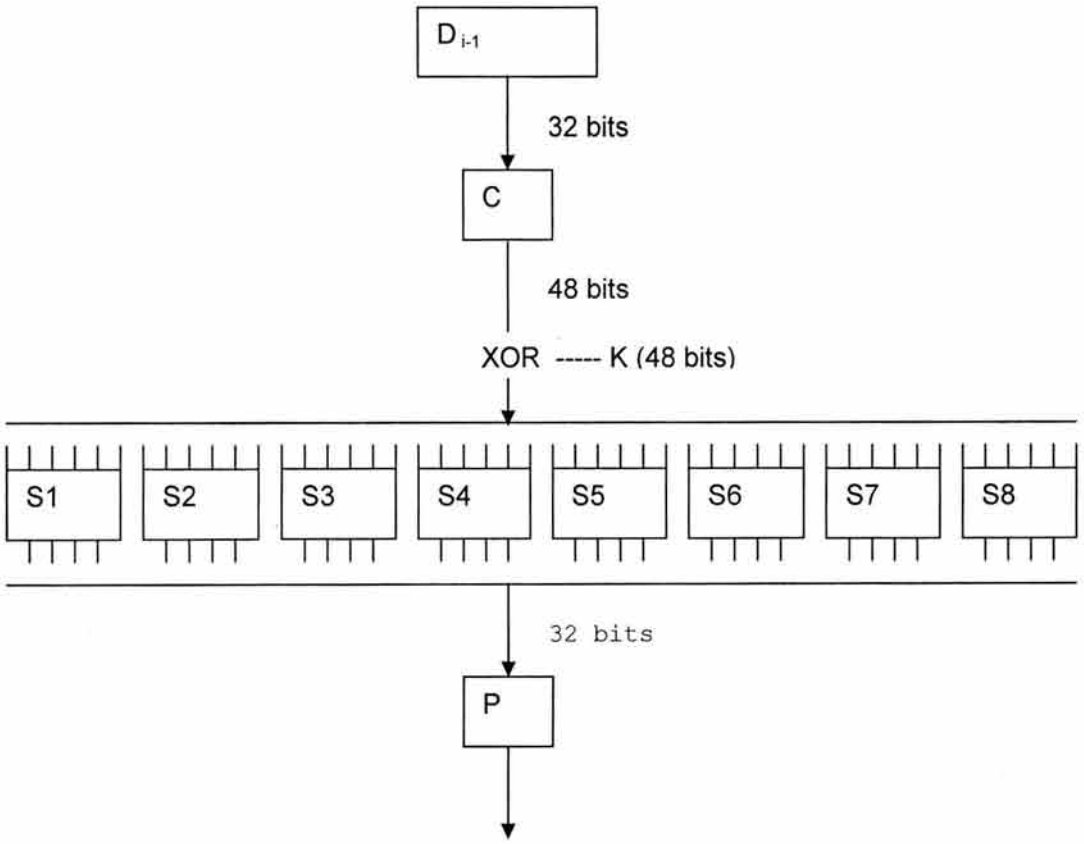
$$L_{i+1} = R_i$$

$$R_{i+1} = L_i \; XOR \, f(R_i, K)$$

En la última iteración (i=16) no se realizará el paso 4.

2.4.3.4.- La Función *f*.

Ahora describiremos la función f, veamos un esquema general:



Los 32 bits de la parte derecha entran en la función *f*. El primer paso consiste en transformar la entrada de 32 bits a 48 bits mediante la tabla C.

Tabla Transformación C					
32	01	02	03	04	05
04	05	06	07	08	09
08	09	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	01

A continuación se realiza un XOR con una subclave de 48 bits. Dado que la clave inicial era de 56 bits (representada como 64 bits) es necesario generar, a partir de esta, subclaves de 48 bits. Más adelante se describe detalladamente el proceso de generación de subclaves.

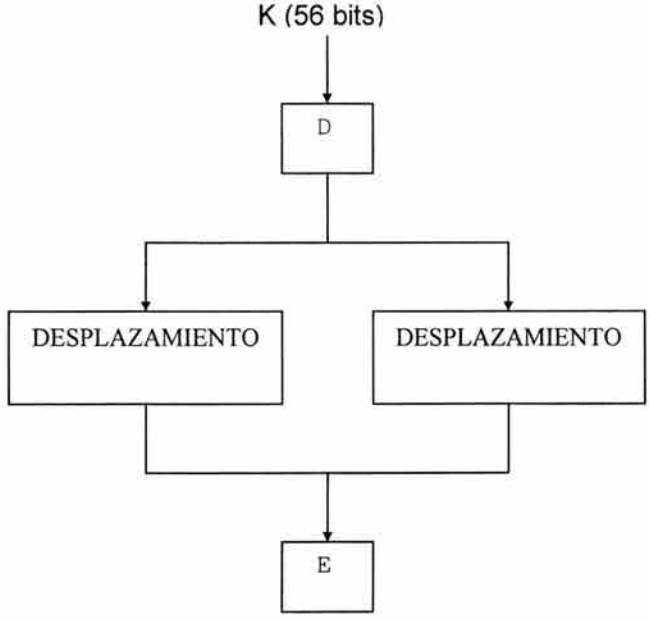
A continuación los 48 bits se dividen en bloques de 6 bits, cada uno de los cuales es utilizado como entrada de lo que se conoce como una caja S. Observe el esquema anterior.

Los 6 bits que forman cada bloque b_1, b_2, b_3, b_4, b_5 y b_6, son utilizados para leer de cada caja S. b_1 y b_6 indican la fila mientras que b_2, b_3, b_4, y b_5 indican la columna. Es decir, si utilizamos como entrada 000011, b1b6 -> 01 indican la primera fila y b2b3b4b5 -> 00001 indican la primera columna.

Las ocho cajas S existentes se muestran en el apéndice.

2.4.3.5.- Generación de las Subclaves K.

Este esquema se realiza para cada una de las iteraciones:



El primer paso consiste en una permutación mediante la tabla D:

Tabla de Permutación D						
57	49	41	33	25	17	09
01	58	50	42	34	26	18
10	02	59	51	43	35	27
19	11	03	60	52	44	36
63	55	47	39	31	23	15
07	62	54	46	38	30	22
14	06	61	53	45	37	29
21	13	05	28	20	12	04

Los 56 bits se dividen en dos partes de 28 bits. Cada una de estas partes rota hacia la izquierda un número X de bits en cada iteración, conforme a la tabla siguiente:

Iteración	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16
Desplazamiento	1	1	2	2	2	2	2	2	2	2	2	2	2	2	2	1

Finalmente, mediante la tabla de compresión E los 56 bits (28+28) del resultado se comprimen formando una subclave de 48 bits (la utilizada en la función f).

Tabla de compresión E					
14	17	11	24	01	05
03	28	15	06	21	10
23	19	12	04	26	08
16	07	27	20	13	02
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

2.4.4.- Descifrado en DES.

Una de las partes positivas de DES es que podemos descifrar los mensajes utilizando el mismo algoritmo que utilizamos para cifrar. La única diferencia consiste en el orden en que se aplican la subclaves, dado que en el descifrado se utilizan en orden inverso. Es decir, primero se utiliza la subclave k_{16}, después la k_{15} ... k_1.

2.4.5.- Claves Débiles en DES.

El algoritmo DES presenta algunas claves débiles. En general, todos aquellos valores de la llave que conducen a una secuencia inadecuada de K_i serán poco recomendables. Distinguiéremos entre claves débiles, que son aquellas que generan un conjunto de dieciséis valores iguales de K_i - y que cumplen $E_k(E_k(M)) = M$ -, claves semidébiles que generan dos valores diferentes de K_i, cada uno de los cuales aparece ocho veces. En cualquier caso, el número de claves de este tipo es tan pequeño en comparación con el número total de posibles claves, que no debe suponer un motivo de preocupación.

Claves débiles para el algoritmo DES expresadas en hexadecimal

Clave	Clave tras aplicar Pi
0101010101010101	0000000 0000000
1F1F1F1F0E0E0E0E	0000000 FFFFFFF
E0E0E0E0F1F1F1F1	FFFFFFF 0000000
FEFEFEFEFEFEFEFE	FFFFFFF FFFFFFF

Claves débiles para el algoritmo DES expresadas en hexadecimal

Clave	Clave tras aplicar P_i
01FE01FE01FE01FE	AAAAAAA AAAAAAA
FE01FE01FE01FE01	5555555 5555555
1FE01FE00EF10EF1	AAAAAAA 5555555
E01FE01FF10EF10E	5555555 AAAAAAA
01E001E001F101F1	AAAAAAA 0000000
E001E001F101F101	5555555 0000000
1FFE1FFE0EFE0EFE	AAAAAAA FFFFFFF
FE1FFE1FFE0EFE0E	5555555 FFFFFFF
011F011F010E010E	0000000 AAAAAAA
1F011F010E010E01	0000000 5555555
E0FEE0FEF1FEF1FE	FFFFFFF AAAAAAA
FEE0FEE0FEF1FEF1	FFFFFFF 5555555

2.4.6.- Debilidades del Algoritmo DES.

Desde la aparición del algoritmo DES han aparecido algunas críticas sobre el criptosistema. La principal es la longitud de la clave, de solo 56 bits, que hacen posible los ataques de fuerza bruta.
Por otra parte, la arbitrariedad de las cajas S, podría ser causa de la existencia de una clave maestra que permitiese descifrar cualquier mensaje.

Durante años, la fuerza del DES residía en la limitación de las máquinas para afrontar la prueba de examinar los miles de millones de claves necesarios (ataque por fuerza bruta). Sin embargo, el desarrollo de la Informática empezó a hacer temblar los cimientos en los que se sustentaba la seguridad de dicho algoritmo.

El aumento de la potencia de las computadoras y procesadores, capaces de realizar, cada vez, más operaciones por segundo, empezó a generar preocupación entre los ciudadanos sobre el uso de un algoritmo fundamentalmente débil como estándar de cifrado. De hecho, en los últimos años se desarrollaron concursos destinados a probar tal realidad, como en 1998, cuando se celebró el DES Challenge II Contest. En él fueron necesarios 40 días y el esfuerzo combinado -a través de Internet-, de la potencia de más de 40.000 Pentiums 166 para descubrir la clave correcta.

Aunque todavía podía considerarse lo suficientemente seguro para muchas comunicaciones, el DES ya empezaba a mostrar sus primeras debilidades, por lo que en el año 2002 se propuso una nueva edición del concurso. Se trataba del DES III Contest., en el que quedó de manifiesto la total debilidad del algoritmo cuando, de forma conjunta, el esfuerzo distribuido de miles de internautas y la EFF (Electronic Frontier Fundation), con su máquina Deep Crack (también conocida como DES Cracker,) consiguieron, en tan sólo 22 horas, romper la seguridad del DES.

El DES Cracker era una computadora construida para probar combinaciones de claves en datos cifrados con DES. Estaba formado por seis cabinas, cada una de las cuales albergaba 29 placas de circuitos y, a su vez, cada una de ellas, alojaba 64 procesadores de búsqueda especialmente desarrollados para tal fin. Esta máquina era capaz de analizar, por si sola, la nada despreciable cantidad de 90 billones de claves por segundo.

El costo de DES Cracker, especialmente fabricado para el citado concurso, fue de 250.000 dólares, costo ridículo si se tiene en cuenta la cantidad de secretos que pueden obtenerse mediante un uso fraudulento de una computadora de este tipo.

Tampoco hay que olvidar que el DES ha llegado a convertirse en el estándar utilizado, en todo el mundo, para el cifrado de datos, operaciones bancarias, económicas, empresariales, etc. durante más de 20 años.

Una forma de solucionar el problema de la longitud de la clave es el uso del cifrado triple, conocido como Triple DES.

2.5.- El algoritmo Triple DES.

2.5.1.- Introducción.

El algoritmo DES tiene un grave problema con la longitud de la clave (56 bits). Por este motivo no es el algoritmo más indicado para cifrar datos importantes.

Para solucionar el problema con la longitud de la clave nace el algoritmo Triple DES, que consiste en utilizar tres veces DES.

La clave utilizada por Triple DES es de 128 bits (112 de clave y 16 de paridad), es decir, dos claves de 64 bits (56 de clave y 8 de paridad) de los utilizados en DES. El motivo de utilizar este de tipo de clave es la compatibilidad con DES. Si la clave utilizada es el conjunto de dos claves DES iguales el resultado será el mismo para DES y para Triple DES.

2.5.2.- Encriptar Mediante Triple DES.

Para encriptar mediante el algoritmo Triple DES seguiremos los siguientes pasos:

1. Dividir la clave de 128 bits en dos partes de 64 bits: k_1 y k_2.
2. Se cifra el texto en claro con k_1. El resultado es conocido como ANTIDES.
3. Se cifra ANTIDES con k_2.
4. Se cifra el resultado con k_1.
5. Si $k_1=k_2$ el resultado coincide con una encriptación mediante DES.

Otra forma de utilizar Triple DES es con una clave de 192 bits (168 bits de clave y 24 bits de paridad). En este caso se cifrará primero con k_1, a continuación con k_2 y finalmente con k_3.

Para ser compatible con DES es necesario que $k_1=k_2=k_3$.

2.6.- El algoritmo IDEA (International Data Encryption Algorithm).

El algoritmo IDEA es bastante más joven que DES, pues data de 1992. Para muchos constituía el mejor y más seguro algoritmo simétrico disponible (hasta antes del 2001). Trabaja con bloques de 64 bits de longitud y emplea una clave de 128 bits. Como en el caso de DES, se usa el mismo algoritmo tanto para cifrar como para descifrar.

IDEA es un algoritmo bastante seguro, y hasta ahora se ha mostrado resistente a multitud de ataques, entre ellos el criptoanálisis diferencial. No presenta claves débiles, y su longitud de clave hace imposible en la práctica un ataque de fuerza bruta.

Como ocurre con todos los algoritmos simétricos de cifrado por bloques, IDEA se basa en los conceptos de confusión y difusión, haciendo uso de las siguientes operaciones elementales:

- XOR.

- Suma módulo 2^{16}.
- Producto módulo $2^{16}+1$.

El algoritmo IDEA consta de ocho rondas. Dividiremos el bloque X a codificar, de 64 bits, en cuatro partes X_1, X_2, X_3 y X_4 de 16 bits que vamos a necesitar. Las operaciones que llevaremos a cabo en cada ronda son las siguientes:

1. Multiplicar X_1 por Z_1.
2. Sumar X_2 con Z_2.
3. Sumar X_3 con Z_3.
4. Multiplicar X_4 por Z_4.
5. Hacer un XOR entre los resultados del paso 1 y el paso 3.
6. Hacer un XOR entre los resultados del paso 2 y el paso 4.
7. Multiplicar el resultado del paso 5 por Z_5.
8. Sumar los resultados de los pasos 6 y 7.
9. Multiplicar el resultado del pasos 8 por Z_6.
10. Sumar los resultados de los pasos 7 y 9.
11. Hacer un XOR entre los resultados de los pasos 1 y 9.
12. Hacer un XOR entre los resultados de los pasos 3 y 9.
13. Hacer un XOR entre los resultados de los pasos 2 y 10.
14. Hacer un XOR entre los resultados de los pasos 4 y 10.

La salida de cada iteración serán los cuatro sub-bloques obtenidos en los pasos 11, 12, 13 y 14 que serán la entrada del siguiente ciclo, en el que emplearemos las siguientes seis subclaves, hasta un total de 48. Al final de todo intercambiaremos los dos bloques centrales (en realidad con eso deshacemos el intercambio que llevamos a cabo en los pasos 12 y 13).

Después de la octava iteración, se realiza la siguiente transformación:

1. Multiplicar X_1 por Z_{49}.
2. Suma X_2 con Z_{50}.
3. Sumar X_3 con Z_{51}.
4. Multiplicar X_4 por Z_{52}.

Las primeras ocho subclaves se calculan dividiendo la clave de entrada en bloques de 16 bits.

Las siguientes ocho se calculan rotando la clave de entrada 25 bits a la izquierda y volviendo a dividirla, y así sucesivamente.

Las subclaves necesarias para descifrar se obtienen cambiando de orden las Z_i y calculando sus inversas para la suma o la multiplicación. Puesto que $2^{16} + 1$ es un número primo, nunca podremos obtener cero como producto de dos números, por lo que no necesitamos representar dicho valor.

Cuando estemos calculando productos, utilizaremos el cero para expresar el número 2^{16}. Esta representación es coherente puesto que los registros que se emplean internamente en el algoritmo poseen únicamente 16 bits.

2.7.- Métodos para Algoritmos de Cifrados por Bloques.

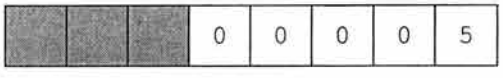
Independientemente del método empleado para codificar, hemos de tener en cuenta lo que ocurre cuando la longitud de la cadena que queremos cifrar no es un múltiplo exacto del tamaño del bloque. Entonces tenemos que añadir información al final para que así lo sea. El mecanismo más sencillo consiste en rellenar con ceros (puede ser otro patrón) el último bloque que se codifica. El problema ahora consiste en saber cuando se descifra por dónde hay que cortar. Lo que se suele hacer es añadir como último byte del último bloque el número de bytes que se han añadido. Esto tiene como inconveniente de que si el tamaño original es múltiplo del bloque, hay que alargarlo con otro bloque entero. Por ejemplo, si el tamaño de bloque fuera 64 bits, y sobran cinco bytes al final, añadiríamos dos ceros y un tres. Si por el contrario no sobrara nada, tendríamos que añadir siete ceros y un ocho.

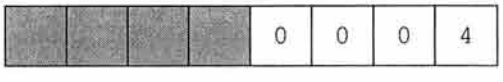
2.7.1.- Modo ECB (Electronic Code Book).

El modo ECB es el método más sencillo y obvio de aplicar un algoritmo de cifrado por bloque. Simplemente se subdivide la cadena que se quiere codificar en bloques del tamaño adecuado y se cifran todos ellos empleando la misma clave.

A favor de este método podemos decir que permite codificar los bloques independientemente de su orden, lo cual es adecuado para codificar bases de datos o archivos en los que se requiera un acceso aleatorio. También es resistente a errores, pues si uno de los bloques sufriera una alteración, el resto quedaría intacto.

Por el contrario, si el mensaje presenta patrones repetitivos, el texto cifrado también los presentará, y eso es peligroso, sobre todo cuando se codifica información muy redundante, o con patrones comunes al inicio y final (como el correo Electrónico).





0	0	0	0	0	0	0	8

Un contrincante puede en estos casos efectuar un ataque estadístico y extraer bastante información.

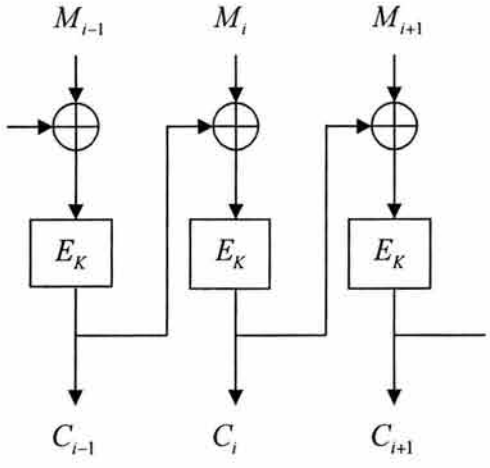
Otro riesgo bastante importante que presenta el modo ECB es el de la substitución de bloques. El atacante puede cambiar un bloque sin mayores problemas, y alterar los mensajes incluso desconociendo la clave, y el algoritmo empleado. Simplemente se escucha una comunicación de lo que se conozca el contenido, como por ejemplo una transacción bancaria a nuestra cuenta corriente.

Luego se escuchan otras comunicaciones y se substituyes los bloques correspondientes al número de cuenta del beneficiario de las transacción por la versión codificada de nuestro número (no necesitamos descifrar). El resultado es que habremos obtenido acceso a la cuenta sin ningún problema.

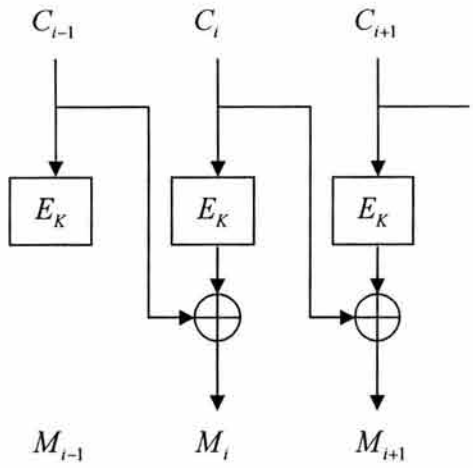
2.7.2.- Modo CBC (Cipher Book Chaining Mode).

El modo CBC incorpora un mecanismo de retroalimentación en el cifrado por bloques. Esto significa que la codificación de bloques anteriores condiciona la codificación del bloque actual, por lo que será imposible sustituir un bloque individual en el mensaje cifrado. Esto se consigue efectuando una operación XOR entre el bloque del mensaje que queremos codificar y el último criptograma obtenido. En cualquier caso, dos mensajes idénticos se codifican de la misma forma usando el modo CBC, más aún, dos mensajes que empiecen igual se codificarán igual hasta que lleguen a la primera diferencia entre ellos. Para evitar esto se emplea un vector de inicialización, que puede ser un bloque aleatorio, como bloque inicial de la transmisión. Este vector será descartado en destino, pero garantiza que siempre los mensajes se codifiquen de manera diferente, aunque tengas partes comunes.

CODIFICACIÓN



DECODIFICACIÓN

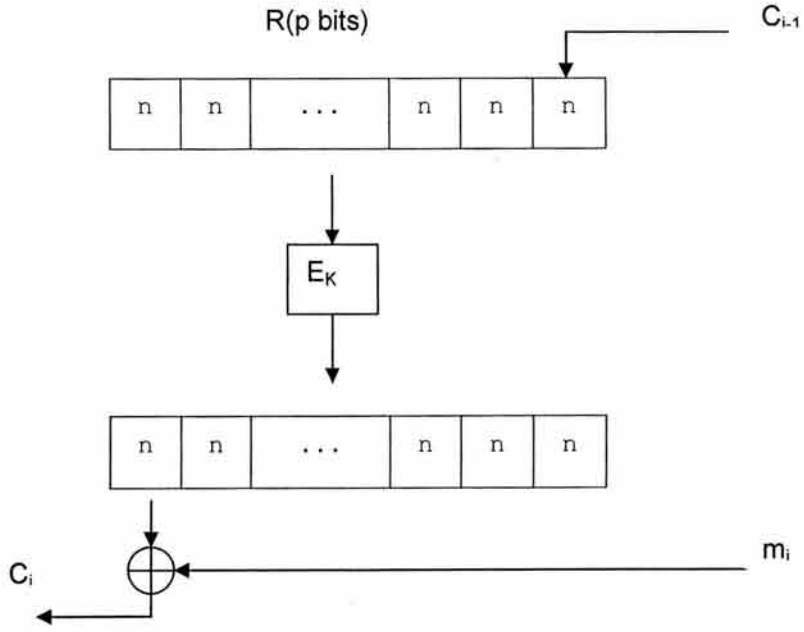


2.7.3.- Modo CFB(Cipher Feedback Mode).

El modo CFB no empieza a codificar (o decodificar) hasta que no se tiene que transmitir (o se ha recibido) un bloque completo de información. Esta circunstancia puede convertir en un serio inconveniente, por ejemplo en el caso de termínales, que deberían poder trasmitir cada carácter que pulsa el usuario de manera individual. Una posible solución sería emplear un bloque completo para transmitir cada byte y rellenar el resto de ceros, pero esto hará que tengamos únicamente 256 mensajes diferentes en nuestra transmisión y que un atacante puede efectuar un sencillo análisis estadístico para comprometerla. Otra opción sería rellenar el bloque con la información aleatoria, aunque seguiríamos desperdiciando gran parte del ancho de banda de la transmisión. El modo de operación CFB permitirá codificar la información en unidades inferiores al tamaño del bloque, lo cual permite aprovechar totalmente la capacidad de transmisión del canal de comunicaciones, manteniendo además un nivel de seguridad adecuado.

Podemos ver el esquema de funcionamiento de este modo de operación.

Sea *p* el tamaño de bloque del algoritmo simétrico, y sea *n* el tamaño de los bloques que queremos trasmitir (*n* ha de divisor de *p*). Sea m_i el i-ésimo bloque del texto plano, de tamaño *n*. empleamos entonces un registro de desplazamiento *R* de longitud *p* y lo cargamos con un vector de inicialización. Codificamos el registro *R* con el algoritmo simétrico y obtenemos en *r* sus *n* bits más a la izquierda. El bloque que deberemos enviar es $c_i = r \oplus m_i$. Desplazamos *R n* bits a la izquierda e introducimos c_i por la derecha como hacíamos en el algoritmo de cifrado.



2.7.4.- Otros Modos.

Existen protocolos criptográficos que no se basan en transmisión de bloques, sino en un mecanismo de secuencia de codificación de flujo (*stream)* de tamaño variable. Estos algoritmos permiten cifrar un mensaje bit a bit de forma continua y enviar cada bit antes que el siguiente sea codificado. Funcionan a partir de lo que se llama un generador de secuencia de clave (*keystream generator*), un algoritmo que genera una clave continua de longitud infinita (o muy grande) bit a bit. Lo que hace es aplicar una operación XOR entre cada bit de texto plano a cada bit de la clave. En el destino existe otro generador idéntico sincronizado para llevar a cabo el descifrado. El problema fundamental es mantener ambos generadores sincronizados, para evitar errores si se pierde algún bit de la transmisión.

Los algoritmos de codificación por bloques pueden ser empleados como generadores de secuencia de clave. Existen para ello otros modos de operación de estos algoritmos, como el OFB (Output-Feedback), que incorpora mecanismos para mantener la sincronía entre los generadores de secuencia origen y destino.

2.8.- El Algoritmo RIJNDAEL.

2.8.1.- Introducción.

El algoritmo de cifrado Rijndael fue desarrollado por el Dr. Joan Daemen nacido en Bélgica, doctorado en Criptografía miembro del CSIC (*Computer Security and Industrial Cryptography*) y por el Dr. Vincent Ridjmen nacido Bélgica, doctorado en los laboratorios ESAT/COSIC de la Universidad Católica de Leuven con la tesis titulada *"Criptoanálisis y diseño de cifrados de bloque iterativos"*.

En 1993 consideran la necesidad de desarrollar un nuevo sistema de cifrado más robusto ante la llegada de un nuevo milenio y con él, el incremento de la capacidad de procesamiento de las computadoras, por lo que la integridad de los algoritmos contemporáneos de cifrado se vería, en un breve periodo de tiempo, seriamente comprometidos. El actual AES (*Advanced Encryption Standard*) elegido por el NIST (*National Institute of Standard and Tecnology*) era el algoritmo DES. Dicho algoritmo se estaba quedando obsoleto, y no por una debilidad teórica si no por el vertiginoso aumento de la velocidad de cómputo usado en el ataque de fuerza bruta simplemente probando todas y cada una de las 72,057,594,037,927,776 claves posibles. Se hace patente la necesidad de encontrar un nuevo AES.

El Dr. Daemen y el Dr. Rijmen inician el desarrollo de un algoritmo al que denominan *Square*. Su principal característica diferenciadora con el respecto de los demás algoritmos existentes es que

trabaja con claves de 128 bits y con bloques de cifrado de igual tamaño. En 1996 finalizan su primer diseño y en la primavera de 1997 lo hacen público.

Coincidiendo con esta fecha, el NIST anuncia el concurso para la selección del próximo AES, título que tenia el algoritmo DES desde la década de los 70. Como medida temporal mientras no se haya seleccionando el AES el NIST recomienda el uso del Triple DES para garantizar la privacidad de los documentos. Anteriormente ese mismo organismo había intentado poner sin éxito un algoritmo secreto, pero el temor de que ese algoritmo contuviese "puertas secretas" que facilitasen el descifrado de la información, el proyecto no tuvo aceptación, por lo que se tomó la decisión de sacar la propuesta a concurso público. En el verano del año 1997 sale a la luz los requisitos que deben tener los algoritmos que se quieran presentar a dicho concurso. Uno de ellos es que los algoritmos deben trabajar con longitudes de claves de 128,192 y 256 bits y longitudes de bloque de no menos 128 bits. Se desestima imponer obligatoriamente el uso de valores de tamaño de bloque 192 y 256 bits por considerarlos inviables. Los doctores Rijmen y Daemen inician la modificación del algoritmo Square para adaptarlo a las condiciones del concurso y en junio de 1998 remiten al NIST un nuevo algoritmo, descendiente del anterior Square y al que llaman Rijndael.(**Rij**men & **Dae**men). En agosto de 1998 finaliza el plazo de presentación de algoritmos. Se han presentado un total de 15 participantes, de los cuales tres de ellos (Rijndael, Crypton y Twofish) están basados en la estructura de. Square. Los algoritmos son analizados por un grupo de expertos designados por el NIST que valoran aspectos tales como su robustez, su estructura, su capacidad de ser implementados tanto en software como en hardware y muchos más aspectos como el trabajo intelectual efectuado en el desarrollo. Se crea también un foro público, de carácter informal, para que toda aquella persona que lo desee pueda acceder a la documentación de los candidatos y así dar su propia opinión sobre ellos.

Tras una selección inicial se publican los cinco finalistas en agosto de 1999: Twofish, Mars, Serpent, RC6 y Rijndael. Rijndael se destaca frente a sus competidores ya que, tanto su tamaño de clave como su tamaño de bloque de cifrado puede ser de 128,192 0 256 bits y, lo que es más importante, pueden ser combinados arbitrariamente entre ellos, por lo que disponemos de nueve posibilidades de asociación clave-bloque, todas ellas posibles. Los otros dos algoritmos basados en Square (Crypton y Twofish) pueden operar con los tres tamaños de clave pero no de manera arbitraria. Además Rijndael es el único de los finalistas que puede operar con distintos tamaños de bloque cifrado.

El 2 de Octubre de 2000 se proclama el algoritmo Rijndael como ganador del concurso y nuevo AES.

Sobre las razones por las que se eligió Rijndael frente a los otros algoritmos candidatos, el NIST dice:

"Tomando todo en consideración, la combinación de seguridad, rendimiento, eficiencia, facilidad de implementación y flexibilidad lo hacían la elección adecuada para el AES.

Específicamente, Rijndael consistentemente obtiene muy buen rendimiento tanto en hardware como en software en una amplia variedad de entornos de computación tanto usado en modo feedback como no feedback. Su preparación de claves es excelente y la agilidad de las claves buena. Rijndael requiere muy poca memoria lo que lo hace excelente para entornos con espacio restringido demostrando aquí también su excelente rendimiento. Las operaciones de Rijndael están entre las más sencillas de defender contra ataques por análisis temporal (timing attack) y criptoanálisis diferencial de potencia (power attack).

Por otro lado parece que se puede proporcionar defensa contra los ataques citados sin afectar significativamente el rendimiento de Rijndael. El algoritmo se ha diseñado con algo de flexibilidad en términos de los tamaños de bloque y de clave y pueden acomodarse a alteraciones en los números de rondas aunque estas modificaciones necesitarán estudio adicional y no se toman en cuenta en este momento. Finalmente la estructura interna de las rondas de Rijndael parece tener buen potencial para beneficiarse del proceso en paralelo".

El NIST apunta también que en términos de seguridad los cinco finalistas son adecuados, pero Rijndael era el que mejor combinaba el resto de características deseables.

Las claves utilizadas por Rijndael de 128, 192 o 256 bits proporcionan un muy grande número de posibilidades. Comparándolo con el DES(56 bits) el AES con 128 bits tendría del orden de 10^{21} claves más. Por tanto si se intentara romper por fuerza bruta una clave de 128 bits con una máquina que identificase una clave DES en 1 segundo de promedio se necesitarían del orden de 10^{14} años.

Confiando en estas claves y contando incluso con los previsibles avances tecnológicos el NIST espera tener AES para más de 20 años.

2.8.2.- Preliminares Matemáticos.

La seguridad del cifrado Rijndael se basa en el desorden provocado en el contenido del texto en claro al aplicarse una serie de permutaciones y otras operaciones matemáticas. Algunas de estas operaciones se realizan a nivel de byte, representando mediante el campo llamado $GF(2^8)$ y otras a nivel de palabras de cuatro bytes.

Para la comprensión del algoritmo es necesario conocer cada una de estas operaciones, por lo cual se deberá introducir una serie de conceptos matemáticos antes de iniciar el estudio del algoritmo propiamente dicho.

2.8.2.1.- Representación de un byte en el Campo $GF(2^8)$.

En el campo $GF(2^8)$ se trata de un campo finito en el que los elementos (en nuestro caso bytes) serán representados como polinomios de grado 7 y con coeficientes binarios, esto es, en {0,1}.

Un byte *b* se compone de 8 bits que representamos como b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0 donde b_7 representa el bit de mayor peso y b_0 al de menor. Así podemos representar el byte como un polinomio cuyos coeficientes son los b_j con j=0...7, y donde estos b_j pueden tomar los valores 0 ó 1.

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0$$

Donde cada b_i representa el bit significativo para cada *i*.

Por ejemplo, un byte que represente el valor hexadecimal '57' (en binario 01010111) se corresponde con el polinomio:

$$0x^7 + 1x^6 + 0x^5 + 1x^4 + 0x^3 + 0x^3 + 1x^2 + 1x^1 = x^6 + x^4 + x^2 + x + 1$$

un byte que represente el valor hexadecimal '83' (en binario 10000011) se corresponde con el polinomio:

$$1x^7 + 0x^6 + 0x^5 + 0x^4 + 0x^3 + 0x^2 + 1x^1 + 1 = x^7 + x + 1$$

2.8.2.2.- Suma.

La suma de dos elementos del campo $GF(2^8)$ es la suma de dos polinomios, por lo que el resultado será otro polinomio. La suma de los coeficientes se corresponde con una suma modulo 2 término a término. Se puede comprobar que esta suma se corresponde con una operación EXOR(denotada por $\oplus$) entre los coeficientes de los polinomios.

Por ejemplo se puede efectuar la suma de los elementos del ejemplo anterior:

$$(57 + 83) = (x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2$$

Podemos comprobar que el conjunto de los polinomios de grado menor o igual que 7 y con coeficientes pertenecientes a Z_2 forman un grupo conmutativo con la suma, es decir, es una operación interna, que cumple la propiedad asociativa, conmutativa, tiene elemento neutro y tiene simétrico. Debido a la existencia de simétrico podemos referirnos a la operación resta, ya que se pueden definir la resta de *a* y *b*, donde *a* y *b* son polinomios, como la suma de a con el simétrico de *b*.

2.8.2.3.- Multiplicación.

Al referirnos a la multiplicación empleada en el algoritmo Rijndael nos estaremos refiriendo realmente a la multiplicación de dos elementos del conjunto $GF(2^8)$, es decir polinomios de grado

menor o igual que 7 y con coeficientes en Z_2 pero cuyo resultado se expresa modulo *m(x)* donde *M(x)* = $x^8+x^4+x^3+x+1$. Nótese que *m(x)* se puede representar en hexadecimal con el valor '11B' y se puede comprobar que es un polinomio irreducible. El propósito de realizar la multiplicación módulo *m(x)* es con el fin de que el resultado obtenido en la operación siga siendo un polinomio de grado menor que 8, por lo que la operación seguiría siendo a nivel de byte.

Por ejemplo, la operación multiplicación de los valores hexadecimales '57' y '83' sería:

$$(x^6+x^4+x^2+x+1)\bullet(x^7+x+1)=$$
$$(x^6+x^4+x^2+x+1)+(x^7+x^5+x^3+x^2+x)+(x^{13}+x^{11}+x^9+x^8+x^7)$$
$$=x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+1$$

Como se puede apreciar este polinomio es de grado mayor que 8 por lo que no pertenece a $GF(2^8)$ y así la operación no se realiza a nivel de byte. Para remediar esto y conseguir que la multiplicación siga siendo una operación interna en $GF(2^8)$ expresamos el resultado obteniendo modulo *m(x)*.

$$(x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+1) \bmod m(x)$$
$$=x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+1 \bmod (x^8+x^4+x^3+x+1)$$
$$=x^7+x^6+1$$

Un caso destacado en cuanto a la multiplicación de polinomios en $GF(2^8)$ es cuando nos surge la multiplicación de un polinomio *b(x)* de grado 7 por el polinomio *c(x)=x*

$$a(x)\bullet b(x)=x\bullet(b_7x^7+b_6x^6+b_5x^5+b_4x^4+b_3x^3+b_2x^2+b_1x^1+b_0)$$
$$=b_7x^8+b_6x^7+b_5x^6+b_4x^5+b_3x^4+b_2x^3b_1x^2+b_0x$$

Donde *a(x)* y *b(x)* son dos polinomios de grado 7.

Como se puede apreciar, este polinomio no pertenece a $GF(2^8)$ ya que su grado es 8. Para que la operación sea interna en $GF(2^8)$ dividimos entre $m(x)=x^8+x^4+x^3+x+1$ como hemos visto anteriormente.

Si el coeficiente b_7 de *b(x)* tiene valor 0, la operación será simplemente una función identidad de *a(x)*b(x)* ya que, como el grado de este polinomio es menor que el grado de *m(x)* el resto de la división entre *m(x)* será el propio polinomio *a(x)*b(x)*.

$$(b_7x^8+b_6x^7+b_5x^6+b_4x^5+b_3x^4+b_2x^3+b_1x^2+b_0x) \bmod (x^8+x^4+x^3+x+1)$$
$$=b_7x^8+b_6x^7+b_5x^6+b_4x^5+b_3x^4+b_2x^3+b_1x^2+b_0x$$

Si el coeficiente b_7 de *b(x)* tiene valor 1, la división de *a(x)*b(x)* entre *m(x)* será en la realidad una resta, ya que ambos polinomios tiene el mismo grado.

$$
\begin{aligned}
&(b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x) \bmod (x^8 + x^4 + x^3 + x + 1) \\
&= (b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x) - (x^8 + x^4 + x^3 + x + 1) \\
&= b_6x^7 + b_5x^6 + b_4x^5 + (b_3 + 1)x^4 + (b_2 - 1)x^3 + b_1x^2 + (b_0 - 1)x - 1 \\
&= b_6x^7 + b_5x^6 + b_4x^5 + (b_3 + 1)x^4 + (b_2 + 1)x^3 + b_1x^2 + (b_0 - 1)x + 1
\end{aligned}
$$

Se puede apreciar que estas dos operaciones se pueden implementar con 4 funciones EXOR, 3 sobre los bits b_3, b_2, b_1 para la resta con los respectivos coeficientes de *m(x)*, y una cuarta función EXOR que comprende el valor del bit b_7 de *b(x)* para saber la operación a realizar, es decir, la función resta o la función identidad.

A esta operación se le denomina *b=xtime(a)*. Si se aplica esta operación reiterativamente encontraremos una forma sencilla y fácil de implementar el producto de un polinomio por una potencia de x.

$$
\begin{aligned}
&b(x) \bullet x = xtime(b(x)) \\
&b(x) \bullet x^2 = (b(x) \bullet x) \bullet x = xtime(b(x)) \bullet x = xtime(xtime(b(x))) \\
&b(x) \bullet x^3 = (b(x) \bullet x^2) \bullet x = (xtime(xtime(b(x)))) \bullet x = xtime(xtime(xtime(b(x))))
\end{aligned}
$$

2.8.2.4.- Representación de Palabras en el Campo $GF(2^8)$.

Recordemos que denotamos como una "palabra" a un conjunto de bytes. En este sentido una palabra formada por cuatro bytes se puede representar como un polinomio de grado menor o igual que tres.

$$
\begin{aligned}
a(x) &= a_3x^3 + a_2x^2 + a_1x + a_0 \\
b(x) &= b_3x^3 + b_2x^2 + b_1x + b_0
\end{aligned}
$$

La operación suma de polinomios se realiza mediante unas operaciones EXOR byte a byte, al igual que se hacia anteriormente al representar un byte mediante un polinomio de grado menor que 7. Esta operación es interna en $GF(2^8)$ ya que la suma de dos polinomios de grado menor que 4 nos dará como resultado otro polinomio de grado menor que 4.

En cuanto a la operación multiplicación, nos encontramos de nuevo con el problema visto anteriormente. La multiplicación puede no ser una operación interna en $GF(2^8)$, por lo que el producto

de dos palabras de 4 bytes puede no ser representable por una palabra de 4 bytes, es decir, mediante un polinomio de grado menor que 4.

$$a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$$
$$b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$$
$$a(x) \bullet b(x) = c(x) = c_6x^6 + c_5x^5 + c_4 + c_3x^3 + c_2x^2 + c_1x + c_0$$

donde

$c_0 = a_0 \bullet b_0$
$c_1 = a_1 \bullet b_0 \oplus a_0 \bullet b_1$
$c_2 = a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2$
$c_3 = a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3$
$c_4 = a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3$
$c_5 = a_3 \bullet b_2 \oplus a_2 \bullet b_3$
$c_6 = a_3 \bullet b_3$

Para solucionar este contratiempo se ha propuesto una solución semejante a la anterior: el resultado de la operación multiplicación se expresa módulo un polinomio de grado 4. Los autores del algoritmo Rijndael han elegido el polinomio $M(x) = x^4+1$ para tal fin. En este caso $M(x)$ no es un polinomio irreducible como se le había exigido al anterior $m(x)$. Esto va a provocar que algunas multiplicaciones que realicemos puedan dar un resultado que no tenga inverso. En este caso del algoritmo Rijndael este caso no se va a presentar nunca ya que siempre se multiplica polinomios que poseen inverso.

$(a(x) * b(x)) \bmod(x^4 + 1) = d(x)$ donde $d(x) = d_3x^3 + d_2x^2 + d_1x + d_0$

Para hallar d_3, d_2, d_1, d_0 aplicamos una sencilla regla:

$$x^j \bmod(x^4 + 1) = x^{j \bmod 4}$$

Así obtenemos los siguientes valores:

$$d_0 = a_0 + b_0 \oplus a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3$$
$$d_1 = a_1 + b_0 \oplus a_0 \bullet b_1 \oplus a_3 \bullet b_2 \oplus a_2 \bullet b_3$$
$$d_2 = a_2 + b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2 \oplus a_3 \bullet b_3$$
$$d_3 = a_3 + b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3$$

Otra forma de expresar esta operación es mediante el uso de matrices

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \bullet \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Un caso especial de la multiplicación de polinomios es la multiplicación de un polinomio *b(x)* por el polinomio *x*

$$b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$$
$$d(x) = x \bullet b(x) = b_3x^4 + b_2x^3 + b_1x^2 + b_0x$$

Dividimos *d(x)* entre *M(x)=x4+1* para obtener un polinomio de grado menor que cuatro

$$c(x) = d(x) \bmod (x^4 + 1) = b_2x^3 + b_1x^2 + b_0x + b_3$$

Esta multiplicación también se puede expresar como el producto de dos matrices, de la misma forma que la matriz anterior pero cuyos elementos son sustituidos todos '00', a excepto los a_1, que se sustituyen por el valor '01'

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 00 & 00 & 00 & 01 \\ 01 & 00 & 00 & 00 \\ 00 & 01 & 00 & 00 \\ 00 & 00 & 01 & 00 \end{bmatrix} \bullet \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Al igual que en el caso de los polinomios de grado menor que siete, que representan a los bits de un byte, si aplicamos reiterativamente esta multiplicación obtenemos una manera rápida y sencilla de multiplicar un polinomio por cualquier potencia de *x*.

2.8.3.- Fundamentos del Diseño de Rijndael.

Los creadores del algoritmo se han basado en tres criterios fundamentales:

- Resistencia contra la totalidad de los ataques conocidos.
- Velocidad, código compacto y operativo en un gran número de plataformas distintas
- Simplicidad de diseño.

Una de las diferencias más notables de Rijndael con respecto al resto de los algoritmos de cifrado existentes radica en que Rijndael no recurre a una estructura interna tipo Redes de Feistel. En una estructura tipo Feistel propiamente dicha, en cada una de las operaciones de cada vuelta los bits sufren una permutación pero la mayoría de estos bits permutan su lugar pero sin variar su valor, es decir, llegan a su nueva posición sin ninguna modificación de valor. En Rijndael, las transformaciones de cada vuelta se componen a su vez de otras tres transformaciones inversibles y distintas entre si que reciben el nombre de *"layers"* (capas). Estas transformaciones están basadas en el principio de diseño llamado *"Wide Trail"* que otorga resistencia al algoritmo frente a ataques de tipo lineal y diferencial. Cada *layer* tiene su propia función específica dentro de esta estrategia:

- Layer mezcla lineal: garantiza una alta difusión sobre múltiples vueltas
- Layer no lineal: permite la aplicación paralela de S-Box para conseguir unas óptimas propiedades no lineales del peor caso
- Layer para la adicción de la clave: Es una función EXOR entre los bits del estado intermedio y la subclave correspondiente a ese estado

Antes de que se proceda a la primera vuelta el algoritmo, se aplica el *layer* para la adicción de la clave. Cualquier *layer* que apliquemos tras la última adicción de clave o antes de la primera en los ataques basados en textos claros conocidos, puede ser descubierta sin conocer la clave y esto seria un fallo en la propia seguridad del algoritmo, como se puede apreciar en la permutación inicial y final del algoritmo DES. Este modo de actuación no es algo nuevo, ya que se aplica en algoritmos tales como IDEA o Blowfish.

En la última vuelta del algoritmo se aplica un *layer* lineal diferente a los aplicados en las anteriores vueltas. Esto se determino así con el fin de que el algoritmo de cifrado y el de descifrado sean lo más similares posible. Esto no reduce en absoluto la seguridad del algoritmo y es una técnica aplicada en algoritmos tales como el DES, en el que se modifica la última operación para conseguir el mismo fin.

2.8.4.- Especificaciones de Rijndael.

Rijndael es un cifrador de bloque y de carácter interactivo, esto es, que realiza varias vueltas de cifrado sobre un mismo conjunto de bits. Su característica diferenciadora respecto al resto de los algoritmos existentes es que Rijndael nos permite especificar un tamaño de bloque de 128, 192 o 256 bits, y lo que es más importante, combinarlos entre sí de diferentes maneras, por lo que disponemos de nueve asociaciones entre el tamaño de clave y del bloque.

2.8.4.1.- Los Estados, la Clave y el Número de Vueltas.

Cada uno de los estados intermedios del algoritmo puede ser representado como una matriz de bytes. Esta matriz tiene 4 filas y el número de columnas, denotado por *Nb*, es variable, siendo igual al tamaño del bloque dividido por 32, es decir, para un tamaño de bloque de 128 bits le corresponden 4 columnas (128/32), para un tamaño de 192 bits le corresponden 6 columnas (192/32) y para 256 bits le corresponden 8 columnas.

Estado intermedio con tamaño de bloque de 128 bits

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

Estado intermedio con tamaño de bloque de 192 bits

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$

Estado intermedio con tamaño de bloque de 256 bits

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$	$a_{0,6}$	$a_{0,7}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$	$a_{1,6}$	$a_{1,7}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$	$a_{2,6}$	$a_{2,7}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$	$a_{3,6}$	$a_{3,7}$

Las claves de los estados intermedios también pueden ser representadas de manera similar mediante una matriz de cuatro filas y un número variable de columnas, que denotamos por *Nk* y que se obtiene al dividir el tamaño de clave por 32, esto es, 4, 6 y 8 columnas para tamaños de clave de 128, 192 y 256 bits respectivamente.

Representación de una clave de 128 bits

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$

Representación de una clave de 192 bits

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$	$k_{0,4}$	$k_{0,5}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$	$k_{1,4}$	$k_{1,5}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$	$k_{2,4}$	$k_{2,5}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$	$k_{3,4}$	$k_{3,5}$

Representación de una clave de 256 bits

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$	$k_{0,4}$	$k_{0,5}$	$k_{0,6}$	$k_{0,7}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$	$k_{1,4}$	$k_{1,5}$	$k_{1,6}$	$k_{1,7}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$	$k_{2,4}$	$k_{2,5}$	$k_{2,6}$	$k_{2,7}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$	$k_{3,4}$	$k_{3,5}$	$k_{3,6}$	$k_{3,7}$

En determinadas ocasiones estas matrices se pueden considerar como un único vector unidimensional de palabras de 4 bytes, donde cada palabra se corresponde con una de las columnas de la matriz. Estos vectores tendrían por tanto una longitud de 4, 6 y 8 palabras según el número de columnas de la matriz. Los índices de estos vectores serían 0..3, 0..5, respectivamente. Si necesitamos referirnos a un byte en concreto de una palabra usaremos la notación *(a,b,c,d)* donde la

letra *a* se corresponde con el byte 1 de la palabra, la letra *b* se corresponde con el 2, la *c* con el 3 y la letra d con el cuarto byte.

La entrada y salida de datos en Rijndael se realiza mediante un vector unidimensional de bytes de 8 bits por byte. Estos bytes están numerados con valores que parten desde 0 hasta *(4*Nb)-1*, es decir, para *Nb=4* el número de bytes seria 16 y estarían numerados desde el 0 hasta el 15. En la tabla podemos ver las relaciones entre el tamaño del bloque, el valor de *Nb* y el rango de índices correspondientes del vector entrada-salida.

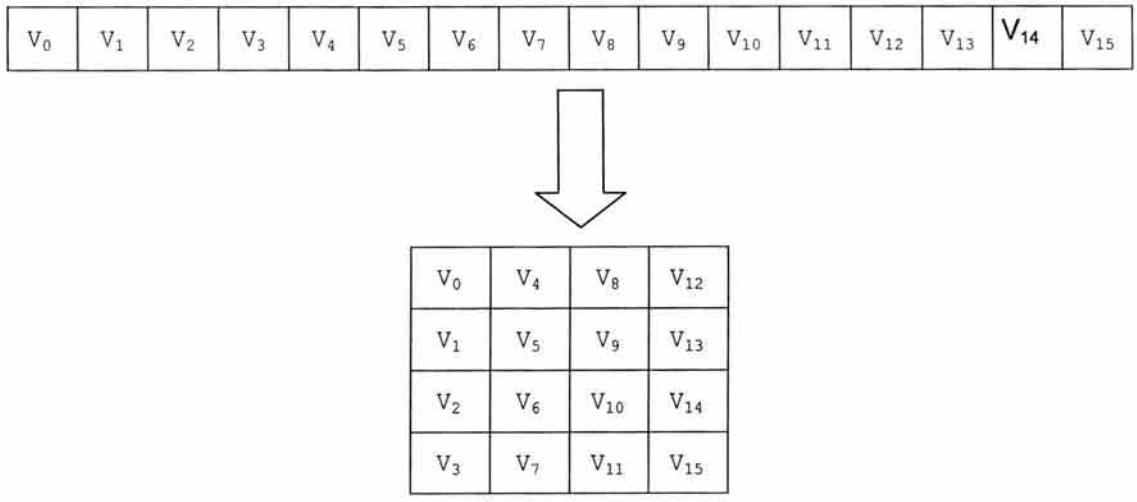
Número de bits	Número de Bytes	Nb	Tamaño del vector de entrada	Rango de índices
128	16	4	15	0..15
192	24	6	24	0..23
256	32	8	32	0..31

Una vez tengamos el vector de entrada completo, necesitamos volcar el contenido en la matriz correspondiente. El modo de realizar esta operación será el siguiente: leeremos el vector secuencialmente desde la posición 0 e iremos rellenando la matriz por columnas, esto es, empezaremos con la columna 0 e iremos rellenado hacia abajo hasta el final para luego proceder de igual forma con la columna siguiente.

Como ejemplo se verá como sería este traspaso de información para el caso de tamaño de bloque de 128 bits. El vector de entrada es de 16 bytes numerados desde el 0 hasta el 15.

Denotaremos por *V* al vector de entrada y sus componentes serán denotados por v_j.

Traslado de información desde un vector de entrada a la matriz.



Para volcar el contenido de la matriz en un vector de salida basta con aplicar el mismo proceso en orden inverso: leemos la matriz por columnas, empezando en la columna 0 y vamos rellenando el vector desde la posición 0 hasta el final. La correspondencia entre las posiciones que ocupa un mismo elemento en la matriz y en el vector se puede enunciar de tal forma que, suponiendo un elemento que ocupa la posición *n* en el vector unidimensional y la posición *(i,j)* en la matriz, la relación entre *n* y *(i,j)* sea la siguiente:

$$i = n \bmod 4$$
$$j = n/4$$
$$n = i + 4 * j$$

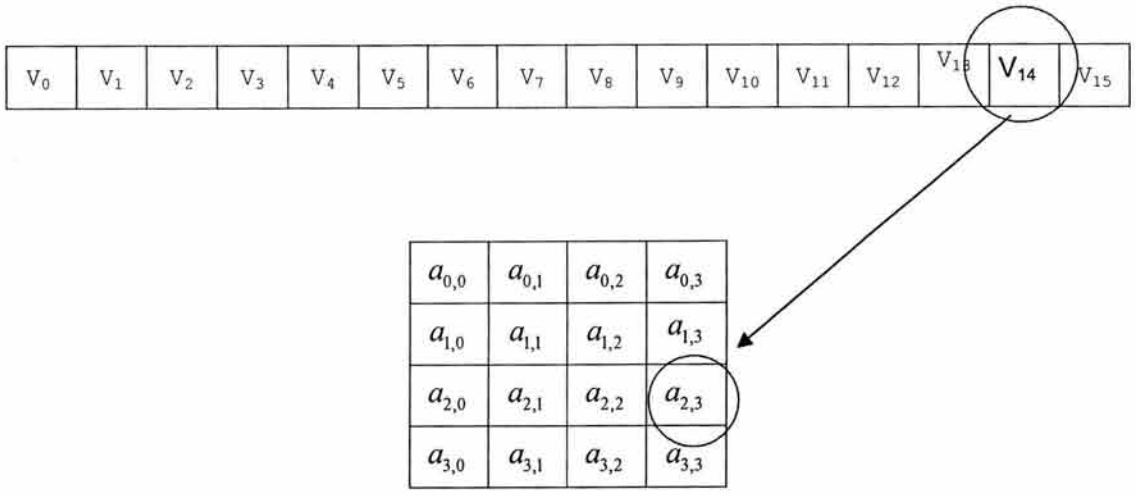
Por ejemplo, suponiendo que queremos hallar la posición en la matriz del elemento de la posición 14 del vector, tenemos que:

$$n = 14$$
$$i = 14 \bmod 4 = 2$$
$$j = 14/4 = 3$$

Vemos que el elemento de la posición 14 del vector ocupa la posición (2,3) en la matriz. Comprobamos el caso contrario; esto es:

$$i = 2$$
$$j = 3$$
$$n = 2 + 4 * 3 = 14$$

Localización de un elemento en la matriz.



El número de vueltas que tendrá el algoritmo es un parámetro variable que depende de los valores *Nb* y *Nk*, y estos a su vez dependen del tamaño de bloque y del tamaño de clave respectivamente. Vamos a denotar el número de vueltas con la notación *Nr*. El valor de *Nr* viene dado por la tabla siguiente:

Número de vueltas del algoritmo según el valor de Nb y Nk.

Nr	*Nb*=4	*Nb*=6	*Nb*=8
Nk=4	10	12	14
Nk=6	12	12	14
Nk=8	14	14	14

Se puede apreciar que el número de vueltas será mayor cuando mayor sea el tamaño del bloque y de la clave.

2.8.4.2.- Operaciones en cada Vuelta.

En cada vuelta se aplica una transformación que a su vez se compone de cuatro subtransformaciones. En notación pseudo código se especificaría de la siguiente manera:

```
Round (State, RoundKey)
{
  ByteSub(State);
  ShiftRow(State);
  MixColumn(State);
  AddRoundKey(State, RoundKey);
}
```

Como se había mencionado anteriormente, la última vuelta difiere de las anteriores en que carece de la operación MixColumn(State). La última vuelta sería de la siguiente manera:

```
FinalRound(State, RoundKey)
 {
   ByteSub(State);
   ShiftRow(State);
   AddRoundKey(State, RoundKey);
 }
```

Transformación SubByte.

Se trata de una sustitución a nivel de byte de carácter no lineal. Se aplica sobre todos y cada uno de los bytes de un estado de manera independiente. Se denota ByteSub(State).

Es una transformación inversible y se compone de dos pasos:

Primero, aplicando la función inversa para la multiplicación en $GF(2^8)$. Cada elemento se sustituye por su valor inverso, salvo el valor *'00'*, que se sustituye por sí mismo ya que carece de inverso para la multiplicación con coeficientes pertenecientes a $GF(2^8)$.

Se aplica la siguiente transformación donde cada x_j representa el bit *j* del valor del byte al que vamos a aplicar la transformación y cuyo valor final se representa por *yj*.

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 00 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \bullet \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

donde cada x_j representa el bit *j* del valor del byte al que vamos a aplicar la transformación y cuyo valor final se representa por y_j.

La Transformación ShiftRow.

Esta transformación consiste en el desplazamiento de forma cíclica a la izquierda de los bytes de cada una de las filas de la matriz representante de un estado. Se denota por ShiftRow(State).

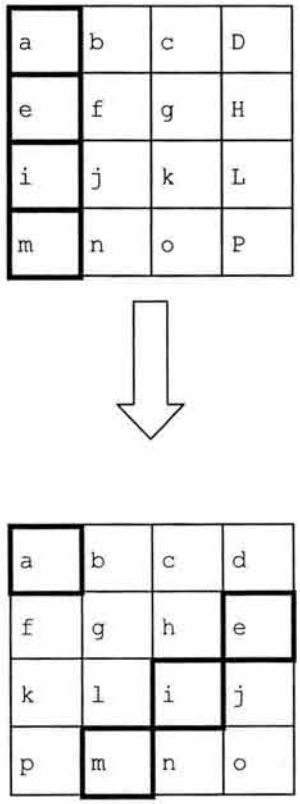
Los bytes de la fila 0 de la matriz no se desplazan. Los bytes de las filas 1,2,3 se desplazan *C1*, *C2* y *C3* posiciones respectivamente a la izquierda. Estos valores de desplazamiento dependen del valor de *Nb*, es decir del número de columnas de la matriz, o lo que es lo mismo, del tamaño del bloque. En la tabla siguiente se muestran dichos valores.

Valor de los desplazamientos *C1, C2, C3* según *Nb*.

NB	C1	C2	C3
4	1	2	3
6	1	2	3
8	1	3	4

Para comprender mejor el funcionamiento de esta transformación vamos a probar la transformación en un estado con *Nb =4*. En este caso los desplazamientos serán de 1, 2 y 3 para *C1*, *C2* y *C3* respectivamente. A la fila 0 no se le aplica la transformación, los bytes de la fila 1 se desplazan 1 posición a la izquierda, los de la fila 2 se desplazan 2 posiciones a la izquierda y los de la fila 3 se desplazan 3 posiciones. Aunque en este caso coincida el número de fila con el valor del desplazamiento, éstos se efectúan siguiendo la Tabla.

Ejemplo de desplazamiento de bytes *C1, C2 y C3*



La operación inversa a esta transformación consiste simplemente en desplazar los bytes las filas 1,2 y 3 un total de *C1, C2* y *C3* posiciones a la derecha.

La transformación MixColumn.

En esta transformación las columnas de la matriz representante del estado son consideradas como polinomios con coeficientes pertenecientes a $GF(2^8)$. Estos polinomios se multiplican modulo $M(x)$ (recordemos que $M(x) = x^4 + 1$) por un polinomio $d(x)$ dado:

$$d(x) = '03'x^3 + '01'x^2 + '01'x + '02' \text{ en hexadecimal}$$
$$d(x) = '11'x^3 + 1x^2 + 1x + 10 \quad \text{en binario}$$
$$d(x) = 3x^3 + 1x^2 + 1x + 2 \quad \text{en decimal}$$

Esta operación puede ser escrita en forma de matriz, donde a_j representa al byte *j* de la columna de la matriz estado y b_j al nuevo byte tras la operación. Esta operación se denota *MixColumn(State)*.

Operaciones de la función MixColumn

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \bullet \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

La inversa de esta operación consiste en multiplicar la columna transformada por el inverso del polinomio anterior *d(x)* para obtener la columna inicial.

Denotamos por *a(x)* al inverso del polinomio *d(x)*, entonces

$$a(x) = '03'x^3 + '01'x^2 + '01'x + '02'$$

por la definición de inverso de un elemento en un grupo sabemos que:

$$a(x) \otimes d(x) = '01'$$
$$a(x) \otimes ('03'x^3 + '01'x^2 + '01'x + '02') = '01'$$

Despejamos y obtenemos que:

$$a(x) = '0B'x^3 + '0D'x^2 + '09'x + '0e'$$
$$a(x) = 1011x^3 + 1101x^2 + 1001x + 11110 \quad \text{en binario}$$
$$a(x) = 11x^3 + 13x^2 + 9x + 14 \quad \text{en decimal}$$

La Adicción de la Subclave.

Esta operación se denota por *AddRoundKey(State, Roundkey)*. Consiste en una operación EXOR entre los elementos de la matriz del estado y los elementos de la matriz de la subclave. Esta subclave

es el resultado de aplicar el proceso llamado key schedule, el cual se tratará más adelante, a la clave principal. La matriz de la clave tiene el mismo número de columnas que la matriz de bloque (*Nb*).

La inversa de la operación *AddRoundKey(State, Roundkey)* es la propia operación. Aplicando la operación a la matriz ya transformada obtendremos de nuevo la matriz inicial, debido a que son operaciones EXOR entre bytes.

Adicción de la clave mediante función EXOR

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$

$\oplus$

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$	$k_{0,4}$	$k_{0,5}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$	$k_{1,4}$	$k_{1,5}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$	$k_{2,4}$	$k_{2,5}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$	$k_{3,4}$	$k_{3,5}$

$=$

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$

Key Shedule.

Mediante esta función se obtienen las diferentes subclaves a partir de la clave principal de cifrado. Se compone a su vez de dos funciones, la KeyExpansion y la RoundKeySelection. La función Key schedule se basa en lo siguiente:

- El número total de subclaves es igual al tamaño del bloque multiplicado por el número de vueltas más 1.

Número de subclaves = *Nb * (Nr + 1)*

Número de Subclaves Generadas

Tamaño de bloque	Tamaño de la clave	Número de vueltas	Número de subclaves
128	4	10	1408
	6	12	1664
	8	14	1920
192	4	12	2496
	6	14	2496
	8	14	2880
256	4	14	3840
	6	14	3840
	8	14	3840

- La clave de cifrado principal se expande y pasa a denominarse Clave Expandida.
- Las subclaves provienen de la Clave Expandida. Para obtenerlas se divide la clave expandida en fragmentos de tamaño *Nb*. El primero de estos fragmentos será la primera subclave; el segundo fragmento será la segunda subclave y así sucesivamente.

Key Expansion.

La clave expandida proviene de la clave principal de cifrado. No existe la posibilidad de especificar una clave expandida concreta. Es un vector de palabras de 4 bytes y se denota por *W[Nb*(Nr + 1)]*. Esta clave expandida se puede crear en memoria usando un "búffer" de *Nk* palabras y así ahorrar trabajo de computación en implementaciones donde la memoria RAM sea escasa. Las primeras *Nk* palabras contienen la clave de cifrado principal. Las restantes palabras son definidas recursivamente en términos de palabras con índices cada vez más pequeños. La función key expansion depende del valor de *Nk* (número de columnas de la matriz representante de la clave, o lo que es lo mismo, número de palabras de 4 bytes que contiene la clave).

Rijndael diferencia el caso en el que $Nk \leq 6$ del caso en que $Nk > 6$ y aplica distintos procesos a estos casos.

Recordemos que *SubByte(W)* es una función que devuelve palabras de 4 bytes en las cuales los bytes han sido permutados. La función *RotByte(W)*

Para $Nk \leq 6$

```
KeyExpansion(byte key[4*Nk] word W[Nb*(Nr + 1)])
 {
   for(i = 0; i < Nk; i++)
        W[i] = (key[4*i], key[4*i+1], key[4*i+2], key[4*i+3]);
        for(i = Nk; i < Nb * (Nr + 1); i++)
        {
         temp = W[i - 1];
         if (i % Nk == 0)
           temp = SubByte(RotByte(temp)) ^ Rcon[i / Nk];
         W[i] = W[i - Nk] ^ temp;
        }
 }
```

2.8.5.- Implementación de Rijndael.

El cifrador Rijndael es susceptible de ser implementado de una manera eficiente en multitud de procesadores y hardware.

2.8.5.1.- Procesadores de 8 bits.

En este tipo de procesadores se puede programar Rijndael simplemente implementando las correspondientes transformaciones. Para la implementación de la función *SubByte* se necesita una tabla de 256 bytes. Las operaciones *SubByte, RoundShift* y *Round Key Addiction* se pueden combinar y ejecutar secuencialmente en los bytes de un estado concreto. La operación *MixColumn* requiere una matriz multiplicativa con coeficientes pertenecientes a *GF(28)*. El proceso para una columna sería:

$$Tmp = a[0] \hat{} a[1] \hat{} a[2] \hat{} a[3];\ \text{donde a es un vector de bytes}$$
$$tm = a[0] \hat{} a[1]; Tm = xtime(Tm); a[0] \hat{} = Tm \hat{} Tmp;$$
$$tm = a[1] \hat{} a[2]; Tm = xtime(Tm); a[1] \hat{} = Tm \hat{} Tmp;$$
$$tm = a[2] \hat{} a[3]; Tm = xtime(Tm); a[2] \hat{} = Tm \hat{} Tmp;$$
$$tm = a[3] \hat{} a[0]; Tm = xtime(Tm); a[3] \hat{} = Tm \hat{} Tmp;$$

El código mostrado se suele implementar en lenguaje ensamblador con el fin de ganar en velocidad. Para prevenir ataques por análisis temporal (*timing attacks*), la función *xtime* debe ser implementada para que se ejecute en un número fijo de ciclos, independientemente de su valor.

La implementación de Rijndael para que se ejecute en un solo ciclo, requiere una gran cantidad de memoria RAM en una SmartCard. Además, la mayoría de las aplicaciones, tales como tarjetas de crédito o monederos electrónicos, trabajan con un número reducido de bloques de datos, por lo que el aumento de la velocidad obtenido al implementar funciones en paralelo apenas compensa el gasto adicional de memoria.

La clave expandida puede ser implementada mediante un "*búffer*" de *4 * max (Nb, Nk)* bytes. Las subclaves se actualizan en cada vuelta. Todas las operaciones relacionadas con esta actualización de clave pueden se pueden implementar a nivel de byte.

2.8.5.2.- Procesadores de 32 bits.

Los diferentes pasos que componen la transformación de cada vuelta se puede combinados mediante tablas, lo que permite unas implementaciones muy rápidas en procesadores con longitudes de palabra de 32 o más bits.

Denotamos una columna de la matriz de estado a la salida de una determinada vuelta como (e) y en función de la matriz de estado a la entrada de esa vuelta (a). Vamos a denotar a_{ij} como al byte de la fila i y la columna j del estado a. Denotaremos por aj a la columna j del estado a.
Para las transformaciones *KeyAddiction* y *MixColumn* tenemos:

Operaciones KeyAddiction y MixColumn en procesadores de 32 Bits.

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix} \quad \text{y} \quad \begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \bullet \begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix}$$

Para las transformaciones *ShiftRow* y *ByteSub* tenemos:

Operaciones ShiftRow y ByteSub

$$\begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix} = \begin{bmatrix} b_{0,j} \\ b_{1,j-c1} \\ b_{2,j-c2} \\ b_{3,j-c3} \end{bmatrix} \quad \text{y} \quad b_{i,j} = S[a_{i,j}]$$

Nótese que los índices de las columnas deben ser expresados módulo *Nb*. Si combinamos estas dos expresiones obtenemos lo siguiente:

Valor de salida de la columna e en función de la matriz de estado a

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \bullet \begin{bmatrix} S[a_{0,j}] \\ S[a_{1,j-c1}] \\ S[a_{2,j-c2}] \\ S[a_{2,j-c3}] \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

La multiplicación de matrices puede ser expresada como una combinación lineal de vectores.

Multiplicación de matrices expresada como combinación lineal de vectores

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = S[a] \begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \oplus S[a_{1,j-c1}] \begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \oplus S[a_{2,j-c2}] \begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \oplus S[a_{3,j-c3}] \begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

Los factores multiplicativos $S[a_{i,j}]$ de los cuatro vectores son obtenidos creando una tabla de valores sobre los bytes de entrada $a_{i,j}$ en la tabla de la S-Box S[256].

Definimos las tablas T0, T1, T2, T3:

$$T_0[a] = \begin{bmatrix} S[a] \bullet 02 \\ S[a] \\ S[a] \\ S[a] \bullet 03 \end{bmatrix} \quad T_1[a] = \begin{bmatrix} S[a] \bullet 03 \\ S[a] \bullet 02 \\ S[a] \\ S[a] \end{bmatrix} \quad T_2[a] = \begin{bmatrix} S[a] \\ S[a] \bullet 03 \\ S[a] \bullet 02 \\ S[a] \end{bmatrix} \quad T_3[a] = \begin{bmatrix} S[a] \\ S[a] \\ S[a] \bullet 03 \\ S[a] \bullet 02 \end{bmatrix}$$

Son cuatro tablas con 256 palabras de cuatro bytes cada una, juntas ocupan un espacio total de 4 kbytes. Usando estas tablas las operaciones de cada vuelta se pueden expresar de la siguiente manera:

$$e_j = T_0[a_{0,j}] \oplus T_1[a_{1,j} - c1] \oplus T_2[a_{2,j} - c2] \oplus T_3[a_{3,j} - c3] \oplus k_j$$

Se puede apreciar que $T_j[a] = RowByte[T_{j-1}[a]]$ por lo que puede expresarse de la forma:

$$e_j = k_j \oplus T_0[b_{0,j}] \oplus RotByte(T_0[b_{1,j} - c1] \oplus RotByte(T_0[b_{2,j} - c2] \oplus RotByte(T_0[b_{3,j} - c3])))$$

Recuérdese que en la vuelta final no se realiza la operación *MixColumn* por lo que debemos emplear las tablas *S* en lugar de las tablas *T* que estamos usando hasta ahora. Esto se puede llevar a cabo sin crear tablas nuevas, simplemente aplicando una máscara a las tablas *T* ya creadas.

2.8.6.- Paralelismo.

Existe un considerable paralelismo entre las operaciones de la transformación de cada una de las vueltas. Las cuatro transformaciones que se efectúan en cada una de las vueltas actúan en paralelo sobre los bytes, las columnas, las filas o un estado concreto. En la implementación con tablas, la mayoría de las operaciones EXOR se pueden llevar a cabo en paralelo.

No debemos olvidar que la obtención de la clave expandida tiene también una componente secuencial. Para hallar el valor de *W[i]* es necesario conocer el valor de *W[i – 1]*. En aplicaciones donde el factor tiempo sea crítico, la clave expandida se calcula una única vez para un gran número de cifrados.

En aplicaciones donde la clave de cifrado cambie con bastante frecuencia, la obtención de la clave expandida y la ejecución de las vueltas del algoritmo se pueden realizar en paralelo.

2.8.7.- Adaptación al Hardware.

El algoritmo Rijndael puede ser perfectamente implementable en hardware diseñado específicamente con tal fin. Aun así, la implementación del algoritmo en software usando procesadores comerciales es muy rápida y el espacio ocupado en disco es pequeño, por lo que su implementación hardware se reducirá probablemente a los siguientes casos:

Procesadores extremadamente rápidos sin restricciones de espacio. En este caso las tablas T pueden ser implementadas en hardware y las operaciones EXOR se pueden realizar en paralelo.
Co-procesadores compactos en las SmartCard para acelerar la ejecución de Rijndael. En este caso se implementará en hardware la S-Box y la operación *xtime*, o la operación *MixColumn* completa.

2.8.8.- El Descifrado de Rijndael.

En la implementación mediante tablas de valores es esencial que el único paso no lineal (operación *SubByte*) sea la primera transformación en cada vuelta y que los bytes de las filas sean permutados (*ShiftRow*) antes de aplicar la función *MixColumn*.

En la inversa de una vuelta concreta, el orden de las transformaciones es el contrario al orden de las operaciones establecido, por tanto, el paso no lineal será el último del proceso y los bytes de las filas serán permutados después de aplicar la inversa de la función *MixColumn*.

La inversa de una vuelta no se puede implementar con las tablas de valores del proceso de cifrado. Esto es una característica del propio diseño de Rijndael. La estructura interna de Rijndael es tal que la secuencia de transformaciones de la inversa es igual al proceso de cifrado en sí, con las transformaciones remplazadas por sus propias inversas y un cambio en la Key Schedule.
Inversa de una variante Rijndael de 2 vueltas.
La inversa de una vuelta normal viene data por:

```
InvRound(State, RoundKey)
{
    AddRoundKey(State, RoundKey);
    InvMixColumn(State);
    InvShiftRow(State);
    InvByteSub(State);
}
```

La inversa de la vuelta final viene dada por:

```
InvFinalRound(State, RoundKey)
{
    AddRoundKey(State, RoundKey);
    InvShiftRow(State);
    InvByteSub(State);
}
```

La inversa de esta variante de 2 vueltas de Rijndael consiste en la inversa de una vuelta normal seguida por la inversa de la vuelta final.

```
AddRoundKey (State, RoundKey);
InvMixColumn (State);
```

puede ser reemplazada por:

```
InvMixColumn (State);
AddRoundKey (State, InvRoundKey);
AddRoundKey(State, ExpandedKey + 2 * Nb);
InvShiftRow(State);
InvByteSub(State);
AddRoundKey(State, ExpandedKey + Nb);
InvMixColumn(State);
InvShiftRow(State);
InvByteSub(State);
AddRoundKey(State, ExpandedKey);
```

2.8.8.1.- Propiedades Algebraicas.

Para obtener el descifrador Rijndael se ha hecho uso de propiedades algebraicas de la composición de transformaciones.

En primer lugar, el orden de las operaciones *ShiftRow* y *ByteSub* es indiferente. Esto es debido a que la operación *ShiftRow* afecta a la posición de los bytes, pero no modifica su valor. En el caso de *ByteSub*, el campo de trabajo es el valor de los bytes, independientemente de cuál sea su posición en la matriz estado.

En segundo lugar podemos apreciar que la secuencia.

```
AddRoundKey (State, RoundKey);
InvMixColumn (State);
```

Puede ser reemplazada por:

```
InvMixColumn (State);
AddRoundKey (State, InvRoundKey);
```

La *InvRoundKey* se halla aplicando la función *InvMixColumn* a la subclave. Este reemplazo se puede hacer tomando como base una propiedad algebraica de las aplicaciones lineales. Dada una aplicación *A*, se puede demostrar que $A(x+k) = A(x) + A(k)$.

2.8.9.- Estructura del Descifrador Rijndael.

Aplicando la propiedad enunciada, podemos rescribir el descifrador Rijndael para la variante de dos únicas vueltas.

```
AddRoundKey(State, ExpandedKey + 2 * Nb);
InvByteSub(State);
InvShiftRow(State);
InvMixColumn(State);
AddRoundKey(State, InvExpandedKey + Nb);
InvByteSub(State);
InvShiftRow(State);
AddRoundKey(State, ExpandedKey);
```

Se puede apreciar que tenemos una operación *AddRoundKey* inicial, una vuelta y una vuelta final. La vuelta y la vuelta final tienen ahora la misma estructura en el descifrador que la que tendrían en el cifrador.

El caso de una variante Rijndael de dos vueltas es una manera sencilla de explicar las similitudes entre el proceso de cifrado y descifrado, pero las deducciones obtenidas pueden ser aplicadas a cualquier variante Rijndael sea cual sea su número de vueltas. Esto nos viene a indicar la facilidad aportada por el algoritmo para modificar su número de vueltas según nuestro criterio sin que por ello varíe la estructura interna del mismo.

Vamos a denotar una vuelta y la vuelta final del algoritmo de la siguiente manera:

```
I_Round (State, I_RoundKey)
{
  InvByteSub(State);
  InvShiftRow(State);
  InvMixColumn(State);
  AddRoundKey(State, InvExpandedKey);
}

I_FinalRound (State, I_RoundKey)
{
  InvByteSub(State);
  InvShiftRow(State);
  AddRoundKey(State, ExpandedKey);
}
```

Visto esto, el descifrador Rijndael puede ser expresado de la siguiente manera:

```
I_Rijndael (State, CipherKey)
{
  I_KeyExpansion (CipherKey, I_ExpandedKey);
  AddRoundKey(State, I_ExpandedKey + Nb * Nr);
  for (i = Nr - 1; i>0; i--)
    Round (State, I_ExpandedKey + Nb * i);
  FinalRound (State, I_ExpandedKey);
}
```

La función *KeyExpansion* para el descifrador se compone de los siguientes pasos:

- Aplicación de la función *KeyExpansion*
- Aplicación de la función *InvMixColumn* a todas las subclaves excepto a la primera y a la última.

Expresado en notación de pseudocódigo:

```
I_KeyExpansion (CipherKey, I_ExpandedKey);
{
  KeyExpansion (CipherKey, I_ExpandedKey);
  for (i = Nr - 1; i>0; i--)
    InvMixColumn(I_ExpandedKey + Nb * i);
}
```

2.8.10.- Implementaciones del Descifrador Rijndael.

La implementación del descifrador Rijndael es similar a la implementación del cifrador, ya que su estructura es también similar, con el uso de una función *MixColumn* que emplea un polinomio distinto y, en algunos casos, una key schedule distinta. A pesar de esto se aprecia una degradación en la ejecución del descifrador en procesadores de 8 bits. Esto se debe a que la ejecución del descifrador se considera menos importante. En muchos cifradores de bloque la operación de descifrado ni siquiera se implementa.

2.8.11.- Fortaleza Estimada de Rijndael.

Se espera de Rijndael que, para cualquier longitud de bloque y de clave, se comporte tan bien como se pueda esperar de un cifrador de bloque con longitud de clave y de bloque dadas. Esto implica que el ataque más eficiente contra la seguridad de Rijndael es un ataque por fuerza bruta.

Emplear pares de texto en claro–texto cifrado conocidos para obtener información sobre otros pares no es más eficiente que el proceso de determinar la clave mediante una búsqueda exhaustiva.

El trabajo necesario para una búsqueda exhaustiva de la clave depende de la longitud de la clave principal de cifrado de tal forma que:

- Para claves de 16 bytes, 2^{127} aplicaciones de Rijndael
- Para claves de 24 bytes, 2^{191} aplicaciones de Rijndael
- Para claves de 32 bytes, 2^{255} aplicaciones de Rijndael

Como es lógico, se ha tomado un considerable margen de seguridad con respecto a todos los ataques conocidos actualmente.

2.8.12.- Ventajas y limitaciones de Rijndael.

Aspectos de implementación:

- Rijndael se puede implementar para ser ejecutado a velocidades inusualmente elevadas para cifradores de bloque.
- Rijndael se puede implementar en SmartCards, empleando un mínimo de memoria RAM y usando un número reducido de vueltas.
- Las operaciones de cada vuelta se pueden paralelizar, lo cual supone un importante avance con vistas a futuros procesadores y hardware específico.
- Como el cifrador no hace uso de operaciones aritméticas, no existe ningún prejuicio con respecto a ningún tipo de arquitectura de procesadores.

Simplicidad de diseño:

- El cifrador es autónomo e independiente. No hace uso de ningún otro componente criptográfico, S-Box procedentes de otros cifradores con buena reputación, tablas aleatorias o cualquier otra artimaña.
- El cifrador no basa su seguridad o parte de ella en operaciones aritméticas ininteligibles o secretas.
- El diseño hermético del cifrador no permite la existencia de puertas traseras ocultas.
- Longitud variable del bloque. Las longitudes de bloque de 192 y 256 bits permiten la construcción de una función tipo hash usando Rijndael como función de compresión y con la característica de que el número de colisiones provocadas sería mínimo. Hoy en día, la longitud de bloque de 128 bits no se considera suficiente para dicho fin.
- Extensiones: El diseño permite la especificación de variaciones con la longitud del bloque y de la clave en el rango comprendido entre los 128 y los 256 bits, tomados en intervalos de 32 bits.
- Aunque el número de vueltas de Rijndael viene fijado en las especificaciones del algoritmo, se puede modificar este parámetro con el fin de solucionar posibles problemas de seguridad o adaptarse a distintas velocidades de ejecución.

Limitaciones:

- Las limitaciones del cifrador están relacionadas con su inversa:
- El descifrador Rijndael es más difícil de implementar en SmartCards que el propio cifrador debido a que requiere una mayor cantidad de código y un número superior de vueltas. A pesar de ello, comparado con otros cifradores, la función inversa es realmente rápida.
- Sin implementaciones software, el cifrador y su inversa emplean diferente código y diferentes tablas
- Sin implementaciones hardware, el descifrador solamente puede aprovechar una parte de la circuitería implementada para el cifrador.

3.- Conclusiones Criptografía Simétrica.

La criptografía simétrica se refiere al conjunto de métodos que permiten tener comunicación segura entre las partes siempre y cuando anteriormente se hayan intercambiado la clave correspondiente que llamaremos clave simétrica. La simetría se refiere a que las partes tienen la misma clave tanto para cifrar como para descifrar.

Este tipo de criptografía se conoce también como criptografía de clave privada o criptografía de clave privada.

Existe una clasificación de este tipo de criptografía en tres familias, la criptografía simétrica de bloques (block cipher), la criptografía simétrica de flujo (stream cipher) y la criptografía simétrica de resumen (hash functions). Aunque con ligeras modificaciones un sistema de criptografía simétrica de bloques puede modificarse para convertirse en alguna de las otras dos formas, sin embargo es importante verlas por separado dado que se usan en diferentes aplicaciones.

La criptografía simétrica ha sido la más usada en toda la historia, ésta ha podido ser implementada en diferentes dispositivos, manuales, mecánicos, eléctricos, hasta los algoritmos actuales que son programables en cualquier computadora. La idea general es aplicar diferentes funciones al mensaje que se quiere cifrar de tal modo que solo conociendo una clave pueda aplicarse de forma inversa para poder así descifrar.

Aunque no existe un tipo de diseño estándar, quizá el más popular es el de Feistel, que consiste esencialmente en aplicar un número finito de interacciones de cierta forma, que finalmente da como resultado el mensaje cifrado. Este es el caso del sistema criptográfico simétrico más conocido, DES.

DES es un sistema criptográfico que toma como entrada un bloque de 64 bits del mensaje y este se somete a 16 interacciones, una clave de 56 bits, en la práctica el bloque de la clave tiene 64 bits, ya que a cada conjunto de 7 bits se le agrega un bit que puede ser usada como de paridad.

Dependiendo de la naturaleza de la aplicación DES tiene 4 modos de operación (ver referencias en: [48][54][56]) para poder implementarse: ECB (Electronic Codebook Mode) para mensajes cortos, de menos de 64 bits, CBC (Cipher Block Chaining Mode) para mensajes largos, CFB (Cipher Block Feedback) para cifrar bit por bit ó byte por byte y el OFB (Output Feedback Mode) el mismo uso pero evitando propagación de error.

En la actualidad no se ha podido romper el sistema DES desde la perspectiva de poder deducir la clave simétrica a partir de la información interceptada, sin embargo con un método a fuerza bruta, es decir probando alrededor de 256 posibles claves, se pudo romper DES en Enero de 1999. Lo anterior quiere decir que, es posible obtener la clave del sistema DES en un tiempo relativamente corto, por lo que lo hace inseguro para propósitos de alta seguridad. La opción que se ha tomado para poder suplantar a DES ha sido usar lo que se conoce como cifrado múltiple, es decir aplicar varias veces el mismo algoritmo para fortalecer la longitud de la clave, esto a tomado la forma de un nuevo sistema de cifrado que se conoce actualmente como triple-DES o TDES.

En la actualidad Rijndael es el nuevo estándar criptográfico del tipo simétrico, el cual maneja claves de 128, 192 y 256 bits, Rijndael no se basa en una estructura de redes de Feistel, la seguridad del cifrado Rijndael se basa en el desorden provocado en el contenido del texto en claro al aplicarse una serie de permutaciones y otras operaciones matemáticas. Algunas de estas operaciones se realizan a nivel de byte, representando mediante el campo llamado $GF(2^8)$ y otras a nivel de palabras de cuatro bytes.

CAPÍTULO 3

Criptografía Moderna

1.- Algoritmos Asimétricos de Cifrado.

Los algoritmos de llave pública, o algoritmos asimétricos, han demostrado su interés para ser empleados en redes de comunicación inseguras. Introducidos por Whitfield Diffier y Martin Hellman a mediados de los años 70, su novedad fundamental con respecto a la criptografía es que las claves no son únicas, si no que forman pares. Hasta la fecha han aparecido multitud de algoritmos asimétricos, la mayoría de los cuales son inseguros. Otros son poco prácticos, bien sea porque el criptograma es considerablemente mayor que el mensaje original, bien sea porque la longitud de la clave es enorme.

Se basan en general en plantearle al atacante problemas matemáticos difíciles de resolver. En la práctica muy pocos algoritmos son realmente útiles. El más popular por su sencillez es RSA, que ha sobrevivido a la multitud de ataques.

Los algoritmos asimétricos emplean generalmente longitudes de clave mucho mayores que los simétricos. Por ejemplo, mientras que para algoritmos asimétricos se considera segura una clave de 128 bits, para algoritmos asimétricos se recomienda claves de al menos 1024 bits.

Además, la complejidad de cálculo que conforman estos últimos los hace considerablemente más lentos que los algoritmos de cifrado por bloques. En la práctica los métodos asimétricos se emplean únicamente para codificar la clave de sesión (simétrica) de cada mensaje.

1.1.- Aplicaciones de los Algoritmos Asimétricos.

Los algoritmos asimétricos poseen dos claves diferentes en lugar de una, K_p y K_P, se denominan clave privada y clave pública. Una de ellas se emplea para codificar, mientras que la otra se usa para decodificar. Dependiendo de la aplicación que le demos al algoritmo, la clave pública será la de cifrado o viceversa. Para estos criptosistemas sean seguros también ha de cumplirse que a partir de una de las claves resulte extremadamente difícil calcular la otra.

1.2.- Protección de la Información.

Una de las aplicaciones inmediatas de los algoritmos asimétricos es el cifrado de la información sin tener que transmitir la clave de decodificación, lo cual permite su uso en canales inseguros.

Supongamos que A quiere enviar un mensaje a B. Para ello solicita a B su clave pública K_P. A genera entonces el mensaje cifrado $E_{K_p}(m)$. Una vez hecho esto únicamente quien posea la clave K_p (clave privada) podrá recuperar el mensaje original m.

La clave que se hace pública es aquella que permite codificar los mensajes, mientras que la clave privada es aquella que permite descifrarlos.

1.3.- Autenticación.

La segunda aplicación de los algoritmos asimétricos es la autenticación de mensajes, con ayuda de funciones resumen, que nos permiten obtener una firma a partir de un mensaje. Dicha firma es mucho más pequeña que el mensaje original, y es muy difícil encontrar otro mensaje que tenga la misma firma. Supongamos que A recibe un mensaje m de B y quiere comprobar su autenticidad. Para ellos B genera un resumen del mensaje r(m) y lo codifica empleando la clave de cifrado, que en este caso será privada. La clave de descifrado se habrá hecho pública previamente, y debe estar en poder de A. B envía entonces a A el criptograma correspondiente a r(m). A puede ahora generar su propia r'(m) y compararla con el valor r(m) obtenido del criptograma enviado por B. Si coinciden, el mensaje será autentico, puesto que el único que posee la clave para codificar es precisamente B.

Nótese que en este caso la clave que emplea para cifrar es la clave privada, justo al revés que para la simple codificación de mensajes.

Muchos de los algoritmos asimétricos presentan claves duales, esto quiere decir que si empleamos una para codificar, la otra permitirá decodificar y viceversa. Esto ocurre con el algoritmo RSA, por lo que un único par de claves es suficiente para codificar y autenticar.

1.4.- Algoritmo RSA.

De entre los algoritmos asimétricos, quizá RSA sea el más sencillo de comprender e implementar. Sus pares de claves son duales, por lo que sirve tanto para codificar como para autenticar. Su nombre proviene de sus tres inventores: Ron Rivest, Asi Shamir y Leonard Adleman. Desde su nacimiento nadie ha conseguido probar o rebatir su seguridad, pero se le tiene como uno de los algoritmos asimétricos más seguros.

RSA se basa en la dificultad para factorizar grandes números. Las claves pública y privada se calculan a partir de un número que se obtiene como producto de dos números primos grandes. El atacante se enfrentará, si quiere recuperar un texto plano a partir del criptograma y la clave pública, a un problema de factorización.

Para generar un par de llaves (K_P, K_p), en primer lugar se escogen aleatoriamente dos números primos grandes, p y q. Después se calcula el producto $n = p \bullet q$.

Escogemos ahora un número e primo relativo con $(p-1)(q-1)$. (e,n) será la clave pública. Nótese que se debe tener inversa módulo $(p-1)(q-1)$, por lo que existirá un número d tal que

$$d \bullet e \equiv 1 (\mathrm{mod} (p-1)(q-1))$$

es decir, que d es la inversa de e módulo $(p-1)(q-1)$. (d,n) será la clave privada. Esta inversa puede calcularse fácilmente empleando el Algoritmo Extendido de Euclides. Si desconocemos los factores de n, este cálculo prácticamente resulta imposible.

La codificación se lleva a cabo según la expresión:

$$c = m^e (\mathrm{mod}\ n)$$

mientras que la decodificación se hará de la siguiente forma:

$$m = c^d (\mathrm{mod}\ n)$$

ya que

$$c^d = (m^e)^d = m^{ed} = m^{k(p-1)(q-1)+1} = (m^k)^{(p-1)(q-1)} m$$

recordemos que $\Phi(n) = (p-1)(q-1)$, por lo que, según la ecuación, $(m^k)^{(p-1)(q-1)} = 1$, los cual nos lleva de nuevo a m.

En la práctica, tomaremos p y q con un número grande de bits, por ejemplo 200, con lo que n tendrá 400 bits. Subdividiremos el mensaje que queramos enviar en bloques de 399 bits (de esta forma garantizamos que el valor de cada bloque sea menor que n) y efectuamos la codificación de cada uno. Obtendremos un mensaje cifrado ligeramente más grande, puesto que estará compuesto por bloques de 400 bits. Para decodificar partiremos el mensaje cifrado en bloques de 400 bits (ya que en este caso sabremos que el valor de cada bloque ha de ser menor que n), y obtendremos bloques de 399 bits.

El atacante, si quiere recuperar la clave privada a partir de la pública, debe conocer los factores p y q de n y esto representa un problema computacionalmente intratable, siempre que p y q (por lo tanto también n) sean lo suficientemente grandes.

En la siguiente descripción del algoritmo señalamos entre paréntesis qué partes del sistema se consideran públicas y cuáles secretas.

1. Encontrar dos grandes números primos, *p* y *q* (secretos), y calcular el número *n* (público) mediante el producto, $n = p \cdot q$.
2. Encontrar la clave de descifrado constituida por una gran número entero impar, *d* (secreto) que es primo con él numero $\Phi(n)$ (secreto), obteniendo mediante $\Phi(n) = (p-1)(q-1)$.
3. Calcular el entero *e* (público) tal que $1 \le e \le \Phi(n)$, mediante la formula: $e \cdot d \equiv 1 (\mathrm{mod}\, \Phi(n))$.
4. Hacer pública la clave de cifrado *(e,n)*.
5. Para cifrar un texto, es necesario previamente codificar el texto en un sistema numérico, bien decimal o bien binario, y dividir en bloques M_i de tamaño *j* o *j-1* de forma que, según sea el alfabeto usando el decimal o el binario cumpla en cada caso $10^{j-1} < n < 10^j$ o $2^{j-1} < n < 2^j$. (Cuando se toma como tamaño *j*, el descifrado del texto puede nos ser único, por lo tanto esta elección se hace sólo cuando la unicidad del descifrado no es importante).
6. Cifrar cada bloque M_i transformándolo en un nuevo bloque de números C_i de acuerdo con la expresión: $C_i \equiv M_i^e \pmod{n}$.
7. Para descifrar el bloque C_i, se usa la clave privada *d* según la expresión $M_i \equiv C_i^d \pmod{n}$.

Veamos el siguiente ejemplo. Si *p=3* y *q=5*, entonces *n=15* y *Φ(n)=8*. Tomamos *d=3*, ya que *MCD(3,8)=1*. Calculamos *e=3* tal que $3 \cdot 3 \equiv 1 (\mathrm{mod}\ 8)$. Consideremos un mensaje *M=7*, su cifrado es entonces $C = 13 \equiv 7^3 (\mathrm{mod} 15)$. El descifrado de *C=13* es, efectivamente, *M=7*, pues $13^3 (\mathrm{mod} 15) \equiv 7$.

Los procesos de cifrado y descifrado pueden ser implementados mediante algunos algoritmos conocidos.

Las dos principales dificultades en la implementación del RSA son:

- Potencias modulares.
- Búsqueda de números primos.

En primer lugar, para las operaciones de cifrado y descifrado es necesario calcular una potencia $a^r \pmod{n}$ de forma eficiente.

Primero se considera la representación en base dos del exponente, el número *r*,

$$r = \sum_{j=0}^{k} x_j 2^j$$

siendo $x_j = 0, 1$ y $k = E(\log_2 r) + 1$. Entonces basta con calcular *k* o *k-1* potencias de la forma $a^{2^j} \pmod{n} (0 \le j \le k)$ para obtener finalmente

$$a^r \equiv \prod_{j=0}^{k} a^{x_j 2^j} \pmod{n}$$

Si el entero *r* es muy grande y se conoce el número $\Phi(n)$, se pude utilizar *I* de identidad de Euler-Fermat que afirma que "para todo entero *a* tal que *1≤a<n* y para algún entero *r*, si $r \equiv r_1 \pmod{\Phi(n)}$ entonces $a^r \equiv a^{r_1} \pmod{n}$. De esa forma puede calcularse previamente el módulo $\Phi(n)$ de *r*, $r_1 \equiv r \pmod{\Phi(n)}$ y facilitar así el cálculo.

Ejemplo de calculo de estas potencias modulares: al menos con números pequeños como *a=5*, *r=71* y *n=53*. Para obtener $a^r \pmod{n} \equiv 5^{71} \pmod{53}$, basta calcular $5^{19} \pmod{53}$ gracias a la equivalencia $71 \equiv 19 \pmod{54}$. Esta última potencia $5^{19} \pmod{53}$ se consigue usando las $k = E(\log_2 19) + 1 = 5$ potencias $5^{2^1}, 5^{2^2}, 5^{2^3}, 5^{2^4} \pmod{53}$. Como en el sistema binario 19 = 10011, el cálculo de $5^{19} \pmod{53}$ se hace de la forma siguiente:

$$5^{19} \pmod{53} = 5^{24+2+1} \pmod{53} \equiv 13 \cdot 25 \cdot 5 \pmod{53} \equiv 35$$

Una dificultad se encuentra en la elección de números primos.

Si se denota como $\pi(x)$ el número de primos en el intervalo que va del 2 a *x*, la probabilidad de que un valor seleccionado aleatoriamente en ese intervalo sea primo será aproximadamente

$$\frac{\pi(x)}{x-1}$$

por lo que antes de encontrar un primo se prevé el examen de una cantidad media de

$$\frac{x-1}{\pi(x)} \simeq \ln x \text{ Valores}$$

Además, para aumentar las dificultades, la seguridad del RSA se basa en que los primos *p* y *q* elegidos han de ser muy grandes. Para encontrar los grandes primos *p* y *q* ese pueden usar varios algoritmos, como el de Solovay-Strassen, y el de Lehmann y Peralta, que sirven para comprobar la primalidad.

La prueba de Solovay-Strassen es una técnica, que permite determinar, con un alto grado de probabilidad, si un número es primo o no.

Se parte de un número *p* de aproximadamente 100 dígitos escogidos aleatoriamente. Este número es el candidato a primo. En primer lugar se hacen las comparaciones sencillas de que dicho número no es divisible por ningún número primo pequeño; así por ejemplo, el número debe ser impar, sus dígitos no deben ser divisibles por 3, el último dígito no debe ser 5 y la suma de los dígitos pares y la de los impares no deben ser iguales en módulo 11, etc. Una vez que *p* ha pasado estos controles, se siguen estos pasos:

1. Escoger cien números $a_1(i=1,..,100)$ aleatoriamente dentro del intervalo *[1,p-1]*.
2. Comprobar si *p* y $a_i \forall i$ son primos entre sí $\left(\frac{a_1}{p}\right) \equiv a_i^{\frac{p-1}{2}} \pmod{n}$ (donde $\left(\frac{a}{b}\right)$ es el llamado símbolo de jacobi para el par *(a, b)* definido de la forma:

si *b* es primo:

$$\left(\frac{a}{b}\right) = \begin{cases} +1 & \text{si } a \text{ es residuo cuadrático de } b \text{ y} \\ -1 & \text{si } a \text{ no es residuo cuadrático de } b \end{cases}$$

si *b* no es primo, pero tiene factorización en números primos $b = p_1 \bullet p_2 ... p_k$, entonces

$$\left(\frac{a}{b}\right) = \left(\frac{a}{p_1}\right) \bullet \left(\frac{a}{p_2}\right) ... \left(\frac{a}{p_k}\right)$$

Si el número *p* pasa todas las comprobaciones del paso 2, entonces se acepta que *p* es primo, ya que la probabilidad de que no sea primo es aproximadamente de $(1/2)^{100}$.

La prueba de Lehmann y Peralta es bastante más fácil de implementar que el anterior. Para esta prueba también es necesario escoger cien números $a_i(i=1,2,...,100)$ aleatoriamente en el intervalo *[1,p-1]*. El desarrollo es como sigue:

1. Si $a_i^{\frac{p-1}{2}} \equiv 1(\text{mod } p)\ \forall\ i=1,...,100$, entonces p es compuesto.
2. Si $\exists i \in \{1,...,100\} a_i^{\frac{p-1}{2}}$ no es congruente con $1(\text{mod } p)$ o $a_i^{\frac{p-1}{2}}$ no es congruente con $-1(\text{mod } p)$, entonces p es compuesto.
3. Si $a_i^{\frac{p-1}{2}} \equiv 1(\text{mod } p)$ o $a_i^{\frac{p-1}{2}} ! \equiv -1(\text{mod } p)\ \forall i=1,...,100$ y $\exists\ i \in \{1,2,...,100\}$ tal que $a_i^{\frac{p-1}{2}} \equiv -1(\text{mod } p)$ entonces p es primo.

En el caso del RSA pueden encontrarse el entero *d* primo con $\Phi(n)=(p-1)(q-1)$ tomando simplemente un número primo mayor que *max{p,q}*.

Para calcular el entero *e* tal que $e \bullet d \equiv 1(\text{mod}\,\Phi(n))$, se puede utilizar el algoritmo euclídeo por ser *d* primo con $\Phi(n)$.

Por último, las operaciones de cifrado y descifrado requieren el cálculo de potencias modulares, lo que se puede llevar a cabo en tiempo polinomial, tal como se ha mencionado. Exactamente son necesarias $2(\log_2 n)$ multiplicaciones modulares, de manera que, por ejemplo, para el cálculo de $2^{18}=(((2^2)^2)^2 2^2$ son necesarias 5 multiplicaciones modulares.

A continuación vemos un par de ejemplos de aplicación del sistema RSA.

1. Supongamos un sistema RSA con p=11, q=17, con los que $n=p \bullet q=187$ y $\Phi(n)=10 \bullet 16=160$. Si se escoge el entero d=7 primo con $\Phi(n)$, se calcula mediante el algoritmo euclídeo el número entero *e* tal que $e \bullet d \equiv 1(\text{mod}\,\Phi(n))$, obteniéndose e=23. por tanto, la clave pública es (23,187), y la clave de descifrado 11.

Codificamos el alfabeto de la siguiente forma: blanco:*00, A:01, B:02,...,Z:26*. Por tanto, el mensaje "SEGURO QUE LLUEVE" queda codificado como M=190 507 211 815 001 721 050 012 122 105 220 500 (divididos en bloques de 3, pues $10^2 < 187 < 10^3$). Se cifran los doce bloques de tres dígitos con los datos anteriores. Para ello, se tienen en cuenta la expresión binaria de 23=10111, de forma que el bloque M_1=190 se cifra como $C_1=190^{23} = 190^{24} \bullet 190^{22} \bullet 190^2$ (mod 187). También en el descifrado para facilitar la recuperación se considera que 7 en el sistema binario es 111, obteniéndose el texto original.

2. Para cifrar el mensaje "JALEA", primero se codifica en binario de la forma 01010 0001 01100 00101 10100. Tomando como parámetros del sistema *p*=31, *q*=43, *n*=1333, $\Phi(n)=1260$,

d=13 y e=97, se divide el mensaje en bloques de tamaño 11 (pues $2^{10}=1024<1333<2048=2^{11}$) y se convierten en enteros:

01010000010 = $2^9 + 2^7 + 2 = 642$
11000010110 = $2^{10}+2^9+2^4+2^2+2=1558$
10000000000 = $2^{10}=1024$

De esta forma se tienen los tres bloques que se deben cifrar usando el hecho de que $97=2^6+2^5+2^0$.

En conclusión, para la selección de los parámetros del sistema el usuario debe comprobar las siguientes condiciones para intentar evitar los ataques mencionados:

- Los números primos *p* y *q* deben tener una diferencia grande (ataque 1).
- Los números primos *p* y *q* deben ser del orden de 100 dígitos (ataque 1).
- Los enteros *(p-1) y (q-1)* deben contener grandes factores primos (ataque 3).
- Los números primos *p* y *q* deben elegirse de manera que $p=2p`+1$ y $q=2q`+1$ con $p`$ y $q`$ números primos impares (ataque 4).

Los primos *p* y *q* del RSA deben ser de la forma $x=2x`+1$ (siendo $x`$ un número primo impar) tal que *x-1* tiene grandes factores primos, con aproximadamente 100 dígitos y de forma que *p-q* sea grande.

No obstante, es lógico pensar que a medida que se profundiza en la investigación se añadirán y descubrirán nuevas propiedades de los parámetros que deban verificar para que el sistema resultante se fortalezca.

Uno de los más recientes ataques al RSA fue realizado por Wiener. Llevó a cabo a un criptoanálisis en tiempo polinomial de un sistema RSA con claves privadas pequeñas. Para ello, utilizó un algoritmo que representa los números racionales como fracciones continuas finitas.

Se concluye que para que la seguridad del sistema RSA quede perfectamente salvaguardada es necesario escoger cuidadosamente las claves a utilizar.

A pesar de las ventajas evidentes de este esquema frente a los sistemas de clave secreta, hay que subrayar que en la actualidad la principal desventaja del RSA estriba en que es mucho más lento que el DES, Rijndael y que los cifrados en flujo. Como cifras se puede señalar que el DES trabaja a unas mil veces más rápido que RSA, existen investigaciones para acelerar el proceso, es de esperar que esta proporción se mantenga, ya que también se realiza para los algoritmos de cifrado simétrico.

1.4.1.- Seguridad y Vulnerabilidades del Algoritmo RSA

Técnicamente no es cierto que el algoritmo RSA deposite su fuerza en el problema de la factorización. En realidad el hecho de tener que factorizar un número para descifrar un mensaje sin la clave privada es una mera conjetura. Nadie ha demostrado que no pueda surgir un método en el futuro que permita descifrar un mensaje sin usar la clave probada y sin factorizar el módulo *n*. De todas formas, existen estudios que demuestran que incluso recuperar sólo algunos bits del mensaje original resulta tan difícil como descifrar el mensaje entero.

Aparte de factorizar *n*, podríamos intentar calcular $\Phi(n)$ directamente, o probar por la fuerza bruta tratando de encontrar la clave privada. Ambos ataques son más costosos computacionalmente que la propia factorización de *n*, afortunadamente.

Otro punto que cabría preguntarse es qué pasaría si los primos *p* y *q* que escogemos realmente fueran compuestos. Recordemos que los algoritmos de prueba de primos que conocemos son probabilísticas, por lo que jamás tendremos la absoluta seguridad de que *p* y *q* son realmente primos. Pero si aplicamos, por ejemplo, treinta pasadas del algoritmo de Rabin-Miller, las probabilidades de que el número escogido pase la prueba y siga siendo primo son de una contra 2^{60}, resulta más fácil que nos toque la primitiva y que simultáneamente nos toque un rayo. Si *p* o *q* fueran compuestos, el algoritmo RSA simplemente no funcionaria.

Para analizar la seguridad del sistema, se supone que el criptoanalista tiene una cantidad ilimitada de pares *(M,C)* de mensajes originales y sus correspondientes criptogramas. Las posibles maneras que tiene de atacar el sistema son las siguientes.

1. Factorizar *n*

De esta forma se obtiene el número $\Phi(n)=(p-1)(q-1)$, y con él la clave privada *d*, puesto que *e* es pública y se cumple:

$$e \cdot d \equiv 1 (\mathrm{mod}\ \Phi(n))$$

Al ser *n* el producto de sólo dos números primos, un algoritmo de factorización requiere como máximo $\sqrt{n}$ pasos, pues uno de los dos factores es necesariamente un primo menor que $\sqrt{n}$. Sin embargo, si *n* fuera el producto de N>2 primos, un algoritmo de factorización necesitaría como máximo $n^{\frac{1}{N}}$ pasos, que es la cota menor que $\sqrt{n}$, por lo que se concluye que es adecuada la obtención de *n* como producto de sólo dos números primos.

Con respecto al estudio del problema de la factorización, hay que mencionar al precursor de la moderna factorización, el algoritmo de fracciones continuas de Morrison-Brillhart, ya que es uno de los más rápidos. Sin embargo, los dos algoritmos de factorización que resultan más prácticos para grandes números enteros corresponden al de factorización con curvas elípticas de Hendrik Lenstra y al de factorización con filtro cuadrático de Carl Pomerance. Ambos algoritmos convierten el problema de factorización de un entero *n* en el problema reencontrar soluciones no triviales ($x! \equiv y(\mathrm{mod}\, n)$ y $x! \equiv -y(\mathrm{mod}\, n)$) de la ecuación: $x^n \equiv y^n(\mathrm{mod}\, n)$. Si se supone que ni *(x+y)* ni *(x-y)* son múltiplos de *n* enteros, se deduce que el *MCD(x+y,n)* o bien el *MCD(x-y,n)* es con seguridad el factor no trivial de *n*, por lo que se resuelve el problema de factorización.

Por ejemplo, si *n*=97343, entonces la ecuación $x^2 \equiv y^2(\mathrm{mod}\ 97343)$ es fácilmente resoluble por ser los factores de *n* dos números primos muy cercanos. Como $312^2 = 1(\mathrm{mod}\ 97343)$ y ni 313 ni 311 son múltiplos de 97343, se concluye que 313 y 311 son los factores de *n*.

2. Calcular $\Phi(n)$.

Como se ve a continuación, esta manera es equivalente a la anterior. Si se tiene $\Phi(n)$, dado que $p+q = n-\Phi(n)+1$ y a partir de la suma se puede calcular $(p-q)^2$ por coincidir con $(p+q)^2 - 4 \bullet n$, luego se consigue la factorización mediante las formulas

$$q = \frac{1}{2}[(p+q)-(p-q)] \text{ y } p = \frac{1}{2}[(p+q)+(p-q)].$$

3. Ataque por iteración.

Si un enemigo conoce *(n,e,C)*, entonces puede generar la secuencia

$$C_1 = C^e(\mathrm{mod}\ n), ..., C_i = C_{i-1}{}^e(\mathrm{mod}\ n)$$

con lo que si existe algún C_j tal que $C = C_j$ se deduce que el mensaje buscado es $M = C_{j-1}$ pues $C_{j-1}{}^e = C_j = C$. Ahora bien, en cuanto la igualdad $C_j = C$ se cumple sólo para un valor de *j* demasiado grande, este ataque se vuelve impracticable. Con respecto a esto Rivest demuestra que si los enteros *p-1* y *q-1* contienen factores primos grandes, la probabilidad de éxito mediante este procedimiento es casi nula para grandes valores de *n*.

4. Ataque de Blakley y Boros.

El sistema RSA, además, tiene una característica muy peculiar, advertida por Blakley y Boros, y es que no siempre esconde el mensaje. A continuación vemos un ejemplo que lo muestra.

Si e=17, n=35 y los mensajes a cifrar son M_1=6 y M_2=7, entonces se obtiene que $6^{17} \equiv 6(\mathrm{mod}\ 35)$ y $7^{17} \equiv 7(\mathrm{mod}\ 35)$. Una situación más peligrosa para el sistema aparece, por ejemplo, con los valores p=97, q=109 y e=865, ya que el criptosistema resultante no esconde ningún mensaje, pues $M^{865} \equiv M(\mathrm{mod}\ 97 \bullet 109)$.

En general, lo ocurrido en el último ejemplo ocurre siempre que e-1 es múltiplo de p-1 y de q-1, pues en ese caso $M^e \equiv M(\mathrm{mod}\ p \bullet q)$. Además, se tiene que para cualquier elección de $n = p \bullet q$ siempre existen al menos 9 mensajes M que nos se cifran en realidad, ya que verifican la ecuación $M^e \equiv M(\mathrm{mod}\, n)$. De esos nueve mensajes hay tres fijos, que son $M \in \{0,1,-1\}$. Para hacer que el sistema RSA sea resistente contra ataques basados en este hecho, es conveniente elegir como claves privadas números primos de la forma $p = 2p`+1$, donde $p`$ es un primo impar.

Aunque en teoría el algoritmo RSA es bastante seguro, existen algunos puntos débiles en la forma de utilizarlo que puede ser aprovechados por un atacante.

1.4.1.1.- Claves Demasiado Cortas.

Actualmente se considera segura una clave RSA con una longitud de n de al menos 768 bits, si bien se recomienda el uso de claves no inferiores de 1024 bits. Hasta hace relativamente poco se recomendaban 512 bits, pero en mayo de 1999, Adi Shamir presentó el denominado dispositivo Twinkle, un prototipo capaz de factorizar números de manera muy rápida, aprovechando los últimos avances de la optimización de algoritmos específicos para esta tarea. Este dispositivo, aún no construido, podría ser incorporado en computadoras de bajo costo y pondría en serio peligro los mensajes cifrados con claves de 512 bits o menos.

Teniendo en cuenta los avances de la tecnología, y suponiendo que el algoritmo RSA no sea roto analíticamente, deberemos escoger la longitud de la clave en función del tiempo que queramos que nuestra información permanezca en secreto. Efectivamente, una clave de 1024 bits parece a todas luces demasiado corta como para proteger información por más de unos pocos años.

1.4.1.2.- Ataques de Intermediario.

El ataque de intermediario puede darse con cualquier algoritmo asimétrico. Supongamos que *A* quiere establecer una comunicación con *B*, y que *C* quiere espiarla. Cuando *A* le solicite a *B* su clave pública K_B , *C* se interpone, obteniendo la clave de *B* y enviando a *A* una clave falsa k_C creada por él. Cuando *A* codifique el mensaje, *C* lo interceptará de nuevo, decodificándolo con su clave propia y empleando K_B para recodificarlo y enviarlo a *B*. Ni *A* ni *B* son concientes de que sus mensajes están siendo interceptados.

La única manera de evitar esto consiste en asegurar a *A* que la clave pública que tiene de *B* es auténtica. Para ello nada mejor que ésta esté firmada por un amigo común, que certifique la autenticidad de la clave. En la actualidad existen los llamados anillos de confianza, que permiten certificar la autenticidad de las claves sin necesidad de centralizar el proceso.

1.4.1.3.- Ataques de Texto Plano Escogido.

Existe una familia de ataques a RSA que explotan la posibilidad de que un usuario codifique y firme un único mensaje empleando el mismo par de llaves. Para que el ataque surta efecto, la firma debe hacerse codificando el mensaje completo, no el resultado de una función resumen sobre él. Por ello se recomienda que las firmas digitales se lleven a cabo siempre sobre una función resumen del mensaje, nunca sobre el mensaje en sí.

Otro tipo de ataque con texto plano escogido podrían ser el siguiente: para falsificar una firma sobre un mensaje *m*, se pueden calcular dos mensajes individuales m_1, m_2, aparentemente inofensivos, tales que $m_1 m_2 = m$, y enviárselos a la víctima para que los firme. Entonces obtendríamos un m_1^d y m_2^d. Aunque desconozcamos *d*, si calculamos $m_1^d m_2^d = m^d \pmod n$ obtendremos el mensaje m firmado.

1.4.1.4.- Ataques de Módulo Común.

Podría pensarse que, una vez generados *p* y *q*, será más rápido generar tantos pares de claves como queramos, en lugar de tener que emplear dos números primos diferentes en cada caso. Sin embargo, si lo hacemos así, un atacante podrá decodificar nuestros mensajes sin necesidad de la llave privada. Sea *m* el texto plano, que codificamos empleando dos claves de cifrado diferente e_1 y e_2. Los criptogramas que obtenemos son los siguientes:

$$c_1 = m^{e_1} \pmod n$$
$$c_2 = m^{e_2} \pmod n$$

El atacante conoce pues *n*, e_1, e_2, c_1 y c_2. Si e_1 y e_2 son primos relativos, el Algoritmo Extendido de Euclides nos permitirá encontrar *r* y *s* tales que

$$re_1 + se_2 = 1$$

Ahora podemos hacer el siguiente cálculo

$$c_1^r c_2^s = m^{e_1 r} m^{e_2 s} = m^{e_1 r + e_2 s} = m^1 (\mathrm{mod}\, n)$$

Esto sólo se cumple si e_1 y e_2 son números primos relativos, pero precisamente eso es lo que suele ocurrir en la gran mayoría de los casos. Por lo tanto, se deben de generar *p* y *q* diferentes para cada par de claves.

1.4.1.5.- Ataques de Exponente Bajo.

Si el exponente de codificación *e* es demasiado bajo, existe la posibilidad de que un atacante pueda romper el sistema. Esto se soluciona rellenando los *m* que se codifican con bits aleatorios por la izquierda. Por ejemplo, si *n* es de 400 bits, una estrategia razonable sería tomar bloques de 392 bits (que es un número exacto de bytes) e incluirles siete bits aleatorios por la izquierda. Cuando codifiquemos simplemente ignoraremos esos siete bits.

Si *d* es demasiado bajo, también existen mecanismos para romper el sistema, por lo que se recomienda emplear valores altos para *d*.

1.4.2.- Firmar y Codificar con RSA.

Con el algoritmo RSA nunca se debe firmar un mensaje después de codificarlo, por el contrario, debe firmarse primero. Existen ataques que aprovechan mensajes primero codificados y luego firmados, aunque se empleen funciones resumen.

1.5.- Algoritmo de Rabin.

Como hemos visto en visto en el sistema RSA sus debilidades nos hacen dudar de su seguridad. Ahora bien, aunque resolver el problema general de la factorización implica romper el sistema RSA, la implicación inversa no sucede. En 1979, Rabin propuso otro sistema basado en la factorización que sí cumple ambas implicaciones.

1.5.1.- Descripción del Algoritmo.

Veamos una descripción del algoritmo donde se señala entre paréntesis qué información ha de ser secreta y cuál pública.

1. Encontrar dos grandes primos p y q(secretos) y definir n mediante su producto $n = p \bullet q$ (público).
2. Encontrar un entero $B < n$ que será parte de la clave pública *(B,n)*.
3. Para cifrar texto, es necesario previamente codificar el texto en un sistema decimal o bien binario y dividirlo en bloques M_i de tamaño *j* o *j-1* de forma que, según sea el alfabeto usado decimal o binario, cumpla $10^{j-1} < n < 10^j$ o bien $2^{j-1} < n < 2^j$. Cuando se toma como tamaño *j*, el descifrado del texto puede ser único, por lo tanto esta elección se hace sólo cuando la unicidad del descifrado no es necesaria).
4. Cifrar el bloque M_i según la expresión $C_i \equiv M_i(M_i + B)(\text{mod } n)$.
5. Para descifrar el criptograma C_i, hay que construir el mensaje $M_i = a \bullet u + b \bullet v$ a partir de las soluciones de las ecuaciones $u^2 + B \bullet u \equiv C_i(\text{mod } p)$ y $v^2 + B \bullet v \equiv C_i(\text{mod } q)$, y los enteros *a* y *b* tales que $a \equiv 1(\text{mod } p)$, $a \equiv 0(\text{mod } q)$, $b \equiv 0(\text{mod } p)$ y $b \equiv 1(\text{mod } q)$.

La base teórica de la función de descifrado está en los siguientes resultados matemáticos.

- Lema 1: Se puede construir una solución de la congruencia $M^2 + B \bullet M \equiv C(\text{mod } p \bullet q)$ mediante las soluciones u y v de las congruencias $u^2 + B \bullet u \equiv C(\text{mod } p)$ y $v^2 + B \bullet v \equiv C(\text{mod } q)$, y de dos enteros *a* y *b* que verifican $a \equiv 0(\text{mod } q)$, y de dos enteros *a* y *b* que verifican $b \equiv 1(\text{mod } q)$, a partir de la expresión $M = a \bullet u + b \bullet v$.

- Lema 2: Si *p* y *q* son primos, siempre se pueden encontrar, mediante el algoritmo euclídeo, los enteros *a* y *b* que satisfagan las condiciones del lema anterior.

- Lema 3: la resolución de la ecuación $u^3 + B \bullet u \equiv C(\text{mod } p)$ es equivalente a la de la ecuación $y^2 \equiv C + (4^{-1})_p B(\text{mod } p)$ donde $(4^{-1})_p$ denota la inversa multiplicativa de 5 en Z_p.

Se llaman residuos cuadráticos módulo *p* a los enteros *d* tales que $y^2 \equiv d(\text{mod } p)$. Para encontrar los residuos cuadráticos modulo *q* son útiles los siguientes resultados. El primero se conoce como criterio de Euler.

- Teorema 4: Un entero a tal que $1 \le a \le p-1$ es un residuo cuadrático módulo un número primo *p* si, y solo si, $a^{\frac{p-1}{2}} \equiv 1 (\text{mod } p)$.
- Lema 5: si *p* es un número primo de la forma *4k-1* y *d* es un residuo cuadrático módulo *p*, entonces una solución de la congruencia $y^2 \equiv d (\text{mod } p)$ viene dada por $y \equiv d^k (\text{mod } p)$.

Mediante ambos resultados queda claro que si los dos primos *p* y *q* son congruentes con *3(mod 4)*, entonces el procedimientos de descifrado se puede realizar en tiempo polinomial.

Resulta recomendable usar el sistema Rabin dos primos *p* y *q* congruentes con 3 en módulo 4.

Aun cuando *p* y *q* no son de la forma, Rabin demuestra que las congruencias $u^2 + Bu \equiv C (\text{mod } p)$ y $v^2 + Bv \equiv C (\text{mod } q)$ se puede resolver no deterministicamente en tiempo polinomial.

Por ejemplo, supongamos que se escoge la clave pública *B*=2, *N*=77 de forma que la clave privada es la factorización de 77; o sea, $p = 7 = 4 \cdot 2 - 1$, $q = 11 = 4 \cdot 3 - 1$. Si el mensaje es *M*=3, entonces $C = M^2 + 2M \equiv 15 (\text{mod } 77)$. Luego para descifrar hay que resolver las ecuaciones $u^2 + 2u \equiv 15 \equiv 1 (\text{mod } 7)$ y $(v+1)^2 \equiv 5 (\text{mod } 11)$.

Las soluciones se consiguen a partir de $(u+1)^2 \equiv 2 (\text{mod } 7)$ y $(v+1)^2 \equiv 5 (\text{mod } 11)$, obteniéndose según el último lema que $u+1 \equiv \pm 2^2 \equiv \pm 4 (\text{mod } 7)$ y $v+1 \equiv \pm 5^3 \equiv \pm 4 (\text{mod } 11)$. Luego u=3 o 2 y v=3 o 6. Usando el algoritmo de Euclides se obtienen *a*=22 y *b*=56 enteros que satisfacen las condiciones del Lema 1 con *p*=7 y *q*=11, por lo que las soluciones de $M^2 + 2M \equiv 15 (\text{mod } 77)$ vienen dadas por:

$$M = \begin{Bmatrix} 3.22 + 3.56 \\ 3.22 + 6.56 \\ 3.22 + 3.56 \\ 2.22 + 6.56 \end{Bmatrix} (\text{mod } 77)$$

El receptor tiene, por tanto, cuatro posibles interpretaciones del criptograma *C*, M_1=3, M_2=17, M_3=58, M_4=72, de las cuales solo una es la correcta.

Este ejemplo muestra una clara desventaja en la aplicación práctica del sistema pues el receptor se encuentra con cuatro posibles descifrados y debe escoger uno. Ahora bien, el enemigo siempre se encuentra con un problema mucho más difícil, según se establece en el siguiente teorema.

- Teorema 6: Romper el sistema Rabin es equivalente a encontrar un algoritmo eficiente para la factorización.

En el sistema de Rabin:

- El receptor tiene que escoger entre varios descifrados.
- El atacante tiene que resolver una factorización.

En la actualidad no existe ningún algoritmo eficiente que sea capaz de factorizar un número de 100 dígitos en menos de 100 años y, además, se puede aventurar que tal algoritmo no se va encontrar en un futuro cercano.

1.6.- Algoritmo de ElGamal.

Fue diseñado en un principio para producir firmas digitales, pero posteriormente se extendió también para codificar mensajes. Se basa en el problema de los logaritmos discretos, que está íntimamente relacionado con el de la factorización de números enteros.

En 1985 el Dr. Taher Elgamal presentó un sistema basado en la aparente intratabilidad del problema del logaritmo discreto. Este nuevo sistema fue desarrollado a partir del conocido algoritmo de distribución de claves que en 1976 desarrollaron Diffie y Hellman

Para generar un par de llaves, se escoge un número primo p y dos números aleatorios a y x menores que p. Se calcula entonces

$$y = a^x \pmod{p}$$

Se muestra un ejemplo del funcionamiento de este sistema. Supóngase que A y B quieren ponerse de acuerdo en la clave secreta común que van a usar ambos para B quieren ponerse de acuerdo en la clave secreta común que van a usar ambos para comunicarse entre si. Para ello escoge un entero aleatorio k_A entre 1 y p-1(secreto) y calcula $a^{K_A} Z_p$ (público). B hace lo mismo con k_B. La clave secreta común que usarán ambos es $a^{K_A K_B}$. Es secreta porque sólo ellos pueden calcularla. En este cifrado se distinguen las claves secreta, pública, de cifrado y de descifrado.

El sistema de ElGamal tiene un funcionamiento distinto al del resto de los sistemas de clave pública, ya que el cifrado se realiza utilizando, además de la clave pública del receptor, la clave secreta del emisor.

1.6.1.- *Descripción de Algoritmo.*

- Serán de información pública un número primo grande p y a, una raíz primitiva de 1 módulo p(tal que $a^{p-1} \equiv 1(\text{mod } p)$ y $a^{d'1}! \equiv 1(\text{mod } p)\ \forall\ d$ tal que *1 < d < p*).
- La clave privada del usuario *B* es un entero K_B escogido dentro del intervalo *[1,p-1]*.
- La clave pública de *B* es el entero $K_B \equiv a^{K_B}(\text{mod } p)$.
- Se supone que el emisor *A* quiere enviar un mensaje $M(1 \le M \le p-1)$ al receptor *B*.

El proceso de cifrado que debe llevar a cabo consistente en:

1. Escoger aleatoriamente un entero K_A tal que $1 \le K_A \le p-1$.
2. Calcular la clave de cifrado a partir de las claves privada de *A* y pública de *B*, $Q \equiv K_B{}^{k_A}(\text{mod } p)$.
3. Cifrar el mensaje *M* según la expresión $C \equiv Q \bullet M(\text{mod } p)$

El proceso de descifrado que hay que llevar a cabo consiste en:

1. Tener Q gracias a la clave pública de *A* K_A, mediante la fórmula $Q \equiv K_A^{K_B}(\text{mod } p)$.
2. Recuperar *M* a partir de $M \equiv (Q^1)_p \bullet C(\text{mod } p)$.

Por ejemplo, supongamos que los datos del sistema son *p*=71 y *a*=7, que cumplen las condiciones $7^{70} \equiv 1(\text{mod } 71)$ y $7^{d-1}! \equiv 1(\text{mod } 71) \forall d$ tal que *1 < d < p*. Si *K*=3 y *K*=2, entonces $Q \equiv 3^2 \equiv 9(\text{mod } 71)$, por lo que el cifrado de *M*=30 es $C_1 \equiv 7^2 \equiv 49(\text{mod } 71)$ y $C_2 \equiv 9 \bullet 30 \equiv 270 \equiv 57(\text{mod } 71)$.

La seguridad de este sistema se basa completamente en el problema del logaritmo discreto, puesto que el enemigo tiene que encontrar la clave privada *k* a partir de la clave pública $K_B \equiv a^{K_B}(\text{mod } p)$, conocidos *a* y *p*.

En el caso binario, el ataque vía resolución del problema del logaritmo discreto resulta fructífero, por lo que obliga al criptógrafo a aumentar el tamaño del cuerpo finito. Actualmente se considera necesario un cardinal del valor aproximado a 2^{600}, lo que en la práctica presenta el inconveniente de tener que aumentar el tamaño de las claves.

Las principales desventajas del sistema de ElGamal:

- El criptograma es el doble de largo que el texto original.
- Hay que escoger un número aleatorio para cada cifrado.
- El tamaño del espacio de claves ha de ser grande.

1.6.2.- Firmas Digitales de ElGamal.

Para firmar un mensaje *m* basta con escoger un número *k* aleatorio, tal que *MCD(k,p-1)=1*, y calcular

$$a = g^k \pmod{p}$$

Luego se emplea el Algoritmo Extendido de Euclides para resolver la ecuación

$$m = (xa + kb) \pmod{(p-1)}$$

La firma la constituye el par *(a,b)*. En cuanto al valor *k*, debe mantenerse en secreto y ser diferente cada vez. La firma se verifica comprobando que

$$y^a a^b = g^m \pmod{p}$$

1.6.3.- Codificación de ElGamal.

Para codificar el mensaje *m* se escoge primero un número aleatorio *k* primo relativo con *(p-1)*, que también será mantenido en secreto. Calculamos entonces las siguientes expresiones

$$a = g^k \pmod{p}$$
$$b = y^k m \pmod{p}$$

El par *(a,b)* es el texto cifrado, de doble longitud que el texto original. Para decodificar se calcula

$$m = \frac{b}{a^x} \pmod{p}$$

1.7.- Los Beneficios y las Desventajas de la Criptografía Asimétrica.

La criptografía asimétrica evita perfectamente los problemas administrativos de clave que presentan la criptografía simétrica. Cada persona solo necesita compartir una clave: su clave pública; como resultado. Las propiedades de escalabilidad y administración de claves son muy superiores con los sistemas de claves públicas/privadas cuando se comparan con los sistemas puros de claves asimétricas.

Un beneficio adicional de los sistemas de clave pública/privada es el hecho de que en dichos sistemas usted no necesita tener una relación prioritaria con alguien para enviarle información cifrada. Con los sistemas de clave simétrica, se debe de establecer de alguna forma una relación anterior, de modo que pueda intercambiar la clave simétrica que se va a usar para una operación de decodificación posterior. Con los sistemas de claves públicas, el emisor sólo necesita observar la clave pública del receptor, codificar el documento y enviárselo. El receptor ya tiene la clave privada equivalente que necesita para decodificar el texto cifrado.

Un último beneficio de los sistemas de clave pública es su naturaleza asimétrica. Esto permite que cada propietario realice operaciones matemáticas con su clave privada, que nadie más en el mundo pueda realizar.

Desafortunadamente los algoritmos asimétricos son lentos, pueden ser de diez a cien veces más lentos que un algoritmo simétrico con una fortaleza comparable. Esto no puede ser mucho problema cuando se esta codificando unos pocos cientos de bytes de longitud, pero si se necesitara codificar una gran cantidad de información se convertiría en un gran problema.

Además de la poca velocidad de los algoritmos asimétricos tienen otro gran problema. Cuando se hace codificaciones utilizando un algoritmo asimétrico, el tamaño del texto cifrado es mayor que el del texto claro original; esto es un tema relevante cuando se usan múltiples niveles de cifrado. Si se utiliza la criptografía asimétrica pura una aplicación codificaría los datos, lo cual aumentaría el tamaño y después la enviaría a través de una sesión segura por la Web, que de nuevo aumentaría el tamaño.

Ello podría funcionar en un túnel cifrado por un proveedor de seguridad de Internet (IPSec), con lo que aumentaría de nuevo el tamaño.

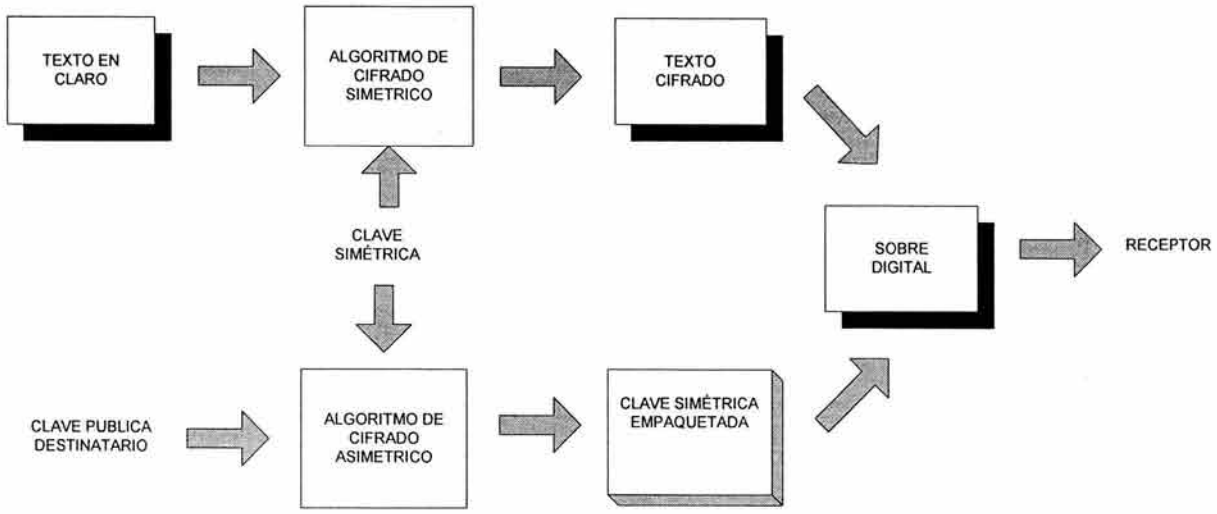
Afortunadamente, existe una manera de salir de estos problemas. En cada área donde una clase de algoritmo es débil, la otra, es fuerte. En esta situación se debe hallar una manera de combinar las dos soluciones, de forma tal que capture las fortalezas de cada una sin heredar sus problemas.

La solución ideal deberá satisfacer las siguientes propiedades:

- La solución debe ser segura.

- El cifrado debe ser rápido.
- El texto cifrado debe ser compacto.
- La solución debe servir en escalas de grandes poblaciones.
- La solución no debe ser vulnerable a la intercepción de claves.
- La solución no debe requerir una relación previa entre partes.
- La solución debe soportar firmas digitales y aceptación.

La combinación de criptografía simétrica y asimétrica satisface cada uno de estos requerimientos. Con el cifrado simétrico puede lograr velocidad y texto compacto, y con la criptografía de clave pública/privada puede lograr escalabilidad, mayor facilidad de administración de la clave, resistencia a la intercepción y firma digital/aceptación.

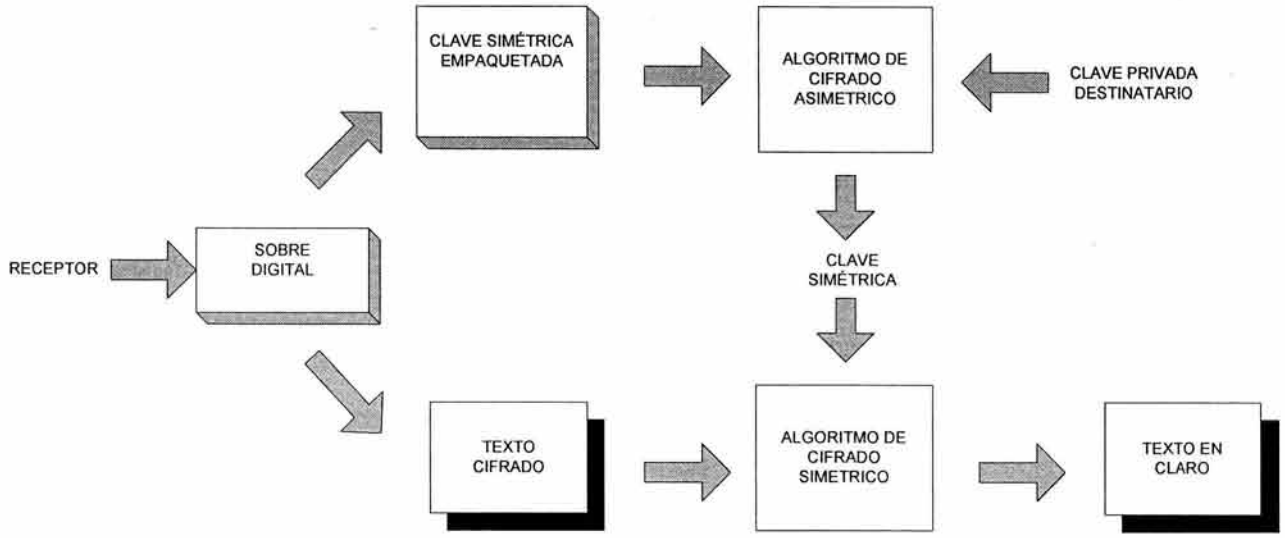


El proceso comienza con la generación de una clave simétrica aleatoria que se utiliza para codificar el texto en claro, produciendo la versión cifrada de la misma. El cifrado simétrico es seguro, rápido y el texto cifrado resultante es compacto. El problema que se tenía con el cifrado simétrico era cómo conseguir la clave simétrica para el receptor. Es aquí donde entra la función de la criptografía asimétrica.

Conseguimos la clave pública del receptor en un directorio y la usamos para codificar sólo la clave simétrica aleatoria. Seguro, la criptografía asimétrica es lenta, pero dado el tamaño de la clave simétrica es muy pequeño (por lo general 128 bits), el tiempo real que se gasta en el cifrado asimétrico es poco. El resultado de este cifrado es una clave de cifrado simétrico aleatorio, codificado con una clave pública asimétrica. Este proceso también es conocido como operación de clave empaquetada.

El último paso en este proceso es unir la clave empaquetada al texto cifrado, como preparación para enviarlo al receptor. En ocasiones, la combinación se conoce como sobre digital, que se envía al receptor a través de Internet. Si un atacante intercepta el sobre digital, no puede usarlo dado que tendría que descifrar primero la clave pública y así obtener el contenido del sobre digital, después tendría que descifrar la clave simétrica para así utilizarla para descifrar el texto codificado en el canal inseguro.

El uso de la codificación de clave pública/privada para envolver la clave simétrica da la solución de escalabilidad, protege contra la intercepción de la clave de cifrado simétrico, no requiere de una relación previa entre las partes involucradas y soporta firmas digitales; por consiguiente, satisface los demás criterios necesitados.



El proceso comienza con la recepción del sobre digital. El primer paso es descomponerlo en las partes que lo constituyen: el texto cifrado y la clave empaquetada. Recodemos que la clave es la clave simétrica cifrada con la clave pública del receptor. Dado que el receptor no puede tener acceso al texto cifrado el siguiente paso es recuperar la clave simétrica. La clave empaquetada se descifra usando la clave privada del receptor. La clave simétrica es usada, entonces, para descifrar el texto cifrado, recuperando el texto claro original. Después se descarta la clave simétrica original porque su uso ha terminado.

Esta solución parece elegante y sencilla. El proceso que se acaba de describir (con algunas variaciones) es el fundamento para la mayoría de las soluciones de cifrado moderno, que van desde el correo electrónico cifrado, pasando por las sesiones cifradas en la Web, hasta las redes privadas virtuales cifradas. Hemos tenido éxito en lograr los beneficios de la criptografía simétrica (seguridad, velocidad, texto compacto) con los beneficios de la criptografía asimétrica (seguridad, escalabilidad, sin relaciones previas).

1.8.- Recapitulación de la Criptografía Asimétrica.

- Con la criptografía asimétrica que esta cifrada con una clave (pública o privada) sólo se puede descifrar con la otra clave (privada o pública).
- El cifrado asimétrico es seguro.
- Dado que no se necesita enviar una clave al receptor, la codificación asimétrica no sufre por la intercepción de claves.
- El número de claves que necesita distribuir es el mismo de participantes, de ahí que la criptografía sirve bien en escala de poblaciones muy grandes.
- La criptografía asimétrica no tiene los problemas complejos de distribución de claves.
- La criptografía asimétrica no exige una relación previa entre las partes para hacer el intercambio de claves.
- La criptografía asimétrica soporta firmas digitales y aceptación.
- El cifrado asimétrico es relativamente lento.
- El cifrado asimétrico expande el texto cifrado.

2.- Comparativa Criptografía Simétrica y Criptografía Asimétrica.

	Algoritmos Simétricos de Cifrado	**Algoritmos Asimétricos de Cifrado**
Algoritmo	El algoritmo utiliza la misma clave para cifrado y descifrado.	El algoritmo utiliza un par de claves, la clave privada y la clave pública, una para descifrar y otra para cifrar respectivamente.
Complejidad de implementación.	Los algoritmos simétricos se basan principalmente en principios de confusión y difusión de Shannon, la mayoría en redes de Feistel, su implementación no es muy compleja.	Los algoritmos asimétricos se basan principalmente los problemas matemáticos complejos cuya solución inversa no existe o es poco eficiente, su implementación es más compleja respecto a los algoritmos simétricos.
Tamaño del criptograma.	El mismo que el tamaño del texto en claro.	Considerablemente mayor que el texto en claro, lo cual es una desventaja.
Velocidad.	Son rápidos y no requieren de gran cantidad de procesamiento de cpu.	Son mucho más lentos que los algoritmos simétricos, utilizan gran cantidad de proceso de cpu ya que realizan complejos cálculos matemáticos, lo cual es una desventaja.

Claves	Usa la misma clave para cifrado y descifrado. Debe ser por lo menos la clave mayor a 128 bits para evitar ataques de búsqueda exhaustiva.	Utilizan dos claves, una publica y una privada, lo que le da gran ventaja en medios inseguros de comunicación. Las claves deben ser de por lo menos 1024 bits para ser seguras.
Administración de claves.	La administración de claves es muy compleja ya que el intercambio de la clave debe hacerse por un medio muy seguro.	La administración es sencilla por medio de anillos de confianza donde se almacenaran y expondrán de manera publica los claves publicas de cifrado. La clave privada es la única que puede descifrar el criptograma cifrado con la clave publica correspondiente, lo que le da ventaja a este tipo de algoritmos para el envió y uso por medios de comunicación inseguros como Internet.
Firmas digitales.	La criptografía simétrica no se ajusta para realizar firmas digitales ni autenticación.	La criptografía asimétrica es ideal para firmal digitalmente un mensaje debido al uso de su par de claves.
Seguridad.	Muy seguro con claves de 128 bits bien seleccionadas.	Seguro con claves de al menos 1024 bits. Debido que la criptografía asimétrica se basa en problemas matemáticos complejos o aparentemente irresolubles corre el riesgo que se descubra un algoritmo eficiente para la resolución de tales problemas por lo que su seguridad se vea gravemente lesionada.

3.- Conclusiones de la Criptografía Asimétrica.

La criptografía asimétrica es por definición aquella que utiliza dos claves diferentes para cada usuario, una para cifrar que se le llama clave pública y otra para descifrar que es la clave privada. El nacimiento de la criptografía asimétrica se dio al estar buscando un modo más práctico de intercambiar las llaves simétricas. Diffie y Hellman, proponen una forma para hacer esto, sin embargo

no fue hasta que el popular método de Rivest Shamir y Adleman RSA publicado en 1978, cuando toma forma la criptografía asimétrica, su funcionamiento esta basado en la imposibilidad computacional de factorizar números enteros grandes.

Actualmente la Criptografía asimétrica es muy usada; sus dos principales aplicaciones son el intercambio de claves privadas y la firma digital. Una firma digital se puede definir como una cadena de caracteres que se agrega a un archivo digital que hace el mismo papel que la firma convencional que se escribe en un documento de papel ordinario. Los fundamentos de la criptografía asimétrica pertenecen a la teoría de números.

En la actualidad la criptografía asimétrica o de clave pública se divide en tres familias según el problema matemático en el cual basan su seguridad.

El algoritmo RSA es el algoritmo asimétrico de cifrado mas usado, su fuerza la deposita en la deposita en problema complejo de factorización de un numero producto de dos números primos muy grandes, la complejidad de implementación del algoritmo RSA radica en la correcta selección de los dos números primos que formaran respectivamente las claves publicas y privadas.

El uso de los algoritmos asimétricos es mas que nada aprovechado en transacciones por medios inseguros de comunicación como Internet donde haciendo uso del intercambio de clave publicas se cifra la sesión entre los dos elementos de la comunicación combinándose con la criptografía simétrica, es decir se aprovecha las ventajas de los algoritmos asimétricos y asimétricos y superando sus debilidades.

CAPÍTULO 4

Aplicaciones Criptográficas.

1.- Métodos de Autenticación.

Por autenticación entenderemos cualquier método que nos permita comprobar de manera segura alguna característica sobre un objeto. Dicha característica puede ser su origen, su no manipulación, su identidad, etc. Consideraremos tres grandes tipos dentro de los métodos de autenticación:

- Autenticación de mensaje: Queremos garantizar la procedencia de un mensaje conocido, de forma que podamos asegurarnos de que no es una falsificación. Este mecanismo se conoce habitualmente como firma digital.
- Autenticación de usuario mediante contraseña: En este caso se trata de garantizar la presencia se un usuario legal en el sistema. El usuario deberá poseer una contraseña secreta que le permita identificarse.
- Autenticación de dispositivo: Se trata de garantizar la presencia de un dispositivo válido. Este dispositivo puede estar solo o tratarse de una llave electrónica que sustituye a la contraseña para identificar a un usuario.

La autenticación de usuario por medio de alguna característica biométrica, como pueden ser las huellas digitales, la retina, el iris, la voz, etc. puede reducirse a un problema de autenticación de dispositivo, solo que el dispositivo en este caso es el propio usuario. Aquí únicamente abordaremos métodos de autenticación basados en técnicas criptográficas.

1.1.- Algoritmos Hash (Reseñas).

Los algoritmos hash son muy comunes en computación y quizá son los que tienen un uso más común para acelerar el proceso de indexación de grandes arreglos o bases de datos de información. A pesar de su amplio uso, la mayoría de las personas no saben cuál es el sentido de un hash o verificación.

Un algoritmo hash tomará un gran bloque de datos y lo comprimirá en una huella digital (fingerprint) o reseña (digest) de los datos originales.

Si tuviéramos el numero 483820 y lo dividiéramos entre 4, el resultado sería 120995. En este sentido, esta cifra es una huella digital para la ecuación (483820 dividido entre 4). Sin importar cuántas veces divida 48320 entre 4, el resultado siempre será 120955. Si usted cambia alguno de los números de la ecuación, la división no dará como resultado 120955. Alternativamente, si solo se tuviera el numero 120955, pero ninguna otra información adicional, sería improbable que se descubriera la ecuación original, dado que existe un número infinito de divisiones que podrían arrojar el mismo resultado.

En muchas formas estas características son las mismas que se asocian con los algoritmos hash. Con un hash, usted toma un gran bloque de datos y calcula una ecuación a través de ellos. El resultado del hash es un valor más pequeño que los datos originales. Incluso, si usted cambia un bit de los datos originales, el valor hash del resultado será diferente; además, cómo con el ejemplo de la división, existen muchos conjuntos de datos diferentes que podrían dar como resultado el mismo valor hash.

Otro ejemplo es el valor de verificación de redundancia cíclica (Cyclic Redundancy Check, CRC), que se pone al final de la mayoría de los mensajes de comunicación.

Cuando los mensajes se van a enviar a través de una línea de comunicaciones, es común ejecutar una ecuación polinómica a través de los bytes del mensaje. Este polinomio da un resultado, el código CRC, el cual se une al final del mensaje antes de enviarlo. Cuando el sistema receptor capta el mensaje con la CRC unido a él, en esencia ejecuta el mismo polinomio a través del mensaje (excluyendo el CRC original) y produce una segunda copia de la CRC. El software, entonces, compara la CRC original con la nueva CRC. Si ambos corresponden, hay un alto grado de confianza de que el mensaje no fue modificado durante su viaje a través de la red.

En un sentido, la CRC actúa como una huella digital o reseña del mensaje, el cual se puede verificar en el receptor del mismo. Existen múltiples mensajes que producirán la misma CRC y si un solo bit del mensaje cambia, el valor de la CRC cambiará. La CRC está presentando las mismas propiedades que descubrimos para un hash.

Todos los algoritmos hash que se utilizan en criptografía están diseñados con algunas propiedades especiales

- No se puede poner a funcionar el hash hacia atrás y recuperar algo de texto claro inicial.
- La reseña resultante no dirá nada sobre el texto claro inicial.

Desde el punto de vista computacional, no es factible descubrir texto claro que verifique un valor especifico. Esto evita que un pirata informático trate de sustituir un documento sin que se presenten fallas en la correspondencia de la reseña.

Existen varios algoritmos hash criptográficos. MD2 es un hash RSA que produce una reseña de 128 bits que se optimiza para microprocesadores de baja tecnología de 8 bits. El MD5 también produce una reseña de 128 bits, pero esta optimizado para procesadores de 32 bits. El hash SHA-1 también esta optimizado para procesadores de alta tecnología y produce una reseña de 160 bits.

1.2.- Firmas Digitales. Funciones Resumen.

Anteriormente vimos que la criptografía asimétrica permitía autenticar información, es decir, poder asegurar que un mensaje *m* proviene de un emisor *A* y no de cualquier otro. También se vio que la autenticación debía hacerse empleando una función resumen y no codificando el mensaje completo.

Sabemos que un mensaje *m* puede ser autenticado codificando con la llave privada K_P el resultado de aplicarle una función resumen, $E_{K_p}(r(m))$. Esa información adicional (que denominamos firma, reseña o signatura del mensaje *m*) sólo puede ser generada por el poseedor de la clave privada K_P. Cualquiera que tenga la llave pública correspondiente estará en condiciones de decodificar y verificar la firma. Para que sea segura, la función resumen *r(x)* debe cumplir además ciertas características:
R(m) es de longitud fija, independientemente de la longitud de m.

- Dado *m*, es fácil calcular r(m).
- Dado *r(m)*, es computacionalmente intratable recuperar m.
- Dado *m*, es computacionalmente intratable obtener un *m'* tal que *r(m) = r(m')*.

1.2.1.- Longitud Adecuada para una Firma Digital.

Para decidir cuál debe ser la longitud apropiada de una firma, veamos primero el siguiente ejemplo: ¿Cuál es la cantidad de personas que hay que poner en una habitación para que la probabilidad de que el cumpleaños de una de ellas sea el mismo día que otra?
Debemos calcular *n* tal que

$$n\frac{1}{365} > 0.5$$

Luego $n > 182$. Sin embargo, ¿Cuál sería la cantidad de gente necesaria para que dos personas cualesquiera tengan el mismo cumpleaños? Cada pareja tiene una probabilidad de 1/365 de compartir el mismo cumpleaños, y en un grupo de n personas hay *n(n-1)/2* parejas diferentes de personas, luego

$$\frac{n(n-1)}{2} \bullet \frac{1}{365} > 0.5$$

Esto se cumple si n>19, una cantidad sorprendentemente mucho menos que 182.

La consecuencia de esta paradoja es que aunque resulte muy difícil dado m calcular un m' tal que $r(m)=r(m')$, es mucho menos costoso buscar dos valores aleatorios m y m', tales que $r(m)=r(m')$.

En el caso de una firma de 64 bits, necesitaríamos 2^{64} mensajes dado un m para obtener el m', pero bastaría con generar aproximadamente 2^{32} mensajes aleatorios para que aparecieran dos con la misma firma. En general, si la primera cantidad es muy grande, la segunda cantidad es aproximadamente su raíz cuadrada. El primer ataque nos llevaría 600,000 años con una computadora que genera un millón de mensajes por segundo, mientras que el segundo necesitaría apenas una hora.

Hemos de añadir pues a nuestra lista de condiciones sobre las funciones resumen la siguiente:

- Debe ser difícil encontrar dos mensajes aleatorios, m y m', tales que $r(m)=r(m')$.

Hoy se recomienda emplear firmas de al menos 128 bits, siendo 160 bits el valor más usado.

1.2.2.- Estructura de una Función Resumen.

Las funciones resumen se basan en la idea de funciones de compresión, que dan como resultado bloques de longitud n a partir de bloques de longitud m. Estas funciones se encadenan de forma iterativa, haciendo que la entrada en el paso i sea función del i-ésimo bloque del mensaje y de la salida del paso $i-1$. En general, se suele incluir en algunos de los bloques del mensaje m, ya sea al principio o al final, información sobre la longitud total del mensaje. De esta forma se reducen las probabilidades de que dos mensajes con diferentes longitudes den el mismo valor en su resumen. A continuación veremos dos algoritmos de generación de firmas digitales: MD5 y SHA.

1.2.3.- Algoritmo MD5.

Se trata de uno de los más populares algoritmos de generación de firmas, debido a su inclusión en las primeras versiones de PGP(Pretty Good Privacy), de una serie de mejoras sobre el algoritmo MD4, diseñado por Ron Rivest, procesa los mensajes de entrada en bloques de 512 bits, y produce una salida de 128 bits.

En primer lugar, el mensaje se alarga hasta que su longitud es exactamente 64 bits inferior a un múltiplo de 512 bits. El alargamiento se lleva a cabo añadiendo un 1 seguido de tantos ceros como

sea necesario. En segundo lugar, se añaden 64 bits que representan la longitud del mensaje original, sin contar los bits añadidos en el proceso anterior. De esta forma tenemos el mensaje como un número entero de bloques de 512 bits, y le hemos añadido información sobre la longitud del mensaje.

A continuación, se inicializan cuatro registros de 32 bits con los siguientes valores (Hexadecimales):

$A = 01234567$
$B = 89ABCDEF$
$C = FEDCBA98$
$D = 76543210$

Posteriormente comienza el lazo principal del algoritmo, que se repetirá para cada bloque de 512 bits del mensaje. En primer lugar copiaremos los valores de *A,B,C* y *D* en otras cuatro variables, *a,b,c* y *d*. Luego definiremos las siguientes cuatro funciones:

$$F(X,Y,Z) = (X \wedge Y) \vee ((\neg X) \wedge Z)$$
$$G(X,Y,Z) = (X \wedge Z) \vee ((Y \wedge (\neg Z))$$
$$H(X,Y,Z) = X \oplus Y \oplus Z$$
$$I(X,Y,Z) = Y \oplus (X \wedge (\neg Z))$$

Ahora representaremos por m_j el j-ésimo bloque de 32 bits del mensaje *m*(0 a 15), y definiremos otras cuatro funciones:

$$FF(a,b,c,d,m_j,s,t_i) \ \ representa \ \ a = b + ((a + F(b,c,d) + m_j + t_i) \lhd s)$$
$$GG(a,b,c,d,m_j,s,t_i) \ \ representa \ \ a = b + ((a + G(b,c,d) + m_j + t_i) \lhd s)$$
$$HH(a,b,c,d,m_j,s,t_i) \ \ representa \ \ a = b + ((a + H(b,c,d) + m_j + t_i) \lhd s)$$
$$II(a,b,c,d,m_j,s,t_i) \ \ representa \ \ a = b + ((a + I(b,c,d) + m_j + t_i) \lhd s)$$

donde la función $a \lhd s$ representa desplazar circularmente el valor a *s* bits a la izquierda.

Las 64 operaciones que se realizan en total quedan agrupadas en cuatro rondas (Ver Apéndice C).

Finalmente, los valores resultantes de *a,b,c* y *d* son sumados con *A,B,C* y *D*, se procesa el siguiente bloque de datos. El resultado final del algoritmo es la concatenación de *A,B,C* y *D*.

Últimamente el algoritmo MD5 ha mostrado ciertas debilidades, aunque sin implicaciones prácticas reales, por lo que se sigue considerando en la actualidad un algoritmo seguro, si bien su uso tiende a disminuir.

1.2.4.- El Algoritmo SHA.

El algoritmo SHA fue desarrollado por la NSA (Nacional Segurity Agency) de EUA, para ser incluido en el estándar DSS (Digital Signature Estándar). Al contrario de los algoritmos de cifrado propuestos por esta organización, SHA se considera seguro y libre de puertas traseras, ya que favorece a los propios intereses de la NSA que el algoritmo sea totalmente seguro. Produce firmas de 160 bits, a partir de bloques de 512 bits del mensaje original.

El algoritmo es similar a MD5, y se inicializa igual que éste, solo que con cinco registros de 32 bits en lugar de cuatro:

$A = 67458701$
$B = EFCDAB89$
$C = 98BADCFE$
$D = 10325476$
$E = C3D2E1F0$

Una vez que los cinco valores están inicializados, se copian en cinco variables *a,b,c,d* y *e*. El ciclo principal tiene cuatro rondas con 20 operaciones cada una:

$$F(X,Y,Z) = (X \wedge Y) \vee ((\neg X) \wedge Z)$$
$$G(X,Y,Z) = X \oplus Y \oplus Z$$
$$H(X,Y,Z) = (Z \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z)$$

La operación *F* se emplea en la primera ronda, la *G* en la segunda y en la cuarta, y la *H* en la tercera. Además se emplean cuatro constantes, una para cada ronda:

$K_0 = 5A827990$
$K_1 = 6ED9EBA1$
$K_2 = 8F1BBCDC$
$K_3 = CA62C1D6$

El bloque de mensaje *m* se divide en 16 partes de 32 bits m_0 a m_{15} y se convierte en 80 bloques de 32 bits w_0 a w_{79} usando el siguiente algoritmo:

$$w_t = m_t \qquad para\, t = 0...15$$
$$w_t = (w_{t-3} \oplus w_{t-8} \oplus w_{t-14} \oplus w_{t-16}) \lhd 1 \quad para\, t = 16...79$$

La NSA introdujo el desplazamiento a la izquierda para corregir una debilidad del algoritmo, pero no ha dado explicaciones sobre la naturaleza de esta debilidad.

1.3.- Autenticación de Dispositivos.

Los algoritmos simétricos pueden ser empleados para autenticar dispositivos, siempre que éstos permitan hacer operaciones de cifrado y descifrado a la vez que impidan acceder físicamente a la clave que llevan almacenada. Un ejemplo de estos mecanismos de autenticación lo tenemos en las tarjetas de emplean los teléfonos GSM. Dichas tarjetas llevan implementado un algoritmo simétrico de cifrado, y usan una clave k almacenada en un lugar de la memoria que no se puede leer desde el exterior. En cada tarjeta se graba una única clave, de la que se guarda una copia en lugar seguro. Si la compañía quiere identificar una tarjeta simplemente genera un bloque de bits aleatorio X y calcula su criptograma $E_K(X)$ asociado. Posteriormente se envía X a la tarjeta para que lo codifique. Si ambos mensajes codificados coinciden, la tarjeta será auténtica. Esta técnica se conoce como autenticación por desafío. Es importante notar que en ningún momento se ve comprometida la clave k.

1.4.- Autenticación de usuario mediante contraseñas.

El sistema de autenticación basa su funcionamiento en una información secreta conocida únicamente por el usuario, que le permite identificarse positivamente frente al sistema. Supondremos que el usuario se encuentra en una terminal segura, es decir, libre de posibles ataques del exterior.

Distinguiremos entonces dos casos claramente diferenciados:

- El sistema se comunica con el usuario, pero éste no puede entrar en él. Un ejemplo es un cajero automático. El usuario carece de acceso a los archivos del sistema y no tiene posibilidad de ejecutar aplicaciones en él, únicamente puede llevar a cabo una serie de operaciones muy restringidas.
- El sistema permite al usuario entrar. Este es el caso de los sistemas operativos como UNÍX, que ofrecen la posibilidad de operar el sistema desde terminales remotas.

El primer caso es el más simple y sencillo de resolver. Basta con que el sistema mantenga la lista de usuarios y sus contraseñas asociadas en un archivo. Como este archivo no puede ser consultado desde el exterior, es imposible averiguar la clave de un usuario. Para protegerse de los ataques de fuerza bruta es suficiente con limitar el número de intentos desde un terminal concreto e introducir retardos cuando la contraseña introducida sea errónea.

El caso *b* es considerablemente más complejo. Por un lado deberemos tomar las mismas medidas que en el caso anterior para protegernos de los ataques de fuerza bruta, y por otro lado hemos de tener en cuenta que cualquier usuario puede acceder a los archivos. En versiones antiguas de UNIX el archivo de contraseñas podía ser descargado por cualquier usuario anónimo por lo que las palabras clave no pueden ser almacenadas como texto plano en dicho archivo. El mecanismo que surge entonces de manera inmediata consiste en almacenar en el archivo de claves la signatura concreta. Los sistemas operativos modernos impiden leer el archivo de contraseñas de forma directa, pero eso no evita que en algunos casos éste pueda quedar comprometido.

1.4.1.- Elección de la Contraseña (ataques mediante diccionario).

El principal problema de las palabras clave son las elecciones poco afortunadas por parte de los usuarios. Desgraciadamente todavía hay personas que emplean su fecha de nacimiento, el nombre de algún familiar o la matrícula del automóvil como contraseña. Un buen atacante podría tratar de generar millones de claves y construir un diccionario. El siguiente paso sería precalcular las signaturas de todas las claves que hay en su diccionario. Si el atacante ha obtenido el archivo con las contraseñas, bastaría con compararlas con las de su diccionario para obtener en pocos segundos una contraseña que le permita entrar en el sistema. Tengamos en cuenta que un diccionario con más de 150,000 claves de ocho caracteres cabe en un disquete de 3 ½, y que han existido casos en los que con diccionarios de este tamaño se ha conseguido averiguar un sorprendente número de claves.

Para protegerse a este tipo de ataques se introduce en el cálculo de la signatura de la contraseña la denominada "sal", que no es ni más ni menos que un conjunto de bits aleatorios que se añaden a la palabra clave antes de calcular su firma. En el archivo de claves se almacenara, junto con la signatura, la sal necesaria para su obtención. Esto obligará al atacante a recalcular todas las signaturas de su diccionario antes de poder compararlas con una de las entradas del archivo de claves del sistema. De todas formas debemos de evitar a toda costa emplear contraseñas que puedan aparecer en un diccionario.

Además de estar bien resguardadas, las palabras clave han de cumplir una serie de condiciones para que puedan considerarse seguras.

- Deben ser memorizadas: Una contraseña jamás debe ser escrita en un papel, por razones obvias.
- Suficientemente complejas: Una buena contraseña debe constar de al menos ocho letras. Una contraseña con únicamente 6 caracteres alfanuméricos (números y letras), tenemos solamente dos mil millones de posibilidades. Teniendo en cuenta que hay programas para PC

capaces de probar más de cuarenta mil claves en un segundo, una clave de estas características podría ser descubierta en menos de quince horas.

- Carecer de significado: Una contraseña jamás debe significar nada, puesto que entonces aumentara la probabilidad de que aparezca en algún diccionario. Evitemos los nombres propios, nombres de lugares o cosas.
- Fáciles de recordar: Puesto que una palabra clave debe ser memorizada, no tiene sentido emplear contraseñas difíciles de recordar. Para esto podemos seguir reglas como que la palabra se pueda pronunciar en voz alta, o que responda a algún acrónimo más o menos complejo. Es importante hacer énfasis a evitar las palabras que signifiquen algo.
- Deben ser modificadas con frecuencia: Hemos de partir de la premisa de que toda palabra clave caerá tarde o temprano, por lo que será muy recomendable que nuestras contraseñas sean cambiadas periódicamente. La frecuencia con que se produzca el cambio dependerá de la complejidad de las claves y del nivel de seguridad que se desee alcanzar. Ante cualquier sospecha, cambiar todas las claves.

2.- Dinero Electrónico.

Lo contrario a la autenticación es la falsificación, cuando se habla de falsificar inmediatamente nos imaginamos dinero. Uno de los objetivos del dinero electrónico es evitar la falsificación. El dinero físico es bastante engorroso. Es incómodo de transportar, se desgasta con facilidad y suele ser susceptible de falsificación. Además debe ser cambiado periódicamente debido a la renovación de las monedas. Para sustituirlo están las tarjetas de crédito y los cheques. El problema que éstos presentan es que rompen el anonimato de quien los emplea, por lo que la privacidad queda comprometida. Existen sin embargo protocolos que permiten el intercambio de capital de una forma segura y anónima. Es lo que denominaremos dinero electrónico.

Las ventajas que reportará su extensión en un futuro próximo son evidentes. Facilitará el comercio electrónico y las compras por Internet, y además garantizará el anonimato en las transacciones comerciales. Hoy por hoy no existe un único protocolo aceptado universalmente, auque si muchas propuestas.

Supongamos que queremos enviar un cheque anónimo. Para ello creamos cien cheques por la misma cantidad, los metemos cada uno en un sobre y los enviamos al banco. El banco abre noventa y nueve al azar y se asegura de que todos lleven la misma cantidad. Al que queda le pone su sello sin abrirlo y nos lo devuelve, restando la cantidad de nuestra cuenta corriente. Tenemos ahora un cheque validado por el banco, pero del que el banco no sabe nada (la probabilidad de que tenga una cantidad diferente de la que el banco su supone es únicamente del uno por ciento). Cuando entreguemos ese

cheque y alguien quiera cobrarlo, bastará que lo lleve al banco, que verificará su sello y abonará su importe, sin conocer su procedencia. Este protocolo se puede implementar mediante criptografía asimétrica de la siguiente forma: se construyen cien órdenes de pago anónimas y se envían al banco, este las comprueba y firma digitalmente una, restando además la cantidad correspondiente de nuestra cuenta. El destinatario podrá cobrar la orden de pago cuando quiera.

El problema que surge con el protocolo anterior es que una orden se puede cobrar varias veces. Para evitar esto bastará con incluir una cadena aleatoria en cada orden de pago, de forma que sea muy fácil tener dos órdenes con la misma cadena. Cada una de las cien órdenes tendrá pues una cadena de identificación diferente. El banco, cuando pague la cantidad, únicamente tendrá que comprobar la cadena de identificación de la orden para asegurarse de que no la ha pagado ya.

Ahora cada orden de pago es única, por lo que el banco puede detectar una orden duplicada, pero no sabe quién de los dos ha cometido fraude: el que paga o el que cobra. Existen mecanismos que permiten saber, cuando la orden aparece duplicada, quién de los dos ha intentado engañar. Si lo ha hecho el cobrador, puede ser localizado sin problemas, pero ¿y si el que envía dos veces la misma orden es el pagador?. El protocolo completo de dinero electrónico hace que la identidad del pagador quede comprometida si envía dos veces la misma orden de pago, por lo que podrá ser sancionado.

Los diferentes protocolos propuestos hoy en día son muy dispares, y que todavía ninguno de ellos ha sido adoptado como estándar. Los clasificaremos en tres grupos, según el tipo de criptografía en que se basen:

- Basados en Criptografía Simétrica: NetBill y NetCheque.
- Basados en Criptografía Asimétrica: Proyecto CAFÉ, ECash, NetCash, CyberCash, iKP de IBM, y Anonymous Credit Cards (ACC) de los laboratorios AT & Bell.
- No basados en Criptografía: ISN, Compuserve y FIRST VIRTUAL Holdings Incorporated.

Por desgracia, ninguno de estos protocolos acaba de imponerse, precisamente porque permiten recuperar el anonimato del comprador (por lo tanto su privacidad), y dificultan el rastreo de capital.

3.- Infraestructura de Claves Públicas (PKI).

3.1.- Introducción a PKI.

La criptografía de claves públicas es una tecnología de claves para comercio electrónico, intranets, extranets y otras aplicaciones Web. Sin embargo, para aprovechar al máximo las ventajas

que ofrece la criptografía de claves públicas, se necesita una infraestructura de apoyo. Muchos sistemas operativos incluyen una infraestructura de claves públicas (PKI) nativa que está diseñada desde el principio para aprovechar al máximo la arquitectura de seguridad.

Estudiaremos fundamentos de los sistemas de seguridad de claves públicas, incluidas las ventajas que ofrecen y los componentes necesarios para implementarlos. También se describe cómo los componentes PKI ofrecen los servicios necesarios a la vez que proporcionan interoperabilidad, seguridad, flexibilidad y facilidad de uso.

La criptografía de claves públicas ofrece grandes ventajas de seguridad cuando se implementa correctamente. Al igual que sucede con otras tecnologías, la criptografía de claves públicas requiere una infraestructura para ofrecer sus ventajas. Sin embargo, la infraestructura de claves públicas, o PKI, no es un objeto físico ni un proceso de software; se trata de un conjunto de servicios útiles proporcionados por una colección de componentes interconectados. Estos componentes funcionan conjuntamente para proporcionar servicios de seguridad basada en claves públicas a las aplicaciones y los usuarios.

Se persiguen dos objetivos: explicar la tecnología de claves públicas y sus usos, y describir las características y las ventajas que proporciona la PKI. La comprensión de ambos temas ayudará a decidir dónde se puede utilizar la tecnología PKI para mejorar los procesos empresariales y aumentar la capacidad de administrar transacciones con otras personas de forma segura.

Se aprenderá qué es una infraestructura de claves públicas, qué ventajas deseables puede ofrecer a las operaciones y cómo la PKI ofrece interoperabilidad, seguridad, flexibilidad y facilidad de uso.

El propósito de una infraestructura de claves públicas (PKI) es facilitar a las organizaciones el uso de la criptografía de claves públicas. La criptografía de claves públicas resulta esencial para el comercio electrónico, Internet, intranet y otras aplicaciones que requieran seguridad distribuida; es decir, seguridad en la que los participantes no forman parte de la misma red y no tienen credenciales de seguridad comunes. La criptografía de claves públicas proporciona la forma mejor y más eficaz de llevarlo a cabo.

La criptografía de claves públicas proporciona tres capacidades que son muy valiosas para las organizaciones. En primer lugar, proporciona privacidad para los datos. En segundo lugar, permite la identificación, o autenticación, de usuarios y equipos. Finalmente, proporciona la función sin rechazo, que es la posibilidad de demostrar que alguien ha efectuado una acción concreta. A continuación se ofrecen algunos ejemplos de cómo las organizaciones pueden utilizar estas capacidades:

- Privacidad.

- Cifrar mensajes de correo electrónico que se enviarán a través de Internet con el fin de evitar que se lean durante su transmisión.
- Cifrar el tráfico de red cuando un cliente visita el sitio Web para proteger su información de pedido y los números de tarjeta de crédito.
- Cifrar una sesión de software entre dos interlocutores remotos para evitar que algún intruso escuche.

- Autenticación.

  - Comprobar la identidad de un visitante en una intranet corporativa para permitir que lea determinados archivos.
  - Demostrar a un cliente que se encuentra en su sitio Web y no en un sitio malintencionado que enmascare el suyo.
  - Proporcionar el informe anual de la empresa a los clientes de forma que les asegure que es auténtico.

- Sin rechazo.

  - Crear un registro obligatorio y que no se pueda modificar de los pedidos de compra de un cliente en el sitio Web de comercio electrónico.
  - Firmar contratos electrónicos con vinculación legal.

La operaciones pueden utilizarse para proporcionar tres capacidades (privacidad, autenticación y sin rechazo) que hacen posible la seguridad distribuida y, por tanto, que se distribuyan correctamente el comercio electrónico, las intranets, las extranets y otros procesos empresariales preparados para Web.

- Privacidad.

La privacidad es necesaria para cualquier tipo de empresa, pero resulta de importancia vital para las que utilizan Internet. Internet permite que cualquier usuario del mundo se comunique con otros, pero no proporciona seguridad. Incluso dentro de la red interna de la empresa, si un usuario obtiene acceso físico al medio de red puede interceptar cualquier dato que se transmita a través de la misma.

La criptografía de claves públicas proporciona privacidad mediante el cifrado de los datos, independientemente si están en forma de mensajes de correo electrónico, números de tarjeta de

crédito enviados a través de Internet o tráfico de red. Como las claves públicas pueden publicarse libremente, usuarios completamente desconocidos pueden establecer comunicaciones privadas con sólo recuperar las claves públicas de cada uno y cifrar los datos.

- Autenticación.

Cualquier transacción implica dos participantes, ya sea un cliente y un servidor o un cliente y un proveedor. En muchas transacciones es deseable que una o ambas partes puedan autenticar o comprobar la identidad de la otra parte. Por ejemplo, antes de que un cliente proporcione su número de tarjeta de crédito a un sitio Web de comercio electrónico, querrá saber que no se está comunicando con un impostor. Una forma de que un cliente pueda hacerlo es conseguir que el sitio Web pruebe que tiene la clave privada correcta. Por ejemplo, un explorador Web podría cifrar una parte de la información mediante la clave pública del sitio y pedir al servidor Web que la descifre, con lo que se demostraría que el servidor tiene la clave privada correcta y se probaría su identidad.

La autenticación también puede utilizarse para asegurar a los clientes que se han generado unos determinados datos y que no se han manipulado. La criptografía de claves públicas permite hacerlo mediante una firma digital, un concepto que es una ampliación de la operación de firma de claves públicas descrito anteriormente. Si el usuario A desea firmar digitalmente su informe anual de la organización, en primer lugar genera una firma única del informe mediante un algoritmo denominado hash. Los algoritmos hash están diseñados especialmente para garantizar que incluso el cambio de un único byte del documento generará un hash completamente distinto. A continuación, cifra el informe y el hash con su clave privada. El usuario B (o cualquier otro usuario) puede comprobar el origen y la autenticidad del informe firmado, si, primero lo descifra con la clave pública del usuario A, calcula su propia versión de la firma y la compara con la firma que ha recibido. Si las dos coinciden, se prueban dos cosas: que el informe no se ha manipulado y que procede del usuario A.

- Sin rechazo.

Las organizaciones requieren la capacidad de llegar a acuerdos vinculantes, tanto en el mundo real como en Internet. Los proveedores y los compradores necesitan tener la seguridad de que si llegan a un acuerdo, la otra parte no podrá rechazarlo más adelante. Las firmas digitales en pedidos, contratos y otros acuerdos electrónicos son legalmente vinculantes en varios países, y su aceptación legal está creciendo rápidamente.

3.1.1.- ¿Qué es una infraestructura de claves públicas?

Ahora que ya se sabe que es la criptografía de claves públicas y por qué resulta valiosa, podemos dar la definición de una PKI. Una PKI es el conjunto de servicios del sistema operativo y de aplicaciones que facilitan el uso de la criptografía de claves públicas. Una PKI permite:

- Administrar claves: una PKI hace que sea fácil emitir claves nuevas, revisar o revocar claves existentes y administrar el nivel de confianza adjuntado a las claves de emisores distintos.
- Publicar claves: una PKI ofrece una forma bien definida para que los clientes busquen y recuperen claves públicas y la información acerca de si una clave específica es válida o no. Sin la posibilidad de recuperar claves y averiguar si son válidas, los usuarios no pueden utilizar los servicios de claves públicas.
- Usar claves: una PKI proporciona una forma sencilla para que los usuarios utilicen las claves, no sólo moviéndolas donde se necesiten, sino también proporcionando aplicaciones fáciles de usar que realicen las operaciones criptográficas de claves públicas, lo que permite proporcionar seguridad al correo electrónico, el comercio electrónico y las redes.

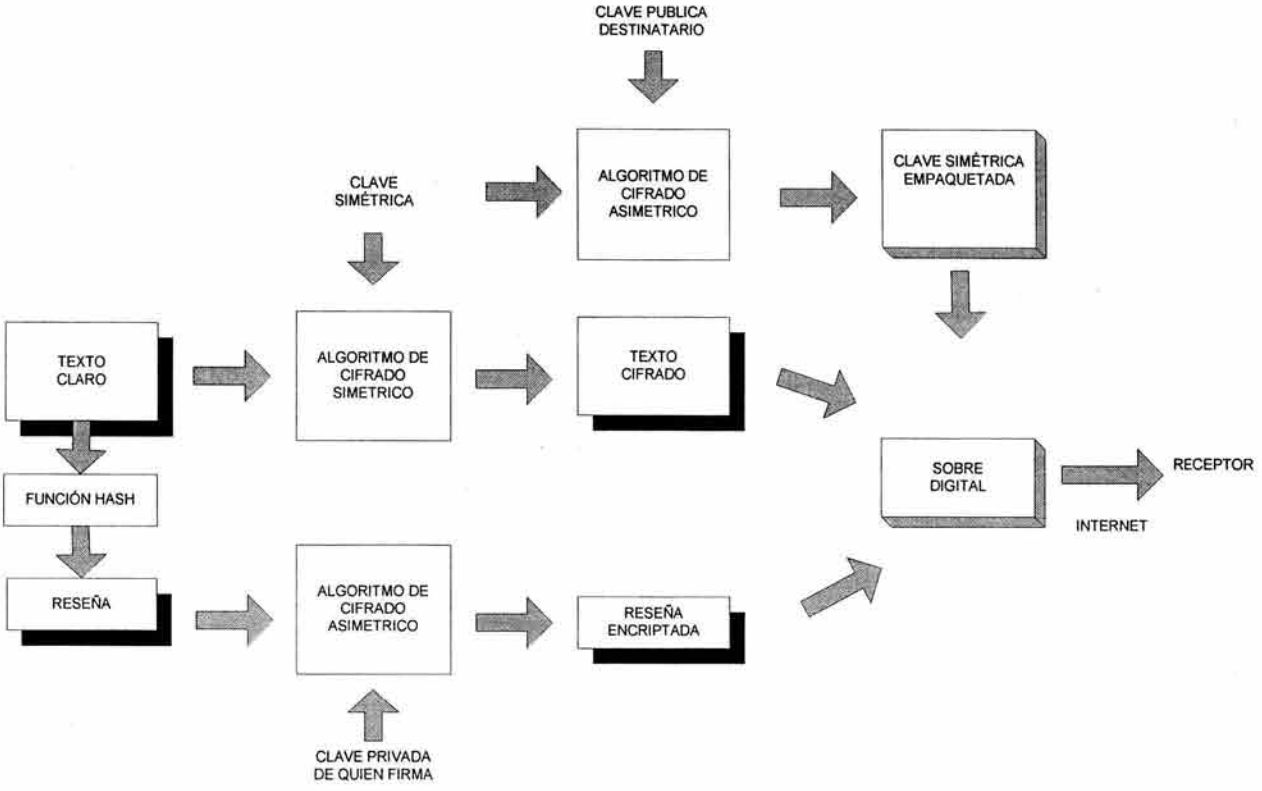
Una vez que existan estas capacidades, los programadores de aplicaciones pueden utilizarlas para crear aplicaciones más seguras. Sin embargo, las aplicaciones que dependen de una forma segura, sencilla y flexible de administrar, publicar y utilizar claves públicas son las que necesitan la PKI.

Una interpretación incorrecta que se ha extendido es pensar en una PKI como en algo físico de hecho es una capacidad, la capacidad de publicar, administrar y utilizar claves públicas de una forma sencilla. Una PKI puede compararse a un sistema de agua de una ciudad. Dicho sistema está compuesto de plantas depuradoras, depósitos, bombas, tuberías principales, etc., así como las cañerías y los tubos de cobre de los hogares. Los distintos objetos del suministro de servicio funcionan conjuntamente para proporcionar a los usuarios la posibilidad de obtener agua según la necesiten. De forma parecida, una PKI consta de un grupo de componentes discretos que funcionan conjuntamente para permitir el uso de claves públicas y criptografía de claves públicas, de un modo integral y transparente.

3.2.- Firmas Digitales.

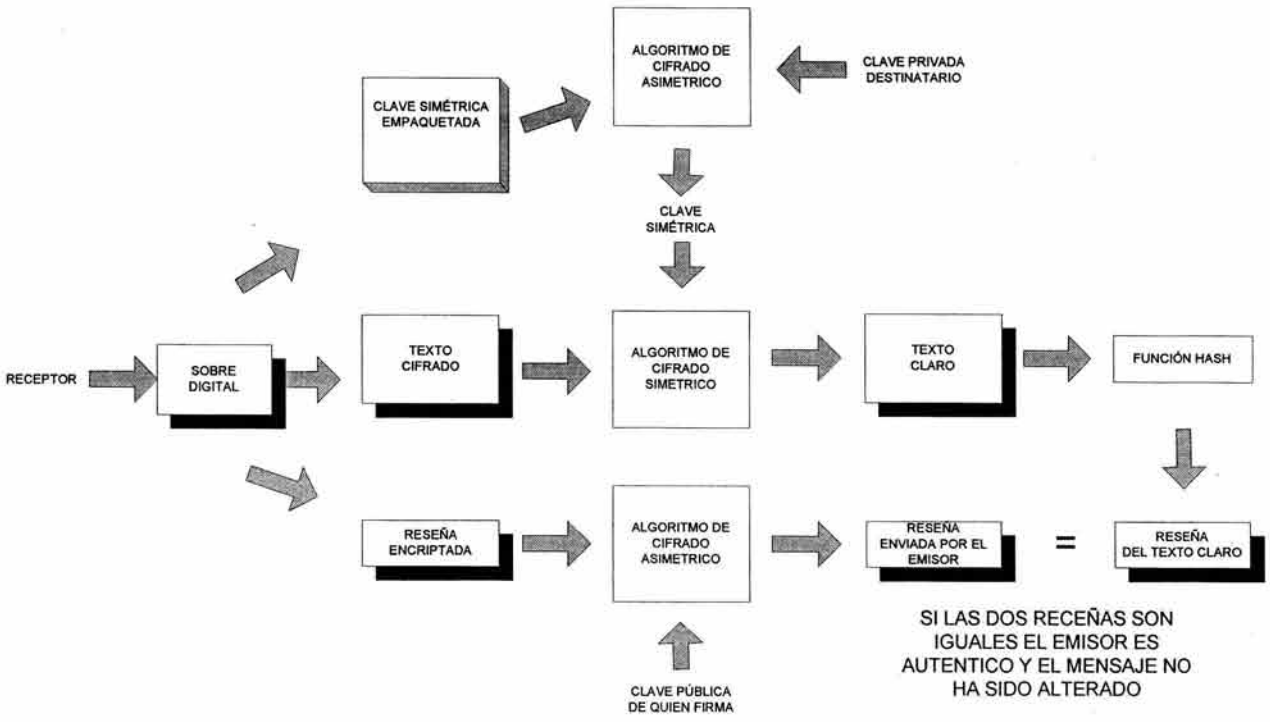
Como se ha visto hasta ahora existe el problema de la autenticación del emisor del mensaje , para lo cual seleccionaremos un algoritmo hash para procesar el mensaje y dar como resultado la reseña, que a continuación se codifica con la clave privada del emisor. Esta reseña cifrada se une al mensaje original cifrado con la clave pública del receptor y se envía al receptor a través de Internet.

Envío del mensaje usando reseña.



El texto cifrado, con la reseña unida a el, viaja a través de Internet hasta el receptor, el receptor nuevamente separa el texto cifrado y la reseña, debido a que el emisor encripta la reseña usando su clave privada el receptor obtendrá su clave pública de un directorio público, con dicha clave descifrara la reseña obteniendo la reseña en claro. Ahora el receptor toma el mensaje cifrado, utilizando la clave privada descifra el mensaje obteniendo el texto en claro. Utilizando el mismo algoritmo hash el receptor genera nuevamente la reseña a partir del texto claro, por último compara las dos reseñas obtenidas, la reseña descifrada enviada por el emisor y la generada a partir del texto claro.

Recepción del mensaje validando la reseña.



Si las dos reseñas corresponden se habrá comprobado dos cosas, que el emisor si es quien dice ser. Esto es seguro porque el receptor utilizo su clave pública para descifrar la reseña cifrada dado que el emisor es el único poseedor de la clave privada podemos garantizar que el emisor es el mismo del quien hemos tomado su clave pública. También sabremos que el mensaje original no fue modificado durante su viaje por Internet debido como se ha mencionado si existiera al menos un bit distinto en el mensaje en claro descifrado la reseña generada seria totalmente distinta y no coincidirían. Esta capacidad de los algoritmos hash para detectar el más pequeño de los cambios en el texto en claro es lo que le da su utilidad para verificar la integridad del mensaje.

Para que el receptor identifique que algoritmo hash se ha usado en la generación de la reseña conjuntamente con la reseña es transmitido adherido un bloque de información que identifica que algoritmo hash produjo la reseña original.

Aún existe un problema, hay una manera de quebrantar este sistema, supongamos que el atacante es bastante inteligente y decide atacar el sistema con algún buen concepto antiguo no lineal. En lugar de encontrar una manera de derrotar el algoritmo hash, o quizás de destruir el cifrado de clave privada, el atacante podría dirigir su atención a otra parte.

La secuencia lógica para verificar la firma sigue un camino como el siguiente:

1. El receptor obtiene la clave pública del emisor de un directorio de claves.
2. El receptor utiliza la clave para descifrar la reseña cifrada.

3. La reseña cifrada se creo utilizando la clave privada del emisor.
4. El emisor tiene la única clave privada que existe.
5. Por consiguiente, si la reseña descifrada y la reseña calculada del texto en claro corresponden, el texto claro procede del emisor que dice ser.

Desafortunadamente el atacante pudo realizar su ataque desde el principio del proceso. Él puede llegar al directorio de claves y sustituir su clave pública por la del verdadero emisor, todo el sistema entra el colapso.

Una vez que el atacante pone en el directorio su clave pública bajo el nombre del verdadero emisor, puede en ese momento enviar el texto que desee, ya que la reseña obtenida y enviada por el falso emisor será formada con la identidad suplantada. El receptor obtendrá del directorio la clave pública de la identidad falsa y sin sospecharlo verificará y obtendrá la reseña que corresponda con la reseña calculada nuevamente del texto en claro, el receptor nunca sospechara de la suplantación de la identidad del emisor ni del contenido del texto en claro.

3.3.- Certificados Digitales.

Un certificado digital simplemente es un documento que dice: "Garantizo que esta clave pública particular está asociada con este usuario en particular".

En su forma más simple esto es a lo que se refieren los certificados digitales. Presentan en lista quien es el propietario de la clave pública y contienen una copia de la clave pública de ese usuario. Entonces, una autoridad de confianza firma la certificación por medio del proceso de firma digital. Se crea un hash para todo certificado y ese hash queda codificado utilizando la clave privada de la autoridad de confianza.

Para verificar la validez de un certificado digital, todo lo que necesita hacer es usar la clave pública de la autoridad de confianza para validar la firma del certificado. Si la verificación del certificado es correcta, se puede estar seguro de la clave privada que está en el certificado pertenece a la persona allí indicada.

Pero, ¿y si el atacante intenta sustituir la clave pública de la autoridad de confianza por una falsa?. La manera como esto se maneja es que la autoridad de confianza creará un certificado con su propia información de identidad, lo mismo que la clave pública de dicha autoridad, y lo firmará. Esto se conoce como certificado autofirmado.

Los programas que usemos o programemos deben manejar con mucho cuidado estos certificados de máximo nivel especial, ya que se basan en la confianza que existe para todos los certificados firmados por esa autoridad. Los navegadores Web ya conocen los certificados autofirmados de muchas de estas autoridades de confianza. Si analizamos la configuración de

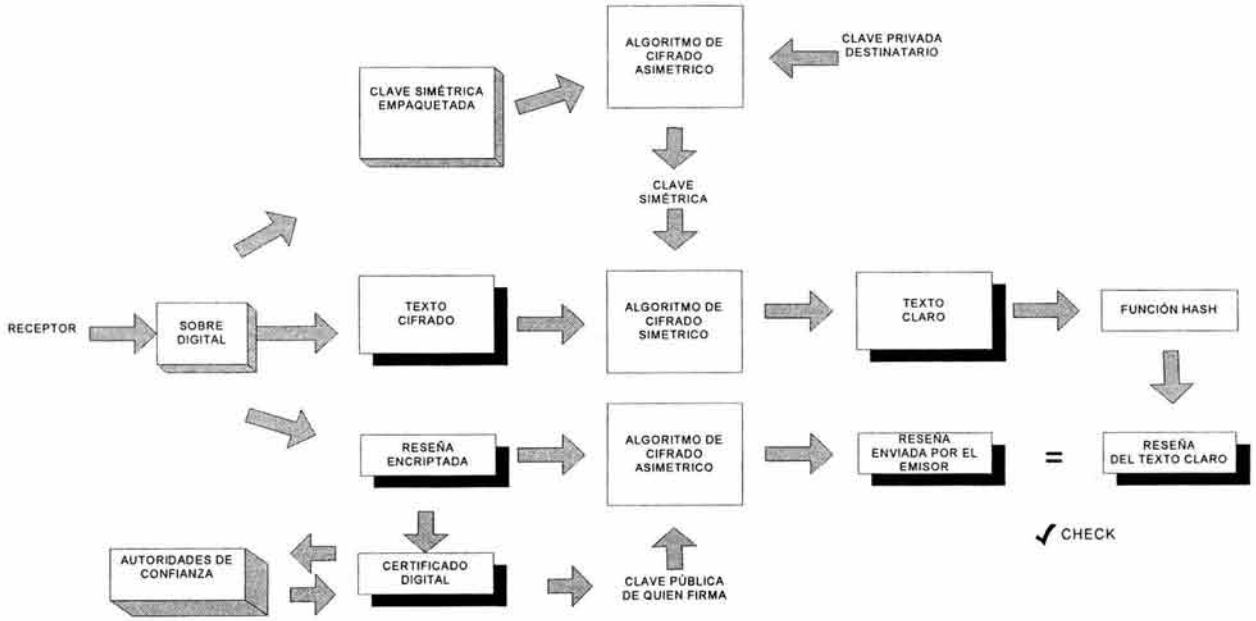
seguridad del algún navegador Web, descubriremos una larga lista de autoridades competentes que identifica el navegador. El fabricante de este último cargó estos certificados después de verificar que la autoridad era real y fiable.

Los certificados digitales son información pública y se quiere que los todos tengan fácil acceso a ellos, pueden existir muchas copias del certificado según sea necesario. Cualquier intento de hacer fraudes con el certificado inmediatamente se detectará cuando éste se utilice. El software que usa el certificado siempre realiza la verificación de la firma en éste y, si ha sido modificado, los valores hash no corresponderán. En este caso el software rehusara usar la clave pública en el certificado y notificará el error.

Con frecuencia los certificados digitales contienen otra información adicional acerca del usuario, como la compañía y quizás la organización dentro de la compañía donde se encuentra este usuario.

Además de la copia de la clave pública del usuario, los certificados también contienen fechas de validez, ya que entran en vigencia en una determinada fecha y expedirán en otra. En su forma habitual, los certificados tienen una vida de pocos años, pero algunos tienen vidas tan cortas que llegan a ser de unas pocas horas en aplicaciones especializadas.

Envío de mensaje validando reseña y usando certificado digital.



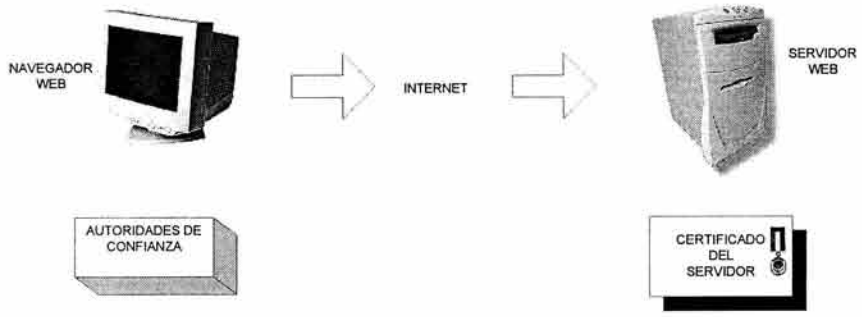
En realidad el emisor incluirá una copia del certificado digital junto con el hash cifrado y el texto en claro. El primer paso es separar los tres componentes; a continuación, el certificado digital se verifica para ver si la firma de este es válida. Al verificar la validez, se comienza por revisar si alguien de confianza para nuestro software firmó el certificado. Esto se hace revisando la firma del certificado, comparándola con la lista de autoridades de confianza que se encuentra en nuestro software.

Si la autoridad que firmó el certificado no ésta incluida en la lista de autoridades de confianza, la verificación se detiene por error. Algunas aplicaciones señalarán el error y preguntarán cómo proceder. Se nos permite pasar por alto el hecho de que no se conoce a quien firma el certificado, continuar. También se le permite agregar el certificado de quien firma a la lista de autoridades de confianza del software.

3.4.- Transacciones Seguras en Internet.

Cuando se navega en la Web, hay ocasiones en las que se necesita capturar datos sensibles como, por ejemplo, la información personal o los números de la tarjeta de crédito. En tales casos es importante que se verifique la autenticidad del servidor al que se le esta enviando la información. Además, es importante que las comunicaciones entre el navegador Web y el servidor Web estén cifradas, de manera que los atacantes no puedan tener acceso a la información mientras esta viajando a través de Internet.

Los servidores Web soportan el protocolo denominado nivel de socket seguro (Segure Socket Layer, SSL) que utiliza técnicas criptográficas.



El servidor Web ha expedido un certificado digital que contiene la identidad del servidor Web, lo mismo que la clave pública de éste. Resulta implícito, aunque no se demuestra, el hecho de que el servidor Web también esta realizando la comparación de la clave privada; además, se observa que el navegador tiene una tabla cargada previamente de autoridades de confianza.

Cuando se usa un SSL del servidor, el navegador Web autentica el servidor Web y se desarrolla un canal de cifrado uno a uno. Sin embargo, este último no autentica el servidor; algo bastante común. Como ejemplo, si usted esta haciendo una compra vía Internet en "Amazon", quiere asegurarse de que esta realmente conectado con "Amazon" y quiere cifrar la información de su tarjeta

de crédito, "Amazon" no necesita autenticar su navegador Web, porque usted se estará identificando directamente, a través de su tarjeta de crédito.

Hay casos en que para que el servidor Web sea necesario autenticar el navegador, como en el caso de que una compañía quiere poner toda su información corporativa en un servidor Web para la misma empresa. En esta situación seria importante garantizar que únicamente los empleados de la empresa tuvieran acceso al servidor, para tal efecto se usa una autenticación SSL del lado del cliente.

El primer paso en el proceso es para que el servidor Web envíe sus certificados digitales al navegador Web. Toda la información en el certificado digital es pública, no importa que este de viaje entre el servidor Web y el navegador.

Entonces, el navegador Web extrae la clave pública del servidor Web del certificado. Antes de que el navegador Web pueda confiar en la clave pública, debe validar el certificado del servidor, revisando si esta firmado por una fuente de la lista de autoridades de confianza. Suponiendo que sea así, el navegador calculará el hash del certificado y lo comparará con el hash que se encuentra en el certificado (descifrado usando la clave pública de la autoridad de confianza). Si los hashes corresponden, el navegador sabe que el certificado no ha sido alterado. A continuación, verificará las fechas de validez codificadas en el certificado para estar seguro que este no ha sido vencido. Suponiendo que no han vencido, hará una verificación más especial asociada con los certificados del servidor Web. Parte de la información de identidad en el certificado es el URL (Uniform Resource Locator) del servidor Web. El navegador hará una revisión extra para garantizar que el nodo que envía la información tenga el mismo URL que esta codificado en la información de identidad. Si todas las verificaciones corresponden, entonces el navegador extraerá la clave pública del servidor Web del certificado del mismo.

Una vez que el navegador tiene la clave pública del servidor, entonces se genera una clave aleatoria de cifrado simétrico que se usará para cifrar la conversación entre el navegador y el servidor. Los algoritmos simétricos de cifrado se utilizan debido a que los cifrados simétricos son rápidos y no expanden los datos durante la operación de cifrado. Para mover la clave de cifrado simétrico hacia el servidor Web, el navegador realiza una operación de empaquetado de clave. La clave simétrica es cifrada utilizando la clave pública del servidor Web, que se extrajo del certificado digital del servidor.

El navegador envía la clave empaquetada al servidor. Como la clave simétrica esta usando la clave pública del servidor, y como este es la única entidad que tiene la clave privada correspondiente, el atacante no puede extraer la clave simétrica.

Ahora que el servidor Web tiene la clave empaquetada, puede usar la clave privada para descifrar la anterior. Esto lleva a la clave simétrica original que genero aleatoriamente el navegador Web.

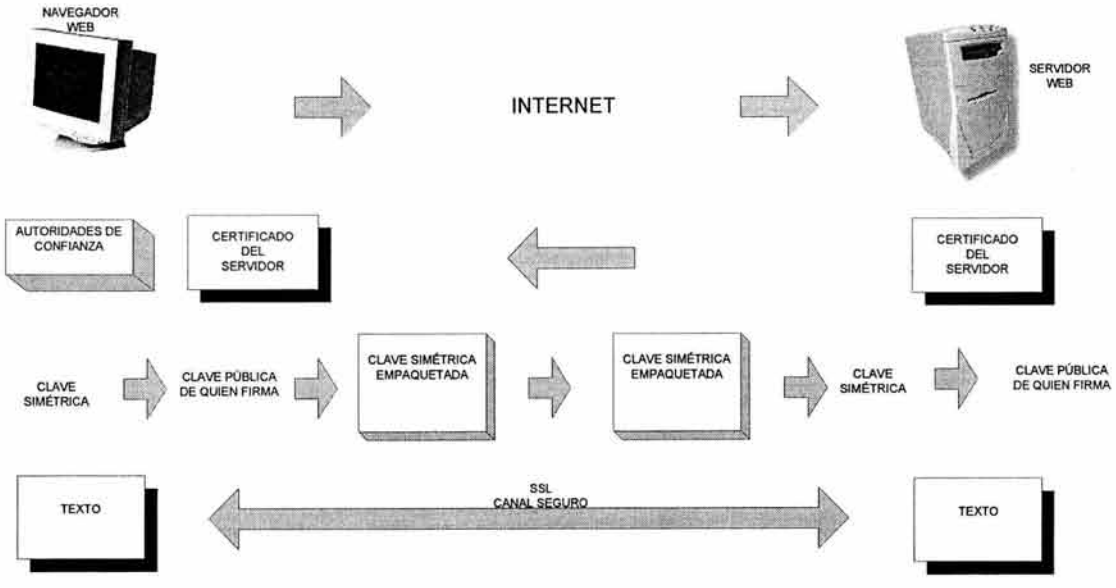
En este punto, tanto el navegador como el servidor tienen una copia de la misma clave de cifrado simétrico. Ambos extremos de la conversación tienen la misma clave simétrica. Ahora pueden comenzar una conversación codificada utilizando el intercambio de la clave simétrica para cifrar y descifrar datos de cada uno.

También el navegador necesita asegurarse de estar hablando con el servidor Web correcto; en otras palabras, debe reconocer la autenticidad del servidor.

La verificación en el procesamiento del certificado donde el navegador Web verifica el URL del servidor, no es una revisión suficiente. Esto se debe a que un sitio Web criminal podría estar fingiendo ser el sitio Web real. En tal caso, todo el tráfico del sitio verdadero se dirigiría hacia el sitio de la estafa. Este tipo de ataque es bastante común en Internet y, con frecuencia, se logra cuando el pirata informático compromete a un servidor DNS (Domain Name Service) y dirige el tráfico hacia el sitio ilegal. Por consiguiente, la simple verificación del URL no es suficiente. Se necesita algo más fuerte.

El navegador Web generó una clave aleatoria de cifrado simétrico y luego la cifró usando la clave pública del servidor Web. El hecho de que el servidor Web pudiera participar en una conversación con el navegador Web, le dice a éste que el servidor ha tenido éxito en descifrar la clave empaquetada y extraer la clave simétrica. El servidor, a su vez, le dice al navegador que esta hablando con el servidor Web real, porque es el único nodo en el universo que tiene la clave privada correspondiente, necesaria para realizar la operación de desempaquetado.

Esquema general de una transacción seguro en Internet usando SSL



3.5.- Identidades de Confianza.

El papel primario de PKI es establecer identidades digitales en las que se pueda confiar. Éstas se pueden usar junto con mecanismos criptográficos para prestar un servicio de seguridad como autenticación, autorización, o validación de una firma digital, para que los usuarios del servicio puedan tener una confianza razonable en que no se les va a engañar.

El problema con la creación de una identidad en que se pueda confiar es encontrar una persona o institución que este preparada para dar fe suficiente de la identidad. Entre las personas dignas de confianza se podría incluir a un amigo de mucho tiempo, el medico de la familia, el juez del tribunal o una institución de confianza. Las instituciones que se encuentran dentro de esta categoría podrían incluir una organización profesional dedicada a acreditar.

Si se puede localizar a un individuo o una identidad que sean dignos de confianza y puedan ofrecer un nivel razonable de credibilidad en el proceso que usan para demostrar la veracidad de la identidad que crean, y el mecanismo de identificación que suministra es razonablemente veraz, entonces puede tener cierto nivel de confianza en el nombre o la identidad que crean. Dentro de este esquema, usted puede confiar o atribuir el mismo grado de confianza que tiene en la identidad original a otra identidad nueva.

Establecer identidades de confianza es un evento que tiene lugar de muchas formas cada día. Un ejemplo seria el uso de una carta de recomendación laboral o las relaciones de negocio de confianza para presentar un tercero que desea también entablar una relación de negocio.

Además de las presentaciones personales, se reconoce que ciertas autoridades tienen el derecho y la capacidad para establecer identidades. Un ejemplo seria la emisión de la credencial de elector federal emitida por el IFE, el pasaporte que por la Secretaria de Relaciones Exteriores que suministra las identidades suficientes que permiten sea reconocido al portador en otras naciones, y el emisor de una tarjeta de crédito que expide una identidad para realizar compras.

Existen algunas expectativas sobre la manera como funcionarán las autoridades que expiden identidades. La identidad emitida por una autoridad es válida en su terreno, pero probablemente no lo sea cuando entre en otros dominios. Un ejemplo seria la tarjeta de crédito que no confiere la suficiente identidad para cruzar la frontera de un país.

Algunas formas de identificación que se expiden se consideran más efectivas o más útiles que otras, usualmente con base en el nivel de confianza que se les otorga la autoridad emisora y la cantidad de pruebas necesarias para establecer dichas identidades. Algunas de ellas, tales como el pasaporte, puede a su vez pueden utilizarse a su vez para establecer la identidad de una persona para solicitar una licencia de conducir.

A partir de la experiencia, esperamos que la mayoría de las identidades sean expedidas por un periodo finito de tiempo y que, usualmente, requieran un proceso de renovación cuando termine el periodo de validez. Esta restricción es en extremo sensible en el terreno de las identidades que no son digitales, evita que la identidad supere el tiempo de vida del poseedor y permite que los derechos asociados se revaliden periódicamente. El proceso de renovación suele ser más simple que el proceso original de solicitud y, con frecuencia, usará la versión anterior que existe en la misma identidad para reducir la prueba para validarla cuando se renueva.

Para los consumidores de una identidad, como en el caso de un proveedor que acepta una tarjeta de crédito, se espera que la autoridad emisora verifique el estado o buen crédito de la identidad, suministrará esa información al consumidor.

3.6.- Autoridades de Certificación.

Dentro de PKI, la autoridad de certificación (Certification Authority, CA) es la autoridad de confianza responsable de crear o certificar identidades. No se trata sólo de poner en funcionamiento una aplicación que pueda generar certificados digitales para que sirvan de identidades electrónicas., esto es comparable con la diferencia que hay entre una oficina de la secretaria de relaciones exteriores que expide pasaportes y un servicio de fotocopiado en color que se puede usar para reproducirlos. El solo hecho que las identidades se puedan crear, no significa que alguien las usará o confiará en ellas.

Es importante notar que la certificación de identidades se refiere al proceso de suministrar la identidad inicial (real) del solicitante de una identidad digital. En la mayoría de las implementaciones de CA, esta función va separada de la Autoridad de Registro (Registration Authority, RA).

La CA desarrolla procedimientos que verifican la identidad de un solicitante que se registra para un certificado y expide un certificado digital que se puede usar como prueba de esa identidad. Las identidades se expiden por un periodo específico y la CA tiene la capacidad de revocarlo y avisar a los usuarios cuando se presenta esa situación.

Las identidades que se usan en la vida diaria han tenido precedentes históricos por parte de las identidades o personas en quienes se confía para expedirlas y los procesos que utilizan. Sin embargo, surgen preguntas acerca de las CA que crean identidades que se van usar en un mundo electrónico. ¿En quien se confiará para operar una CA? ¿Qué tan ampliamente se utilizarán las identidades? ¿Qué proceso y pruebas se utilizarán para establecer las identidades? ¿Qué mecanismos se pueden suministrar en una red digital para brindar una identidad razonablemente segura?.

En primer lugar se debe buscar para encontrar operadores de una CA que generen confianza y los cuales son las instituciones competentes mencionadas previamente que siempre han conocido, y

entre las que se encuentran los bancos, las compañías emisoras de tarjetas de crédito, algunas entidades gubernamentales tales como la Secretaria de Hacienda o la Secretaria de Relaciones Exteriores o posiblemente un grupo de acreditación de profesionales como el de los médicos o abogados, etc.

Además de las entidades de confianza tradicionales, has surgido nuevas formas de autoridades que expiden identidades. Muchas de las CA públicas han tenido que establecer una reputación de confianza sin contar con un nombre o una reputación comercial existente previamente. Las CA públicas han establecido una reputación confiable al construir instalaciones muy seguras y tener especial cuidado con el establecimiento y la auditoria de los procedimientos y el personal operativo.

3.7.- Certificado Digital.

Un certificado digital forma una asociación entre una identidad y la pareja de claves pública y privada que posee el tenedor de la identidad.

Luego de que se ha concluido que una autoridad emisora en particular puede ser de confianza para establecer una identidad con un propósito especifico, el siguiente paso es producir un documento que se pueda usar para certificar que ha recibido una identidad validad. Un certificado digital es la forma electrónica de este documento, para uso electrónico, se necesita producir un documento o certificado digital que suministra suficiente información para permitir que un tercero se sienta satisfecho de que se es el poseedor correcto de la identidad.

La forma particular del certificado digital más común es el X.509, que recibe su nombre por el estándar que define su contenido y uso (ver apéndice para detalle de la estructura).

Para propósitos de este estudio, no vamos a referir al individuo u otra entidad identificada por el certificado como el propietario del certificado. La entidad que lo emite y desea usarlo para comprobar una identidad es el usuario del certificado que, en ocasiones, se conoce más formalmente como la parte confiante.

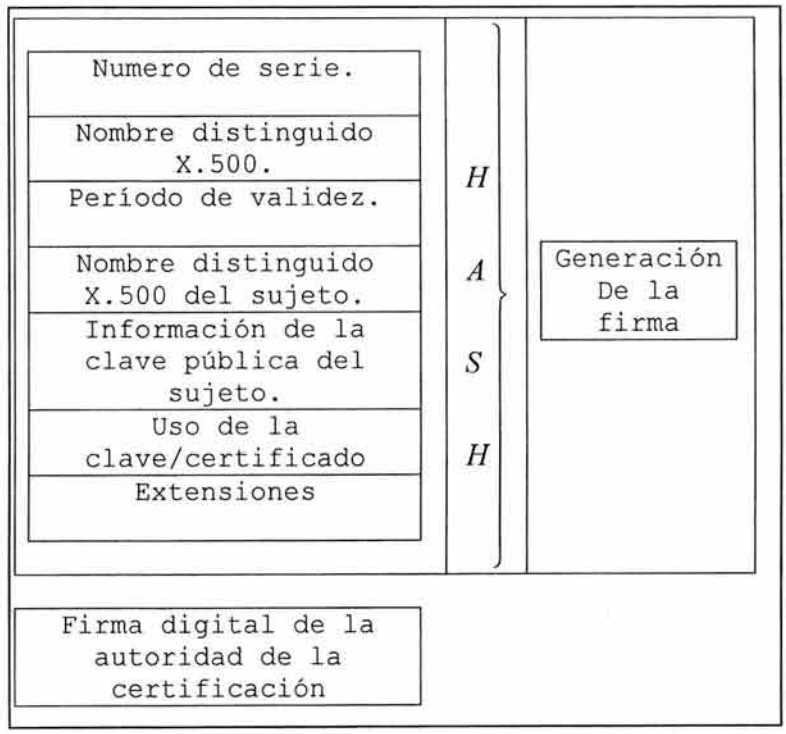
Los campos en un certificado digital X.509 reflejan exactamente el contenido de un pasaporte legal con esta imagen de alto nivel. Una vez que comenzamos a mirar el contenido en detalle (ver apéndice), se describirán los campos que son únicos para la especificación y los requerimientos de un certificado digital basado en PKI.

La información en el navegador Web para un certificado X.509 muestra los siguientes datos:

- Sujeto: Los nombres y apellidos del individuo o entidad que se van a identificar con el certificado. Se puede incluir información de identificación adicional en el nombre del sujeto o en otros campos específicos del certificado.

- Clave pública: La clave pública corresponde a la clave privada del sujeto.
- Expedidor: Identifica la fuente de confianza que generó y firmó el certificado.
- Número de serie: Un número de serie permite que este certificado se identifique como único, incluso si se omite otro certificado con la misma información.
- Válido desde: La fecha de iniciación especifica el tiempo desde cuando el certificado se puede utilizar.
- Válido hasta: La fecha final determina el tiempo máximo hasta cuando el certificado se puede utilizar. Junto con la fecha inicial forma el periodo de validez para el certificado.
- Uso de clave/certificado: La información de uso de clave/certificado describe los valores válidos para la pareja de claves pública/privada del sujeto.
- Firma digital: La firma digital del expedidor, que se generó utilizando su clave privada, verifica la identidad del sujeto. El valor hash que se utiliza para construir la firma de la CA en el certificado permite verificar el contenido del mismo no haya sido adulterado.

Los nombres del sujeto y el expedidor están definidos utilizando convenciones que se definen en el estándar X.500. Esto permite establecer un nombre único mediante la definición de un concepto conocido como un nombre distinguido. Un nombre distinguido puede incluir el uso de información diferenciadora, tal como la organización para la que el individuo trabaja, la dirección en donde se le localiza o la manera de contactarlo a través de Internet. La información de un certificado se representa gráficamente a continuación.



3.7.1.- Terminología.

Definiremos algunos términos que identifican a los participantes en el uso de los certificados.

- Una identidad destino es una persona u objeto que se identifica mediante un certificado, o un usuario de un certificado que identifica otra entidad destino. Una entidad destino incluye personas, nodos de red (tales como servidores Web, barreras de seguridad o enruteadores), programas ejecutables y casi cualquier objeto que tenga una identidad única y al que se le pueda asignar un certificado.
- El propietario del certificado es la entidad destino que se identifica en el campo del sujeto del certificado y, en ocasiones, se puede identificar como el sujeto del certificado.
- El usuario del certificado es una entidad destino que recibe un certificado y lo utiliza con el fin de establecer la identidad de un propietario de certificado. Como el usuario del certificado se basa en la identidad establecida en el mismo, en ocasiones se conoce como la parte confiante.

3.7.2.- Tipos y Atributos de los Certificados.

Además de la información que se utiliza para establecer directamente la identidad del propietario del certificado, se incluyen otros atributos o información con destino a una aplicación en particular.

Los certificados que se utilizan para identificar a los emisores de correo electrónico incluirán la dirección electrónica de dicho emisor; usualmente en Internet tomará la forma de una dirección para correo electrónico, esto permite comparar directamente la identidad en el certificado con la dirección de origen de correo electrónico en el mensaje.

Las computadoras o los componentes de red, como los enrutadores, también se debe de identificar, pero tienen requerimientos de nombre o de identidad diferentes de los usuarios humanos. Por ejemplo, cuando se crean enlaces de red seguros, utilizando tecnología de una red privada virtual (VPN), resulta esencial autenticar la identidad de las máquinas que se comunican. En este caso, la identidad suele ser las direcciones de red tcp/ip que las máquinas que trabaja en cooperación utilizarán en los paquetes de red.

Otra información podría identificar la política que se usó para crear el certificado o la política que describe donde y cuando se puede usar el certificado. Por ejemplo, el certificado sólo puede ser válido en una red segura o cifrando información altamente sensible en una red que no es segura.

3.7.3.- Aspectos de Seguridad de los Certificados.

Los certificados tienen varias características de seguridad que resultan interesantes.

En si mismo, el certificado es una estructura pública que contiene información pública, incluida la clave pública que se utiliza en el proceso de validar la identidad. Como resultado, el certificado se puede publicar libremente y se puede acceder a él en la forma más amplia posible.

Permitir un acceso amplio al certificado reduce los aspectos asociados con la distribución del certificado a los usuarios que necesitan acceder a él. Sin embargo, cuando se planea la infraestructura de un certificado, se debe considerar el alcance de uso de éste.

Los aspectos de privacidad de la información que se incluyen en el certificado se deben considerar con especial cuidado. El uso de algunos tipos de información de identificación puede, de hecho, restringir en dónde se debe de publicar un certificado, se debe analizar bastante cuál es la información que se almacena.

Los certificados son estructuras autoprotegidas. Si el contenido de un certificado se modifica, o si se ha usado una fuente no confiable para expedir el certificado, normalmente el usuario respectivo recibirá la advertencia por parte del software de su aplicación. Como resultado, los certificados se pueden enviar hacia ambientes hostiles, sin brindarles protección adicional y sin la necesidad se contar con un repositorio de alta seguridad.

Pueden existir otras razones para proteger un banco repositorio para ofrecer seguridad o privacidad, que incluye evitar la eliminación de las Listas de revocación de certificados (Certificate Renovation Lists, CRL).

La firma, por parte de una fuente autorizada, permite que el usuario del certificado confié en la identidad del propietario de tal certificado. Además, la presencia de la clave pública permite que se verifique que el propietario posee la clave privada correspondiente.

3.8.- Componentes de la Infraestructura de Claves Públicas.

3.8.1.- Autoridad de Certificación.

La CA es responsable de establecer identidades y crear los certificados digitales que forman la asociación entre una identidad y una pareja de claves pública/privada. A nivel mecánico, comprende el conjunto de componentes y servicio de software y hardware que se usan durante este proceso. También incluyen el personal, los procedimientos de operación, el ambiente y las directivas que definen cómo se establecen las identidades y cuál es la forma de certificado digital que se expide.

La CA está integrada por varios subcomponentes o servicios distintos que se tratan a continuación. Los más importantes incluyen un CS, una RA y un repositorio de certificados.

Una CA define las reglas que permiten que los subscriptores y usuarios del certificado se sientan satisfechos en cuanto a que las identidades que certifica están disponibles para los propósitos establecidos y sean confiables. Las reglas que describen la manera como las diferentes facetas de una CA están limitadas y operan, se definen en un documento llamado Declaración de Prácticas de Certificación: Una Declaración de Prácticas de Certificación (Certification Practices Statement, CPS) para la CA que expidió el certificado, debe estar disponible para el usuario del certificado. Si no se encuentra disponible CPS, esto puede producir una duda razonable sobre la veracidad de la CA y reducir la confianza en las entidades que lo expiden.

3.8.2.- Autoridad de Registro.

La RA es responsable del registro y la autenticación inicial de los suscriptores, que son los usuarios a quienes se les expide un certificado después de que les ha sido aprobada una solicitud de registro. Estas interacciones también pueden incluir la revocación del certificado y los demás servicios que los subscriptores necesitan cuando interactúan con PKI. Una RA y sus interfaces se pueden implementar como parte de un servidor de certificados o pueden formar un componente independiente.

Una persona puede realizar las obligaciones de una RA. Todo el proceso de validación de la identidad se puede desarrollar como un conjunto de procedimientos manuales. El envío de un certificado solicitado por parte de un individuo calificado y autenticado es una culminación válida de las responsabilidades de la RA.

Las normas comerciales que controlan la generación de certificados y el registro del suscriptor del certificado varían ampliamente, pero se deben describir en la CPS para la CA. Los administradores de seguridad y los asesores legales dentro de las empresas que utilizarán los certificados expedidos por la CA deberán revisar los aspectos de la CPS

3.8.3.- Servidor de Certificado.

El servidor de certificado es el componente de una Autoridad de Certificación en el cual muchas personas piensan cuando utilizan el término CA. Es la máquina o servicio responsable de expedir los certificados con base en la información suministrada durante el proceso de registro. La clave pública del usuario se combina con otra información de identificación y la estructura del certificado resultante se firma bajo la clave privada de la CA.

Los aspectos de una CPS que controla a un servidor de certificados incluyen descripciones de la manera como las claves son seguras para la CA, información que se pondrá en el certificado y con qué frecuencia se genera la información de revocación.

3.8.4.- Repositorio de Certificados.

Los certificados y las claves públicas correspondientes que se necesiten deben estar disponibles para el público antes de que puedan entrar en funcionamiento. Si un mecanismo de publicación cuenta con apoyo para la difusión de certificados públicos, un repositorio será el sitio usual para publicar los certificados. Estos repositorios, que usualmente utilizan como parte de PKI, son directos, ocasionalmente directorios X.500, pero lo más común es que sean directorios LDAP (Lightweight Directory Access Protocol). Como se verá, LDAP en realidad es una descripción del método de acceso y el protocolo que se utilizan para localizar información en un directorio. Un directorio que cumpla las condiciones de LDAP se podría implementar como cualquier otro archivo plano en una base de datos relacional, e, incluso, en un directorio X.500, considerando que cumple los requerimientos LDAP.

3.8.5.- Validación del Certificado.

Los usuarios del certificado necesitan validar los certificados que reciben. La validación de un certificado individual requiere:

- Verificación de la firma del firmante del certificado.
- Garantizar que el certificado está vigente, comprobando su periodo de validez.
- Verificar el cumplimiento entre el uso que se le pretende dar al certificado y cualquier directiva de restricciones específicas para el certificado por la CA.
- Verificar que el certificado no hay sido revocado (cancelado) por la CA.

El proceso de validar las cadenas de certificados suele ser complejo, en particular cuando se usa a través de empresas. Su puede realizar en un ambiente de clientes, por un servicio que el cliente puede utilizar para realizar la misma tarea.

3.8.6.- Servicio de Recuperación de Claves.

Las parejas de clave pública/privada pueden generarse localmente en un almacenamiento de claves, dentro de una aplicación como un navegador o en un dispositivo físico al estilo de una tarjeta inteligente. Alternativamente, la pareja de claves se puede crear en un servidor central de generación de claves.

En cualquier caso, existe la necesidad de suministrar un mecanismo que permita almacenar las claves de cifrado y recuperarlas en caso de perdida. Otros casos que exigen se presentan cuando las

claves de cifrado están penalizadas por las agencias legales. Esta situación permite la operación continuada de los procesos de cifrado, incluso si un desastre afecta al poseedor de las claves. Si por ejemplo, se ha cifrado información importante para una empresa y por algún motivo el tenedor de las claves pública/privada las pierde, la empresa querrá usar el servicio de recuperación de claves para recuperar la información vital.

3.8.7.- Servidor de Hora.

La hora segura, como las firmas digitales, es una construcción electrónica que permite establecer un registro de tiempo verificable. Se necesita un reloj preciso que aumente de manera monotónica, en donde el timbre se transfiera de manera segura, de modo que no se pueda interceptar o reemplazar. Finalmente, el registro de tiempo está firmado de manera que se puede verificar el emisor del valor de la hora confiable.

Varias operaciones se benefician del concepto de hora segura. Ellas incluyen archivos de registro de auditoria seguros, sistemas de reconocimiento de recibo, sistemas de flujo de trabajo y documentos electrónicos, incluidos los contratos. Para ofrecer un registro de tiempo que se pueda autenticar en el futuro, se necesita un reloj seguro que suministre de alguna manera confiable, con características como la monotonicidad (el tiempo sólo debe ir hacia delante). Además, debe estar en capacidad de demostrar que el documento al cual se anexó el indicador de tiempo no ha cambiado (lo cual normalmente también requiere el uso de una firma digital).

Si se incluye un servidor de hora como parte de PKI, se suministran registros de tiempo digitales para uso de los servicios o aplicaciones en niveles. El valor de una empresa de servicio para respaldar aplicaciones como la verificación de contratos, dependerá de la preparación que tenga un tercero involucrado para confiar en sus registros de hora. El uso de un tercer proveedor de registros de hora confiable puede ser necesario en algunos casos.

3.8.8.- Servidor de Firmas.

Las firmas digitales se pueden generar mediante aplicaciones que administran documentos o transacciones a los cuales se aplica una firma. Si la aplicación no presta dicho soporte, o si se prefiere un servicio central de firma y verificación, se puede usar un servidor separado para realizar esta función para las transacciones del usuario. Un servidor de firmas también puede formar la base de un servicio de terceros como el que suministran las notarias digitales.

3.9.- Administración del Ciclo de Vida de la Clave y el Certificado.

La administración del ciclo de vida de la clave y el certificado se refiere a las operaciones necesarias para administrar las claves y los certificados desde su creación hasta su retiro.

En principio, la creación de la pareja de claves pública/privada se necesita para enlazarlas al certificado que se utiliza para establecer la identidad de una entidad destino. La pareja de claves cuenta con otra información de identificación como parte del proceso de registro con una CA para un certificado. La CA expide el certificado después de que ha verificado la información suministrada por usted para establecer su identidad.

Antes de que los certificados se puedan usar para establecer identidades se deben distribuir en diferentes formas a los usuarios. El propietario del certificado puede transmitirlo, o la CA lo puede almacenar en un repositorio, para su recuperación posterior.

El certificado que se expide para usted tiene un tiempo de vida limitado. Cuando llega la fecha de terminación se considera que ha expirado y se debe reexpedir un certificado. Como parte del proceso de reexpedición para una identidad particular, parte de la información puede haber cambiado. Alternativamente, como las claves tienen una expectativa de vida útil que se basa en su longitud, tanto el certificado como la clave se deben actualizar.

Puede ser necesario que la CA que expidió un certificado para una entidad destino lo invalide. En este caso, el certificado debe ser revocado y la CA debe publicar la información de esta revocación.

Como las claves se pueden perder y es necesario recuperarlas para descifrar la información previamente cifrada, se deben permitir procesos de almacenamiento de claves y recuperación de claves.

3.10.- Protocolos Basados en PKI.

El nivel de socket seguro (SSL) es el protocolo más ampliamente conocido y adoptado que se basa en PKI. Sin embargo, antes que existiera el SSL, existía el Diffie-Hellman. Recientemente, han ganado popularidad los protocolos IPSec y S/MIME. Comenzamos con Diffie-Hellman, dado que ofrece los puntos de sostén fundamentales para la mayor parte de los protocolos PKI.

3.10.1.- Intercambio de Claves Diffie-Hellman.

En 1976, Whitfield Diffie y Martin Hellam, publicaron el algoritmo Diffie-Hellman, el primero de clave pública, el cual permitía que dos partes computaran un secreto compartido. Su uso es muy amplio, debido a sus características de venta principal: no necesita cifrado, por lo que sus costos de implantación son bajos. Uno de los usos más comunes de un secreto compartido Diffie-Hellam es

como base de claves de cifrado adicionales que las partes usarán para proteger la información que intercambian.

El concepto fundamental subyacente del algoritmo Diffie-Hellman es la dificultad matemática de calcular logaritmos discretos en un campo finito. El algoritmo comienza con dos valores, p y q, que no son secretos, pero que tienen atributos particulares. El parámetro *p* puede ser un número primo. El parámetro *g* (llamado generador) debe ser un número entero. Los parámetros *p* y *g* también deben contar con la siguiente propiedad: para cada número, *n*, entre 1 y *p*-1 incluso, existe un valor g^k tal que $n=g^k \bmod p$. A continuación el procedimiento:

1. Las dos partes, a las que llamaremos lado 1 y lado 2, eligen los parámetros *p* y *g* que usarán.
2. Cada lado genera un valor aleatorio, digamos valor1 y valor2. Ambos números aleatorios son menores que *p*-2. Cada lado conserva su valor aleatorio privado.
3. Ambos lados calculan valores públicos basados en *g*, *p* y sus respectivos valores privados; $g^{valor1} \bmod p$ para el lado 1, y $g^{valor2} \bmod p$ para el lado 2.
4. Cada lado envía al otro su valor público.
5. El lado 1 calcula $n=(g^{valor1})^{valor2} \bmod p$. El lado 2 calcula $n=(g^{valor2})^{valor1} \bmod p$. Debido a las propiedades básicas de *g*, ahora tienen un valor compartido, *n*.

Una deficiencia del algoritmo Diffie-Hellman es que es vulnerable a los ataques de un intruso en el medio, porque no autentica a los dos lados en el intercambio. Esto significa que un tercero podrá interceptar la comunicación y manipular los intercambios, de modo que los participantes originales aún creen que se están comunicando directamente. En cambio, el tercero negocia valores compartidos separados con los dos participantes. Cuando las partes originales usan el valor compartido que han calculado, como en una clave de cifrado, el intruso puede descifrar, ver copiar o alterar el tráfico, y volver a cifrarlo antes de enviarlo. Ni siquiera los participantes originales podrán detectar al intruso.

Una serie de variaciones fortalecen al algoritmo Diffie-Hellman mediante la adición de características para prevenir los ataques. Un ejemplo es un protocolo Diffie-Hellman autenticado, como el denominado protocolo de estación-a-estación, mismo; sin embargo, las dos partes también firman de manera digital los dos valores públicos $g^{valor1} \bmod p$ y $g^{valor2} \bmod p$, y después intercambian las firmas. Cada lado verifica la forma digital para garantizar que no ha habido interceptación de las comunicaciones. Como un intruso no tiene la clave privada de ninguno de los lados, cualquier firma que trate de insertar no se verificará.

3.10.2.- Capa de Sockets Seguros (SSL).

El protocolo SSL (Secure Sockets Layer) permite establecer conexiones seguras a través de Internet, de forma sencilla y transparente. La idea consiste en interponer una fase de codificación de los mensajes antes de enviarlos por la red. Una vez que se ha establecido la comunicación, cuando una aplicación quiere enviar información a otra computadora, la capa SSL la recoge y la codifica, para luego enviarla a su destino a través de la red. Análogamente, el módulo SSL de la otra computadora se encarga de decodificar los mensajes y se los pasa como texto plano a la aplicación destino.

Una comunicación SSL consta fundamentalmente de dos fases:

- Fase de saludo (handshaking): Consiste básicamente en una identificación mutua de los interlocutores, para la cual se emplea habitualmente los certificados X.509. Tras el intercambio de claves públicas, los dos sistemas escogen una clave de sesión, de tipo simétrico.
- Fase de comunicación: En esta fase se produce el auténtico intercambio de información, que se codifica mediante la clave de sesión acordada en la fase de saludo.

Cada sesión de SSL lleva asociado un identificador único que evita la posibilidad de que un atacante escuche la red y repita exactamente lo mismo que ha oído, aún sin saber lo que significa, para engañar a uno de los interlocutores.

Las ventajas de este protocolo son evidentes, ya que liberan a las aplicaciones de llevar a cabo las operaciones criptográficas antes de enviar la información, y su transparencia permite usarlo de manera inmediata sin modificar apenas los programas ya existentes. Desde hace tiempo los principales navegadores de Internet incorporaban un módulo SSL, que se activa de forma automática cuando es necesario. Desgraciadamente las versiones de exportación tanto de Netscape como Internet Explorer trabajan con claves de sesión de 40 bits, que pueden ser descifradas en cuestión de pocas horas por cualquier PC más o menos potente, por lo que en ningún caso pueden ser merecedoras de nuestra confianza. Afortunadamente, este problema se puede subsanar con utilidades comerciales adicionales (plug-ins), que restauran su total funcionalidad a estos programas.

La capa de sockets seguros (Segure Sockets Layer, SSL) establece un canal seguro del nivel de trasporte entre dos partes. El SSL ofrece privacidad de las comunicaciones a través del cifrado simétrico, e integridad mediante los códigos de autenticación de mensajes (Message Authentication Codes, MAC). EL SSL usa PKI básicamente para autenticar a las partes durante el establecimiento de la conexión. Tanto el protocolo Seguridad del nivel de transporte (Transport-Level Security, TLS) como el protocolo, y seguridad a nivel de transporte inalámbrico (Wireless Transsport-Level Security, WTLS) del Forum del protocolo del acceso inalámbrico (Wireless Access Protocol, WAP) son descendientes directos del SSL.

Netscape Communications Corporation desarrolló originalmente el SSL, a mediados de la década de 1990. La versión 2 de SSL fue la primera que se usó ampliamente; la versión 3 mejoró la

eficiencia, la flexibilidad y el conjunto de características del protocolo. El TLS es una forma superior del SSLv3, con funciones criptográficas fortalecidas.

El SSL es un protocolo con dos capas que opera sobre un protocolo de transporte confiable, usualmente TCP. Para dar una idea de la manera que estos protocolos funcionan, describiremos el SSL versión 3 (SSLv3).

La capa más baja del SSLv3 está integrada por el protocolo SSL Record. Esta capa encapsula los datos del protocolo de niveles más altos. Los datos del protocolo están protegidos con el cifrado y los algoritmos MAC negociados en el protocolo Forma de Contacto SSL. Un código de autenticación de mensaje usa funciones criptográficas que brindan un mecanismo para detectar cambios en el mensaje. El algoritmo MAC del SSLv3 implica unir el hash de una clave junto con los datos del protocolo y el número de secuencia del mensaje.) Esta combinación de cifrado y algoritmo MAC se conoce como especificación de cifrado corriente. Las transformaciones del protocolo de registro incluyen un número de secuencia, de modo que se pueden detectar mensajes compresión, alterados o reenviados. Opcionalmente, protocolo de registro realiza la compresión, si se negoció un algoritmo de compresión.

La capa más alta del SSLv3 consta de los mensajes que transporta el nivel de registro, los cuales incluyen el protocolo Especificación de cambio de cifrado de trama (Change Cipher Spec), el protocolo Alarma (Alert) y el protocolo Toma de contacto SSL (Handshake). Antes de sumergirse en este último, trataremos los otros mensajes de nivel más alto.

Partes importantes de SSL:

- El protocolo Especificación de Cambio de Cifrado indica un cambio en las cifras usadas.

Consta de un mensaje único que ésta cifrado con la especificación de cifrado corriente. Tanto el cliente como el servidor envían el mensaje de especificación de cambio de cifrado para indicar que van a comenzar a usar nuevas especificaciones de cifrado y claves.

- El protocolo Alarma transfiere mensajes acerca de un evento, incluidas la severidad y la descripción del mismo.

Estos eventos son, básicamente, condiciones de error, como un MAC malo, un certificado que ha expirado un parámetro ilegal. El protocolo de Alarma también se usa para compartir información acerca de una terminación de conexión planeada.

- El protocolo Toma de Contacto en la manera como un cliente y un servidor negocian los parámetros de seguridad que usaran para el canal seguro.

Estos parámetros incluyen la versión de protocolo, el algoritmo de cifrado y las claves criptográficas que usarán. Además, el cliente autentica el servidor y, opcionalmente, el servidor autentica el cliente. PKI entra en juego en la autenticación cliente-servidor.

- El cliente inicia la conexión SSL enviando al servidor un mensaje de saludo del cliente (Client Hello Message).

Dicho mensaje contiene las capacidades de seguridad que desea el cliente, incluidas la versión del protocolo, las cifras que soporta en orden de preferencia y cualquier método de compresión que soporte. Además, el cliente genera un numero aleatorio y lo envía en el mensaje de saludo en el ID de una sesión.

- El servidor responde con su mensaje de saludo (Server Hello Message).

Si soporta una versión del protocolo, un cifrado y un método de compresión en la lista del cliente, esos datos se indican en el mensaje mencionado. Si el cliente y el servidor carecen de versiones del protocolo, cifras o métodos de compresión comunes, la conexión falla. Si el cliente y el servidor tienen una sesión preexistente con este ID, el servidor regresa el valor ID de la sesión del cliente en el saludo del servidor. Si no es así, el servidor responde con un valor aleatorio diferente, generado por el servidor, que indica una nueva sesión.

- El servidor envía su certificado al cliente.

Casi siempre es un certificado X509v3 con el certificado del servidor. El cliente debe verificar la firma del certificado, revisar que una Autoridad de Certificación en que el cliente confié haya firmado el certificado, y garantizar que el certificado esté dentro de su periodo de validez. Observe que a lo largo de la verificación de la firma también se necesita verificar la cadena de certificados firmante, lo mismo que hacer una revisión para comprobar que no se encuentra en la última lista de revocación de certificados (CRL) de la CA. En realidad, el cliente casi siempre es un navegador Web, y los navegadores no son consistentes en el nivel de revisión que implantan. En la actualidad, ningún navegador verifica los certificados con las CRL; sin embargo, los navegadores confrontan los certificados con un conjunto de firmantes de confianza. Si el firmante del certificado del servidor no está en la lista de confianza, el usuario verá una ventana emergente o de aparición instantánea, que señala que el certificado no es de confianza. Verificar el certificado requerirá que el usuario rastree el

certificado de la CA y lo instale en la lista de firmantes de confianza de su navegador. La mayoría de los usuarios no tienen ninguna idea de cómo hacer esto y, por lo general, acepta el certificado de todas maneras.

- El servidor solicita el certificado del cliente.

Esta especificación del protocolo hace que sea opcional y, de hecho, permite tres opciones: el servidor no solicita el certificado del cliente, el servidor requiere un certificado del cliente o la conexión falla, o el servidor solicita un certificado del cliente, pero continúa la configuración de la conexión si el cliente no suministra uno. Para este ejemplo, supondremos que el servidor requiere un certificado del cliente.

- El servidor envía el mensaje de "saludo hecho" (Server Hello Done Message) para indicar que esta listo con su lado de la negociación inicial.
- El cliente envía su certificado al servidor.
- El servidor verifica el certificado del cliente.

La naturaleza exacta de esta validación corresponde al implantador. Nosotros supondremos que el servidor valida la firma, verifica el periodo de validez, garantiza que el certificado este firmado por una de las CA de confianza del servidor, y que el certificado no aparece en la CRL más reciente de la CA.

- El cliente genera un valor secreto (usando un generador aleatorio criptográficamente seguro) pre-maestro y lo cifra con la clave pública del certificado del servidor.

Después envía este valor cifrado al servidor en un mensaje de intercambio de clave del cliente. El protocolo SSLv3 también permite usar otros métodos para intercambiar el secreto pre-maestro, incluidos Diffie-Hellman y Fortalezza (sistemas de cifrado basados en hardware, usados por el gobierno de los Estados Unidos).

- El servidor descifra el secreto pre-maestro.

La función de conversión usa los hashes MD5 y SHA-1 y los valores aleatorios del servidor y de los clientes intercambiados previamente. El secreto maestro se usa para generar claves para cifrado y cálculos MAC. Alternativamente el cliente y el servidor podrían usar el algoritmo Diffie-Hellman para generar el secreto pre-maestro.

- El cliente convierte el cifrado pendiente en el cifrado actual y envía el mensaje Change Cipher Spec al servidor.

Este mensaje significa que el servidor deberá activar la especificación del cifrado que se acaba de negociar.

- El servidor envía el mensaje Change Cipher Spec al servidor, como confirmación.
- En este punto termina la toma de contacto, y los dos lados pueden intercambiar datos usando los servidores de privacidad e integridad de SSL.

Ambos lados convierten el secreto pre-maestro en el secreto maestro, mediante el cálculo de una serie de hashes que usan el secreto pre-maestro, el valor aleatorio del saludo del cliente y el valor aleatorio del saludo del servidor. Entonces, el secreto maestro se usa para deducir las claves para el cifrado y los algoritmos MAC, incluidos un MAC secreto de escritura del cliente, un MAC secreto de escritura del servidor, una clave de escritura del cliente y una clave de escritura del servidor, lo mismo que un vector de iniciación (initialization vector, IV) de escritura del cliente y un vector de inicialización (IV) de escritura del servidor.

Un cliente puede resumir una sesión dentro de un periodo de suspensión dado mediante la solicitud de una sesión con el mismo ID de sesión de una sesión negociada previamente. La capacidad para resumir una sesión fue una característica agregada en el SSLv3 para reducir las operaciones auxiliares del establecimiento de la conexión. Se agregaron otras características para mejorar la eficiencia de la toma de contacto y añadir flexibilidad. Por ejemplo, el SSLv3 ofrece más cifrados que la v2, agrega soporte hash SHA y al algoritmo DSA, ofrece compresión y tiene capacidad para negociar la versión del protocolo. Se redujo la cantidad de mensajes necesarios para realizar la toma de contacto y se agrego el mensaje Change Cipher Spec. El método de generación de claves en SSLv3 se basa en el secreto maestro y los valores aleatorios, no en el ID de la conexión como en SSLv2.

TLS, el protocolo de seguridad de la capa de transporte, es muy similar al SSLv3; tan similar, de hecho, que el texto de la especificación es idéntico en muchas secciones. Las diferencias entre los dos protocolos no son grandes, pero los hacen incompatibles. Las principales diferencias están en las funciones hash y en las funciones de generación de claves criptográficas. El TLS usa la función de autenticación de mensajes HMAC, la cual difiere de la función de claves MAC, especificada en SSLv3. La función de generación de secreto maestro de TLS se basa en una función pseudoaleatoria que usa HMAC. La función de generación de claves SSLv3 es diferente. Por consiguiente, los dos

protocolos no generaran las misma claves dando los mismos resultados y no pueden interoperar. Algunas de las otras diferencias incluyen estos cambios en TLS: muchos más mensajes del protocolo de alerta, no soporta Fortalezza y el mensaje de la finalización de toma de contacto de computa usando algoritmos diferentes.

3.10.3.- IPSec.

IPSec define un marco de referencia seguro y un conjunto de servicios de seguridad para las comunicaciones a nivel de red (IP), partes de cual emplea PKI. IPSec se puede usar con ambientes IPv4 e IPv6. Opera en uno de dos modos: modo de túnel o modo de transporte. En el modo de túnel, todo el paquete de IP esta cifrado y se convierte en la parte de los datos de un nuevo paquete IP más grande, al que se agregan un nuevo encabezado IP y un encabezado IPSec. Si se usa el servicio de encapsulado de carga útil de seguridad (Encapsulating Security Payload, ESP), el paquete también tendrá una cola de datos IPSec. En el modo de transporte, el encabezado IPSec se inserta directamente en el paquete IP. El modo de túnel se usa, básicamente, con puertas o nodos de enlace, y proxys. Los sistemas intermedios implantan los servicios IPSec; los extremos no saben acerca de IPSec. En el modo de transporte, ambos extremos deben implantar IPSec; los sistemas intermedios no realizan ningún procesamiento de IPSec en el paquete.

IPSec ofrece seguridad fuerte y gran flexibilidad. Como resultado, es bastante complicado entender. Usaremos un ejemplo para demostrar la manera como dos sistemas podrían utilizar algunas de las opciones de IPSec. En nuestro ejemplo, un representante de ventas ha instalado en su laptop un software cliente de red privada cliente (VPN), basada en IPSec. Cuando el representante se conecta a Internet, el cliente VPN filtra el tráfico, buscando paquetes IP con destino a su oficina principal, con esto impide que cualquier paquete pase normalmente hasta allí (inseguro). Sin embargo, cuando el cliente ve un paquete que está dirigido a la oficina principal, lo intercepta. Después, usa los servicios IPSec para transmitir el paquete con seguridad a la casa matriz y garantizar que todo el tráfico que salga de ella también sea seguro.

- Lo primero que el cliente VPN hace es establecer una asociación de seguridad IPSec con el servidor de comunicaciones de la oficina matriz.

Una asociación de seguridad (Security Association, SA) define el contexto un contexto de seguridad entre dos partes. El protocolo de asociación de seguridad de Internet y administración de claves (Internet Security Association and Key Management Protocol, ISAKMP) es el marco de referencia que define la manera como el cliente VPN y el servidor configuran una asociación.

- Con el ISAKMP, el cliente y el servidor negocian el algoritmo de cifrado, el algoritmo hash, el mecanismo de autenticación y el mecanismo de establecimiento de la clave que usaran para los servicios IPSec.

Obsérvese que el ISAKMP no obliga a algoritmos o mecanismos en particular, de modo que se puede suministrar un máximo de flexibilidad. Sin embargo, exige el uso de firmas digitales dentro del componente de autenticación; esto significa que el cliente VPN y el servidor deben tener certificados de clave pública IPSec para poder establecer una asociación de seguridad. También significa que el cliente y el servidor deben conocer las opciones de la asociación de seguridad que cada uno soporta, pues de otro modo, el cliente y el servidor pueden estar incapacitados para negociar configuraciones comunes de la asociación de seguridad.

La primera parte de la SA del ISAKMP incluye las dos partes que negocian un canal seguro sobre el cual negociarán asociaciones de seguridad posteriores. Cada SA subsiguiente es específica para un protocolo de seguridad, dentro del contexto de una sola SA del ISAKMP. Este protocolo no está limitado a IPSec y se puede usar para otros protocolos, como el de Seguridad de capa de transporte (Transport Layer Segurity, TLS), aunque su uso actual está dedicado básicamente a los protocolos de encabezado de autenticación IPSec (IPSec Aunthetication Header) y carga útil de encapsulamiento (Encapsulating Payload). El ISAKMP consolida negociaciones de autenticación y de clave que hacen comúnmente en protocolos seguros, haciendo más eficientes las operaciones de seguridad.

- En este punto el cliente VPN y el servidor han negociado una SA del ISAKMP para el tráfico desde el cliente VPN hasta el servidor.
- Después, negocian una segunda SA. Cada SA está identificada de manera única mediante la combinación de un índice de parámetros de seguridad (Segurity Parameters Index, SPI) en cada paquete IP, en el protocolo de seguridad y en la dirección IP de destino.

Como resultado, cada SA es de un sentido (una dirección IP destina, una SA). Para que el cliente VPN y el servidor intercambien tráfico IPSec bidireccional, deben negociar dos asociaciones de seguridad: una para el tráfico que va del cliente hacia el servidor (el servidor es el destino) y una para el tráfico que va del servidor al cliente (el cliente es el destino).El cliente VPN y el servidor han completado ahora el primer paso en la configuración de las comunicaciones IPSec.

- El conjunto de SA determina el contexto básico de la seguridad.

El cliente VPN y el servidor han llegado a un acuerdo sobre los algoritmos que van a usar y se han autenticado mutuamente. Después, completan una segunda fase de negociación SA especifica para los servicios IPSec usaran. Para aplicar los demás servicios de seguridad de IPSec a sus comunicaciones, tales como control de acceso, la integridad de falta de conexión, autenticación de, origen de los datos, protección contra repetición o confidencialidad, el cliente y el servidor deben negociar el uso del encabezado de autenticación (Authetication Header AH), la carga útil de seguridad de encapsulamiento (Encapsulating Segurity Payload, ESP), o una combinación de ambas.

El encabezado de autenticación IPSec ofrece integridad, autenticación del origen de datos y servicios contra la repetición. El servicio de integridad del encabezado de autenticación usa un valor de verificación de integridad (Integrity Check Value, ICV) que se calcula sobre todo el paquete IP, excepto para los valores de campo del encabezado que pueden cambiar durante la transmisión (por ejemplo, tiempo de vida). El ICV puede usar un valor hash, un código de autenticación de mensaje de clave (como HMAC) o una firma digital; el algoritmo ICV es específico en la SA del IPSec. En general se usa un hash simple o en clave para las comunicaciones punto a punto. La clave que se calcula con una clave de secreto compartido o una firma digital. La prevención de la repetición se basa en un aumento monótono de una secuencia numérica. No se permite que la secuencia numérica se "envuelva"; de modo que, cuando un contador alcance un valor máximo, el ciclo no puede regresar a cero. IPSec obliga a que se cree una nueva SA, si un contador alcanza el límite. La nueva SA tendrá un contador nuevo y una nueva clase de cifrado.

El protocolo de carga útil de seguridad de encapsulamiento (ESP) ofrece varias características generales que son similares al AH, incluida la integridad a través de un ICV, autenticación de origen de datos con los MAC en clave y protección contra la repetición a través de números de secuencia. Los paquetes ESP están formateados de una manera levemente diferente de los paquetes AH; el ICV se agrega al final del paquete IPSec, mientras que el AH pone el ICV en el encabezado IPSec. No obstante, la diferencia más grande entre los dos en el servicio de confidencialidad de ESP: el ESP suministra confidencialidad a través del cifrado. El algoritmo de cifrado se negocia en la SA.

De regreso al ejemplo, el representante de ventas con el cliente VPN esta enviando detalles contractuales sensibles a la oficina principal, de modo que el cliente VPN está configurado para usar ESP con servicios de autenticación de origen, integridad y confidencialidad. El cliente y el servidor usan firmas digitales durante la autenticación del establecimiento de su SA. Dado que el cliente VPN está preconfigurado con una clave HMAC compartida, el cliente y el servidor acuerdan usar los MAC con claves o cifrado DES triple, pasa suministrar integridad, autenticación de origen y confidencialidad.

- Para la negociación de claves se pueden usar otros protocolos IPSec.

Dos de ellos son el protocolo de determinación de claves de Oakley y el protocolo de intercambio de clave de Internet (Internet Key Exchange). OKALEY usa el algoritmo Diffie-Hellman como el mecanismo fundamental para negociar un valor compartido, pero agrega valor y autenticación a las direcciones. La validación de la dirección de OAKLEY usa "cookies" aleatorias de 64 bits para impedir ataques de negación del servicio y suministrar un identificador débil de dirección de colega. Los mecanismos de autenticación incluyen firmas digitales, cifrado de clave pública y cifrado de clave simétrica con claves previamente intercambiadas. Los participantes de OAKLEY también permiten que las dos partes se identifiquen entre si y negocien tres algoritmos para usar: un algoritmo de cifrado, un algoritmo hash y un mecanismo de autenticación. El intercambio de clave se puede completar en tres mensajes. Sin embargo, si las partes usan las características opcionales del protocolo, el número de mensajes será mayor.

IKE implementa un subconjunto de OAKLEY. Como con OALLEY , los mecanismos de autenticación IKE incluyen firmas digitales, cifrado de clave pública y claves simétricas precompartidas. IKE también soporta la negociación del grupo Diffie-Hellman y puede ofrecer una condición de perfecto secreto para claves e identidades donde el compromiso de una clave única sólo expone los datos protegidos con esa clave y no afecta a las demás claves que se usan en el protocolo.

3.10.4.- S/MIME.

Las extensiones seguras/multipropósito de correo de Internet (secure/multipurpose Internet Mail Extensions, S/MIME) agregan servicios de autenticidad del mensaje, integridad y privacidad a las aplicaciones de mensajes electrónicos. S/MIME no está limitado al correo electrónico y lo pueden usar otros mecanismos de transportes que soportan MIME, como es el caso de http. S/MIME puede asegurar una parte de un mensaje único, partes de múltiples mensajes o un mensaje completo. S/MIME usa la codificación con la sintaxis criptográfica para mensajes (Cryptographic Messsage Syntax, CMS) con el fin de envolver, firmar o firmar y envolver mensajes.

Los agentes de usuario de correo pueden agregar automáticamente servicios de seguridad criptográfica al correo que se va a enviar e interpretar servicios criptográficos que se han aplicado para recibir correo. Usualmente, los mensajes S/MIME se transmiten en codificación con base 64 para facilitar la transmisión de datos binarios. La especificación del servicio criptográfico S/MIME da a SHA-1 como la reseña preferida, y a DSA como el algoritmo de firma predefinido. Si se usa RSA, el tamaño mínimo de la clave RSA debe ser 768 bits.

La aplicación/pkcs7 mime tipo MIME se usa para llevar objetos CMS de varios tipos, incluidos EnvelopedData y signedData. Por ejemplo, los mensajes que van sólo en sobres están formateados como objetos envelopedData CMS. Una copia del contenido de la clave de cifrado se cifra para cada

receptor y para el emisor. Después el objeto CMS se inserta en una entidad MIME de aplicación/pkcs7-mime. Los mensajes que van solo en sobre no cuentan con integridad de mensaje. Los mensajes solamente firmados ofrecen integridad de mensaje, pero no confidencialidad. Los receptores sin cliente S/MIME no pueden ver mensajes de aplicación/pkcs7-mime, pero pueden ver mensajes de múltiples partes firmadas. Los mensajes firmados y cifrados se crean anidando mensajes firmados y solo cifrados. Los mensajes se pueden anidar en cualquier orden.

3.10.5.- Protocolo de Registro de Hora.

El protocolo de registro de hora (Time Stamp Protocol, TSP) ofrece pruebas de que los datos existieron en un número en particular, a través de los servicios de la Autoridad de registro de hora (Time Stamp Authority, TSA). En la actualidad, el TSP se halla bajo desarrollo del grupo de trabajo de PKIX.

TSP es un protocolo simple de solicitud/respuesta. La entidad que solicita un registro de hora envía un mensaje de solicitud correspondiente a la TSA; dicho mensaje contiene el hash de datos que se le va a poner registro de hora. El TSA regresa el registro de hora en un mensaje de respuesta, que incluye el estado de la solicitud (es decir si fue aprobada o negada) y el registro de hora, en la forma de un mensaje de datos firmados, formateado de conformidad con el CMS.

El trabajo de PKIX acumula varios requerimientos de una TSA. Por ejemplo, el servidor TSP debe usar un reloj confiable, darle el registro de hora a un solo hash de datos y no incluye ninguna identificación de la entidad que solicita el registro de la hora.

WTLS

La seguridad del nivel de transporte inalámbrico (Wireless Transport-Level Security, WTLS) es la especificación WAP de Forum para servicios de seguridad que suministran privacidad, integridad de datos y autenticación entre aplicaciones. WTLS ofrece una funcionalidad similar a la de TLS, pero esta adaptado al ambiente inalámbrico. Por ejemplo, WTLS agrega optimización de toma de contacto y regeneración de clave dinámica para mejorar el desempeño en redes inalámbricas con un ancho de banda bajo, con un tiempo de espera relativamente largo. A diferencia de TLS, WTLS puede funcionar con redes orientadas por conexiones o datagramas.

WTLS usa los mismos conceptos de protocolos de TLS y SSL, con un protocolo de registro y un protocolo de toma de contacto. Usando este último, el cliente y el servidor acuerdan la revisión del protocolo, seleccionan algoritmos de cifrado mutuamente aceptables, se autentican mutuamente y generan el secreto maestro compartido. WTLS soporta la ejecución del protocolo de toma de contacto con datagramas, en este caso, de datos de servicio de transporte. El WTLS también soporta una toma de contacto de secreto compartido si el cliente y el servidor ya han establecido un secreto

compartido. Esto facilita una toma de contacto abreviada, permitiendo un mejor desempeño para las redes inalámbricas.

3.11.- Estándares de Formato (PKI).

La interoperatividad entre aplicaciones PKI requiere saber qué se supone que significan los bits que se reciben desde otra aplicación. Los estándares ofrecen acuerdos comunes para sintaxis y significados de datos. El estándar de formato que más prevalece para las aplicaciones PKI es el estándar X.509, dado que define la estructura básica para los certificados de clave pública. Los estándares PKCS de RSA Laboratorios son los estándares principales para definir lo que significan estos bits de datos. Estos estándares definen aspectos, como darle el formato apropiado a una clave privada o a una clave pública. Otros estándares importantes son el certificado PKIX y el perfil CRL, el trabajo del proyecto IEEE P1363 de la definición de algoritmos criptográficos, y el grupo de trabajo de ITEF en firmas digitales XML.

3.11.1.- X.509.

Un certificado es esencialmente una clave pública y un identificador, firmados digitalmente por una autoridad de certificación, y su utilidad es demostrar que una clave pública pertenece a un usuario concreto. El formato de certificados X.509 es el más común y extendido en la actualidad.

El estándar X.509 sólo define la sintaxis de los certificados, por lo que no está atado a ningún algoritmo en particular, y contempla los siguientes campos:

- Versión.
- Número de serie.
- identificador del algoritmo empleado para la firma digital.
- Nombre del certificador.
- Periodo de validez.
- Nombre del sujeto.
- Clave pública del sujeto.
- identificador único de certificador.
- identificador único de sujeto.
- Extensiones.
- Firma digital de todo lo anterior generada por un certificador.

Estos certificados se estructuran de forma jerárquica, de tal forma que nosotros podemos verificar la autenticidad de un certificado comprobando la forma de la autoridad que lo emitió, que a su vez tendrá otro certificado expedido por otra autoridad de rango superior. De esta forma vamos subiendo en la jerarquía hasta llegar al nivel más alto, que deberá estar ocupado por un certificador que goce de la confianza de toda la comunidad. Normalmente las claves públicas de los certificadores de mayor nivel se suelen publicar incluso en papel para que cualquiera pueda verificarlas.

El mecanismo que debe de emplearse para conseguir un certificado X.509 es enviar nuestra clave pública a la autoridad de certificación, después de habernos identificado positivamente frente a ella. Existen autoridades de certificación que, frente a una solicitud, generan un par de llave pública-privada y lo envían al usuario. Hemos de hacer notar que en este caso, si bien tendremos un certificado válido, nuestro certificador podrá descifrar todos nuestros mensajes.

El X.509 es el sector de estandarización de la Unión Internacional de Telecomunicaciones (ITU-T) y el estándar de formato de certificado de la Organización Internacional de Normalización (Internacional Organization for Standardization, ISO). Definido como parte de las series de estándares de ITU-ISO sobre servicios de directorios, el X.509 es el estándar fundamental que define la estructura del certificado de clave pública. Se publico por primera vez en 1988, se revisó en 1993 y, de nuevo, en 1996, la versión actual, X.509 versión 3, agregó soporte para campos de extensión que aumentan en gran medida la flexibilidad del certificado. Un certificado X.509 X.509v3 incluye un conjunto obligatorio de campos en un orden predefinido, lo mismo que campos opcionales de extensión. El X.509 permite una flexibilidad considerable, incluso dentro de sus campos obligatorios, ya que ofrece múltiples opciones de codificación para la mayoría de los campos (para el detalle técnico del estándar del certificado X.509 ver apéndice).

3.11.2.- PKIX.

El grupo de trabajo de la infraestructura de claves públicas (PKIX) en el área de seguridad de IETF está definiendo una serie de estándares para el uso de certificados de clave pública en Internet. El grupo de trabajo de PKIX se formó en octubre de 1995 a fin de desarrollar los estándares de Internet necesarios para soportar PKI interoperables. El primer renglón de trabajo del grupo fue crear un perfil que limitara las estructuras de datos de los certificados, extensiones y valores de datos para un conjunto específico de opciones. La mayor flexibilidad del estándar X.509 dificulta la interoperatividad. Al restringir las opciones permitidas, el grupo PKIX espera aumentar la capacidad de una PKI para interoperar con otra.

3.11.3.- IEEE P1363.

El proyecto P1363 de IEEE comenzó en 1994 orientado a desarrollar estándares criptográficos, incluidas las especificaciones estándar para la criptografía de clave pública. El proyecto comenzó con las técnicas primitivas de especificación matemática y las técnicas criptográficas para los algoritmos RSA y Diffie-Hellman y desde entonces ha ampliado su trabajo para incluir sistemas criptográficos de curva elíptica. Aunque no está limitado a los estándares de formateo, el trabajo del grupo P1363 es importante para garantizar la interoperabilidad de PKI, dado que suministra una especificación de referencia para el campo de las técnicas comunes de clave pública, las firmas digitales y la identificación de varias familias, tales como logaritmos discretos, factorización de enteros y curvas elípticas. También cubre consideraciones criptográficas relacionadas, como la administración de claves y la generación segura de números aleatorios.

3.11.4.- PKCS.

Inicialmente, RSA Laboratories desarrolló los Estándares de la infraestructura de claves públicas (Public Key Cryptography Standards, PKCS) en colaboración con la industria, la academia y los representantes del gobierno (EUA) para avanzar en la interoperabilidad de la criptografía de claves públicas. Encabezada todavía por RSA, el trabajo de PKCS se ha expandido a través del tiempo para cubrir al creciente grupo de estándares de formateo, algoritmos y las API de PKI. Los estándares PKCS ofrecen definiciones fundamentales de formatos de datos y algoritmos que se encuentran virtualmente en todas las implantaciones de PKI actuales.

Los estándares PKCS son como sigue:

- PKCS #1 – Estándar de cifrado RSA.
- PKCS #2 – Cubre el cifrado RSA de las reseñas de mensajes y fue incorporado en el PKCS #1.
- PKCS #3 – Estándar de acuerdo de clave Diffie-Hellman.
- PKCS #4 – Originalmente especifico la sintaxis de claves RSA, pero fue incluido en el PKCS # 1.
- PKCS #5 – Estándar de cifrado basado en contraseñas
- PKCS #6 – Estándar de sintaxis de certificado extendido.
- PKCS #7 – Estándar de sintaxis de mensaje criptográfico.
- PKCS #8 - Estándar de sintaxis de información de clave privada.
- PKCS #9 – Tipos de Atributos seleccionados.
- PKCS #10 – Estándar de sintaxis de solicitud de certificación.
- PKCS #11 – Estándar de interfaz criptográfica de señal.

- PKCS #12 – Estándar de sintaxis de intercambio de información personal.
- PKCS #13 – Estándar de criptografía de curva elíptica.
- PKCS #14 – Estándar de generación de número de pseudoaleatorios.
- PKCS #15 – Estándar de la sintaxis de información criptográfica de señal.

3.11.5.- XML.

El lenguaje de marcación extensible es un mecanismo flexible para definir formatos de datos digitales. Un uso natural de XML es definir formatos para objetos de datos firmados. Al entender esto, el grupo de trabajo de firmas digitales en XML, de IETF está defiendo elementos XML para facilitar la interoperabilidad de objetos de datos firmados.

El concepto básico en el trabajo de IEFT es que las firmas en XML se pueden aplicar a objetos de datos arbitrarios. El firmante calcula primero una reseña de objeto de datos (contiene virtualmente cualquier tipo de datos digitales). Después, la reseña se pone en un elemento XML que, por sí mismo, está reseñado y firmado criptográficamente. Los elementos de la firma XML delimitan los datos que se firman y también pueden contener otra información relacionada con la firma, tal como un registro de tiempo.

3.12.- Interfaces de Programación de Aplicaciones

Una manera para agregar capacidades PKI a aplicaciones nuevas o existentes es usar un conjunto de las API que ya implantan funciones PKI. CryptoAPI de Microsoft, la Arquitectura común de seguridad de datos (Common Data Segurity Architecture, CDSA) del Open Group y la Interfaz de programa de aplicación de servicio de seguridad general (Generic Security Service API, GSS-API) de IETF ofrecen un conjunto estandarizado de interfaces para PKI y funciones de seguridad relacionadas. Cualquier aplicación PKI necesitará el acceso a certificados de claves públicas; el Protocolo lightweight de acceso a directorios (Lightweight Directory Access Protocol, LDAP) se ha convertido en el estándar de facto para el almacenamiento de certificados.

3.12.1.- CryptoAPI de Microsoft

CryptoAPI de Microsoft es un amplio conjunto de funciones de seguridad que son parte de los sistemas operativos Windows, CryptoAPI resume funciones criptográficas y oculta su implantación real; de hecho, permite múltiples implantaciones de las mismas funciones criptográficas por medio de los proveedores de servicios criptográficos (Cryptographic Service provider, CSP). Un CSP es un

modulo independiente que implanta funciones criptográficas accesibles a través de interfaces comunes de CryptoAPI. En otras palabras, un CSP empaca u oculta la implantación criptográfica de modo que el usuario no necesita preocuparse al respecto. Microsoft suministra un CSP base; otros proveedores ofrecen las CSP que contienen su propia implantación de algunas o todas las funciones de CryptoAPI.

CryptoAPI suministra funciones en estas áreas generales:

- Las funciones de criptografía básica soportan la selección, inicialización y terminación de un CSP. Estas funciones también incluyen funciones de manipulación de claves criptográficas para generar, intercambiar y almacenar claves criptográficas. Otras funciones incluyen codificación y decodificación de objetos, cifrado y descifrado de datos, generación de hash, y creación y verificación de firmas digitales.
- Las funciones de certificado y almacenamiento de certificados administran colecciones de certificados, listas de revocación de certificados y listas de confianza de certificados.
- Las funciones de verificación de certificados soportan la manipulación de listas de confianza de certificados (Certificate Trust Lists, CTL) y cadenas de certificados, tales como ubicar una CTL dentro de un mensaje firmado y validación de cadenas de certificados.
- Las funciones de mensaje manipulan mensajes PKCS #7 y operan específicamente en construcciones de mensajes. Las funciones de mensaje incluyen cifrar y descifrar mensajes y datos de mensaje, firmar mensajes y datos de mensaje, y verificar mensajes firmados. Las funciones de mensaje de bajo nivel operan directamente en mensajes en PKCS #7. CryptoAPI también incluye un conjunto de funciones de cómo CryptSignAndEncryptMessage, la cual realiza el formateo PKCS #7, cifrado y firma de una llamada API.
- Las funciones auxiliares cubren funciones adicionales que no se ajustan en otras categorías. Ellas incluyen manipulación de extensiones de certificado y comparación de atributos de certificado, y funciones de manipulación de ID objeto.

Otra característica de CryptoAPI es que las aplicaciones pueden especificar pocos detalles, si los hay, de algunos algoritmos criptográficos. Esto se puede hacer implantando funciones criptográficas más simples, pero puede hacer que la interoperabilidad con otras CSP u otras herramientas criptográficas sea extremadamente difícil.

3.12.2.- Arquitectura Común de Seguridad de Datos (CDSA).

La arquitectura común de seguridad de datos (Common Data Segurity Architecture, CDSA) del Open Group se basa en un trabajo que originalmente presentó Intel Architecture Labs. CDSA es una

arquitectura de seguridad por capas que se caracteriza por su capacidad para extenderse y su interoperabilidad. Ofrece flexibilidad a través de módulos de seguridad plug-in y servicios de seguridad API. CDSA está disponible en el Open Group como una implantación de fuente abierta y caracteriza los servicios de confianza basados en certificados de claves públicas.

El administrador de servicios de seguridad comunes (Common Security Services Manager, CSSM) es el núcleo de CDSA. CSSM administra servicios de seguridad y módulos agregados, define varias API para acceder a servicios de seguridad y define interfaces de proveedor de servicios para módulos de servicio de seguridad. Los servicios internos de CSSM ofrecen aseguramiento de integridad y administración de contexto de seguridad; los servicios de integridad verifican de integridad interna de los componentes CSSM y la integridad, identidad y autorización de otros componentes CSSM. Los servicios de administración de contexto de seguridad administran los parámetros necesarios para operaciones criptográficas.

CSSM tiene seis administradores de módulos principales:

- El administrador de servicios criptográficos ofrece una API común para acceder a todos los CSP que se usan en el sistema.
- El administrador de servicios de directiva de confianza procesa las solicitudes para servicios de seguridad que requieren la revisión y aprobación de la directiva.
- El administrador de servicios de la librería del certificado manipula certificados y listas de revocación de certificados. Sus funciones incluyen crear, firmar, verificar y extraer campos de los certificados. Las librerías incorporadas facilitan el procesamiento de los certificados.
- El administrador de servicios de almacenamiento de datos ofrece almacenamiento seguro y persistente para objetos de seguridad, incluidos certificados y las CRL. Soporta información de consulta acerca de objetos de seguridad, tales como la hora de creación del objeto, el tamaño y la hora de modificación.
- El administrador de servicios de autorización es un servicio de autorización general para determinar si se permite una operación en un objeto específico, con base en un subconjunto de credenciales.
- El administrador de módulo electivo ofrece flexibilidad para extender la arquitectura con administradores adicionales.

3.12.3.- Interfaz de Programa de Aplicación de Servicio de Seguridad General (GSS-API).

La interfaz de programa de aplicación de servicio de seguridad general (Generis Security Service API, GSS-API) ofrece servicios de seguridad general mientras oculta la implantación de los servicios.

El grupo de trabajo de tecnología de autenticación común del IEFT diseño el GSS-API para usarlo en intercambios de colega a colega que requieren seguridad, tales como protocolos de comunicaciones. Las especificaciones GSS-API se publicaron originalmente a comienzos de la década de 1990 y han sido revisadas varia veces para acomodar nuevas tecnologías, tales como Java.

Las principales funciones de GSS-API:

- Iniciar contextos de seguridad.
- Autenticar colegas.
- Administración de credenciales.
- Funciones de seguridad de mensaje, que incluyen firma de mensajes, verificación sello y eliminación del sello. GSS-API soporta integridad por mensaje y autenticación del origen de los datos, lo mismo que confidencialidad del mensaje.

Probablemente, GSS-API se usa más ampliamente con ambientes Kerberos, pero también soporta funciones de clave pública, incluidos el establecimiento de claves y firmas digitales para verificar la integridad.

3.12.4.- Protocolo Lightweight de Acceso a Directorios (LDAP).

El IEFT desarrolló el Protocolo Lightweight de acceso a directorios (Lightweight Directory Access Protocolo, LDAP) como una parte simplificada en contacto directo con el cliente para el protocolo de acceso a directorios X.500. El LDAP reduce la complejidad de usar servicios de directorio de X.500 mediante suministro de, aproximadamente, 90 por ciento de toda la funcionalidad del protocolo X.500, a un 10 por ciento del costo de procesamiento.

Para los ambientes PKI, los servicios LDAP se han convertido en los mecanismos preferidos para almacenar certificados. Dado que los productos LDAP están ampliamente disponibles, el LDAP da un método estándar para almacenar y recuperar certificados y datos de CRL. Además, los juegos de desarrollo están rápidamente disponibles con el fin de desarrollar aplicaciones LDAP.

La interoperabilidad entre los directorios LDAP es el siguiente desafió que se debe de enfrentar. El IETF está trabajando para estandarizar esquemas LDAP que pertenecen a certificados y a los CRL, los cuales aumentan la interoperabilidad entre los productos LDAP, tales como servidores de certificados y productos de autenticación, que basan la autenticación o la autorización en el contenido del certificado y la estructura del directorio. Como una aplicación adicional, implantaciones individuales del directorio LDAP pueden usar esquemas ajustados a la estructura de la corporación. Algunas compañías diseñan sus esquemas LDAP corporativos para equilibrar su estructura

corporativa; por ejemplo, un esquema puede tener unidades organizacionales para cada subsidiaria corporativa, y otra puede tener unidades organizacionales para cada sitio geográfico o cada área funcional corporativa. En algunos esquemas, los usuarios están identificados con ID usuario único; en otros, mediante su dirección de correo electrónico. Si los productos PKI no pueden ajustar sus esquemas de publicación para acomodarlos al esquema corporativo, puede ser difícil agregar certificados a un directorio LDAP existente.

3.12.5.- Aplicaciones e Implantaciones de PKI.

Los navegadores y los clientes de correo electrónico son los ejemplos más prevalecientes de la tecnología de claves públicas actuales. Los navegadores Web Netscape e Internet Explorer has soportado la tecnología de claves públicas virtualmente desde su concepción. Ambos soportan la generación de parejas de claves y la creación de solicitud de certificado, lo mismo que el almacenamiento de claves privadas y certificados. Con Internet Explorer, la mayor parte de este soporte se presenta en forma de controles Active X que llaman funciones CryptoAPI de Microsoft, mientras que, por lo común, las claves y los certificados de Microsoft se almacenan en el registro. Con los navegadores de Netscape, la funcionalidad se construye en el navegador mismo; esos navegadores, por ejemplo, mantienen su propia clave privada y su repositorio de certificados. En esencia, todos los principales clientes de correo electrónico de la actualidad soportan el protocolo S/MIME. Un usuario puede firmar, cifrar, o firmar y cifrar mensajes de correo electrónico con unas cuantas pulsaciones en los botones de ratón.

A pesar del trabajo de las entidades que establecen los estándares, como IEFT, la interoperabilidad entre los productos sigue siendo problema. Por ejemplo, las claves y certificados de Internet Explorer no funcionarán con los navegadores de Netscape. Las claves privadas de Microsoft se almacenan en un formato "pvk" específico que es incompatible con las claves formateadas en PKCS.

Los productos también son inconsistentes en su enfoque para garantizar sus implantaciones de clave pública. Netscape requiere protección de contraseña para las claves privadas; Microsoft deja las contraseñas como opcionales. La función del navegador de generación de claves para generar parejas de claves en un navegador de Netscape permite que los usuarios elijan la longitud de la clave que van a generar. Los desarrolladores no pueden limitar el comportamiento de los usuarios, dado que la generación de claves tiene pocas opciones. Como en la actualidad las funciones de generación de claves de Internet Explorer se basan en CAPI (CryptoAPI). Permiten un control considerablemente mayor a su creador.

Las características del usuario son otra área en la cual las aplicaciones de clave pública están madurando. Si un usuario pierde la clave privada (o la contraseña para liberar la clave privada), la

mayoría de los productos actuales obliga al usuario a generar una nueva pareja de claves y obtener un nuevo certificado. Instalar una clave privada y un certificado en múltiples productos o en múltiples computadoras puede ser un desafió, incluso para quienes cuentan con conocimiento técnico. La mayor parte de los productos soporta la capacidad para exportar e importar claves y certificados; no obstante, los formatos que soportan difieren en los procedimientos para importar y exportar.

4.- PGP(Pretty Good Privacy).

PGP (Pretty Good Privacy) se trata de un proyecto iniciado por Phill Zimmerman en 1993, cuando se carecía de herramientas sencillas pero a la vez potentes que permitiesen a los usuarios "comunes" hacer uso de una criptografía seria.

Con el tiempo PGP se ha convertido en uno de los mecanismos más populares para utilizar criptografía, tanto por usuarios particulares como grandes empresas. Se ha publicado su código fuente, y se permite su uso gratuitamente para fines no comerciales.

Se trata ya de un estándar internacional, RFC 2440, y dispone de numerosas aplicaciones (codificación de almacenamiento, codificación automática de tráfico TCP/IP, etc). Principalmente sus dos funciones originales en qué consiste PGP y más importantes: asegurar la confidencialidad de un texto, y la autentificación de un emisor frente a su receptor (firma digital).

4.1.- Fundamentos de PGP.

PGP funciona con criptografía asimétrica (aunque por cuestiones de eficiencia también hace uso de criptografía simétrica), y su punto fuerte radica en la facilidad que ofrece a los usuarios comunes para generar las claves (algo que como antes se ha mencionado no es en absoluto trivial) y gestionarlas.

PGP proporciona lo que se denomina "anillo de claves" (llavero), que es un único archivo donde el usuario puede guardar todas sus claves, con facilidad para realizar inserción y extracción de claves de manera sencilla.

Además proporciona un mecanismo de identificación y autentificación de claves (certificación de que una clave pública es realmente de quien dice que es) para evitar los "ataques de intermediario". Estos ataques consisten en que una persona que intervenga el canal de comunicación nos proporcione una clave pública falsa del destinatario al que deseamos enviar el mensaje. En ese caso, encriptaríamos con dicha clave, y el atacante podría desencriptar la información, encriptarla con la verdadera clave pública que ha intervenido, y enviársela al destinatario sin que nadie se percate de su presencia.

Para autentificar claves, PGP permite que los usuarios "firmen claves", por lo que podemos confiar en la autenticidad de una clave siempre que ésta venga firmada por una persona de confianza. Así la "autoridad de certificación" de otro tipo de sistemas (entidades que aseguran la autenticidad de las claves), en PGP son los propios usuarios.

Además, cada clave tiene una "huella digital" (fingerprint), que se trata de una secuencia lo suficientemente larga para que sea única, pero lo suficientemente corta para poder ser comunicada de viva voz o escrita en papel. Así, si queremos asegurarnos de la autenticidad de una clave, solo hemos de preguntar a su autor la huella digital de su clave.

Como ejemplo, la clave pública se puede obtener en el servidor de oficial de PGP (keyserver.pgp.com) y la huella digital es:

A119 1272 002F 37B4 62ED 7A9A 694B 0D34 312E DD52

Si la clave que alguno tiene en su anillo como mía no tuviese esta huella digital, es que no es verdadera (alguien podría haberla subido a keyserver en mi lugar o a una web indicando que es mi clave).

Cuando una clave secreta queda comprometida, o se sustituye por una nueva (por ejemplo porque el valor de la información que deseamos proteger hace necesaria la utilización de un algoritmo más seguro o de clave más larga), puede ser revocada por su autor. Solo tiene que generar y distribuir un "certificado de revocación" que informará a los usuarios de PGP que esa clave ya no es válida. Por supuesto, para poder emitir dicho certificado es necesario tener la clave secreta.

Como cualquier herramienta, PGP proporciona un nivel de seguridad muy bueno y gran rendimiento si se utiliza correctamente. Sin embargo un uso inadecuado puede convertirlo en algo completamente inútil. Para que ésto no ocurra debemos cuidar algunos aspectos:

- Escoger contraseñas adecuadas: PGP para extraer del anillo una clave privada requiere al usuario una contraseña. Por supuesto, ésta ha de ser segura (memorizable, larga, aleatoria, complicada, etc.)
- Proteger los archivos sensibles: hemos de proteger los anillos de claves (PUBRING.PKR y SECRING.SKR), tanto de intrusos como de su pérdida (conviene tener backup).
- Firmar sólo las claves de cuya autenticidad estemos seguros: hemos de ser cuidadosos para que las "redes de confianza" funcionen adecuadamente, o podemos certificar claves falsas (lo que nos engañaría a nosotros y a aquellos que confíen en nosotros como autoridad de certificación).

4.2.- Estructura de PGP.

4.2.1.- Codificación de Mensajes.

Como el lector ya sabe, los algoritmos simétricos de cifrado son considerablemente más rápidos que los asimétricos. Por esta razón PGP cifra primero el mensaje empleando un algoritmo simétrico con una clave generada aleatoriamente (clave de sesión) y posteriormente codifica la clave haciendo uso de la llave pública del destinatario. Dicha clave es extraída convenientemente del anillo de claves publicas a partir del identificador suministrado por el usuario, todo ello de forma transparente, por lo que únicamente debemos preocuparnos de indicar el mensaje a codificar y la lista de identificadores de los destinatarios. Nótese que para que el mensaje pueda ser leído por múltiples destinatarios basta con que se incluya en la cabecera la clave de sesión codificada con cada una de las claves publicas correspondientes.

Cuando se trata de decodificar el mensaje, PGP simplemente busca en la cabecera las claves publicas con las que esta codificado y nos pide una contraseña. La contraseña serviría para que PGP abra nuestro anillo de claves privadas y compruebe si tenemos una clave que permita decodificar el mensaje. En caso afirmativo, PGP descifrará el mensaje. Nótese que siempre que queramos hacer uso de una clave privada, habremos de suministrar a PGP la contraseña correspondiente, por lo que si el anillo de claves privadas quedara comprometido, un atacante aún tendrá que averiguar nuestra contraseña para descifrar nuestros mensajes. No obstante, si el anillo de claves privadas quedara comprometido, lo mejor sería revocar todas las claves que tuviera almacenadas y generar otras nuevas.

Como puede comprenderse, gran parte de la seguridad de PGP reside en la calidad del generador aleatorio que se emplea para generar las claves de sesión, puesto que si alguien logra predecir la secuencia de claves que estamos usando, podría descifrar todos nuestros mensajes independientemente de los destinatarios a los que vayan dirigidos. Afortunadamente, PGP utiliza un método de generación de números pseudoaleatorios muy seguro, una secuencia aleatoria pura es imposible de conseguir, como se dijo antes, y protege criptográficamente la semilla aleatoria que necesita. No obstante, consideraremos sensible al archivo que contiene dicha semilla (normalmente RANDSEED.BIN), y por lo tanto habremos de evitar que quede expuesto.

4.2.2.- Firma Digital.

En lo que se refiere a la firma digital, las primeras versiones de PGP obtienen en primer lugar la signatura MD5, que posteriormente se codifica empleando la clave privada RSA correspondiente. Las versiones actuales implementan el algoritmo DSA. La firma digital o signatura puede ser añadida al

archivo u obtenida en otro archivo aparte. Esta opción es muy útil si queremos armar un archivo ejecutable, por ejemplo.

4.2.3.- Armaduras ASCII.

Una de las funcionalidades más útiles de PGP consiste en la posibilidad de generar una armadura ASCII para cualquiera de sus salidas. Obviamente, todas las salidas de PGP (mensajes codificados, claves públicas extraídas de algún anillo, armas digitales, etc.) consisten en secuencias binarias, que pueden ser almacenadas en archivos. Sin embargo, en la mayoría de los casos puede interesarnos enviar la información mediante correo electrónico, o almacenarla en archivos de texto.

Recordemos que el código ASCII es de 7 bits, eso quiere decir que los caracteres situados por encima del valor ASCII 127 no están definidos, y de hecho diferentes computadoras y sistemas operativos los interpretan de manera distinta. También hay que tener en cuenta que entre los 128 caracteres ASCII se encuentran muchos que representan códigos de control, como el retorno de carro, el fin de archivo, el tabulador, etc. La idea es elegir 64 caracteres imprimibles (que no sean de control) dentro de esos 128 caracteres. Con los 64 caracteres escogidos podremos representar exactamente 6 bits, por lo que una secuencia de tres bytes (24 bits) podría representarse mediante cuatro de estos caracteres. Esta cadena de caracteres resultante se trocea colocando en cada línea un número razonable de símbolos, por ejemplo 72. El resultado es una secuencia de caracteres que pueden ser tratados como texto estándar y, por tanto, manipulados en cualquier editor de texto y, por supuesto, enviados por correo electrónico.

4.2.4.- Gestión de Claves.

PGP, como ya se ha dicho, almacena las claves en unas estructuras denominadas anillos. Un anillo no es más que una colección de claves, almacenadas en un archivo. Cada usuario tendrá dos anillos, uno para las claves públicas (PUBRING.PKR) y otro para las privadas (SECRING.SKR).

Cada una de las claves, además de la secuencia binaria correspondiente para el algoritmo concreto donde se emplee, posee una serie de datos, como son el identificador del usuario que la emitió, la fecha de expiración, la versión de PGP con que fue generada, y la denominada huella digital (fingerprint). Este último campo es bastante útil, pues se trata de una secuencia hexadecimal lo suficientemente larga como para que sea única, y lo suficientemente corta como para que pueda ser escrita en un papel, o leída de viva voz. La huella digital se emplea para asegurar la autenticidad de una clave.

4.2.5.- Distribución de Claves y Redes de Confianza.

PGP, como cualquier sistema basado en clave pública, es susceptible a ataques de intermediario. Esto nos obliga a establecer mecanismos para asegurarnos de que una clave procede realmente de quien nosotros creemos. Uno de los mecanismos que permite esto es la huella digital, pero no el único. PGP permite a un usuario armar claves, y de esta forma podremos confiar en la autenticidad de una clave siempre que ésta venga armada por una persona de confianza. Hay que distinguir entonces dos tipos de confianza: aquella que nos permite creer en la validez de una clave, y aquella que nos permite fiarnos de una persona como certificador de claves. La primera se puede calcular automáticamente, en función de que las armas que contenga una clave pertenezcan a personas de confianza, pero la segunda ha de ser establecida manualmente. No olvidemos que el hecho de que una clave sea auténtica no nos dice nada acerca de la persona que la emitió. Por ejemplo, yo puedo tener la seguridad de que una clave pertenece a una persona, pero esa persona puede dedicarse a armar todas las claves que le llegan, sin asegurarse de su autenticidad, por lo que en ningún caso merecerá nuestra confianza.

Cuando una clave queda comprometida, puede ser revocada por su autor. Para ello basta con generar y distribuir un certificado de revocación que informará a todos los usuarios de que esa clave ya no es válida. Para generarlo es necesaria la clave privada, por lo que en muchos casos se recomienda generar con cada clave su certificado de revocación y guardarlo en lugar seguro, de forma que si perdemos la clave privada podamos revocarla de todas formas.

Afortunadamente, las últimas versiones de PGP permiten nombrar revocadores de claves, que son usuarios capaces de invalidar nuestra propia clave, sin hacer uso de la llave privada.

5.- La Criptografía y la Sociedad Moderna.

5.1.- Contexto Actual.

La libertad de expresión y al derecho a la intimidad es un derecho constitucional del individuo (Articulo 16 de la Constitución Política de los Estados Unidos Mexicanos); que como consecuencia de la progresiva implantación de los sistemas de telecomunicación electrónicos, resulta crecientemente amenazada. Muchas personas que navegan en Internet (Internautas) poseen e intercambian información confidencial que desean permanezca como tal. Desde los números de tarjeta de crédito utilizados en las transacciones comerciales, hasta los mensajes enviados a un amigo que no se quiere sean leídos por nadie más. Cada individuo cuenta con sus propias razones para desear conservar en privado la información almacenada en su computadora.

Ahora bien, ¿de qué herramientas puede valerse el internauta para proteger sus datos de ojos indiscretos? Precisamente, la criptografía viene en ayuda del internauta y del ciudadano en general

para ofrecer los mecanismos de protección de la información necesarios. La criptografía se presenta así como esa delgada barrera que vela por la conservación de la privacidad en las comunicaciones, sean éstas para transmitir datos bancarios o simplemente información no relevante pero personal y confidencial. En la nueva Sociedad de la Información, no puede entenderse una democracia sin garantía de privacidad.

Sin embargo, esta lógica aspiración de los ciudadanos a comunicarse libremente con quien quieran sin que nadie se interponga, parece chocar de frente con los planes de seguridad nacional de los Estados, que contemplan la criptografía más como amenaza terrible en manos de terroristas y narcotraficantes que como instrumento garante de la intimidad de los ciudadanos. Surge aquí una situación en la que entran en conflicto los intereses del ciudadano y del Estado.

Un guante sirve para proteger las manos del frío o de arañazos cuando se trabaja o para proteger a un paciente de gérmenes y bacterias cuando se le está operando. Existen guantes de todas las formas, materiales y tamaños. Pero todos poseen un rasgo en común: proteger las manos. Precisamente esta característica impulsa a los ladrones y criminales a ponérselos cuando realizan sus delitos, con el fin de no dejar sus huellas dactilares como evidencia. ¿Y si el Estado decidiera prohibir los guantes por temor a que un criminal los utilice? Cualquier persona con un guante en su poder sería encarcelado. Si la aprobación de esta medida se antojase inaceptable a la opinión pública, imagine entonces que en su lugar, en vez de prohibir los guantes, se obligase a pegar en todos ellos una copia de plástico de la huella dactilar del propietario. De esa forma, si un individuo los utilizara al perpetrar sus crímenes, estaría dejando sus huellas.

El dilema que agita la conciencia de muchos gobiernos, especialmente tras el terrible atentado del 11 de septiembre del 2000 en Estados Unidos. Se ha reabierto la polémica de si prohibir por completo la criptografía, cuya misión es proteger la información, o recurrir a mecanismos tan ilógicos como el equivalente a imprimir huellas dactilares sobre los guantes.

¿Es el cifrado de la información un bien para los ciudadanos, en la medida en que les proporciona privacidad en sus comunicaciones digitales, o una amenaza para la sociedad, al permitir también a terroristas y delincuentes comunicarse en secreto y burlar las líneas telefónicas intervenidas por la policía? La libertad de cifrar facilitaría que todos, incluidos los criminales, envíen correos seguros y utilicen Internet de forma confidencial. Prohibir o restringir el uso de la criptografía impediría, en principio, que los criminales burlen las escuchas policiales, pero también permitiría que la policía, y en realidad cualquiera, espíe las comunicaciones de los ciudadanos.

Mientras un grupo aboga por la libertad de cifrado y otra, por las restricciones criptográficas, un tercer grupo propone soluciones de compromiso, como el almacenamiento centralizado de una copia de las claves de los usuarios. ¿Se imagina al Gobierno guardando una copia de la llave de su casa? ¿Qué ocurre si alguien roba esas llaves? ¿O si un empleado corrupto las copia a su vez y las vende al mejor postor?

Evidentemente, los criminales seguirán haciendo uso de la criptografía, pero sin entregar las claves al gobierno, mientras que el ciudadano honrado, pobre de él, las depositará sin oponerse, exponiendo su privacidad a todo tipo de ataques. Mientras los políticos polemizan, no se dude en utilizar la criptografía.

5.2.- Internet, Criptografía y la Política Exterior.

La tecnología de la criptografía se ha convertido en un gran campo de batalla en la evolución del comercio electrónico y del Internet. Al igual que en los Estados Unidos, los bancos y las corporaciones mexicanas están diversificando sus operaciones para ofrecer servicios financieros on-line, muchos de estos servicios están elaborados en base a programas de Internet, que venden las transnacionales americanas como Netscape y Microsoft.

Por asuntos de seguridad el Departamento de Comercio de los Estados Unidos, ha prohibido las exportaciones y ventas de programas codificados a países extranjeros para ser usados en comunicaciones a través de Internet, así como también a ciudadanos que no sean americanos o residentes en ese país.

El mundo industrializado parece profundamente dividido sobre el asunto, en el sentido de que sí los gobiernos pueden legalmente incursionar en las comunicaciones electrónicas de sus ciudadanos. Los mensajes por Internet son fáciles de interceptar. Muchos individuos y corporaciones están protegiendo la seguridad de sus comunicaciones y por ende las transacciones comerciales a través de equipos sofisticados y codificadores de información.

La Organización de Cooperación Económica y de Desarrollo (OCED), convoco a una reunión a finales de 1997 para debatir la proposición de los Estados Unidos, con el apoyo de Francia e Inglaterra, pero no del resto de los miembros de las naciones de la OCDE, a fin de configurar una política global con respecto a Internet, es decir, el de permitir a las agencias de seguridad penetrar la red y obtener información que les sea conveniente, con el propósito de combatir el crimen organizado, fiscalizar las operaciones de traficantes de drogas y controlar a los terroristas. El documento final de la OCDE deja en libertad a sus miembros para reglamentar o no, la tecnología de la criptografía, debido al hecho de que para Internet no existen fronteras nacionales.

Francia e Inglaterra declararon fuera de ley el uso privado de sistemas codificadores de información, mientras que otras naciones como Australia, Canadá, Dinamarca y Finlandia, están llevando a cabo políticas para proteger la privacidad de sus comunicaciones. Al principio Japón rechazó la proposición de los Estados Unidos, pero luego se supo que se había acercado a éste, mientras que Alemania quedó profundamente dividida al respecto.

Los oponentes al proyecto de resolución de los Estados Unidos reconocieron que, las resoluciones finales fueron una verdadera desilusión, pero que ese país tuvo más éxito en

concienciar con sus argumentos al resto de los países, que de lograr el respaldo de la conferencia para la obtención de sistemas criptográficos por parte de los fabricantes.

Las políticas nacionales en materia de criptografía pueden permitir el acceso legal a textos ordinarios, o a claves criptográficas, o a datos codificados, según el informe final de la reunión de los países miembros de la OCDE.

Muchos otros países, dentro o fuera de la OCED, todavía les queda mucho para enfrentar el problema de la codificación (data-scrambling). Los propios Estados Unidos tienen una política nacional algo contradictoria que permite a los ciudadanos de utilizar dentro de las fronteras nacionales sistemas criptográficos (data-scrambling software) que ellos deseen, pero a su vez restringe la exportación de tecnologías más modernas de para la codificación de datos.

De esta manera y para poder penetrar al mercado europeo, América Online y Netscape tuvieron que valerse de la empresa alemana Brokat Informationssysteme GmbH, especializada en sistemas de seguridad de forma criptográfica, exigidos por bancos y otras instituciones financieras de mercados de capitales, a fin de proteger sus transacciones financieras, en contra de una mafia organizada electrónicamente, ya que por resolución del Gobierno de los Estados Unidos, no pueden exportar sus programas codificados a Europa.

Hoy día, los programas codificados en Internet son cruciales, por ser la única manera de proporcionar seguridad y confiabilidad, así como proteger a compañías que transmitan informaciones sensitivas.

Netscape y otras empresas de los Estados Unidos, solamente pueden vender sus programas criptográficos de alta calidad en los Estados Unidos, mientras que compañías europeas especializadas en datos cifrados, obtienen permisos de importación del gobierno americano, para vender sus productos a clientes dentro del territorio de los Estados Unidos.

Esta discrepancia ha hecho que compañías americanas pierdan anualmente varios millones de dólares, y que no puedan competir libremente con otros gigantes de la computación europea como es el caso de la Siemens-Nixdorf, división de computadoras de la Siemens AG, quien recientemente comenzó a vender programas para servidores de Internet de alta seguridad, compitiendo así con los productos de Netscape.

Esta aparente contradicción no impidió, que en meses recientes, el gobierno norteamericano desarrollase una fuerte presión en el Congreso para llevar adelante su proposición. Bajo este esquema y para resolver algunos conflictos sobre la política para controlar las comunicaciones a través de Internet, circuló un proyecto de ley donde plantea la necesidad de controlar el uso doméstico de sistemas criptográficos (data-software), así como establecer un sistema nacional control criptográfico.

Aunque el mercado de programas criptográficos para correo electrónico a través de Internet de por sí es muy limitado, la clave es la venta de tecnología dentro de un amplio mercado, como lo es el

del comercio electrónico en constante crecimiento. La criptografía representa la primera línea de defensa, contra personas ávidas de obtener información sobre tarjetas de crédito, transacciones financieras, cuentas privadas en la bolsa de valores, compras con tarjetas de crédito, penetración en cuentas bancarias, etc., todas realizadas a través de Internet. De manera que la criptografía juega un papel muy importante en la venta de servidores de Internet, y de sistemas procesadores de alta confiabilidad para transacciones bancarias y financieras.

La razón por la cual Estados Unidos impide a compañías norteamericanas exportar programas criptográficos avanzados, es que las agencias como el FBI y la Agencia de Seguridad Nacional (NSA), temen que perderían posibilidad de intervenir en las comunicaciones entre personas sospechosas, como terroristas, criminales y traficantes de drogas. Aunque la política de los Estados Unidos no tiene como objetivo estorbar la investigación, desarrollo y divulgación de potentes programas para la codificación, ha creado, debido a su oposición, una oportunidad propicia, para que otras compañías fuera de sus fronteras se dediquen al desarrollo de programas criptográficos.

La tecnología se basa en una serie de algoritmos y fórmulas matemáticas, que son de acceso público y pueden buscarse y obtenerse en Internet. Para muchos ejecutivos en el área de las computadoras, el verdadero misterio consiste en el por qué, el gobierno de los Estados Unidos continúa restringiendo la exportación de tecnología codificada.

El esfuerzo internacional fracasó, debido a que ciertos países se negaron adoptar un enfoque común, brecha que ha permitido a personas u organizaciones inescrupulosas, buscar por medio de la criptografía sus propósitos fuera de ley, y los países individualmente, debido a una falta de leyes coherentes al respecto.

Los gobiernos pueden aún albergar esperanzas de imponer el acceso de las agencias encargadas de controlar el crimen organizado, para controlar los mensajes electrónicos, o para restringir la divulgación de software de alta tecnología de codificación, ya que nuevas y más recientes versiones, pueden ser desarrolladas y fácilmente transmitidas a través de Internet.

Aunque la OCED no tiene autoridad para imponer una política internacional al respecto, sus recomendaciones son usadas frecuentemente por las naciones miembros para elaborar sus propias políticas exteriores y comerciales.

5.3.- El Derecho a Cifrar.

Es común observar que las nuevas tecnologías de la información llevan asociados nuevos crímenes y, por tanto; la necesidad de nuevas leyes. Así vemos cómo el narcotráfico está utilizando las mismas tecnologías de la sociedad red para organizar sus turbios delitos: la misma Unión Europea acaba de aprobar un nuevo tratado de cibercrimen para facilitar la cooperación entre las

diversas fuerzas del orden de nuestra Comunidad en la investigación de los delitos surgidos del uso de las tecnologías digitales.

Extrañamente, casi todo el mundo parece olvidar que tras esas nuevas obligaciones y leyes debería haber también nuevos derechos, materiales y formales, que respondan al cambio de una sociedad industrial a una de la información. Aquí vamos a argumentar a favor de uno de estos nuevos derechos, el derecho a cifrar.

5.3.1.- La Sociedad Transparente.

La intimidad es el estado natural del ser humano. Si queremos hablar con un amigo de algo confidencial no tengo más que apartarme un poco de la multitud y podemos hablar tranquilamente sin miedo a ser oídos. Hablándole al oído tengo la garantía de que incluso personas cercanas no escucharán lo que le digo. El simple gesto de correr una cortina hace que mi dormitorio sea un espacio inobservable para terceros no deseados.

O al menos así era hasta nuestro salto a la sociedad de la información. Cada vez nos acercamos más a una sociedad transparente: una sociedad en la que todas nuestras acciones quedan registradas en los más variados dispositivos: cámaras, servidores de Internet, cuentas de correo electrónico, registros de llamadas telefónicas, etc.

No se trata simplemente de que cada vez las tecnologías invasoras de nuestra intimidad sean mejores, sino de algo mucho más radical: las nuevas tecnologías de la información y la comunicación en su estado natural ya son sistemas de espionaje. Cualquier servidor que ofrece páginas HTML por defecto graba absolutamente todos y cada uno de nuestros movimientos en un sitio Web: qué páginas hemos visitado, cuánto tiempo hemos pasado en ellas, a qué nuevo sitio nos hemos desviado, etc. etc. La necesidad de que nuestro celular sea localizado para recibir una llamada en cualquier momento hace que -por defecto- los proveedores de telefonía móvil mantengan bases de datos con las posiciones de nuestros teléfonos móviles (y por consiguiente las nuestras) las veinticuatro horas; la misma naturaleza del correo electrónico hace que todos los mensajes que recibimos existan en diferentes copias en muchos servidores; no hay ningún impedimento técnico para que el administrador de sistema lea nuestro correo, la compañía telefónica grabe nuestras llamadas o el proveedor de Internet cree un detallado perfil de cuáles son nuestros gustos y preferencias en Internet. Sólo la ética o el miedo al castigo de la ley evitan que estos atentados a nuestra intimidad no sean un lugar común.

Es este cambio tecnológico, que nos lleva a un cambio social, el que exige también cambios en nuestra concepción de los derechos humanos. De la sociedad opaca hemos pasado a la sociedad transparente. El derecho a la criptografía en una sociedad en la que bastaba cerrar las cortinas para tener intimidad era obviamente un sinsentido, así que no es de extrañar que no esté escrito en

ninguna declaración de derechos humanos o constitución. Pero la amenaza de la sociedad transparente hace obvia la necesidad de debatir sobre el derecho a cifrar.

5.3.2.- Definición de Intimidad.

¿Qué entendemos por intimidad? La mejor definición es: "el poder para controlar aquello que otros pueden saber acerca de ti". Esta definición es general, no está inmersa en el mundo de las comunicaciones, como lo están otras definiciones más en uso, y recupera perfectamente las ideas claves del derecho a la intimidad. A veces, se confunde "intimidad" con "secretismo". En la definición está muy claro que aquella persona que defiende su derecho a la intimidad no tiene por qué estar ocultando nada; simplemente ejerce su derecho a que los canales de información acerca de él o ella estén bajo su supervisión. No hace falta ser un criminal para correr las cortinas de nuestra habitación, ni tampoco ser terrorista para cifrar nuestros correos electrónicos.

El derecho a la intimidad es el poder para controlar aquello que otros puedan saber acerca de ti y para mantener privadas aquellas acciones que culturalmente una sociedad considera no es de buen gusto hacer públicas.

Contra los defensores de "privacidad" como término técnico indicamos que "intimidad" no se usa exclusivamente -como suelen argumentar- en el sentido de "mantener privadas aquellas acciones que culturalmente una sociedad considera no es de buen gusto hacer públicas".

5.3.3.- Raíces Filosóficas de la Intimidad.

Pero más allá de la constatación biológica en el ser humano de una necesidad innata de intimidad, tenemos que enfrentarnos al hecho de que cualquier sociedad humana necesita en ciertos momentos saber qué está haciendo una persona en concreto. Si la policía sospecha que preparo una acción terrorista, es razonable suponer que tiene derecho a violar mi intimidad y escuchar mis conversaciones telefónicas, leer mi correo electrónico y de papel, seguirme las veinticuatro horas, etc. Igualmente, si hay sospechas de que una persona da un trato vejatorio a su familia, parece razonable invadir la intimidad de esa familia para comprobar la veracidad de los hechos. Así pues, necesitamos encontrar algún principio general que nos permita justificar estas invasiones puntuales a la intimidad de ciertos individuos sin que el derecho general a disfrutarla se ponga en peligro. Se presentan tres concepciones diferentes de la intimidad que hacen posible estas invasiones.

En primer lugar tenemos la intimidad en sentido utilitario. Partiendo de la premisa biológica básica de la necesidad de mantener una esfera privada, en la que yo soy el único árbitro de quién ha de saber qué, la idea es minimizar la intrusión. El Estado, a través de sus diferentes agencias, ha de

intentar siempre reducir la invasión de nuestra esfera privada al mínimo. Entre dos formas de invasión elegir siempre la que afecte psicológicamente nuestra necesidad de "estar solos" al mínimo.

Seguidamente está la idea de intimidad como dignidad. Así, el derecho a la intimidad se seguiría de la idea general de derecho a mantener la dignidad humana. Así pues, cualquier invasión de nuestra esfera privada es un ataque a nuestra dignidad, lo sepamos o no, nos afecte psicológicamente o no. Por tanto, sólo cuando derechos más importantes están en peligro -vidas humanas en un atentado terrorista, la dignidad de la familia- está legitimado el estado a invadir la esfera privada de los sospechosos.

Finalmente, tenemos una concepción substantiva de la intimidad. La intimidad como el derecho a regular la capacidad del estado para intervenir en nuestra esfera privada.

Así pues, debemos escoger entre la intimidad como utilidad y la intimidad como dignidad. La elección no es fácil y dependerá básicamente del modelo de derechos humanos que uno tenga en mente. Si postulamos la dignidad humana como principio básico, claramente adoptaremos una visión de la intimidad como dignidad. Por el contrario, si tomamos un acercamiento más pragmático de los derechos, la posición utilitaria es sin duda más razonable.

5.3.4.- ¿Somos vigilados?.

¿Viola nuestra intimidad una vigilancia inobservada?. Si los espías son suficientemente cuidadosos, uno puede ser vigilado durante años sin saberlo. El mundo digital no hace sino acrecentar esta facilidad. De hecho, como recogen diversos documentos del Parlamento Europeo una red multinacional de espionaje, dirigida desde Estados Unidos por la NSA habría estado vigilando las comunicaciones telefónicas y electrónicas de forma masiva durante bastantes años.

La respuesta a este dilema depende, básicamente del modelo de intimidad que escojamos. Si nos inclinemos por la apuesta utilitarista de minimizar las invasiones, parece claro que "ojos que no ven, corazón que no siente" de forma que el Estado podría intervenir nuestras comunicaciones e invadir nuestra intimidad sin demasiados problemas, en tanto en cuanto lo hiciera de forma inobservable. Por el contrario, la defensa de la dignidad nos obligaría a rechazar cosas como ECHELON y derivados pues aunque nos pasara inadvertida la vigilancia, se vulneraría nuestra dignidad humana, al considerársenos sospechosos sin haber dado la menor prueba de serlo. En el mundo de la vigilancia electrónica masiva, uno es culpable hasta que se pueda demostrar que es inocente.

- ECHELON: Red de satélites espías conformada de forma secreta (hasta 1998) por los gobiernos de EUA, Inglaterra, Canadá y Australia, cuyo fin era interceptar y analizar de forma

pasiva, literalmente, todas las telecomunicaciones electrónicas, de voz y datos alrededor del mundo.

A nivel práctico observemos cómo intelectuales, ONGs, gobiernos, incluso la Unión Europea se han indignado en grado sumo ante la existencia de la red de espionaje mundial ECHELON. Si el modelo nuestro por defecto fuera el de la minimización, sería extraño que nos molestase tanto una invasión sistemática e indetectable como ECHELON.

Finalmente, no olvidemos que lo que hoy pasa sin detectar, puede descubrirse al día siguiente. De nuevo, ECHELON es un excelente ejemplo de ello.

5.3.5.- La Autorregulación de la Intimidad

Para entender el papel de la criptografía en todo esto necesitamos indicar una propiedad formal más de la intimidad, que normalmente pasa inadvertida y es muy poco comentada. Si recordamos nuestra definición de intimidad, allí hablábamos del "poder para controlar lo que los otros saben de ti". Muchos analistas pasan por el alto la cuestión de "poder para controlar", y simplemente piensan en la intimidad como una libertad negativa. "Dos conceptos de libertad": la idea de que te dejen en paz, de tener un espacio en el que uno no es observado.

Sin embargo, el derecho a la intimidad implica algo más: poder para controlar. Es decir, que sea el propio sujeto el que decida qué información revela, en cada momento y a quién. Si uno tomara una visión puramente negativa de la intimidad no habría nada malo en ello, incluso podría verse como positivo que el Estado regulara de forma clara " mantener privadas aquellas acciones que culturalmente una sociedad considera no es de buen gusto hacer públicas". Sin embargo, nadie lo considera así. Lo importante es que, de derecho, es el sujeto el que decide qué información de su vida se conoce y quién la recibe. Un estado que prohibiera que contáramos detalles de nuestra vida íntima a terceros, por muy buenas intenciones que tuviera, violaría nuestro derecho a la intimidad de la misma forma en que lo haría otro estado que monitorizara sistemáticamente nuestras conversaciones telefónicas sin tener una buena razón policial para hacerlo.

Resumiendo: Para respetar el derecho a la intimidad no es suficiente con garantizar que no hay terceros monitorizando: hay que darle al sujeto el poder de decidir qué información privada revela, cuando, de qué manera y a quien.

5.3.6.- El Derecho a Cifrar.

No resulta difícil entender el motivo de argumentar la existencia de un nuevo derecho, el derecho a cifrar. Revisemos las premisas:

- En el mundo digital, la posibilidad de que terceros espíen nuestras comunicaciones o monitoricen nuestros movimientos en la Red es más bien la norma que la excepción. Internet y las otras tecnologías de comunicación son básicamente inseguras y el usuario medio está totalmente indefenso ante asaltos de terceras personas.
- Existe claramente un derecho a la intimidad, recogido en las diversas constituciones democráticas y en las Declaraciones Universales y de la Unión Europea de los Derechos Humanos.
- Según se sigue de la definición de intimidad establecida es condición necesaria para disfrutar del derecho a la intimidad la posibilidad de que sea uno mismo el que decida la información que libera y a quien.
- Actualmente, la única forma en que un usuario de las tecnologías de la información que no dispone de un presupuesto multimillonario puede proteger directamente sus comunicaciones electrónicas es mediante la criptografía.

De estas cuatro premisas se sigue rápidamente la necesidad de instaurar la criptografía como derecho. Más específicamente, la criptografía actuaría como derecho material que garantiza la posibilidad de un derecho formal básico como el derecho a la intimidad.

Un crítico podría objetar que la persona que utiliza la criptografía para cifrar sus comunicaciones no sabe realmente que le están escuchando, no es equivalente a correr las cortinas si vemos a un desconocido enfrente de nuestra casa. Cifrar sistemáticamente las comunicaciones de uno equivale más bien a paranoia que a ejercicio de intimidad. Contra ese argumento debemos considerar que el "ciberespacio" debería ser considerado un espacio público.

Psicológicamente, cuando enviamos un correo electrónico o mantenemos una conversación telefónica en nuestra casa tenemos una falsa sensación de intimidad. Pero en realidad, esas comunicaciones electrónicas se desplazan por un ciberespacio público en el que resulta absurdamente fácil interceptar las comunicaciones. Por ello, es forzoso que para proteger nuestras comunicaciones utilicemos el mismo tipo de medidas que en los espacios públicos.

5.3.7.- Conflicto de Derechos.

Nos queda por examinar si el uso de la criptografía no pondría en peligro derechos más básicos, con lo que su defensa como derecho material que da sentido a la intimidad en la sociedad de la información quedaría ciertamente maltrecha.

Cualquier constitución o ley penal reconoce la necesidad de que el estado y las fuerzas del orden puedan intervenir nuestras comunicaciones y, en general, invadir nuestra intimidad, en casos de fuerza mayor, como establecer los culpables de un delito.

Por primera vez, la criptografía hace que sea imposible -una imposibilidad matemática-, contra la que el poder tecnológico, económico, legal o político del estado no puede hacer nada- que las fuerzas del orden interfieran nuestras comunicaciones electrónicas. Ante un mensaje cifrado en RSA o Rijndael, un policía no puede sino encogerse de hombros, pues no hay forma humana de descifrar lo que ahí se pone.

¿Es aceptable una restricción así a los poderes del estado? Algunos creen que no y de hecho comparan a los activistas en pro del cifrado de las comunicaciones electrónicas como las versiones digitales de la Asociación del Rifle en Estados Unidos, que defienden la libertad para adquirir y poseer armas. Los argumentos que hemos estado viendo aquí los comparan al pueril "Las armas no matan a la gente: la gente mata a la gente" que habremos oído más de una vez en los labios de los defensores de las armas en EUA.

El argumento no se sostiene, y que hay diferencias importantes entre el derecho a poseer armas y el derecho a la criptografía.

En primer lugar, observemos la diferencia entre objeto para cometer un delito y objeto que facilita la perpetración de un delito, ocultando su preparación. Son dos cosas muy diferentes. Ambas cuestionables, pero en grado y sentido diferente. Una pistola es un objeto para cometer directamente un crimen (un asesinato, un robo a mano armada), mientras que la criptografía serviría, como mucho, para camuflar las comunicaciones entre los criminales que serían previas al delito.

En segundo lugar, hay que distinguir también entre objeto con función de arma y aparato que ocasionalmente se utiliza como arma. Con la excepción de las escopetas de caza y las armas deportivas, las pistolas, subfusiles, ametralladoras y otras armas que la Asociación del Rifle quiere seguir manteniendo legales son objetos creados y diseñados exclusivamente con la misión de herir e incluso matar a otros seres humanos. Para ello se hicieron y ésa es su función básica.

Por el contrario, un cuchillo de cocina, un hacha para cortar leña o la proverbial sierra mecánica de las películas son objetos construidos con funciones pacíficas y que ocasionalmente, en manos criminales, pueden utilizarse para cometer delitos. Los tristemente célebres atentados del 11 de septiembre de 2001 contra las Torres Gemelas y el Pentágono muestran como un objeto tan útil como un avión de pasajeros puede llegar también a ser un arma de destrucción masiva.

Así pues, la criptografía no puede compararse a las armas por los motivos antes mencionados:

- Si pensamos en un uso delictivo, la criptografía no es el objeto utilizado para cometer el crimen -como una pistola- sino una forma de mantener en secreto la acción.

- La criptografía pública como PGP no fue creada con la intención de ayudar a los criminales, sino con el muy loable uso de proteger nuestra intimidad en el mundo de las tecnologías de la información y la comunicación.

De la misma forma en que no tendría sentido prohibir los aviones de pasajeros porque han sido utilizados como arma terrorista, tampoco suena razonable prohibir la criptografía porque los terroristas responsables de los atentados del 11 de septiembre usaran la criptografía para comunicarse.

5.3.8.- La Imposibilidad de Prohibir la Criptografía

Sin embargo, la no comparación de la criptografía con las armas no las exime automáticamente de la prohibición. El plutonio no es directamente un arma, tiene usos pacíficos, pero dada su evidente peligrosidad, su distribución está muy controlada y, desde luego, ningún particular puede argüir su "derecho" a tener plutonio. Más bien es al contrario: es un delito que un particular tenga plutonio, sea lo que sea lo que tenga en mente hacer con él. ¿No podría pasar lo mismo con la criptografía? ¿Que su uso en manos criminales y terroristas sea tan peligroso que, a la hora de la verdad, lo mejor fuera prohibirla, aunque tenga una importancia clara a la hora de garantizar la intimidad de las personas?

Prohibir la criptografía es una ilusión, algo imposible. Si los estados prohibieran su uso, se erosionaría gravemente la intimidad de los ciudadanos, mientras que terroristas y crimen organizado podría seguir utilizando esa tecnología fácilmente. La razón central es la inmaterialidad de las tecnologías criptográficas. Pensar que prohibir la criptografía es como prohibir las armas o el plutonio es caer en la falacia de confundir los bits con los átomos.

Nos referimos aquí a la obviedad de que es muy difícil prohibir una tecnología. En Europa, a pesar de que poseer un arma es mucho más complejo que en Estados Unidos y que ciertas armas no son accesibles legalmente al público general (como subfusiles o armas automáticas) lo cierto es que esas armas pueden conseguirse en Europa. La caída de la Unión Soviética, y la subsiguiente "aprehensión" de tecnología nuclear por la mafia rusa, hace que exista un mercado negro de plutonio. Sin embargo, no es menos cierto que en Europa hay muchas menos armas que en Estados Unidos y -por consiguiente- muchos menos crímenes violentos, y no es menos cierto que tener reservas de plutonio no está precisamente al alcance de cualquiera: sólo grupos muy bien preparados y financiados, como las bandas terroristas de Osama Bin Laden, serían capaces de disponer de semejante material nuclear. Pero la criptografía es algo muy diferente, es algo inmaterial, es software, ni siquiera software. En su esencia es simplemente matemáticas y por ello prohibirla es una ilusión.

Si mañana -sueño imposible- los gobiernos del mundo decidieran prohibir las armas automáticas, sin duda se seguirían cometiendo crímenes con ellas un tiempo. Después de todo, sería muy difícil

apropiarse de todas las armas que existen, y algunos seguirían quizás fabricándolas, pero a la larga, habríamos acabado, o reducido a un estado prácticamente anecdótico, los crímenes con armas automáticas. En el caso -imposible de todas formas- de que la OTAN consiguiera hacer desaparecer todos los programas de criptografía del mundo, nada impediría que los terroristas o el crimen organizado se fabricaran su propio software. Ese conocimiento es público, bien establecido, e imaginarse que algún día podría desaparecer de la Tierra por arte de magia es una ilusión.

Otra posibilidad es que, aunque la criptografía siga siendo asequible, la pena asociada a la posesión de este tipo de software disuadiera a los criminales de usarla. Sin embargo, como se ha argumentado varias veces, cuando hablamos de crímenes ya suficientemente penados como el terrorismo o el tráfico de drogas, el aumento de pena que la posesión de herramientas criptográficas implicaría sería tan ridículo que a estos criminales no les importaría lo más mínimo cifrar.

¿Vamos a abandonar nuestro derecho real a la intimidad en la sociedad de la información por la promesa ilusoria de seguridad? Lo cierto es que no hay forma de impedir que los criminales usen la criptografía. Por otro lado la criptografía no es un arma, sino una herramienta de comunicación que a veces puede usarse para facilitar el encubrimiento de un crimen, aunque no sea ésta su función. Finalmente, su prohibición efectiva es un imposible.

CONCLUSIONES

La criptografía se encuentra basada en su totalidad en principios matemáticos de los que podremos destacar:

- Teoría de la Información: Principios de difusión y confusión de la información.
- Teoría de Números: Generación de números primos y pruebas de primalidad, aritmética modular, exponenciación, logaritmos discretos, factorización de grandes números.
- Probabilidad y Estadística: Análisis de frecuencias y generación de números pseudoaleatorios.
- Teoría de la Complejidad Computacional: Análisis de Algoritmos, resolución de problemas np-complejos.

Un estudio serio de los algoritmos criptográficos se debe de realizar en base a sus principios matemáticos, de ahí la importancia para el diseñador de software conocer las bases matemáticas en que se sustentan los algoritmos criptográficos. Debido a que los algoritmos criptográficos actuales se utilizan en sistemas informáticos también se deben conocer las implementaciones óptimas desde el punto de vista computacional para así aumentar y sustentar su seguridad.

Aquí es muy importante hacer notar que todo aquel profesional que desee realmente aplicar y sustentar la seguridad de un sistema informático no únicamente se debe basar en la implementación tal cual de un algoritmo inmerso en un producto comercial sin el conocimiento teórico en el cual algoritmo criptográfico se sustenta. En el mercado existen múltiples productos de seguridad informática que esconden los algoritmos que emplean lo cual nos da una fuerte desconfianza de sus bases teóricas y por lo tanto de su seguridad, el profesional deberá desconfiar de estos productos no probados ni analizados por la comunidad criptográfica.

Como hemos visto los algoritmos simétricos de cifrado nos proporcionan una fuerte seguridad basada en los principios de difusión y confusión de la información, existen los basados en Redes de Feistel que no son mas que la aplicación reiterativa de cifrados producto clásicos, en este tipo de cifrados destaca DES que hasta mediados de los 90 ofrecía una buena seguridad pero con el considerable aumento de la capacidad de proceso de las computadoras su seguridad ha sido superada por medio de ataques de búsqueda exhaustiva de las claves. Ante este problema surge el algoritmo Rijndael como nuevo estándar criptográfico simétrico cuyas bases matemáticas se basan en un conjunto de reglas básicas de operaciones en un grupo finito $GF(2^8)$, Rijndael es muy eficiente y adaptable debido a sus distintas configuraciones del tamaño de la clave hasta de 256 bits, Rijndael aun se encuentra ganando la confianza de los desarrolladores de software pero todo indica su

aceptación y confianza. En resumen los algoritmos simétricos de cifrado son rápidos, eficientes, no aumenta el tamaño del criptograma y en caso de Rijndael seguros, pero tienen la gran desventaja de que la misma clave se usa para cifrar y descifrar, mucha de su fortaleza recae en el propio usuario que define la clave que en la mayoría de los casos es mal seleccionada.

Los algoritmos asimétricos de cifrado o de clave pública resuelven el problema de compartir la clave de cifrado lo que le da una enorme ventaja sobre los métodos simétricos, su seguridad se basa en la resolución de problemas matemáticos complejos, es decir, en la no existencia de algoritmos eficientes para la resolución de un problema matemático complejo dado, lo que no garantiza que en un futuro cercano si exista y por lo tanto su seguridad se encuentre entre dicho. Tienen como ventaja utilizar una pareja de claves de cifrado y descifrado lo que le da su gran utilidad para su uso en canales de comunicación inseguros como Internet, también se utilizan para realizar firmas digitales y autenticar al emisor de un mensaje, tienen como desventaja que son lentos y mucho más difíciles de implementar que los algoritmos simétricos, su par de claves no son escogidas arbitrariamente por el usuario sino generadas aleatoriamente cumpliendo con un importante número de condiciones matemáticas previas, otra desventaja es que el criptograma generado es mucho mas grande que el texto original lo que la una enorme inconveniente al momento de la transmisión. El algoritmo RSA es el más aceptado y matemáticamente más sencillo de implementar por lo que es importante su conocimiento. El diseñador de software debe de conocer todas sus ventajas he inconvenientes al momento de la implantación de los algoritmos de cifrado asimétricos para así darle el mejor provecho posible.

En la actualidad la mejor manera de obtener provecho de los algoritmos simétricos y asimétricos de cifrado es por medio de la combinación de sus virtudes, es decir, un sistema informático seguro combinará varios algoritmos simétricos y asimétricos de manera que conociendo a detalle sus ventajas y desventajas se obtenga el mejor provecho posible. El diseñador y desarrollador de software debe de conocer las distintas formas de conformar su solución de seguridad.

La infraestructura de claves públicas (PKI) es una solución combinada de algoritmos criptográficos, métodos y estándares mundialmente aceptados que interoperan de manera conjunta para ofrecer la mejor solución posible, su conocimiento es primordial en el diseño de sistemas de comercio electrónico y del sistema financiero, la infraestructura de claves públicas introduce los conceptos de certificados digitales, firma electrónica, Autoridades Certificadoras, así como el uso de distintos protocolos seguros de comunicación. El estándar X.509, el uso de SSL e IPSec son ejemplos de los estándares y protocolos actualmente aceptados en una PKI. El diseñador y desarrollador de software debe conocer como crear su infraestructura de claves públicas para así dar la mejor solución a sus necesidades.

La criptografía es una herramienta muy poderosa si es utilizada correctamente, los gobiernos han entendido su importancia y su lugar históricamente ganado, pero también existe la preocupación de

su mal uso por criminales, en algunos países existen normas estrictas sobre su uso, en México su normalización y reglamentación apenas comienza, actualmente en México la firma electrónica es legalmente reconocida como prueba o comprobante de contratos comerciales. El legítimo derecho que tiene todo ciudadano a proteger su privacidad no tiene porque contravenir el deber que tiene todo gobierno de proteger a sus ciudadanos contra actividades criminales.

APÉNDICE

A.- Certificados X.509.

1.- Tipos de Certificados.

Existen dos tipos de certificados principalmente: certificados de entidad destino y certificados de CA.

Los certificados de entidad destino los expide una autoridad de certificación a una entidad que no expide certificados para otra entidad.

Los certificados CA los expide una Autoridad de certificación a una entidad que también es autoridad de certificación, de manera que puede expedir certificados de entidad destino. Un certificado CA se distingue por la presencia de un campo denominado restricciones básicas, establecidas explícitamente para indicar que ha expedido un certificado CA.

Se pueden identificar varios casos de certificados CA:

- Certificados autoexpedidos: El certificados autoexpedido tiene los nombres del expedidor y del sujeto junto al nombre de la autoridad de certificación que expidió el certificado. Esta forma de certificado CA se puede utilizar para permitir operaciones especiales, como la validación de una nueva pareja de claves públicas cuando se presenta un evento del cambio de clave en la CA. En este caso, la nueva clave está dentro del certificado y la clave antigua se utiliza para firmar el certificado, permitiendo que la nueva clave CA sea de confianza.
- Certificados autoafirmados: Un certificado autofirmado es un caso especial del certificado autoexpedido. En este caso, la clave pública certificada en el certificado expedido por la Autoridad de certificación, corresponde a la clave privada utilizada para firmar el certificado, permitiendo que la nueva clave CA sea de confianza.
- Certificados cruzados: En un certificado cruzado, el sujeto y el expedidor son autoridades de certificación diferentes. Los certificados cruzados los utiliza una Autoridad de certificación para certificar una entidad de otra Autoridad de certificación.

2.- Formato del certificado.

La autoridad de certificación crea un certificado mediante la firma de una información recopilada acerca de la entidad. Esta información incluye la clave pública y el nombre distinguido de la entidad, y puede incluir un identificador único que contiene información adicional acerca de la entidad.

X.509 define la forma de un certificado expedido por una Autoridad de certificación con un nombre CA, para un sujeto con un nombre distinguido CA que utiliza la siguiente forma simbólica:

CA<<A>>=CA{V,SN,AI,CA,UCA,A,UA,Ap,T^}

En este caso, el certificado contiene la siguiente información:

V	El número de versión del certificado.
SN	El número de serie.
Ai	Identifica el algoritmo de firma que se utilizo para firmar el certificado.
CA	El nombre distinguido de la CA expedidora.
UCA	Un identificador único para la CA expedidora (opcional).
A	EL nombre distinguido del sujeto identificado por el certificado.
UA	Un identificador único para el sujeto (opcional).
Ap	La clave pública del Sujeto A.
T^	El periodo de validez del certificado descrito por una fecha inicial y una fecha de terminación, durante las cuales el certificado tiene validez.

X.509 también describe el formato de un certificado que utiliza una descripción ASN.1 que, incluso, es menos legible que la forma simbólica que menciono previamente.

Los campos estándar definidos para cada certificado incluyen lo siguiente:

- Versión Number (número de versión): Indica el formato y el contenido permisible definido dentro de un certificado de una versión en particular. La versión más reciente de los certificados X.509 es la numero 3. Aunque algunos certificados de la versión 1 se pueden ver ocasionalmente, los certificados de la versión 2 fueron una aberración a corto plazo que se reemplazo rápidamente con la versión 3.
- Serial Number (número de serie): El número de serie para cada certificado expedido por una CA particular es único.

- Signatura (firma): Identifica el tipo de función hash y el algoritmo de firma utilizado para firmar el certificado. El campo de la firma representa la descripción del algoritmo utilizado, lo mismo que la firma real generada para el certificado. Un identificador típico de un algoritmo de firma podría ser md5WithRSAEncryption que indica que el algoritmo hash utilizado fue md5 (Message Digest #.5 definido por RSA Labs) y el algoritmo descifrado utilizado fue RSA.
- Issuer (expedidor): Identifica la Autoridad de certificación que expidió y firmo el certificado.
- Validity (validez): Define las fechas de iniciación y terminación entre las cuales el certificado se puede considerar válido.
- Subject (sujeto): Nombre del individuo o entidad que se va a identificar con el certificado que corresponde a la clave pública que se encuentra en el certificado.
- SubjectPublicKeyinfo (información de la clave pública del sujeto): Contiene la clave pública que va a certificar el sujeto del certificado. Además de la clave pública también identifica el algoritmo para el cual se puede usar la clave pública.
- IssuerUniqueidentifier (identificador único del expedidor): Permite que el expedidor del certificado se identifique donde el nombre del expedidor puede haber sido reutilizado. Éste es un campo opcional y sólo se puede usar si el número de la versión del certificado es v2 o v3.
- SubjectUniqueidentifier (identificador único del sujeto): Permite que el sujeto del certificado sea identificado donde el nombre del sujeto pueda haber sido reutilizado. Éste es un campo opcional y solamente se puede usar si el número de la versión del certificado es v2 o v3.
- Extensions (extensiones): Permite codificar información adicional en los certificados sin requerir modificación en el formato del mismo. Las extensiones estándar las define X.509 y se describen en la siguiente sección. Además, las extensiones privadas las puede definir cualquier organización.

3.- Extensiones de Certificado.

Las extensiones de certificado permiten contar con información adicional específica para un certificado. El conjunto de extensiones definidas o aceptadas por una comunidad particular puede variar. Se pueden definir perfiles que describen el conjunto de extensiones que utilizan los participantes dentro de la comunidad y cuál es la interpretación que deberá poner en uso una extensión. Por ejemplo, el grupo de trabajo PKIX ha definido extensiones únicas para usar dentro de Internet.

Trataremos la diferencia entre extensiones críticas y no críticas, y después pasaremos a describir diferentes clases de extensiones.

0.3.1.1.- Indicadores de carácter crítico.

Las extensiones de certificado contienen un indicador que señala si la extensión debe ser considerada crítica.

El sentido general del indicador crítico es que, cuando se establezca como válido, indica que la extensión se debe procesar. Si un usuario de un certificado no reconoce o no puede procesar de alguna forma un certificado que contiene una extensión crítica el certificado debe ser considerado inválido. Las reglas del procesamiento para la semántica de algunas extensiones también se pueden ver afectadas por el uso del indicador crítico.

Si una extensión no esta marcada como crítica, el usuario del certificado puede pasar por alto la extensión. Incluso si la segunda comunidad no reconoce el significado del tipo de extensión.

0.3.1.2.- Extensiones de claves.

Esta clave de extensión incluye información acerca de las claves usadas por autoridades de certificación y entidades destino. También incluye restricciones sobre la manera como se pueden usar las claves.

0.3.1.3.- Authority Key Identifier.

La extensión AuthorityKeyIdentifier (identificador de la clave de autoridad) es una extensión no critica que se usa para especificar cuáles de las múltiples claves públicas que poseen una autoridad de certificación se emplean para la verificación de una firma en un certificado. Esto permite que la autoridad de certificación opere usando múltiples conjuntos de claves, y permite al usuario del certificado identificar cual es el conjunto de claves que debe utilizar.

La extensión AuthorityKeyIdentifier se debe de incluir en todos los certificados CA para ayudar en la construcción de una cadena de certificados, excepto cuando el certificado CA sea autofirmado.

La especificación explicita de las claves que se van a usar permite que las CA manejen casos como la superposición de claves para la Autoridad de certificación, en las que múltiples certificados CA y claves están en uso activo. También es útil para localizar claves que han expirado con fines de verificación de firmas de largo plazo, en las cuales se necesita la validación de la cadena para firmas de certificados, cuando todos los certificados y claves de la cadena pueden haber expirado.

La clave CA específica se puede identificar mediante la especificación de un valor para el campo KeyIdentifier de esta extensión o mediante uso de una combinación de los campos del número de serie del certificado CA (authorityCertSerialNumber) y el nombre de CA (authorityCertIssuer). Se pueden usar ambos mecanismos, pero la forma KeyIdentifier permite una identificación más específica cuando se construyen rutas de certificado.

0.3.1.4.- Subject Key Identifier

La extensión SubjectKeyIdentifier (identificador de clave de sujeto) identifica el conjunto especificó de claves al que pertenece una clave pública, cuando el sujeto del certificado tiene más de un conjunto de claves en uso. A diferencia de AuthorityKeyIdentifier, solamente se usa la forma KeyIdentifier. Esta extensión siempre está marcada como no critica

La extensión SubjectKeyIdentifier se debe de incluir en todos los certificados CA para ayudar en la construcción de una cadena de certificado. Es opcional, pero se recomienda para los certificados de entidad destino.

KeyUsage.

La extensión KeyUsage (uso de clave) define el propósito para el cual se puede usar la clave pública. Tiene las siguientes configuraciones posibles:

- KeyCertSign (clave para firma de certificado): La clave se usa para verificar firmas en certificados. Esta configuración sólo es válida en certificados CA.
- CRLSign (firma CRL): La clave se usa para verificar la firma CA en una lista de revocación de certificados.
- NonRepudation (aceptación): La clave se usa cuando ofrece un servicio de aceptación. Un tercero puede usar esta clave brindando una forma de servicio notarial, en la cual se verifica la firma de una entidad destino, para prevenir una posterior negación de que la firma se usó.
- DigitalSignature (firma digital): Identifica claves usadas con fines de firmas digitales, diferentes de los tipos de claves de firma que se presentaron previamente.
- KeyEncipherment (clave de cifrado): Esta clave se usa para cifrar otras claves o información de seguridad. Se podría usar para transportar con seguridad las claves. Cuando se especifica, se puede restringir para indicar que la clave de cifrado sólo se puede usar Encipher Only (sólo cifrar) o DecipherOnly (sólo descifrar).
- DataEncipherment (cifrado de datos): Esta clave se usa para cifrado de los datos del usuario. Las claves u otros datos de seguridad no están cubiertos por este tipo de clave. La clave KeyEncipherment se debe usar para estos propósitos.
- KeyAgreement (acuerdo de clave): Esta clave se usa en el proceso de establecer o llegar aun acuerdo sobre cuál es la clave que se debe de usar para operaciones futuras. El algoritmo Diffie-Hellman es un ejemplo de un protocolo de acuerdo de clave. La extensión KeyUsage se puede marcar como critica o no critica. Se recomienda que esta extensión se marque como critica.

Si la extensión se marca como critica, la clave se puede usar únicamente para su propósito designado. Cuando la indicación es critica, la CA está indicado que la clave se está certificando sólo

para ese propósito, un concepto importante si surgen temas de responsabilidad cuando una clave se usa para otros propósitos.

Si la extensión se marca como no crítica, se ubica en el estado de un campo de advertencia. La clave se puede usar para otros propósitos a discreción del usuario del certificado. El campo también se puede determinar como no crítico, si el propósito primario de la configuración es seleccionar el certificado correcto y la pareja de claves, cuando un usuario puede tener más de una.

0.3.1.5.- Extended key Usage.

La extensión ExtendedKeyUsage (uso extendido de la clave) permite especificar información de uso adicional para la clave. Cualquier organización puede definir la información específica de uso de la clave. Cualquier organización puede definir la información especifica de uso de clave y los identificadores correspondientes (se aplican las mismas reglas para registrar esos identificadores como otros objetos). Por ejemplo, Microsoft agrega identificadores Objeto (Object Identifiers, OID), que incluyen la especificación de:

- Uso de conexión de tarjeta inteligente.
- Uso de la autenticación del cliente.
- Uso de correo electrónico seguro.

Y muchos otros identificadores específicos de uso de clave, extendidos para la aplicación

La extensión se puede marcar como crítica o no crítica. En el primer caso, sólo se puede usar para el propósito específico. En el segundo, las indicaciones de uso de clave son recomendaciones. Periodo de uso de la clave privada.

Cuando se usan claves para firmas digitales, existen casos en los cuales el período de validez para la clave privada puede ser diferente del periodo de validez para el certificado. Con frecuencia, es necesario validar firmas digitales usando la clave pública certificada mucho después de que ya no es válido usar la clave privada para generar firmas. La extensión PrivateKeyUsage siempre se marca como no crítica.

La clave privada tiene un periodo de validez especificado como un campo notBefore (no antes) que indica la fecha más antigua desde cuando la clave privada se puede usar para generar un firma digital, y un campo noAfter (no después) que indica la última fecha en que se puede usar la clave privada. El tratamiento y uso de esa extensión varía con las diferentes comunidades PKI y debe definirse mediante directivas de seguridad y PKI relevantes.

4.- Extensiones de Directiva.

Las extensiones de directiva permiten la especificación y restricción de las directivas que se usan para definir el uso de certificados.

0.4.1.1.- Directivas de certificado.

Ésta es una lista de identificadores y calificadores de directiva que pertenecen al uso del certificado. Los identificadores de directiva se pueden usar en el proceso de la validación de una ruta del certificado. Si la extensión de directiva se especifica dentro de un certificado CA expedido por otra Autoridad de certificación, identifica las condiciones de directiva bajo las cuales se puede usar esa ruta de certificado.

La lista de identificadores de directiva se encuentra en el campo policyIdentifier (identificador de directiva) de la extensión y es un identificador de una directiva de certificado en particular. Los identificadores d directiva se pueden crear según los requiera cualquier organización y se pueden registrar ante su entidad favorita nacional para estándares (ANSI en Estados Unidos), a fin de permitir identificadores únicos. Los calificadores se pueden aplicar opcionalmente a cualquier identificador de directiva especifico. Su busca que los calificadores transporten información legible para una persona o apuntadores, como una noticia de usuario o un URL, apuntando hacia una versión legible de la directiva.

Cuando se está realizando la validación de ruta, se pueden usar identificadores de directiva, pero no calificadores de directiva. Los calificadores de directiva que se encuentren durante el proceso se pueden presentar a los usuarios del certificado para determinar la aplicabilidad de la ruta que se validó.

Si la extensión se marca como crítica, el certificado únicamente se puede usar en cumplimiento de los requerimientos de directivas especificadas. La interpretación de cualquier calificador se puede definir mediante la directiva aplicable.

Si la extensión de marca como no crítica, el uso del certificado no necesariamente está restringido por las directivas de la lista. Los calificadores de directiva se pueden interpretar o pasar por alto, a discreción del usuario del certificado.

Una CA puede hacer referencia al identificador especial de directiva anyPolicy, para indicar que se puede confiar en el certificado con el fin de usarlo en todas las directivas posibles.

0.4.1.2.- Policy Mappings.

La extensión PolicyMappings (mapa de directiva) únicamente es válida dentro de certificados CA. Los mapas de directiva se usan para crear una correspondencia entre directivas de certificado

(especificadas por identificadores de directiva), definidas por autoridades de directiva en diferentes dominios de directiva.

Indica que la Autoridad de certificación expedidora ha reconocido que la CA sujeto que está certificando tiene un identificador de directiva que se considera equivalente a un identificador de directiva reconocido por la CA expedidora. La extensión consta de una lista de parejas de identificadores de directiva especial, anyPolicy, no es válido para los mapas de directiva.

La extensión se puede marcar como crítica o no crítica. Se recomienda marcarla como crítica cuando no hay claridad absoluta del significado que tiene una extensión de mapa de directiva no crítica. Esta extensión se debe marcar como no crítica.

5.- Extensiones Subject e Issuer Information.

Estas extensiones soportan el uso de formas alternativas de nombres para sujetos y expedidores de certificados. Para el uso de correo electrónico en Internet o nombres de dominio se han implantado diferentes esquemas de nombres, direcciones X.400 y nombres EDI. También son nombres válidos los identificadores uniformes de recursos (nombres de recurso de la Web) y direcciones IP. En general, es posible usar cualquier esquema estructurado de nombres.

0.5.1.1.- Subject Alternative Name.

La extensión SubjectAlternativeName (nombre alternativo del sujeto) cuenta una lista de diferentes formas de nombres que se pueden usar para identificar al sujeto del certificado. Si la extensión se marca como crítica, el campo Subject puede contener un nombre nulo y el usuario del certificado debe entender y procesar, por lo menos, uno de los nombres en la lista SubjectAlternativename. Cualquier nombre que no se reconozca se puede pasar por alto. Si el campo del nombre del sujeto es nulo, esta extensión se debe marcar como crítica.

0.5.1.2.- Issuer Alternative Name.

La extensión IssuerAlternativeName cuenta con una lista de diferentes formas de nombres para identificar al expedidor del certificado. Si la extensión está marcada como crítica, el campo del sujeto puede contener un nombre nulo y el usuario del certificado debe procesar, por lo menos, uno de los nombres en la lista. Cualquier nombre que no se reconozca se puede pasar por alto. Esta extensión debe marcarse como no critica.

0.5.1.3.- Subject Directory Attributes.

La extensión SubjectDirectoryAttributes (atributos de directorio del sujeto) es una extensión no crítica que permite la inclusión de atributos de directorio que son aplicables al sujeto del certificado.

6.- Extensiones de restricción de ruta de certificado.

CertificationPathConstraints (restricciones de ruta de certificación) se usan cuando se procesan rutas de certificado. Las restricciones se aplican a los tipos de certificados que una CA certificada puede expedir a los tipos que se permiten en certificados subsiguientes en una ruta de certificación.

0.6.1.1.- Basic Constraint.

La extensión BasicConstraints (restricciones básicas) permite identificar los certificados CA. Ésta extensión cuenta con campos para CA y PathLenConstraints (restricción para longitud de ruta). Una entidad cuya clave pública está certificada en un certificado en donde un campo de restricciones básicas de CA esta configurado como verdadero, para permitirle firmar certificados

Si el campo CA está configurado como verdadero, el campo PathLenConstraints también se puede configurar para indicar el número máximo de certificados CA que pueden seguir en la ruta de certificación. Si la longitud de ruta está en 0, la CA sujeto solamente para expedir certificados de entidad destino y no puede expedir certificados para otras CA.

X.509 recomienda que la extensión se marque como crítica, pues, de otro modo, las entidades destino pueden ser autorizadas inadvertidamente para expedir certificados. Si el campo se configura como falso, la clave pública dentro del certificado no se puede usar para verificar firmas en otros certificados. Si el campo se configura como falso, la clave pública dentro del certificado no se puede usar para verificar firmas en otros certificados. Si el campo CA está configurado como verdadero y la longitud de ruta está especificada, los usuarios del certificado deben verificar la ruta de certificación de acuerdo con el valor de la longitud de ruta.

Si no se encuentra esta extensión o está marcada como no crítica, el certificado se debe considerar un certificado de entidad destino, y la clave pública no se pude usar para verificar firmas en otros certificados. Esta extensión se debe como crítica en un certificado CA:

0.6.1.2.- Name Constraints.

La extensión NameContraints (restricciones de nombre) define los espacios del nombre con los cuales los campos Subject o SubjectNameAlternative deben cumplir en los certificados subsiguientes. La extensión permite la especificación de darles a los árboles un nombre que se debe de incluir o excluir. La extensión se puede marcar como crítica o no critica.

0.6.1.3.- Policy Constraints.

La extensión PolicyConstraints (restricciones de directiva) le permite al usuario usar el campo requiereExplicityPolicy para indicar que los certificados subsiguientes en una ruta de certificación deben contener identificadores de directivas aceptables.

En la extensión, el campo inhibitPolicyMapping(inhibir mapa de directiva) le permite al usuario evitar el uso de mapas entre diferentes directivas para certificados subsiguientes en una cadena de certificación. Por lo común, esto se implanta cuando las cadenas de certificados cruzan dominios de directivas, u se considera improbable que cualquier operación de mapas de directiva sensible correspondiente tenga un magnifico resultado.

La extensión se puede identificar como crítica o no crítica, pero se recomienda dejarla en crítica.

Las CA no deben expedir certificados cuando las restricciones de directiva están en una secuencia nula, es decir, se debe especificar inhibitPolicyMapping o requireExplicitPolicy.

Inhibit Any Policy Constraint.

La extensión InhibitAnyPolicyConstraint (inhibir cualquier restricción de directiva) permite la exclusión del uso del identificador de directiva anyPolicy. Esto permite el control de una CA usando cualquier directiva, en oposición a una directiva explícita. Se puede fijar como crítica o no crítica.

Formato de la lista de revocación de certificados.

En la revocación de certificado, como cuando un certificado queda inválido por una autoridad expedidora, antes de su fecha de expiración normal, por razones como compromiso de clave o un cambio en el estado del propietario del certificado. Se debe suministrar un mecanismo para permitir a los usuarios del certificado revisar el estado de revocación del certificado.

X.509 permite tres casos:

- El certificado no se permite revocar.
- La autoridad de certificación que expidió el certificado lo revoca
- Una Autoridad, diferente a la que expidió el certificado lo revoca porque la autoridad de certificación que lo expidió le ha delegado la responsabilidad de la revocación.

El mecanismo de revocación especificado por X.509 es el uso de lista de revocación de certificados (CRL), aunque la especificación permite utilizar esquemas alternativos.

X.509 establece una distinción entre la fecha y la hora cuando una Autoridad de certificación revoca un certificado, y la fecha y la hora cuando se publicó por primera vez el estado de revocación. La fecha de la revocación real se especifica junto la entrada del certificado en la CRL. La fecha de noticia de la revocación se especifica en el encabezado de la CRL cuando se pública

. El sitio de la información de la revocación puede variar para diferentes autoridades de certificación. El certificado mismo puede contener un apuntador hacia donde se localiza la revocación, alguna forma de direccionamiento indirecto de puede implantar donde una lista de revocación puede señalar el usuario del certificado hacia una localización diferente. El usuario del certificado puede reconocer de un directorio u otro repositorio o mecanismo donde se pueda obtener la información de la revocación, con base en la información de la configuración establecida durante la iniciación.

Para mantener la consistencia y las condiciones de auditoria, a la Autoridad de certificación se le exige que:

- Mantenga un registro de auditoria de la revocación del certificado.
- Suministre información sobre el estado de la revocación.
- Publique las CRL, incluso cuando la CRL es una lista vacía (suponiendo que use las CRL de alguna manera).

Los campos estándar definidos para la lista de revocación de certificados incluyen lo siguiente:

- Versión number (numero de versión): la versión más reciente definida para actualizar es la v2. Las CRL que contienen extensiones críticas se deben marcar como versión 2.
- Signatura (firma): Identifica el tipo de función hash y algoritmo de firma usado para firmar la lista de revocación.
- Issuer (expedidor): El nombre de la autoridad que expidió y firmo la lista de revocación.
- ThisUpdate (esta actualización): Define la fecha y la hora en que se publicará la siguiente lista de revocación.

Los certificados que han sido revocados por una Autoridad de certificación se listan como una serie de certificados revocados. Cada entrada identifica el certificado a través de su número de serie e incluye la fecha de revocación, especificando la fecha y hora cuando la autoridad de certificación revocó el certificado. Opcionalmente, cada entrada puede especificar extensiones (crlEntryExtensions) que ofrecen información adicional acerca del certificado revocado.

La lista de revocación también puede ofrecer campos de extensión opcionales, que suministran información adicional sobre la revocación.

7.- Lista de revocación del certificado y extensiones de entrada CRL.

Además de los campos estándar definidos para un CRL, descrita anteriormente, existen extensiones que se aplican a la CRL como un todo y a entradas individuales dentro de la CRL. Las siguientes secciones definen estas extensiones y presentan una lista de para que se usan.

0.7.1.1.- Authority Key Identifier.

La extensión AuthorityKeyIdentifier (identificador de clave de autoridad) es una extensión no crítica usada para especificar que una clave pública en particular, propiedad de una Autoridad de certificación, se usa para verificación de una firma en una CRL. Esto permite que la Autoridad de certificación opere usando múltiples conjuntos de claves y permite que el usuario de un certicado identifique explícitamente cuál es el conjunto de claves que deberá usar para validar la firma en la CRL. Esta extensión se debe incluir en todas las CRL y se debe marcar como crítica.

La misma justificación para usar la extensión AuthorityKeyIdentifier es aplicable a los certificados.

Cuando este campo se usa para un certificado, la clave de la CA especifica se puede identificar especificando un valor para el campo KeyIdentifier (identificador de clave) de esta extensión o usando una combinación de los campos de número de serie del certificado CA, authorityCertSerialNumber, y de nombre de la CA, authorityCertIssuer. Ambos mecanismos se pueden usar, pero la forma keyIdentifier permite la identificación más especifica cuando se construyen rutas de certificados.

0.7.1.2.- Issuer Alternative Name.

La extensión IssuerAlternativeName (nombre alternativo del expedidor) cuenta una lista de formas de nombres diferentes que se pueden usar para identificar al expedidor de la CRL. Si la extensión se marca como crítica, el campo Expedidor puede contener un nombre nulo y el usuario del certificado debe procesar, por lo menos, uno de los nombres de la lista. Cualquier nombre que no se reconozca se puede pasar por alto. Esta extensión se debe marcar como crítica.

CRL number.

La extensión CRLNumber es una extensión CRL que contiene un número de secuencia que permite a un usuario de CRL determinar si se han visto todas las CRL anteriores a la actual. El número aumenta secuencialmente. La extensión siempre se marca como no crítica.

0.7.1.3.- Reason Code.

La extensión ReasonCode (código de razón) es una extensión de entrada CRL que identifica la razón para la revocación del certificado. Los códigos de razón definidos actualmente incluyen los siguientes:

- KeyCompromise (compromiso de clave): Se cree que la clave principal del sujeto del certificado ha sido comprometida. Este código de razón es aplicable a certificados de entidades destino.
- CACompromise (compromiso de CA): Se cree que la clave privada de una CA ha sido comprometida. Este código de razón se usa cuando se revocan certificados CA.
- AffiliationChanged (cambio de afiliación): Se ha cambiado el nombre del sujeto u otra información en el certificado. Este código de razón no implica que la clave privada ha sido comprometida.
- Superseded (superado): Este certificado ha sido superado por un certificado más reciente. Este código de razón no implica que la clave privada ha sido comprometida.
- CessationOfOperation (cesación de operación): El certificado ya no se requiere para el propósito para el que se expidió. Este código de razón no implica que la clave privada ha sido comprometida.
- PrivilegeWithdraw (retiro de privilegio): Se ha retirado un privilegio que se especifico dentro de un certificado.
- CertificateHold (retener certificado): El certificado ha sido suspendido efectivamente o se deja en espera. Si se usa este código de razón, se puede especificar la extensión HoldInstrutionCode (conservar el código de instrucción). Los certificados que han sido suspendidos pueden ser revocados o liberados más adelante, y la entrada retirada de la CRL.
- RemoveFromCRL (retirar de CRL): Las CRL delta usan explícitamente este código de razón para indicar que un certificado ha expirado o que ha sido liberado del estado de retención.

La extensión ReasonCode siempre se marca como crítica.

0.7.1.4.- Hold Instruction Code.

La extensión HoldInstructionCode es una extensión de entrada de una CRL que permite un código de instrucción que identifica cómo procesar un certificado que tiene un código de razón de retener certificado. En X.509 no existen códigos de instrucción estándar, aunque el grupo de trabajo PKIX ha especificado algunos. Las extensiones siempre se consideran no críticas.

0.7.1.5.- Invalidity Date.

La extensión Invaliditydate (fecha de no válido) es una extensión de entrada CRL que identifica la fecha en que se sabe o sospecha que una clave privada quedó comprometida. La extensión siempre es no crítica.

0.7.1.6.- CRL Scope.

La extensión CRLScope (alcance CRL) es una extensión CRL que define el alcance del distintos tipos de las CRL, incluidas:

- CRL simples: Ofrecen información de revocación acerca de certificados expedidos por una Autoridad de certificación única.
- CRL indirectas: Suministran información de revocación de múltiples autoridades de certificación.
- CRL delta: Actualiza información de revocación expedida previamente.
- CRL delta indirectas: Ofrecen información de revocación que actualiza múltiples CRL base, expedidas por una o más autoridades de certificación.

0.7.1.7.- Status Referral

La extensión StatusReferral CRL (CRL de referencia de estado) se usa para suministrar información enlazada que conducirá a los usuarios de certificado a una fuente de información de revocación. Si se usa la extensión StatusReferral, la CRL no contiene noticias de revocación. Además, se requiere que los usuarios del certificado pasen por alta esta CRL como una fuente de noticias de revocación. La información dentro de la extensión (o algún otro mecanismo fuera de enlace) se debe usar para localizar una fuente real de noticias de revocación.

La extensión StatusReferral puede contener una lista de CRL e información acerca del estado de cada CRL. Este mecanismo se podría usar para públicar información acerca de las CRL (como la hora de la última actualización), que permite a los usuarios juzgar la aplicabilidad de las CRL que previamente se han almacenado. La información podría resultar en una solicitud de descarga de una CRL que contenga noticias de revocación.

Alternativamente, la extensión StatusReferral se puede usar como un mecanismo de redirección. Por ejemplo, debido a los cambios de configuración, la fuente original de información de revocación identificada en un certificado puede haber cambiado. La extensión StatusReferral permite señalar una nueva fuente, sustituyendo la CRL original en ese lugar.

La extensión se ha diseñado para ser extensible, algo que permitiría, en el futuro, hacer referencia a (o posiblemente actualizar) esquemas de revocación que no usan CRL.

Esta extensión siempre está marcada como crítica para evitar que los usuarios de certificados que no pueden interpretar la extensión se basen erróneamente en el contenido de la CRL para noticias de revocación. A los usuarios de certificados que no reconocen la extensión se les pide pasar por alto la CRL, evitando su uso como una fuente de noticias de certificados.

CRL Stream Indetifier

La extensión CRLStreamIdentifier (identificador de corriente CRL) es una extensión no crítica que permite que una Autoridad de certificación utilice múltiples listas de revocación de certificados. Cada Autoridad de certificación debe asignar un identificador único para cada corriente de CRL que desee mantener. La combinación de las extensiones CRLNumber y CRLStreamIdentifier permite crear un identificador único para cualquier CRL de cualquier Autoridad de certificación.

Ordered List.

La extensión OrderedList (lista ordenada) es una extensión no crítica que permite el orden de los certificados dentro de la lista RevokedCertificates (certificados revocados) que se va a especificar. Los certificados se pueden listar en secuencia ascendente usando AscSerialNum (número de serie ascendente) para listarlos basados en el campo de número de serie de cada entrada en la lista o usando AscRevDate (fecha de revocación ascendente) para ordenar las entradas con base en el campo RevocationDate (fecha de revocación).

Si esta extensión no está presente, no se deben hacer suposiciones en cuanto al orden de la lista RevokedCertificates.

0.7.1.8.- Delta Information.

La extensión DeltaInformation (información delta) es una extensión no crítica que identifica que las CRL delta que también están disponibles corresponden a la CRL que contiene la extensión. La extensión sólo se puede usar en las CRL que no son delta e identifica el sitio en donde se pueden localizar las CRL delta. Además, la extensión puede especificar de manera óptima la hora en que se expedirá la siguiente CRL delta.

8.- Puntos de distribución de CRL y Extensiones de CRL delta.

Dado que una CRL estándar contiene todos los certificados revocados que no han expirado, es probable que se vuelva muy grande. Cargar las listas de revocación de certificados para los usuarios del certificado puede ser un proceso lento en conexiones de red con bajo ancho de banda. Los requerimientos de memoria pueden restringir a los usuarios del certificado, al punto de que no sea posible procesar una lista de revocación larga.

Se han creado dos alternativas para permitir que la información re revocación sea más flexible. Las CRL delta permiten la distribución de las actualizaciones de revocación donde se propagan las listas de cambios, en lugar de publicar toda lista de noticias de revocación. Las CRL con puntos de distribución CRL se pueden usar para identificar un sitio en el cual se pueda encontrar la información de revocación de cada certificado. Éste puede ser un directorio en el que cada entrada de certificado

contiene un atributo CRL que indica al estado de revocación para el certificado o un sitio en el cual se pueda encontrar una CRL.

De las extensiones que se presentan aquí, las siguientes son extensiones de certificado:

- CRLDistributionPoints.
- FreshestCRL.

Las siguientes son extensiones CRL:

- IssuingDistributionPint
- DeltaCRLIndicator.
- BaseUpdate

La siguiente es una extensión de entrada CRL:

- Certificate Issuer.

9.- Extensiones de Certificado.

Las siguientes son extensiones específicas para certificados:

0.9.1.1.- CRL Distribution Points.

La extensión CRLDistributionPoints (puntos de distribución de CRL) es una extensión de certificado que identifica a los puntos de distribución de la CRL que el usuario del certificado deberá usar para verificar el estado de revocación de este certificado. La extensión se puede marcar como crítica o no crítica. En el primer caso, el usuario del certificado debe recuperar y verificar una CRL, desde uno de los puntos de distribución de la lista.

Los puntos de distribución se identifican como una lista de entradas que contienen un DistributionPointName (nombre de punto de distribución) que especifica los nombres de los puntos de distribución que se pueden usar para buscar una CRL. El nombre del expedidor de la CRL también se suministra en el campo CRLIssuer, y es el nombre predeterminado que se usa para buscar la CRL si no se suministra explícitamente DistributionPointNames.

Cada entrada también identifica los tipos de información de revocación soportados por el punto de distribución referenciado. Si se omite el campo ReasonFlag (indicador de razón), la CRL que suministra el punto de distribución contendrá una entrada de revocación para cuando este certificado

se revoque, sin considerar el motivo de la revocación. El campo ReasonsFlag soporta los siguientes indicadores como razones de revocación.

- Unused.
- KeyCompromise.
- AddfiliationChange.
- CACompromise.
- Superseded.
- CessationOfOperation.
- CertificateHold.
- PrivilegeWithdrawn.
- AACompromise.

0.9.1.2.- Freshest CRL.

La extensión Freshest CRL es una extensión de certificado que identifica la CRL frehest que se debe usar cuando se verifica el estado de revocación del certificado. Si se marca como crítica, el usuario del certificado debe obtener y revisar la CRL indicada antes de usar en certificado. Si se marca como no crítica, el usuario del certificado puede hacer uso de cualquier esquema disponible localmente para verificar el estado de revocación del certificado.

10.- Extensiones CRL.

Las siguientes son extensiones específicas de CRL.

0.10.1.1.- Issuing Distribution Point.

La extensión IssuingDistributionPoint (punto de distribución de expedición) es una extensión de CRL que identifica el punto de distribución que expidió esta CRL. Además, la extensión identifica el tipo de información de revocación que se transporta en la CRL. Esto permite un tipo de información de revocación que se transporta en el punto de distribución. Si el campo se omite, la CRL contiene toda la información de revocación aplicable en el punto de distribución.

La información de revocación posible representada en la CRL se identifica en el campo IssuingDistributionPointsSyntax e incluye los siguientes indicadores:

- OnlyContaintsUserCerts (sólo contiene certificados de usuario): La información de revocación se lleva solamente para certificados de entidad destino.
- OnlyContaintsAuthorityCerts (solo contiene certificados de autoridad): La información de revocación se transporta solamente para certificados CA.
- OnlySomeReasons (solo algunas razones): El indicador OnlySomReasons es una lista que, si se presenta, incluye el conjunto de códigos de razón de revocación.
- IndirectCRL (CRL indirecta): Si se establece, esto indica que la CRL puede contener noticias de revocación de autoridades diferentes a las del expedidor del la CRL.
- OnlyContaintsAttributeCerts (solo contiene certificados de atributo): La CRL se usa para llevar noticias de revocación para un tipo especial de certificados que se usan en sistemas de administración de privilegios, conocidos como certificados de atributo.

Delta CRL indicador.

La extensión Delta CRL indicator es una extensión crítica CRL que identifica esta CRL como una CRL delta, en lugar de una CRL base. La CRL base correspondiente se identifica usando un numero CRL expedido por la Autoridad de certificación. Debe contener todos los certificados revocados para un alcance en particular, que forma el contexto base para las CRL deltas subsecuentes que se expidan

0.10.1.2.- Base Update

La extensión Base Update es una extensión CRL no crítica que se usa dentro de las CRL delta. Indica la fecha de iniciación.

Extensiones de entrada CRL.

Las siguientes extensiones son específicas para las entradas CRL.

11.- Certificate Issuer

La extensión CertificateIssuer (expedidor de certificado) es una extensión de entrada CRL crítica que identifica al expedidor del certificado responsable por la entrada en la CRL. Se usa cuando el indicador IndirectCRL se fija en la extensión IssuingDistributionPoint (en la CRL) para especificar que se soportan CRL indirectas. En el caso de CRL indirectas, múltiples CA suministran contribuciones, de modo que es necesario identificar explícitamente a la CA responsable.

B.- Leyes y Normas Mexicanas Referentes a la Seguridad Informática.

1.- Código Penal Federal

Libro Segundo.

Título Noveno Revelación de Secretos y Acceso Ilícito a Sistemas y Equipos de Informática.

Capítulo II Acceso Ilícito a sistemas y Equipos de Informática.

Articulo 211 bis 1

Al que sin autorización modifique, destruya o provoque perdida de información contenida en sistemas o equipos de informática protegidos por algún mecanismo de seguridad, se le impondrá de seis meses a dos años de prisión y de cien a trescientos días multa.

Al que sin autorización conozca o copie información contenida en sistemas o equipos de informática protegidos por algún mecanismo de seguridad, se le impondrán de tres a un año de prisión y de cincuenta a ciento cincuenta días de multa.

Articulo 211 bis 2

Al que sin autorización modifique, destruya o provoque perdida de información contenida en sistemas o equipos de informática del estado, protegidos por algún mecanismo de seguridad, se le impondrá de uno a cuatro años de prisión y de doscientos a seiscientos días multa.

Al que sin autorización conozca o copie información contenida en sistemas o equipos de informática del estado, protegidos por algún mecanismo de seguridad, se le impondrán de seis a dos años de prisión y de cien a trescientos días de multa.

Articulo 211 bis 3

Al que estando autorizado para acceder a sistemas y equipos de informática del estado, indebidamente modifique, destruya o provoque perdida de información que contengan, se le impondrán de dos a ocho años de prisión y de trescientos a novecientos días de multa.

Al que estando autorizados para accedes a sistemas y equipos de informática del estado, indebidamente copie información que contengan, se le impondrán de uno a cuatro años de prisión y de ciento cincuenta a cuatrocientos días de multa.

Articulo 211 bis 4

Al que sin autorización modifique, destruya o provoque perdida de información contenida en sistemas o equipos de informática de las instituciones que integran el sistema financiero, protegidos por algún mecanismo de seguridad, se le impondrán de seis meses a cuatro años de prisión y de cien a seiscientos días de multa.

Al que sin autorización conozca o copie información contenida en sistemas o equipos de informática de las instituciones que integran el sistema financiero, protegidos por algún mecanismo de seguridad, se le impondrán de tres meses a dos años de prisión y de cincuenta a trescientos días de multa.

Articulo 211 bis 5

Al que estando autorizado para acceder a sistemas y equipos de informática de las instituciones que integran el sistema financiero, indebidamente modifique, destruya o provoque perdida de información que contengan, se le impondrán de seis meses a cuatro años de prisión y de cien a seiscientos días multa.

Al que estando autorizado para acceder a sistemas y equipos de informática de las instituciones que integran el sistema financiero, indebidamente copie información que contengan, se le impondrán de tres meses a dos años de prisión y de cincuenta a trescientos días multa.

Las penas previstas en este artículo se incrementaran en una mitad cuando las conductas sean cometidas por funcionarios o empleados de las instituciones que integran el sistema financiero.

Articulo 211 bis 6

Para efectos de los artículos 211 Bis 4 y 211 Bis 5 anteriores, se entiende por instituciones que integran el sistema financiero, las señaladas en el artículo 400 Bis de este código.

Articulo 211 bis 7

Las penas previstas en este artículo se aumentaran hasta en la mitad cuando la información obtenida se utilice en provecho propio o ajeno.

2.- Ley Federal de Protección de Datos Personales (Iniciativa).

(Solo se muestran los artículos e incisos importantes para la seguridad y protección de datos)

Artículo 2.

1. Esta ley es aplicable a los datos de carácter personal que figuren en archivos, registros, bancos o bases de datos de personas físicas o jurídicas, públicas o privadas, y a todo uso posterior, incluso no automatizado, de datos de carácter personal registrados en soporte físico susceptible de tratamiento automatizado.

2. Esta ley no es aplicable a los archivos, registros, bases o bancos de datos:

II. Cuyo titular sea una persona física y tengan un fin exclusivamente personal;

Artículo 4.

1. Para los efectos de esta ley se entiende por:

I. Datos personales: La información de persona física o jurídica determinada o determinable;

II. Datos sensibles: Aquellos que revelan el origen racial, étnico, opiniones políticas, convicciones religiosas, filosóficas o morales, afiliación sindical, salud o vida sexual;

III. Archivo, registro, base o banco de datos: Conjunto de datos personales organizados, tratados automatizadamente;

IV. Tratamiento de datos: Operaciones y procedimientos sistemáticos que tienen por objeto recolectar, guardar, ordenar, modificar, relacionar, cancelar y cualquiera otra que implique el procesamiento de datos, o su cesión a terceros a través de comunicaciones, consultas, interconexiones y transferencias;

V. Responsable del archivo, registro, base o banco de datos: Persona física o jurídica que ostenta la titularidad del archivo, registro, banco o base de datos;

VI. Datos informáticos: Los datos personales tratados automatizadamente;

VII. Usuario de datos: Toda persona física, jurídica, pública o privada que trata datos personales de manera voluntaria, ya sea en archivos, registros, bancos de datos propios o a través de conexión con los mismos;

VIII. Disociación de datos: Todo tratamiento de datos personales que impida asociarlos a persona determinada o determinable; y,

IX. Interesado: La persona física o jurídica a la que conciernen los datos personales.

Artículo 5.

2. La colecta de datos se debe hacer por medios lícitos que garanticen el respeto a las garantías individuales y, especialmente, de los derechos al honor y a la intimidad de la persona a la que conciernen.

3. Los datos sólo pueden ser utilizados para los fines que motivaron su obtención, o para fines compatibles con estos.

Capítulo II

Artículo 8.

1. La colecta y el tratamiento automatizado de los datos requieren del consentimiento previo del interesado, salvo que la ley disponga otra cosa.

2. El interesado, sin su responsabilidad, tiene el derecho de revocar su consentimiento para el tratamiento automatizado de datos, dando aviso oportuno e indubitable al titular del archivo, registro, base o banco de datos, salvo que la ley disponga otra cosa.

Artículo 9.

1. Ninguna persona está obligada a proporcionar datos personales de carácter sensible que le conciernen.

2. Los datos sensibles sólo pueden ser colectados y tratados por razones de interés general previstas en la ley, cuando previamente el interesado ha otorgado su consentimiento, o cuando se colecten y traten con fines estadísticos o científicos, siempre que no se puedan atribuir a persona determinada o determinable.

3. Queda prohibida la formación de archivos, registros, bases o bancos que revelen datos sensibles, salvo lo dispuesto en esta ley.

Artículo 11.

1. Queda prohibido registrar datos personales en archivos, registros o bancos de datos que no reúnan condiciones técnicas de integridad o seguridad.

2. El responsable del archivo, registró, base o banco de datos debe adoptar todas las medidas técnicas y de organización necesarias para evitar la adulteración, pérdida, inexactitud, insuficiencia, falta, consulta, reserva, cancelación o tratamiento de datos no autorizado.

3. El reglamento de esta ley determinará los requisitos y condiciones mínimas de seguridad y de organización, en función del estado de la tecnología, la naturaleza de los datos almacenados y los riesgos a que estén expuestos.

Artículo 14.

1. Se prohíbe la transferencia de datos personales con Estados u organismos internacionales, que no proporcionen niveles de seguridad y protección cuando menos equivalentes a los que se proporcionan en el Estado Mexicano.

2. La prohibición no rige en los supuestos siguientes:

III. Transferencias bancarias o bursátiles, conforme a la legislación que le resulte aplicable;

IV. Cuando la transferencia se acuerde en un tratado, convenio o instrumento internacional vigente en el que el Estado Mexicano sea parte; y,

V. Cuando la transferencia tenga por objeto la cooperación internacional para la lucha contra el crimen organizado, el terrorismo, el narcotráfico y delitos contra la humanidad.

Artículo 15.

1. Toda persona tiene derecho de solicitar al Instituto Federal de Protección de Datos Personales, informes sobre la existencia de archivos, registros, bases o bancos de datos personales, sobre las finalidades y la identidad de sus responsables.

Artículo 26.

1. Quienes se dediquen a la prestación de servicios de información sobre la solvencia patrimonial y de crédito sólo podrán tratar automatizadamente datos de carácter personal obtenidos de fuentes accesibles al público, facilitados por el interesado o con su consentimiento previo.

5. Sólo se pueden archivar, registrar o ceder los datos personales que sean significativos para evaluar la solvencia económica financiera de los afectados durante los últimos cinco años. Dicho plazo se reducirá a dos años cuando el deudor cancele o de otro modo se extinga la obligación, debiéndose hacer constar dicho hecho.

Capítulo III
Del Instituto Federal de Protección de Datos Personales

Artículo 29.

1. El Instituto Federal de Protección de Datos Personales es el organismo público descentralizado de la administración pública federal, con personalidad jurídica y patrimonio propios que tiene por objeto el control de los responsables de los archivos, registros, bases o bancos de datos personales y la protección de éstos.

Artículo 30.

1. Compete al Instituto Federal de Protección de Datos Personales el ejercicio de las atribuciones siguientes:

IV. Vigilar que las normas sobre integridad y seguridad de los datos personales se respeten y apliquen por los titulares de los archivos, registros, bases o bancos de datos correspondientes;

Con ese objeto, podrá solicitar a la autoridad judicial competente autorización para inspeccionar los inmuebles, equipos, herramientas y programas de captura y tratamiento de datos;

V. Solicitar la información que requiera para el cumplimiento de su objeto a las entidades públicas y privadas titulares de los archivos, registros, bases o bancos de datos personales, garantizando en todo caso la seguridad, la integridad y confidencialidad de la información;

VI. Imponer las sanciones administrativas que correspondan a los infractores de esta ley;

VII. Formular y presentar las denuncias y querellas por violaciones a lo dispuesto en esta ley.

Capítulo IV
De las sanciones

Artículo 40.

1. Son infracciones graves a esta Ley:

I. Colectar o tratar datos de carácter personal para constituir, o implementar archivos, registros, bases o bancos de datos de titularidad pública, sin la previa autorización de la normativa aplicable;

II. Colectar o tratar automatizadamente datos de carácter personal para constituir, o implementar archivos, registros, bases o bancos de datos de titularidad privada, sin el consentimiento del interesado o de quien legítimamente puede otorgarlo.

VI. Mantener archivos, registros, bases o bancos de datos, inmuebles, equipos o herramientas sin las condiciones mínimas de seguridad requeridas por las disposiciones aplicables.

3.- Código de Comercio y de la Ley Federal de Protección al Consumidor.

Titulo II
Del Comercio Electrónico.

Artículo 89.- En los actos de comercio podrán emplearse los medios electrónicos, ópticos o cualquier otra tecnología. Para efecto del presente Código, a la información generada, enviada, recibida, archivada o comunicada a través de dichos medios se le denominará mensaje de datos.

Artículo 90.- Salvo pacto en contrario, se presumirá que el mensaje de datos proviene del emisor si ha sido enviado:

I.- Usando medios de identificación, tales como claves o contraseñas de él, o

II.- Por un sistema de información programado por el emisor o en su nombre para que opere automáticamente.

Artículo 91.- El momento de recepción de la información a que se refiere el artículo anterior se determinará como sigue:

I.- Si el destinatario ha designado un sistema de información para la recepción, ésta tendrá lugar en el momento en que ingrese en dicho sistema, o

II.- De enviarse a un sistema del destinatario que no sea el designado o de no haber un sistema de información designado, en el momento en que el destinatario obtenga dicha información.

Para efecto de este Código, se entiende por sistema de información cualquier medio tecnológico utilizado para operar mensajes de datos.

Artículo 92.- Tratándose de la comunicación de mensajes de datos que requieran de un acuse de recibo para surtir efectos, bien sea por disposición legal o por así requerirlo el emisor, se considerará que el mensaje de datos ha sido enviado, cuando se haya recibido el acuse respectivo.

Salvo prueba en contrario, se presumirá que se ha recibido el mensaje de datos cuando el emisor reciba el acuse correspondiente.

Artículo 93.- Cuando la ley exija la forma escrita para los contratos y la firma de los documentos relativos, esos supuestos se tendrán por cumplidos tratándose de mensaje de datos siempre que éste sea atribuible a las personas obligadas y accesible para su ulterior consulta.

En los casos en que la ley establezca como requisito que un acto jurídico deba otorgarse en instrumento ante fedatario público, éste y las partes obligadas podrán, a través de mensajes de datos, expresar los términos exactos en que las partes han decidido obligarse, en cuyo caso el fedatario público, deberá hacer constar en el propio instrumento los elementos a través de los cuales se atribuyen dichos mensajes a las partes y conservar bajo su resguardo una versión íntegra de los mismos para su ulterior consulta, otorgando dicho instrumento de conformidad con la legislación aplicable que lo rige.

Artículo 94.- Salvo pacto en contrario, el mensaje de datos se tendrá por expedido en el lugar donde el emisor tenga su domicilio y por recibido en el lugar donde el destinatario tenga el suyo.

Artículo 1298-A.- Se reconoce como prueba los mensajes de datos. Para valorar la fuerza probatoria de dichos mensajes, se estimará primordialmente la fiabilidad del método en que haya sido generada, archivada, comunicada o conservada."

4.- Delitos informáticos, casos de ejemplo.

- Kevin David Mitnick.

Kevin David Mitnick es quizás el más famoso hackers de los últimos tiempos. Nacido el 6 de Agosto de 1963 en Van Nuts, California, desde muy niño sintió curiosidad por los sistemas de comunicación electrónica y fue auto cultivando un obsesivo deseo por investigar cosas y lograr objetivos aparentemente imposibles, hasta llegar a poseer una genial habilidad para ingresar a servidores sin autorización, robar información, interceptar teléfonos, crear virus, etc.
Cuando el gobierno acusó a Kevin de haber substraído información del FBI, relacionada a la investigación de Ferdinand Marcos y de haber penetrado en computadoras militares, en 1992, él decidió defenderse en la clandestinidad, convirtiéndose en un fugitivo de la justicia durante casi tres años.

Mitnick fue arrestado por el FBI en Raleigh, North Carolina, el 15 de Febrero de 1995.
Kevin descubrió y reveló información de alta seguridad perteneciente al FBI, incluyendo cintas del consulado de Israel, en Los Ángeles. Sus incursiones costaron millones de dólares al FBI y al gobierno norteamericano y obligó a este departamento policial a mudar sus centros secretos de comunicación a sitios inaccesibles.

Mitnick, quién fue liberado en Enero del 2000, después de permanecer casi 5 años en un prisión federal, le costó al estado norteamericano y a empresas privadas, millones de dólares al ser objeto de hurto de su software, información y alteración de los datos de las mismas. Entre los agraviados se

incluyen corporaciones tales como Motorola, Novell, Nokia y Sun Microsystems, el FBI, el Pentágono y la Universidad de Southern California.

El se convirtió en un símbolo entre la comunidad internacional de hackers, después de que el FBI lo investigara y persiguiera infructuosamente durante tres años, y cuya captura se produjo en 1995, cuando los investigadores rastrearon sus huellas hasta llegar a un departamento en Raleigh, en Carolina del Norte.

Bajo un acuerdo de petición de clemencia, la Juez del U.S. District, Mariana Pfaelzer prohibió a Mitnick a acceder a computadoras, teléfonos celulares, televisión o cualquier equipo electrónico que pudiese ser usado en Internet. La Juez pensó que Mitnick no estaría en condiciones de obtener ningún recurso económico por encima del salario mínimo.

Sin embargo, un reciente reporte en Internet informó que Mitnick había ganado $20,000 dólares por concepto de dictado de conferencias sobre seguridad en Internet, en diversos foros.

- Vladimir Levin.

Vladimir Levin, un graduado en matemáticas de la Universidad Tecnológica de San Petesburgo, Rusia, fue acusado de ser la mente maestra de una serie de fraudes tecnológicos que le permitieron a él y la banda que conformaba, substraer más de 10 millones de dólares, de cuentas corporativas del Citibank.

En 1995 fue arrestado por la Interpol, en el aeropuerto de Heathrow, Inglaterra, y luego extraditado a los Estados Unidos.

Las investigaciones establecieron que desde su computadora instalada en la empresa AO Saturn, de San Petersburgo, donde trabajaba, Levin irrumpió en las cuentas del Citibank de New York y transfirió los fondos a cuentas aperturadas en Finlandia, Israel y en el Bank of America de San Francisco.

Ante las evidencias y manifestaciones de sus co-inculpados, Vladimir Levin se declaró culpable. Uno de sus cómplices, Alexei Lashmanov, de 28 años, en Agosto de 1994 había hecho alarde entre sus conocidos, en San Petersburgo, acerca de sus abultadas cuentas bancarias personales en Tel Aviv, Israel.

Estos conspiradores habían obtenido accesos no autorizados al Sistema de Administración de Dinero en Efectivo del Citibank (The Citibank Cash Management System), en Parsipanny, New Jersey, el cual permite a sus clientes acceder a una red de computadoras y transferir fondos a cuentas de otras instituciones financieras, habiendo realizado un total de 40 transferencias ilegales de dinero.

Lashmanov admitió que él y sus cómplices habían transferido dinero a cinco cuentas en bancos de Tel Aviv. Incluso trató de retirar en una sola transacción US $ 940,000 en efectivo de estas cuentas.

Otros tres cómplices, entre ellos una mujer, también se declararon culpables. Esta última fue descubierta "in fraganti" cuando intentaba retirar dinero de una cuenta de un banco de San Francisco. Se estima en un total de 10.7 millones de dólares el monto substraído por esta banda.

Las investigaciones y el proceso tuvieron muchas implicancias que no pudieron ser aclaradas ni siquiera por los responsables de la seguridad del sistema de Administración de Dinero en Efectivo, del propio Citibank. Jamás se descartó la sospecha de participación de más de un empleado del propio banco.

A pesar de que la banda substrajo más de 10 millones de dólares al Citibank, Levin fue sentenciado a 3 años de prisión y a pagar la suma de US $ 240,015 a favor del Citibank, ya que las compañías de seguros habían cubierto los montos de las corporaciones agraviadas.

Los técnicos tuvieron que mejorar sus sistemas de seguridad contra "crackers" y Vladimir Levin ahora se encuentra en libertad.

C.- Tablas

1.- Tabla de Vigenère

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
A	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
B	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a
C	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b
D	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c
E	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d
F	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e
G	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f
H	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g
I	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h
J	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i
K	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j
L	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k
M	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l
N	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m
O	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n
P	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
Q	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
R	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q
S	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r
T	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s
U	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t
V	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u
W	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
X	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w
Y	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x
Z	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y

2.- DES.- S-BOX

S-1

14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
00	15	07	04	14	02	13	01	10	06	12	11	09	05	03	08
04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

S-2

15	01	08	14	06	11	03	04	09	07	02	13	12	00	05	10
03	13	04	07	15	02	08	14	12	00	01	10	06	09	11	05
00	14	07	11	10	04	13	01	04	08	12	06	09	03	02	15
13	08	10	01	03	15	04	02	11	06	07	12	00	05	14	09

S-3

10	00	09	14	06	03	15	05	01	13	12	07	11	04	02	08
13	07	00	09	03	04	06	10	02	08	05	14	12	11	15	01
13	06	04	09	08	15	03	00	11	01	02	12	05	10	14	07
01	10	13	00	06	09	08	07	04	15	14	03	11	05	02	12

S-4

07	13	14	03	00	06	09	10	01	02	08	05	11	12	04	15
13	08	11	05	06	15	00	03	04	07	02	12	01	10	14	09
10	06	09	00	12	11	07	13	15	01	03	14	05	02	08	04
03	15	00	06	10	01	13	08	09	04	05	11	12	07	02	14

S-5

02	12	04	01	07	10	11	06	08	05	03	15	13	00	14	09
14	11	02	12	04	07	13	01	05	00	15	10	03	09	08	06
04	02	01	11	10	13	07	08	15	09	12	05	06	03	00	14
11	08	12	07	01	14	02	13	06	15	00	09	10	04	05	03

S-6

12	01	10	15	09	02	06	08	00	13	03	04	14	07	05	11
10	15	04	02	07	12	09	05	06	01	13	14	00	11	03	08
09	14	15	05	02	08	12	03	07	00	04	10	01	13	11	06
04	03	02	12	09	05	15	10	11	14	01	07	06	00	08	13

S-7

04	11	02	14	15	00	08	13	03	12	09	07	05	10	06	01
13	00	11	07	04	09	01	10	14	03	05	12	02	15	08	06
01	04	11	13	12	03	07	14	10	15	06	08	00	05	09	02
06	11	13	08	01	04	10	07	09	05	00	15	14	02	03	12

S-8

13	02	08	04	06	15	11	01	10	09	03	14	05	00	12	07
01	15	13	08	10	03	07	04	12	05	06	11	00	14	09	02
07	11	04	01	09	12	14	02	00	06	10	13	15	03	05	08
02	01	14	07	04	10	08	13	15	12	09	00	03	05	06	11

3.- Tabla.- Las 4 Rondas de MD5.

Primera Ronda:	Segunda Ronda:
FF(a,b,c,d,m_0,7,D76AA478)	GG(a,b,c,d,m_1,5,F61E2562)
FF(d,a,b,c,m_1,12,E8C7B756)	GG(d,a,b,c,m_6,9,C040B340)
FF(c,d,a,b,m_2,17,242070DB)	GG(c,d,a,b,m_{11},14,265E5A51)
FF(b,c,d,a,m_3,22,C1BDCEEE)	GG(b,c,d,a,m_0,20,E9B6C7AA)
FF(a,b,c,d,m_4,7,F57C0FAF)	GG(a,b,c,d,m_5,5,D62F105D)
FF(d,a,b,c,m_5,12,4787C62A)	GG(d,a,b,c,m_{10},9,02441453)
FF(c,d,a,b,m_6,17,A8304613)	GG(c,d,a,b,m_{15},14,D8A1E681)
FF(b,c,d,a,m_7,22,FD469501)	GG(b,c,d,a,m_4,20,E7D3FBC8)
FF(a,b,c,d,m_8,7,698098D8)	GG(a,b,c,d,m_9,5,21E1CDE6)
FF(d,a,b,c,m_9,12,8B44F7AF)	GG(d,a,b,c,m_{14},9,C33707D6)
FF(c,d,a,b,m_{10},17,FFFF5BB1)	GG(c,d,a,b,m_3,14,F4D50D87)
FF(b,c,d,a,m_{11},22,895CD7BE)	GG(b,c,d,a,m_8,20,455A14ED)
FF(a,b,c,d,m_{12},7,6B901122)	GG(a,b,c,d,m_{13},5,A9E3E905)
FF(d,a,b,c,m_{13},12,FD987193)	GG(d,a,b,c,m_2,9,FCEFA3F8)
FF(c,d,a,b,m_{14},17,A679438E)	GG(c,d,a,b,m_7,14,676F02D9)
FF(b,c,d,a,m_{15},22,49B40821)	GG(b,c,d,a,m_{12},20,8D2A4C8A)
Tercera Ronda:	**Cuarta Ronda:**
HH(a,b,c,d,m_5,4,FFFA3942)	*II*(*a*, *b*, *c*, *d*, m_0, 6, *F*4292244)
HH(d,a,b,c,m_8,11,8771F681)	*II*(*d*, *a*, *b*, *c*, m_7, 10, 432*AFF*97)
HH(c,d,a,b,m_{11},16,6D9D6122)	*II*(*c*, *d*, *a*, *b*, m_{14}, 15, *AB*9423*A*7)
HH(b,c,d,a,m_{14},23,FDE5380C)	*II*(*b*, *c*, *d*, *a*, m_5, 21, *FC*93*A*039)
HH(a,b,c,d,m_1,4,A4BEEA44)	*II*(*a*, *b*, *c*, *d*, m_{12}, 6, 655*B*59*C*3)
HH(d,a,b,c,m_4,11,4BDECFA9)	*II*(*a*, *b*, *c*, *d*, m_3, 10, 8*F*0*CCC*92)
HH(c,d,a,b,m_7,16,F6BB4B60)	*II*(*c*, *d*, *a*, *b*, m_{10}, 15, *FFEFF*47*D*)
HH(b,c,d,a,m_6,23,04881D05)	*II*(*b*, *c*, *d*, *a*, m_1, 21, 85845*DD*1)
HH(a,b,c,d,m_9,4,D9D4D039)	*II*(*a*, *b*, *c*, *d*, m_8, 6, 6*FA*87*E*4*F*)
HH(d,a,b,c,m_{12},11,E6DB99E5)	*II*(*d*, *a*, *b*, *c*, m_{15}, 10, *FE*2*CE*6*E*0)
HH(c,d,a,b,m_{15},16,1FA27CF8)	*II*(*c*, *d*, *a*, *b*, m_6, 15, *A*3014314)
HH(b,c,d,a,m_2,23,C4AC5665)	*II*(*b*, *c*, *d*, *a*, m_{13}, 21, 4*E*0811*A*1)
	II(*a*, *b*, *c*, *d*, m_4, 6, *F*7537*E*82)
	II(*d*, *a*, *b*, *c*, m_{11}, 10, *BD*3*AF*235)
	II(*c*, *d*, *a*, *b*, m_2, 15, 2*AD*7*D*2*BB*)
	II(*b*, *c*, *d*, *a*, m_9, 21, *EB*86*D*391)

D.- Códigos Fuentes

1.- Sustitución Simple.

```
/*************************************************************************/
/*  TESIS- TECNICAS CRIPTOGRAFICAS                                       */
/*  AUTOR: Guillermo Gomez Villa                                         */
/*  FECHA: Mayo 2004                                                     */
/*************************************************************************/

/************ sustitucion.h *****************/
char alfabeto[]={"abcdefghijklmnñopqrstuvwxyzABCDEFGHIJKLMNÑOPQRSTUVWXYZ1234567890 .,()"};
char      sub[]={"XbfgY,(j4kWaNZ13dR5w7 .)n8ñ0opTrsE9P2Fv6cVzLAeltBCGxHyUDMJmqÑOhiQKuSI"};

/*Encuentra el indice correcto*/
int Busca(char *s, char ch)
{
   register unsigned int t;
   for(t=0; t<strlen(s); t++)
      if(ch==s[t])
         return t; /* Recorre el arreglo en busca de la  */
                   /*letra correspondiente en posición   */
   return -1;
}
void Sustitucion(char ps_Opt)
{
   char texto[100];
   int  unsigned i;
   clrscr();
   puts("          TECNICAS CRIPTOGRAFIAS");
   puts("");
   puts("       CIFRADO POR SUSTITUCION SIMPLE");
   puts("");
   printf("Alfabeto={%s}\n",alfabeto);
   printf("Matriz de Substitucion:\n");
   printf("     Sub={%s}\n\n ",sub);
   if (ps_Opt=='e' ||ps_Opt=='E')
   {
      puts("");
      printf("%s","Texto en claro(MAX 100):"); fgets(texto,100-1,stdin);
      puts("");
      puts("Resultado: ");
      for(i=0;i<strlen(texto)-1;i++) /* CIFRADO POR SUSTITUCION */
            printf("%c",alfabeto[Busca(sub,texto[i])]);
   }
   else if (ps_Opt=='d' ||ps_Opt=='D')
   {
      puts("");
      printf("%s","Texto Cifrado(MAX 100):"); fgets(texto,100-1,stdin);
      puts("");
      puts("Resultado: ");
      for(i=0;i<strlen(texto)-1;i++) /* DESCIFRADO POR SUSTITUCION */
            printf("%c",sub[Busca(alfabeto,texto[i])]);
   }
   puts("");puts("");puts("");
   puts("        <Presione cualquier tecla para continuar>");
   getch();
}
```

```
/*****************************************************************************/
```

2.- Vigenère.

```
/*********************************************************************/
/*  TESIS- TECNICAS CRIPTOGRAFICAS                                   */
/*  AUTOR: Guillermo Gomez Villa                                     */
/*  FECHA: Mayo 2004                                                 */
/*********************************************************************/

/************* vigenere.h ******************************/

char alfa_vige[]={"abcdefghijklmnñopqrstuvwxyz"};

void Vigenere(char ps_Opt)
{
   char texto[100],texto_cifrado[100],clave[20],tmp;
   unsigned int  ind_txt,ind_clave;
   ind_clave=0;
   ind_txt=0;
   clrscr();
   puts("          TECNICAS CRIPTOGRAFIAS");
   puts("");
   puts("             CIFRADO VIGENERE");
   puts("");
   printf("Alfabeto={%s}\n",alfa_vige);
   puts("");
   if (ps_Opt=='e' ||ps_Opt=='E')
   {
      puts("");
      printf("%s","Texto en Claro(MAX 100):"); fgets(texto,100-1,stdin);
      printf("%s","Clave de Cifrado(MAX(20):"); fgets(clave,20-1,stdin);
      puts("");
      puts("Resultado: ");
      while(ind_txt<(strlen(texto)-1))
      {
         if(isalpha(texto[ind_txt]))
         {
             if (texto[ind_txt] >= 'A' && texto[ind_txt] <= 'Z')
             {   /* todo se convierte a minusculas */
                 texto[ind_txt] = texto[ind_txt] - 'A' + 'a';
             }
            tmp = (texto[ind_txt] + clave[ind_clave] - 2*'a') % ALPHA;
            while (tmp < 0) tmp += ALPHA;
            tmp += 'a';
            texto_cifrado[ind_txt]=tmp;
         }
         else texto_cifrado[ind_txt]=texto[ind_txt];
         if(++ind_clave>=(strlen(clave)-1)) ind_clave = 0;
         ind_txt++;
      };
      texto_cifrado[ind_txt]='\0';
      printf("%s",texto_cifrado);
   }
   else if (ps_Opt=='d' ||ps_Opt=='D')
   {
      puts("");
      printf("%s","Texto Cifrado(MAX 100):"); fgets(texto_cifrado,100-1,stdin);
      printf("%s","Clave de Cifrado(MAX(20):"); fgets(clave,20-1,stdin);
      puts("");
      puts("Resultado: ");
      while(ind_txt<(strlen(texto_cifrado)-1))
      {
```

```
        if(isalpha(texto_cifrado[ind_txt]))
        {
            if (texto_cifrado[ind_txt] >= 'A' && texto_cifrado[ind_txt] <= 'Z')
            {   /* todo se convierte a minusculas */
                texto_cifrado[ind_txt] = texto_cifrado[ind_txt] - 'A' + 'a';
            }
           tmp = ( texto_cifrado[ind_txt] - clave[ind_clave] ) % ALPHA;
           while (tmp < 0) tmp += ALPHA;
           tmp += 'a';
           texto[ind_txt]=tmp;
        }
        else texto[ind_txt]=texto_cifrado[ind_txt];
        if(++ind_clave>=(strlen(clave)-1)) ind_clave = 0;
        ind_txt++;
      };
      texto[ind_txt]='\0';
      printf("%s",texto);
   }
   puts("");puts("");puts("");
   puts("        <Presione cualquier tecla para continuar>");
   getch();
}

/*******************************************************/
```

3.- Simple Xor.

```
/*************************************************************************/
/*  TESIS- TECNICAS CRIPTOGRAFICAS                                       */
/*  AUTOR: Guillermo Gomez Villa                                         */
/*  FECHA: Mayo 2004                                                     */
/*************************************************************************/

/******************** sxor.h ***************************************/
void Simple_XOR(char ps_Opt)
{
   FILE *fi, *fo;
   char *cp,clave[20],arch_in[50],arch_out[50];
   int c;
   char texto[100],texto_cifrado[100];
   clrscr();
   puts("          TECNICAS CRIPTOGRAFIAS");
   puts("");
   puts("            CIFRADO  SIMPLE XOR");
   puts("");
   if (ps_Opt=='e' ||ps_Opt=='E')
   {
      puts("");
      printf("%s","Ruta y Nombre Archivo de Entrada:"); scanf("%s",arch_in);
      printf("%s","Ruta y Nombre Archivo de Salida:"); scanf("%s",arch_out);
      printf("%s","Clave de Cifrado(MAX(20):"); scanf("%s",clave);
      if ((cp = clave) && *cp!='\0')
      {
        if((fi = fopen(arch_in,"rb")) != NULL)
        {
            if((fo = fopen(arch_out,"wb")) != NULL)
            {
              while((c=getc(fi)) != EOF)
              {
                 if(!*cp)
                 {
                    cp = clave;
```

```
                }
                c ^= *(cp++);
                putc(c,fo);
              }
              fclose(fo);
            }
            else
            {
               puts("");
               puts("Archivo de entrada no encontrado !!");
               puts("");
            }
            fclose(fi);
        }
        else
        {
          puts("");
          puts("Archivo de entrada no encontrado !!");
          puts("");
        }
    }
}
if (ps_Opt=='d' ||ps_Opt=='D')
{
   puts("");
   printf("%s","Ruta y Nombre Archivo de Entrada:"); scanf("%s",arch_in);
   printf("%s","Ruta y Nombre Archivo de Salida:"); scanf("%s",arch_out);
   printf("%s","Clave de Cifrado(MAX(20):"); scanf("%s",clave);
   if ((cp = clave) && *cp!='\0')
   {
     if((fi = fopen(arch_in,"rb")) != NULL)
     {
         if((fo = fopen(arch_out,"wb")) != NULL)
         {
           while((c=getc(fi)) != EOF)
           {
              if(!*cp)
              {
                 cp = clave;
              }
              c ^= *(cp++);
              putc(c,fo);
           }
           fclose(fo);
         }
         else
         {
            puts("");
            puts("Archivo de entrada no encontrado !!");
            puts("");
         }
         fclose(fi);
     }
     else
     {
       puts("");
       puts("Archivo de entrada no encontrado !!");
       puts("");
     }
   }
}
puts("");puts("");puts("");
puts("        <Presione cualquier tecla para continuar>");
getch();
}
```

```
/*************************************************************************/
```

4.- Transposición.

```
/*************************************************************************/
/*  TESIS- TECNICAS CRIPTOGRAFICAS                                       */
/*  AUTOR: Guillermo Gomez Villa                                         */
/*  FECHA: Mayo 2004                                                     */
/*************************************************************************/

/************************transposicion.h  ******************************/
#define AX 3
#define AY 4

void Transposicion(char ps_Opt)
{
   /* BLOQUE DE DECLARACIONES */
   char texto[100],texto_cifrado[100];
   char matriz_a[AX][AY];
   int  ind_txt,ind_cif,fin_cadena,ln_lengh,x,y;
   int  col, res;
   fin_cadena=FALSO;
   ind_txt=0;
   col=0;
   res=0;
   clrscr();
   puts("          TECNICAS CRIPTOGRAFIAS");
   puts("");
   puts("         CIFRADO POR TRANSPOSICION");
   puts("");
   if (ps_Opt=='e' ||ps_Opt=='E')
   {
      puts("");
      printf("%s","Texto en Claro(MAX 100):"); fgets(texto,100-1,stdin);
      ln_lengh = strlen(texto)-1;
      if(ln_lengh>0)
      {
         puts("");
         puts("Resultado: ");
         do
         {
            for(x=0;x<AX;x++) for(y=0;y<AY;y++) matriz_a[x][y]='_';
            for(x=0;x<AX && !fin_cadena;x++)
              for(y=0;y<AY && !fin_cadena;y++)
              {
                 matriz_a[x][y]=texto[ind_txt++];
                 if(ind_txt==ln_lengh) fin_cadena = VERDADERO;
              }
            ind_cif=0;
            texto_cifrado[0]='\0';
            for(y=0;y<AY;y++)
              for(x=0;x<AX ;x++)
               if (matriz_a[x][y]!='_')
                  texto_cifrado[ind_cif++]=matriz_a[x][y];
               texto_cifrado[ind_cif]='\0';
            printf("%s",texto_cifrado);
         }while(!fin_cadena);
      }
   }
   else if(ps_Opt=='d' ||ps_Opt=='D')
   {
      puts("");
```

```
      printf("%s","Texto Cifrado (MAX 100):"); fgets(texto,100-1,stdin);
      ln_lengh = strlen(texto)-1;
      if(ln_lengh>0)
      {
         puts("");
         puts("Resultado: ");
         do
         {
            col = 0;
            col = (ln_lengh-ind_txt) / AY ;
            res = (ln_lengh-ind_txt) % AY ;
            for(x=0;x<AX;x++) for(y=0;y<AY;y++) matriz_a[x][y]='_';
            for(y=0;y<AY && !fin_cadena;y++)
              for(x=0;x<AX && !fin_cadena;x++)
              {
                 if(x<col) matriz_a[x][y]=texto[ind_txt++];
                 else if (x >= col && res > 0 )
                 {
                    matriz_a[x][y]=texto[ind_txt++];
                    res--;
                    break;
                 }
                 if(ind_txt==ln_lengh) fin_cadena = VERDADERO;
              }
            ind_cif=0;
            texto_cifrado[0]='\0';
            for(x=0;x<AX ;x++)
              for(y=0;y<AY ;y++)
                 if (matriz_a[x][y]!='_')texto_cifrado[ind_cif++]=matriz_a[x][y];
            texto_cifrado[ind_cif]='\0';
            printf("%s",texto_cifrado);
         }while(!fin_cadena);
      }
   }
   puts("");puts("");puts("");
   puts("        <Presione cualquier tecla para continuar>");
   getch();
}

/*****************************************************************************/
```

5.- DES.

```
/*****************************************************************************/
/*  TESIS- TECNICAS CRIPTOGRAFICAS                                           */
/*  AUTOR: Guillermo Gomez Villa                                             */
/*  FECHA: Mayo 2004                                                         */
/*****************************************************************************/
/*********************** des.h ***********************************************/
/*****************************************************************************/
/* Implementación del algoritmo DES                                          */
/* Basado en la implementación DES codificada por James Gillogly             */
/* Codigo de uso publico y libre bajo licencia GNU                           */
/* Modificado y adecuado para su uso en compilador Borland C                 */
/*****************************************************************************/


int optind = 1,      /* indice arreglo de argumentos */
    optopt;          /* Bandera de validación        */
char    *optarg;     /* argumento asociado con la opcion */

#define BADCH   (int)'?'
#define EMSG    ""
```

```
#define tell(s) fputs(*nargv,stderr);fputs(s,stderr); \
        fputc(optopt,stderr);fputc('\n',stderr);return(BADCH);

getopt(int nargc,char **nargv,char *ostr);
void dodecrypt(FILE *fp1, FILE *fp2);
void doencrypt(FILE *fp1, FILE *fp2);
static unsigned long byteswap(unsigned long x);
static round(int num,unsigned long *block);
static int spinit();
desinit(int mode);
int desdone(void);
void DES(int argc,char *argv[]);
int setkey(char *key);
void Alg_Des(char ps_Opt);

getopt(int nargc,char **nargv,char *ostr)
{
    static char *place = EMSG;  /* letra de opcion en proceso */
    register char   *oli;       /* letra del indice del arreglo de argumentos */

    if(!*place)
    {   /* actualizacion de puntero de busqueda */
        if(optind >= nargc || *(place = nargv[optind]) != '-' || !*++place)
        {
           return(EOF);
        }
        if (*place == '-')
        {    /* se encontro "-" */
            ++optind;
            return(EOF);
        }
    }
    /* Validación de la opcion */
    if ((optopt = (int)*place++) == (int)':' || !(oli = strchr(ostr,optopt)))
    {
        if(!*place) ++optind;
        tell(": opcion incorrecta -- ");
    }
    if (*++oli != ':')
    {  /* no necesita argumentos */
        optarg = NULL;
        if (!*place) ++optind;
    }
    else
    {   /* necesita argumentos */
        if (*place)
            optarg = place; /* sin espacios en blanco */
        else if (nargc <= ++optind)
        {   /* no hay argumentos */
            place = EMSG;
            tell(": la opcion requiere un argumento -- ");
        }
        else
            optarg = nargv[optind];     /* espacios en blanco */
        place = EMSG;
        ++optind;
    }
    return(optopt);         /* regresa la letra de opcion */
}

/*****************************************************************************/

/************************** RUTINAS DES ************************************/

#define NULL    0
```

```
#ifdef  LITTLE_ENDIAN
unsigned long byteswap();
#endif

/*****************************************************************************/
/* Tablas definidas en el los documentos del  Data Encryption Standard  */
/*****************************************************************************/
/* Tabla de permutacion inicial IP */
static char ip[] = {
    58, 50, 42, 34, 26, 18, 10,  2,
    60, 52, 44, 36, 28, 20, 12,  4,
    62, 54, 46, 38, 30, 22, 14,  6,
    64, 56, 48, 40, 32, 24, 16,  8,
    57, 49, 41, 33, 25, 17,  9,  1,
    59, 51, 43, 35, 27, 19, 11,  3,
    61, 53, 45, 37, 29, 21, 13,  5,
    63, 55, 47, 39, 31, 23, 15,  7
};

/* Tabla de permutacion pinal IP^-1 */
static char fp[] = {
    40,  8, 48, 16, 56, 24, 64, 32,
    39,  7, 47, 15, 55, 23, 63, 31,
    38,  6, 46, 14, 54, 22, 62, 30,
    37,  5, 45, 13, 53, 21, 61, 29,
    36,  4, 44, 12, 52, 20, 60, 28,
    35,  3, 43, 11, 51, 19, 59, 27,
    34,  2, 42, 10, 50, 18, 58, 26,
    33,  1, 41,  9, 49, 17, 57, 25
};

/* Matriz de operacion de expancion
 * Esta es solo de referencia solamente; no es utilizada en el codigo
 * la funcion f() se implementa implicitamente para mejorar velocidad
 */
#ifdef notdef
static char ei[] = {
    32,  1,  2,  3,  4,  5,
     4,  5,  6,  7,  8,  9,
     8,  9, 10, 11, 12, 13,
    12, 13, 14, 15, 16, 17,
    16, 17, 18, 19, 20, 21,
    20, 21, 22, 23, 24, 25,
    24, 25, 26, 27, 28, 29,
    28, 29, 30, 31, 32,  1
};
#endif

/* Tabla de seleccion de permutacion (key) */
static char pc1[] = {
    57, 49, 41, 33, 25, 17,  9,
     1, 58, 50, 42, 34, 26, 18,
    10,  2, 59, 51, 43, 35, 27,
    19, 11,  3, 60, 52, 44, 36,

    63, 55, 47, 39, 31, 23, 15,
     7, 62, 54, 46, 38, 30, 22,
    14,  6, 61, 53, 45, 37, 29,
    21, 13,  5, 28, 20, 12,  4
};

/* rotacion de numeros izquierdos de pc1 */
static char totrot[] = {
    1,2,4,6,8,10,12,14,15,17,19,21,23,25,27,28
};
```

```
/* Tabla de seleccion de clave permutada  */
static char pc2[] = {
    14, 17, 11, 24,  1,  5,
     3, 28, 15,  6, 21, 10,
    23, 19, 12,  4, 26,  8,
    16,  7, 27, 20, 13,  2,
    41, 52, 31, 37, 47, 55,
    30, 40, 51, 45, 33, 48,
    44, 49, 39, 56, 34, 53,
    46, 42, 50, 36, 29, 32
};

/* Tablas de las cajas S (S-boxes) */
static char si[8][64] = {
    /* S1 */
    14,  4, 13,  1,  2, 15, 11,  8,  3, 10,  6, 12,  5,  9,  0,  7,
     0, 15,  7,  4, 14,  2, 13,  1, 10,  6, 12, 11,  9,  5,  3,  8,
     4,  1, 14,  8, 13,  6,  2, 11, 15, 12,  9,  7,  3, 10,  5,  0,
    15, 12,  8,  2,  4,  9,  1,  7,  5, 11,  3, 14, 10,  0,  6, 13,

    /* S2 */
    15,  1,  8, 14,  6, 11,  3,  4,  9,  7,  2, 13, 12,  0,  5, 10,
     3, 13,  4,  7, 15,  2,  8, 14, 12,  0,  1, 10,  6,  9, 11,  5,
     0, 14,  7, 11, 10,  4, 13,  1,  5,  8, 12,  6,  9,  3,  2, 15,
    13,  8, 10,  1,  3, 15,  4,  2, 11,  6,  7, 12,  0,  5, 14,  9,

    /* S3 */
    10,  0,  9, 14,  6,  3, 15,  5,  1, 13, 12,  7, 11,  4,  2,  8,
    13,  7,  0,  9,  3,  4,  6, 10,  2,  8,  5, 14, 12, 11, 15,  1,
    13,  6,  4,  9,  8, 15,  3,  0, 11,  1,  2, 12,  5, 10, 14,  7,
     1, 10, 13,  0,  6,  9,  8,  7,  4, 15, 14,  3, 11,  5,  2, 12,

    /* S4 */
     7, 13, 14,  3,  0,  6,  9, 10,  1,  2,  8,  5, 11, 12,  4, 15,
    13,  8, 11,  5,  6, 15,  0,  3,  4,  7,  2, 12,  1, 10, 14,  9,
    10,  6,  9,  0, 12, 11,  7, 13, 15,  1,  3, 14,  5,  2,  8,  4,
     3, 15,  0,  6, 10,  1, 13,  8,  9,  4,  5, 11, 12,  7,  2, 14,

    /* S5 */
     2, 12,  4,  1,  7, 10, 11,  6,  8,  5,  3, 15, 13,  0, 14,  9,
    14, 11,  2, 12,  4,  7, 13,  1,  5,  0, 15, 10,  3,  9,  8,  6,
     4,  2,  1, 11, 10, 13,  7,  8, 15,  9, 12,  5,  6,  3,  0, 14,
    11,  8, 12,  7,  1, 14,  2, 13,  6, 15,  0,  9, 10,  4,  5,  3,

    /* S6 */
    12,  1, 10, 15,  9,  2,  6,  8,  0, 13,  3,  4, 14,  7,  5, 11,
    10, 15,  4,  2,  7, 12,  9,  5,  6,  1, 13, 14,  0, 11,  3,  8,
     9, 14, 15,  5,  2,  8, 12,  3,  7,  0,  4, 10,  1, 13, 11,  6,
     4,  3,  2, 12,  9,  5, 15, 10, 11, 14,  1,  7,  6,  0,  8, 13,

    /* S7 */
     4, 11,  2, 14, 15,  0,  8, 13,  3, 12,  9,  7,  5, 10,  6,  1,
    13,  0, 11,  7,  4,  9,  1, 10, 14,  3,  5, 12,  2, 15,  8,  6,
     1,  4, 11, 13, 12,  3,  7, 14, 10, 15,  6,  8,  0,  5,  9,  2,
     6, 11, 13,  8,  1,  4, 10,  7,  9,  5,  0, 15, 14,  2,  3, 12,

    /* S8 */
    13,  2,  8,  4,  6, 15, 11,  1, 10,  9,  3, 14,  5,  0, 12,  7,
     1, 15, 13,  8, 10,  3,  7,  4, 12,  5,  6, 11,  0, 14,  9,  2,
     7, 11,  4,  1,  9, 12, 14,  2,  0,  6, 10, 13, 15,  3,  5,  8,
     2,  1, 14,  7,  4, 10,  8, 13, 15, 12,  9,  0,  3,  5,  6, 11
};

/*32-bit Tabla de la funcion de permutacion P usada en la salida de las S-boxes*/
static char p32i[] = {
    16,  7, 20, 21,
```

```
    29, 12, 28, 17,
     1, 15, 23, 26,
     5, 18, 31, 10,
     2,  8, 24, 14,
    32, 27,  3,  9,
    19, 13, 30,  6,
    22, 11,  4, 25
};
/****** Fin de la definicion de tablas DES ************/

/* Las tablas de las operaciones de búsqueda se       */
/* inicializan una vez solamente en el arranque       */
static long (*sp)[64];      /* Combinando las cajas S y P  */

static char (*iperm)[16][8];    /* Permutacion Inicial y Final */
static char (*fperm)[16][8];

/* 8 6-bit sub-claves por cada una de las las 16 rondas, */
/*  inicializado por setkey()                            */
static unsigned char (*kn)[8];

/*  El bit 0 es el extremo izquierdo en el byte */
static int bytebit[] = {
    0200,0100,040,020,010,04,02,01
};

static int nibblebit[] = {
     010,04,02,01
};
static int desmode;

/* Inicializa el arreglo perm */
static perminit(perm,p)
char perm[16][16][8];       /* 64-bit, cualquiera inicial o final */
char p[64];
{
    register int l, j, k;
    int i,m;

    /* limpia el arreglo de permutacion */
    for (i=0; i<16; i++)
        for (j=0; j<16; j++)
            for (k=0; k<8; k++)
                perm[i][j][k]=0;

    for (i=0; i<16; i++)
        for (j = 0; j < 16; j++)
        for (k = 0; k < 64; k++)
        {   l = p[k] - 1;
            if ((l >> 2) != i)
            continue;
            if (!(j & nibblebit[l & 3]))
            continue;
            m = k & 07;
            perm[i][j][k>>3] |= bytebit[m];
        }
        return 0;
}

/* Inicializa la tabla de busqueda para las cajas combinadas S y P */
static int spinit()
{
    char pbox[32];
    int p,i,s,j,rowcol;
    long val;
```

```
    /* calcula la pbox, la inversa de p32i. */
    for(p=0;p<32;p++){
        for(i=0;i<32;i++){
            if(p32i[i]-1 == p){
                pbox[p] = i;
                break;
            }
        }
    }
    for(s = 0; s < 8; s++){          /* Para cada S-box */
        for(i=0; i<64; i++){         /* para cada posible entrada */
            val = 0;
            rowcol = (i & 32) | ((i & 1) ? 16 : 0) | ((i >> 1) & 0xf);
            for(j=0;j<4;j++){   /* Para cada bit de salida */
                if(si[s][rowcol] & (8 >> j)){
                 val |= 1L << (31 - pbox[4*s + j]);
                }
            }
            sp[s][i] = val;

#ifdef  DEBUG
            printf("sp[%d][%2d] = %08lx\n",s,i,sp[s][i]);
#endif
        }
    }
    return 0;
}

/* Reserva espacio y inicializa la busqueda de los arreglos DES
 * mode == 0: Standard Data Encryption Algorithm
 * mode == 1: DEA Sin permutacion inicial y final para velocidad
 * mode == 2: DEA Sin permutacion ademas con clave de 128 bytes
 *            (independencia completa de las sub-claves por cada ronda)
 */
desinit(int mode)
{
    if(sp != NULL){
        /* Ha sido inicializado */
        return 0;
    }
    desmode = mode;

    if((sp = (long (*)[64])malloc(sizeof(long) * 8 * 64)) == NULL){
        return -1;
    }
    spinit();
    kn = (unsigned char (*)[8])malloc(sizeof(char) * 8 * 16);
    if(kn == NULL){
        free((char *)sp);
        return -1;
    }
    if(mode == 1 || mode == 2)  /* No permutations */
        return 0;

    iperm = (char (*)[16][8])malloc(sizeof(char) * 16 * 16 * 8);
    if(iperm == NULL){
        free((char *)sp);
        free((char *)kn);
        return -1;
    }
    perminit(iperm,ip);

    fperm = (char (*)[16][8])malloc(sizeof(char) * 16 * 16 * 8);
    if(fperm == NULL){
        free((char *)sp);
        free((char *)kn);
```

```
        free((char *)iperm);
        return -1;
    }
    perminit(fperm,fp);

    return 0;
}
/* Libera espacio usado por DES */
int desdone(void)
{
    if(sp == NULL)
        return 0; /* Ha sido inicializado */

    free((char *)sp);
    free((char *)kn);
    if(iperm != NULL)
        free((char *)iperm);
    if(fperm != NULL)
        free((char *)fperm);

    sp = NULL;
    iperm = NULL;
    fperm = NULL;
    kn = NULL;
    return 0;
}

/* Permuta inblock con perm */
static permute(inblock,perm,outblock)
char *inblock, *outblock;       /* el resultado dentro de outblock,64 bits */
char perm[16][16][8];           /* se define 2K bytes en perm */
{
    register int i,j;
    register char *ib, *ob;     /* puntero para entrar o salir del bloque  */
    register char *p, *q;

    if(perm == NULL){
        /* sin permutacion, solo copia */
        for(i=8; i!=0; i--)
            *outblock++ = *inblock++;
        return 0;
    }
    /* Limpia el bloque de salida */
    for (i=8, ob = outblock; i != 0; i--)
        *ob++ = 0;

    ib = inblock;
    for (j = 0; j < 16; j += 2, ib++) { /*  */
        ob = outblock;
        p = perm[j][(*ib >> 4) & 017];
        q = perm[j + 1][*ib & 017];
        for (i = 8; i != 0; i--){
            *ob++ |= *p++ | *q++;
        }
    }
    return 0;
}

/* La funcion non-linear (r,k), el corazon del DES */
static long f(r,subkey)
unsigned long r;        /* 32 bits */
unsigned char subkey[8];    /* clave de 48-bits por cada ronda */
{
    register unsigned long rval,rt;
#ifdef  TRACE
    unsigned char *cp;
```

```
    int i;

    printf("f(%08lx, %02x %02x %02x %02x %02x %02x %02x %02x) = ",
        r,
        subkey[0], subkey[1], subkey[2],
        subkey[3], subkey[4], subkey[5],
        subkey[6], subkey[7]);
#endif
    /* Corre E(R) ^ K por las cajas S & P combinadas
     * Este codigo toma ventaja de la regularidad y conveniencia en
     * E, sabiendo que cada grupo de 6 bits en E(R) que alimenta
     * una S-box simple es un segmento contiguo de R.
     */
    rt = (r >> 1) | ((r & 1) ? 0x80000000 : 0);
    rval = 0;
    rval |= sp[0][((rt >> 26) ^ *subkey++) & 0x3f];
    rval |= sp[1][((rt >> 22) ^ *subkey++) & 0x3f];
    rval |= sp[2][((rt >> 18) ^ *subkey++) & 0x3f];
    rval |= sp[3][((rt >> 14) ^ *subkey++) & 0x3f];
    rval |= sp[4][((rt >> 10) ^ *subkey++) & 0x3f];
    rval |= sp[5][((rt >> 6) ^ *subkey++) & 0x3f];
    rval |= sp[6][((rt >> 2) ^ *subkey++) & 0x3f];
    rt = (r << 1) | ((r & 0x80000000) ? 1 : 0);
    rval |= sp[7][(rt ^ *subkey) & 0x3f];
#ifdef  TRACE
    printf(" %08lx\n",rval);
#endif
    return rval;
}

#ifdef  LITTLE_ENDIAN
/* Intercambia un Byte por un long */
static unsigned long byteswap(unsigned long x)
{
    register char *cp,tmp;

    cp = (char *)&x;
    tmp = cp[3];
    cp[3] = cp[0];
    cp[0] = tmp;

    tmp = cp[2];
    cp[2] = cp[1];
    cp[1] = tmp;

    return x;
}
#endif


/* Realiza una ronda de cifrado DES  */
static round(int num,unsigned long *block)
{
    /* Las rondas son nomnbradas del 0 al 15. Por cada ronda de derecha es
     * seguida de una funcion f() y el resultado se aplica con un XOR exclusivo
     * a la otra mitad izquierda; en cada ronda impar el reves es realizado.
     */
    if(num & 1){
        block[1] ^= f(block[0],kn[num]);
    } else {
        block[0] ^= f(block[1],kn[num]);
    }
    return 0;
}

/* Inicializa y prepara la clave  */
```

```
int setkey(char *key)
{
    char pc1m[56];      /* lugar donde se modifica pc1  */
    char pcr[56];
    register int i,j,l;
    int m;

    /* En modo 2, los 128 bytes de sub-clave son asigandos directamente de la
     * clave del usuario, la asignacion es usada completamente independiente
     * de cada sub-clave por cada ronda.
     *
     */
    if(desmode == 2){
        for(i=0;i<16;i++)
            for(j=0;j<8;j++)
                kn[i][j] = *key++;
        return 0;
    }
    /* Limpia la clave  */
    for (i=0; i<16; i++)
        for (j=0; j<8; j++)
            kn[i][j]=0;

    for (j=0; j<56; j++) {      /* convierte pc1 a los  bits de la clave */
        l=pc1[j]-1;
        m = l & 07;
        pc1m[j]=(key[l>>3] &
            bytebit[m])
            ? 1 : 0;
    }
    for (i=0; i<16; i++) {      /* un trozo de clave por cada iteracion*/
        for (j=0; j<56; j++)    /* se rota pc1 ala derecha para acumular */
            pcr[j] = pc1m[(l=j+totrot[i])<(j<28? 28 : 56) ? l: l-28];
            /* rota las mitades derecha y izquierda independientemente */
        for (j=0; j<48; j++){   /* seleccion los bits individialmente */
            /* verifica el bit que se asigna a kn[j] */
            if (pcr[pc2[j]-1]){
                /* lo enmascara si este esta ahi */
                l= j % 6;
                kn[i][j/6] |= bytebit[l] >> 2;
            }
        }
    }
    return 0;
}
/* Realiza la encriptacion del bloque de 64 bits */
void endes(char *block)
{
    register int i;
    unsigned long work[2];      /* Area de almacenamiento de trabajo */
    long tmp;

    permute(block,iperm,(char *)work);  /* permutación inicial */
#ifdef LITTLE_ENDIAN
    work[0] = byteswap(work[0]);
    work[1] = byteswap(work[1]);
#endif

    /* Realiza las 16 rondas */
    for (i=0; i<16; i++)
        round(i,work);

    /* intercambia las mitades derecha y izquierda */
    tmp = work[0];
    work[0] = work[1];
    work[1] = tmp;
```

```
#ifdef LITTLE_ENDIAN
    work[0] = byteswap(work[0]);
    work[1] = byteswap(work[1]);
#endif
    permute((char *)work,fperm,block);  /* inversa de la permutacion inicial */
}

/* Reliza la desencriptación del bloque de 64-bits */
void dedes(char *block)
{
    register int i;
    unsigned long work[2];  /* Area de trabajo y alamacenamiento de datos */
    long tmp;
    permute(block,iperm,(char *)work);  /* Permutación inicial */
#ifdef LITTLE_ENDIAN
    work[0] = byteswap(work[0]);
    work[1] = byteswap(work[1]);
#endif
    /* intercambio de midad izquierda y derecha */
    tmp = work[0];
    work[0] = work[1];
    work[1] = tmp;
    /* reliza las 16 rondas en orden inverso */
    for (i=15; i >= 0; i--)
        round(i,work);
#ifdef LITTLE_ENDIAN
    work[0] = byteswap(work[0]);
    work[1] = byteswap(work[1]);
#endif
    permute((char *)work,fperm,block);  /* inversa de la permutacion inicial */
}

/* covienrte de  hexadecimal a ascii nybble a binario */
int htoa(char c)
{
    if(c >= '0' && c <= '9')
        return c - '0';
    if(c >= 'a' && c <= 'f')
        return 10 + c - 'a';
    if(c >= 'A' && c <= 'F')
        return 10 + c - 'A';
    return -1;
}
/* convierte bytes de hex/ascii a binario */
void gethex(register char *result,register char *cp,register int cnt)
{
    while(cnt-- != 0)
    {
        *result = htoa(*cp++) << 4;
        *result++ |= htoa(*cp++);
    }
}
#ifdef  DEBUG
put8(register char *cp;)
{
    int i;
    for(i=0;i<8;i++)
    {
        fprintf(stderr,"%02x ",*cp++ & 0xff);
    }
}
#endif

/************************ FIN RUTINAS DES*************************************/
```

```
char iv[8]; /* vector inicial para  modo CBC  */
int block;
void DES(int argc,char *argv[])
{
    FILE *fp1=NULL, *fp2=NULL;
    int c,cnt,encrypt,decrypt,hexflag;
    register int i;
    char key[8],tkey1[20],tkey2[20],*akey,*arch1,*arch2;
    extern char *optarg;
    optind=1;
    hexflag = block = encrypt = decrypt = 0;
    akey = NULL;
    while((c = getopt(argc,argv,"hedk:s:f:b")) != EOF)
    {
        switch(c){
        case 'h':
            hexflag++;
            break;
        case 'e':
            encrypt++;
            break;
        case 'd':
            decrypt++;
            break;
        case 'k':
            akey = optarg;
            break;
        case 'b':
            block++;
            break;
        case 'f':
            arch1=optarg;
            break;
        case 's':
            arch2=optarg;
            break;
        }
    }
    if((fp1=fopen(arch1, "rb"))==NULL)
    {   /*abre archivo origen para lectura*/
         printf(" !!! No se puede abrir archivo de entrada\n !!!");
         return ;
    }
    if((fp2=fopen(arch2, "wb"))==NULL)
    {   /*abre archivo destino para escritura*/
        printf(" !! No se puede abrir archivo de salida !!!");
        return ;
    }
    if(encrypt == 0 && decrypt == 0)
    {
        fprintf(stderr,"Utilize: des -e|-d [-h] [-k key] -f arch_entrada -s arch_salida
\n");
        return ;
    }
    if(akey == NULL)
    {
        /* No hay clave indicada en la linea de comando, se optiene del prompt*/
        memset(tkey1,0,sizeof(tkey1));
        memset(tkey2,0,sizeof(tkey2));
        for(;;)
        {
            akey = getpass("Introduca la clave: ");
            strncpy(tkey1,akey,sizeof(tkey1));
            akey = getpass("Introduca la clave nuevamente: ");
            strncpy(tkey2,akey,sizeof(tkey2));
            if(strncmp(tkey1,tkey2,sizeof(tkey1)) != 0)
```

```
            {
                fprintf(stderr,"Las claves no coinciden, trate de nuevo\n");
            }
            else
                break;
        }
        akey = tkey1;
    }
    if(hexflag)
    {
        for(i=0;i<16;i++)
        {
            if(htoa(akey[i]) == -1)
            {
                fprintf(stderr,"No es hexadecimal el caracter en la clave\n");
                return;
            }
        }
        gethex(key,akey,8);
    }
    else
    {
        strncpy(key,akey,8);
        /* prepara la clave, obtiene el bit de paridad */
        for(cnt = 0; cnt < 8; cnt++)
        {
            c = 0;
            for(i=0;i<7;i++)
                if(key[cnt] & (1 << i))
                    c++;
            if((c & 1) == 0)
                key[cnt] |= 0x80;
            else
                key[cnt] &= ~0x80;
        }
    }
    i = strlen(akey);
    i = (i < 8) ? i : 8;
    memset(akey,0,i);
    desinit(0);
    setkey(key);
    /* Inicializa las inversas y todo a cero */
    memset(iv,0,8);
    if(encrypt)
    {
        doencrypt(fp1,fp2);
        puts("");
        printf("\n        !! Cifrado Terminado !!\n");
    } else
    {
        dodecrypt(fp1,fp2);
        puts("");
        printf("\n      !! Descifrado Terminado !!\n");
    }
    fclose(fp1);
    fclose(fp2);
    return;
}

/* Encripta  un entrada estandar a una salida estandar */
void doencrypt(FILE *fp1,FILE *fp2)
{
    char work[8],*cp,*cp1;
    int cnt,i;
    for(;;)
    {
```

```
        if((cnt = fread(work,1,8,fp1)) != 8)
        {
            /* pone el conteo de bytes residuales en el ultimo bloque
             * Note que la basura esta a la izquierda en los otros bytes,
             * si hay alguna; esta caracteristica no es un error,
             * este es ser corregido al momento de la desencriptacion.
             */
            work[7] = cnt;
        }
        if(!block)
        {
            /* CBC mode; cadena a la ultima palabra cifrada*/
            cp = work;
            cp1 = iv;
            for(i=8; i!=0; i--)
                *cp++ ^= *cp1++;
        }
        endes(work);    /* bloque encriptado */
        if(!block)
        { /* salva la cadena del texto cifrado a la cadena */
            memcpy(iv,work,8);
        }
        fwrite(work,1,8,fp2);
        if(cnt != 8)
            break;
    }
}
void dodecrypt(FILE *fp1,FILE *fp2)
{
    char work[8],nwork[8],ivtmp[8],*cp,*cp1;
    int cnt,i;

    cnt = fread(work,1,8,fp1);
    for(;;)
    {
        if(!block)
        { /* Salva el entrante texto cifrado para la cadena */
            memcpy(ivtmp,work,8);
        }
        dedes(work);
        if(!block)
        { /* desencadena el bloque, salva el texto
             cifrado para el siguiente*/
            cp = work;
            cp1 = iv;
            for(i=8; i!=0; i--){
                *cp++ ^= *cp1++;
            }
            memcpy(iv,ivtmp,8);
        }
        /* Salva el buffer pendiente para la siguiente lectura */
        memcpy(nwork,work,8);
        /* Trata de leer el siguiente bloque */
        cnt = fread(work,1,8,fp1);
        if(cnt != 8)
        {   /* Este solo puede ser 0 si no es 8 */
            /* el previo bloque notifico del numero de bytes
             */
            cnt = nwork[7];
            if(cnt < 0 || cnt > 7){
                fprintf(stderr,"!! Archivo corrupto o clave incorrecta !!\n");
            } else if(cnt != 0)
                fwrite(nwork,1,cnt,fp2);
            return ;
        }
        else
```

```
        {  /* todo esta correcto para escribir el previo bloque */
           fwrite(nwork,1,8,fp2);
        }
    }
}

void Alg_Des(char ps_Opt)
{
   char clave[20],arch_in[50],arch_out[50];
   char *argumento[8];
   clrscr();
   puts("          TECNICAS CRIPTOGRAFIAS");
   puts("");
   puts("             CIFRADO  DES");
   puts("");
   puts("");
   printf("%s","Nombre y Ruta de Archivo de Entrada:"); scanf("%s",&arch_in);
   printf("%s","Nombre y Ruta de Archivo de Salida:"); scanf("%s",&arch_out);
   printf("%s","Clave de Cifrado(MAX(20):"); scanf("%s",&clave);
   if (ps_Opt=='e' ||ps_Opt=='E')
   {
      argumento[1]="-e";
      argumento[2]="-k";
      argumento[3]=clave;
      argumento[4]="-f";
      argumento[5]=arch_in;
      argumento[6]="-s";
      argumento[7]=arch_out;
      DES(8,argumento);
   }
   if (ps_Opt=='d' ||ps_Opt=='D')
   {
      argumento[1]="-d";
      argumento[2]="-k";
      argumento[3]=clave;
      argumento[4]="-f";
      argumento[5]=arch_in;
      argumento[6]="-s";
      argumento[7]=arch_out;
      DES(8,argumento);
   }
   puts("");puts("");puts("");
   puts("        <Presione cualquier tecla para continuar>");
   getch();
}
/*****************************************************************************/
```

6.- Rijndael.

```
/*****************************************************************************/
/*  TESIS- TECNICAS CRIPTOGRAFICAS                                           */
/*  AUTOR: Guillermo Gomez Villa                                             */
/*  FECHA: Mayo 2004                                                         */
/*****************************************************************************/

/************ rijndael.h *****************/

/************************** RIJNDAEL ************************************/
/* Rijndael Block Cipher - rijndael.c
   Basado en la implementacion de  Mike Scott
*****************************************************************************/
```

```
#define BYTE unsigned char       /* 8 bits  */
#define WORD unsigned long       /* 32 bits */

/* rota x un bit a la izquierda */

#define ROTL(x) (((x)>>7)|((x)<<1))

/* Rota 32-bits word a la izquierda por 1, 2 o 3 bytes  */

#define ROTL8(x) (((x)<<8)|((x)>>24))
#define ROTL16(x) (((x)<<16)|((x)>>16))
#define ROTL24(x) (((x)<<24)|((x)>>8))


/* Prepara Datos */

static BYTE InCo[4]={0xB,0xD,0x9,0xE};  /* Coeficientes Inversos */

static BYTE fbsub[256];
static BYTE rbsub[256];
static BYTE ptab[256],ltab[256];
static WORD ftable[256];
static WORD rtable[256];
static WORD rco[30];

/* Datos Parametro-Dependencia  */

int Nk,Nb,Nr;
BYTE fi[24],ri[24];
WORD fkey[120];
WORD rkey[120];

static WORD pack(BYTE *b)
{ /* paquete de bytes dentro de un 32-bit Word */
    return ((WORD)b[3]<<24)|((WORD)b[2]<<16)|((WORD)b[1]<<8)|(WORD)b[0];
}

static void unpack(WORD a,BYTE *b)
{ /* desempaca bytes de un word */
    b[0]=(BYTE)a;
    b[1]=(BYTE)(a>>8);
    b[2]=(BYTE)(a>>16);
    b[3]=(BYTE)(a>>24);
}

static BYTE xtime(BYTE a)
{
    BYTE b;
    if (a&0x80) b=0x1B;
    else        b=0;
    a<<=1;
    a^=b;
    return a;
}

static BYTE bmul(BYTE x,BYTE y)
{ /* x.y= AntiLog(Log(x) + Log(y)) */
    if (x && y) return ptab[(ltab[x]+ltab[y])%255];
    else return 0;
}

static WORD SubByte(WORD a)
{
    BYTE b[4];
    unpack(a,b);
    b[0]=fbsub[b[0]];
```

```
    b[1]=fbsub[b[1]];
    b[2]=fbsub[b[2]];
    b[3]=fbsub[b[3]];
    return pack(b);
}

static BYTE product(WORD x,WORD y)
{ /* producto punto de dos arreglos de 4-byte  */
    BYTE xb[4],yb[4];
    unpack(x,xb);
    unpack(y,yb);
    return bmul(xb[0],yb[0])^bmul(xb[1],yb[1])^bmul(xb[2],yb[2])^bmul(xb[3],yb[3]);
}

static WORD InvMixCol(WORD x)
{ /* Multiplication de Matrices*/
    WORD y,m;
    BYTE b[4];

    m=pack(InCo);
    b[3]=product(m,x);
    m=ROTL24(m);
    b[2]=product(m,x);
    m=ROTL24(m);
    b[1]=product(m,x);
    m=ROTL24(m);
    b[0]=product(m,x);
    y=pack(b);
    return y;
}

BYTE ByteSub(BYTE x)
{
    BYTE y=ptab[255-ltab[x]];  /* Inverso Multiplicativo */
    x=y;  x=ROTL(x);
    y^=x; x=ROTL(x);
    y^=x; x=ROTL(x);
    y^=x; y^=0x63;
    return y;
}

void gentables(void)
{ /* Generacion de Tablas */
    int i;
    BYTE y,b[4];

  /* usa 3 raices primitivas pra generar las de potencia y log  */

    ltab[0]=0;
    ptab[0]=1;  ltab[1]=0;
    ptab[1]=3;  ltab[3]=1;
    for (i=2;i<256;i++)
    {
        ptab[i]=ptab[i-1]^xtime(ptab[i-1]);
        ltab[ptab[i]]=i;
    }

  /* Prepara una transformacion:- cada bit es xor con un bit */

    fbsub[0]=0x63;
    rbsub[0x63]=0;
    for (i=1;i<256;i++)
    {
        y=ByteSub((BYTE)i);
        fbsub[i]=y; rbsub[y]=i;
```

```
    }

    for (i=0,y=1;i<30;i++)
    {
        rco[i]=y;
        y=xtime(y);
    }

  /* calcula la tabla siguiente y la reversa */
    for (i=0;i<256;i++)
    {
        y=fbsub[i];
        b[3]=y^xtime(y); b[2]=y;
        b[1]=y;          b[0]=xtime(y);
        ftable[i]=pack(b);

        y=rbsub[i];
        b[3]=bmul(InCo[0],y); b[2]=bmul(InCo[1],y);
        b[1]=bmul(InCo[2],y); b[0]=bmul(InCo[3],y);
        rtable[i]=pack(b);
    }
}

void gkey(int nb,int nk,char *key)
{ /* tamaño de bloque=32*nb bits. Clave=32*nk bits */
  /* posibles valores nb,bk = 4, 6 or 8          */
  /* key comes as 4*Nk bytes                     */
  /* Key Scheduler. Crea una clave expandida */
    int i,j,k,m,N;
    int C1,C2,C3;
    WORD CipherKey[8];

    Nb=nb; Nk=nk;

  /* Nr es el numero de rondas */
    if (Nb>=Nk) Nr=6+Nb;
    else        Nr=6+Nk;

    C1=1;
    if (Nb<8) { C2=2; C3=3; }
    else      { C2=3; C3=4; }

  /* pre-calcula los incrementos siguientes y reversos */
    for (m=j=0;j<nb;j++,m+=3)
    {
        fi[m]=(j+C1)%nb;
        fi[m+1]=(j+C2)%nb;
        fi[m+2]=(j+C3)%nb;
        ri[m]=(nb+j-C1)%nb;
        ri[m+1]=(nb+j-C2)%nb;
        ri[m+2]=(nb+j-C3)%nb;
    }

    N=Nb*(Nr+1);

    for (i=j=0;i<Nk;i++,j+=4)
    {
        CipherKey[i]=pack((BYTE *)&key[j]);
    }
    for (i=0;i<Nk;i++) fkey[i]=CipherKey[i];
    for (j=Nk,k=0;j<N;j+=Nk,k++)
    {
        fkey[j]=fkey[j-Nk]^SubByte(ROTL24(fkey[j-1]))^rco[k];
        if (Nk<=6)
        {
            for (i=1;i<Nk && (i+j)<N;i++)
```

```
                fkey[i+j]=fkey[i+j-Nk]^fkey[i+j-1];
        }
        else
        {
            for (i=1;i<4 &&(i+j)<N;i++)
                fkey[i+j]=fkey[i+j-Nk]^fkey[i+j-1];
            if ((j+4)<N) fkey[j+4]=fkey[j+4-Nk]^SubByte(fkey[j+3]);
            for (i=5;i<Nk && (i+j)<N;i++)
                fkey[i+j]=fkey[i+j-Nk]^fkey[i+j-1];
        }

    }

 /* Ahora la clave expandida de descifrado se usa en orden reverso */

    for (j=0;j<Nb;j++) rkey[j+N-Nb]=fkey[j];
    for (i=Nb;i<N-Nb;i+=Nb)
    {
        k=N-Nb-i;
        for (j=0;j<Nb;j++) rkey[k+j]=InvMixCol(fkey[i+j]);
    }
    for (j=N-Nb;j<N;j++) rkey[j-N+Nb]=fkey[j];
}

/*Proceso principal de cifrado*/
void encrypt(char *buff)
{
    int i,j,k,m;
    WORD a[8],b[8],*x,*y,*t;

    for (i=j=0;i<Nb;i++,j+=4)
    {
        a[i]=pack((BYTE *)&buff[j]);
        a[i]^=fkey[i];
    }
    k=Nb;
    x=a; y=b;

/* Estado alternado entre a y b */
    for (i=1;i<Nr;i++)
    { /* Nr es el nuemro de rondasnds. podria ser mayor. */

/* Si Nb es preparado - se expande al siguiente
   ciclo y codigo duro en los valores de fi[]  */

        for (m=j=0;j<Nb;j++,m+=3)
        { /* reparto con  cada elemento de  32-bit de los estados  */
          /* Este es el bit tiempo-critico */
            y[j]=fkey[k++]^ftable[(BYTE)x[j]]^
                 ROTL8(ftable[(BYTE)(x[fi[m]]>>8)])^
                 ROTL16(ftable[(BYTE)(x[fi[m+1]]>>16)])^
                 ROTL24(ftable[x[fi[m+2]]>>24]);
        }
        t=x; x=y; y=t;      /* intercambio de punteros*/
    }

/* Ultima Ronda  */
    for (m=j=0;j<Nb;j++,m+=3)
    {
        y[j]=fkey[k++]^(WORD)fbsub[(BYTE)x[j]]^
             ROTL8((WORD)fbsub[(BYTE)(x[fi[m]]>>8)])^
             ROTL16((WORD)fbsub[(BYTE)(x[fi[m+1]]>>16)])^
             ROTL24((WORD)fbsub[x[fi[m+2]]>>24]);
    }
    for (i=j=0;i<Nb;i++,j+=4)
    {
```

```
        unpack(y[i],(BYTE *)&buff[j]);
        x[i]=y[i]=0;    /* clean up stack */
    }
    return;
}

void decrypt(char *buff)
{
    int i,j,k,m;
    WORD a[8],b[8],*x,*y,*t;

    for (i=j=0;i<Nb;i++,j+=4)
    {
        a[i]=pack((BYTE *)&buff[j]);
        a[i]^=rkey[i];
    }
    k=Nb;
    x=a; y=b;

/* Estado Alternados entre a y b */
    for (i=1;i<Nr;i++)
    { /* Nr es el numero de rondas. Podrian ser mas. */

        for (m=j=0;j<Nb;j++,m+=3)
        { /* This is the time-critical bit */
            y[j]=rkey[k++]^rtable[(BYTE)x[j]]^
                 ROTL8(rtable[(BYTE)(x[ri[m]]>>8)])^
                 ROTL16(rtable[(BYTE)(x[ri[m+1]]>>16)])^
                 ROTL24(rtable[x[ri[m+2]]>>24]);
        }
        t=x; x=y; y=t;      /* swap pointers */
    }

/* Ultima Ronda- */
    for (m=j=0;j<Nb;j++,m+=3)
    {
        y[j]=rkey[k++]^(WORD)rbsub[(BYTE)x[j]]^
             ROTL8((WORD)rbsub[(BYTE)(x[ri[m]]>>8)])^
             ROTL16((WORD)rbsub[(BYTE)(x[ri[m+1]]>>16)])^
             ROTL24((WORD)rbsub[x[ri[m+2]]>>24]);
    }
    for (i=j=0;i<Nb;i++,j+=4)
    {
        unpack(y[i],(BYTE *)&buff[j]);
        x[i]=y[i]=0;    /* clean up stack */
    }
    return;
}
int RIJNDAEL(char opt);
int RIJNDAEL(char opt)
{   int i,nb,nk;
    int salir;
    char key[32];
    char block[32];
    char debug[1]={"N"};
    char arch_entrada[20],arch_salida[20];
    FILE *fp1,*fp2;
    salir = FALSO;
    for (i=0;i<32;i++) key[i]=0;
    key[0]=1;

    gentables();

    puts("");
    printf("Archivo de Entrada:");scanf("%s",arch_entrada);
    printf("Archivo de Salida :");scanf("%s",arch_salida);
```

```
printf("\nClave: ");scanf("%s",key);
printf(" Debug?(S/N) :");scanf("%s",debug);
if((fp1=fopen(arch_entrada, "rb"))==NULL)
{  /*abre archivo origen para lectura*/
   puts(""); puts("");
   printf(" !! No se puede abrir archivo de entrada !! \n ");
   return(0);
}
if((fp2=fopen(arch_salida, "wb"))==NULL)
{  /*abre archivo destino para escritura*/
   puts(""); puts("");
   printf(" !! No se puede crear archivo de salida !! \n ");
   return(0);
}
fseek(fp1, SEEK_SET, 0);
do
{
   for (i=0;i<32;i++) block[i]=0;
   if (fread(block,sizeof(block),1,fp1)==NULL)
   {
     if(strlen(block)>0)
     {
       salir = 1 ;
       block[strlen(block)]='\0';
     }
     else  break;
   }
   nb=8;
   nk=8;
   if (debug[0]=='S' ||debug[0]=='s')
   {
     clrscr();
     printf("\nTamano de Bloque= %d bits, Tamano de Clave= %d bits\n",nb*32,nk*32);
     printf("\n");
   }
   gkey(nb,nk,key); /* Preparacion de la clave */
   if(opt=='E' || opt=='e' )
   {
     if (debug[0]=='S' ||debug[0]=='s')
     {
        printf("TextoClaro     = ");
        printf("%s",block);
        printf("\n");
        printf("TextoClaro hex = ");
        for (i=0;i<nb*4;i++) printf("%02x",block[i]);
     }
     encrypt(block); /* Encriptacion del bloque */
     if (debug[0]=='S' ||debug[0]=='s')
     {
        printf("\n");
        printf("Cifrado hex    = ");
        for (i=0;i<nb*4;i++) printf("%02x",(unsigned char)block[i]);
     }
     fwrite(block,sizeof(block),1,fp2);
   }
   else if(opt=='D' || opt=='d' )
   {
     if (debug[0]=='S' ||debug[0]=='s')
     {
        printf("Cifrado hex    = ");
        for (i=0;i<nb*4;i++) printf("%02x",(unsigned char)block[i]);
        printf("\n");
     }
     decrypt(block);      /* Desencriptacion del bloque */
     if (debug[0]=='S' ||debug[0]=='s')
     {
```

```
            printf("Descifrado     = ");
            printf("%s",block);
            printf("\n");
            printf("Descifrado hex = ");
            for (i=0;i<nb*4;i++) printf("%02x",block[i]);
          }
          fwrite(block,sizeof(block),1,fp2);
        }
        if (debug[0]=='S' ||debug[0]=='s')
        {
           printf("\n\n");
           puts("     <PRESIONE CUALQUIER TECLA PARA CONTINUAR>  ");
           getch();
        }
    }while(salir!=VERDADERO);
    fclose(fp1);
    fclose(fp2);
    if(opt=='E' || opt=='e' )
            printf("\n        !! Cifrado Terminado !!\n");
    else if(opt=='D' || opt=='d' )
            printf("\n      !! Descifrado Terminado !!\n");
    return 0;
}

void Alg_Rijndael(char ps_Opt)
{
   clrscr();
   puts("          TECNICAS CRIPTOGRAFIAS");
   puts("");
   puts("            CIFRADO  RIJNDAEL");
   puts("");
   RIJNDAEL(ps_Opt);
   puts("");
   puts("");puts("");puts("");
   puts("        <Presione cualquier tecla para continuar>");
   getch();
}
/*****************************************************************************/
```

7.- RSA.

```
/*****************************************************************************/
/*  TESIS- TECNICAS CRIPTOGRAFICAS                                           */
/*  AUTOR: Guillermo Gomez Villa                                             */
/*  FECHA: Mayo 2004                                                         */
/*****************************************************************************/
/******************* rsa.h ***************************************/
#define MAX_NUM_RAND  65000

unsigned long int inverse (unsigned long int, unsigned long int);
unsigned long int gcd (unsigned long int, unsigned long int);
void seedRandom (void);
unsigned long int findPrime (unsigned long int, unsigned long int);
void findKeys (unsigned long int *, unsigned long int *, unsigned long int *);
/*****************************************/
void seedRandom (void)
{
  randomize();
  return;
}
/*
   Aleatoriamente busca el numero primo entre  un numero menor y otro mayor
   (http://www.utm.edu/research/primes/glossary/WheelFactorization.html)
```

```
   para prueba de primalidad. Esto es mucho mas rapido con avanzada pruebas de
   primalidad y especialmente con prueba de probabilidad de primalidad,
   cualquier seria implementacion RSA deberia usar.
*/
unsigned long int findPrime (unsigned long int lower, unsigned long int upper)
{
  /* Un conjunto de arreglos que usaremos para pruebas de primalidad */
  unsigned long int test [] = { 2, 3, 5, 7 };
  unsigned long int divisors [] = { 1, 11, 13, 17, 19, 23, 29, 31, 37, 41,
                    43, 47, 53, 59, 61, 67, 71, 73, 79, 83,
                    89, 97, 101, 103, 107, 109, 113, 121, 127, 131,
                    137, 139, 143, 149, 151, 157, 163, 167, 169, 173,
                    179, 181, 187, 191, 193, 197, 199, 209 };
  unsigned long int numDivisors = 48, numTest = 4, newPrime = 0, testProduct = 2 * 3 * 5 *
7;
  unsigned long int root = 0, multiple = 0, counter = 0, currentPrime = 0;

  /* Ahora nosotros generamos un numero aleatorio en un especifico rango y realizamos
     pruebas de factorizacion para verificar primalidad hasta encontrar un numero primo
  */
  do
    {
      newPrime = random (MAX_NUM_RAND) % (upper - lower) + lower - 2;
    }
  while (newPrime % 2 == 0);
  currentPrime = 0;
  while (! currentPrime)
    {
      newPrime += 2;
      if (newPrime > upper)
    {
      newPrime -= lower;
    }
      if (newPrime < lower)
    {
      newPrime += lower;
    }
      currentPrime = newPrime;
      /* Prueba la factorizacion con otros primos  */
      for (counter = 1; counter < numTest && currentPrime; ++ counter)
    {
      if (currentPrime % test [counter] == 0)
        {
              currentPrime = 0;
        }
    }
      if (currentPrime)
    {
      root = ceil ( sqrt (currentPrime) );
      for (counter = 1, multiple = 0; currentPrime && divisors [counter] + multiple <= root;
++ counter)
        {
          if (currentPrime % (divisors [counter] + multiple) == 0)
        {
          currentPrime = 0;
        }
          /* Incremento en multiplos cuando es necesario */
          if (counter == numDivisors)
        {
          counter = 0;
          multiple += testProduct;
        }
        }
    }
    }
  return currentPrime;
```

```
}

unsigned long int exponentiate (unsigned long int p, unsigned long int q,
                unsigned long int m)
{
  unsigned long int square = 0, result = 0;
  /* M debe de ser menor a 65536 o se puede generar un desborde de memoria */
  /* Realiza el primer paso*/
  square = p % m;
  if (q & 0x1)
    {
      result = square;
    }
  q = q >> 1;
  while (q != 0)
    {
      square = (square * square) % m;
      if (q & 0x1)
    {
      if (result == 0)
        {
          result = square;
        }
      else
        {
          result = (result * square) % m;
        }
    }
      q = q >> 1;
    }
  return result;
}

unsigned long int gcd (unsigned long int a, unsigned long int b)
{
  unsigned long int r = a, oldR = 0;
  while (r != 0)
    {
      oldR = r;
      r = a % b;
      a = b;
      b = r;
    }
  return oldR;
}

unsigned long int inverse (unsigned long int number, unsigned long int mod)
{
  long int quotient = 0, x = 0, y = 1,
  oldx = 1, oldy = 0, newx = 0, newy = 0;
  if (gcd (number, mod) != 1)
  {
    return 0;
  }
  /* Algoritmo:
     do {
     quotient <- Divide oldx * number + oldy * mod by x * number + y * mod
     newx <- Subtract x * quotient from oldx
     newy <- Subtract y * quotient from oldy
     oldx <- x
     oldy <- y
     x <- newx
     y <- newy
     } while ( x * number + y * mod != 1)
  */
  while ( (x * number) + (y * mod) != 1 )
```

```
  {
    quotient = ((oldx * number) + (oldy * mod)) / ((x * number) + (y * mod));
    newx = oldx - (x * quotient);
    newy = oldy - (y * quotient);
    oldx = x;
    oldy = y;
    x = newx;
    y = newy;
  }
  if (x < 0)
  {
    x += mod;
  }
  return x;
}

/*
  Encuentra las claves para el Criptosistema RSA,  almacenados en e (publico) and d
(privado)
  con tamaño n
*/
void findKeys (unsigned long int * e, unsigned long int * d, unsigned long int * n)
{
  unsigned long int p = 0, q = 0, phi = 0;

  /* Busca el aleatoriamente y obtiene p, q */
  seedRandom ();

  /* Optiene p, q (length = 8,8 bits para p*q = n es al menos de 15 bits 15 bits y n < 2^16
(desbordamiento) */
  p = findPrime (128, 255);
  do
  {
    q = findPrime (128, 255);
  } while (q == p);
//  fprintf (stderr, "\n p = %lu,\n q = %lu\n", p, q);
  /* n = pq, phi = (p - 1)(q - 1) */
  *n = p*q;
  phi = (p - 1) * (q - 1);

  /* selecciona e el cual el gcd (e, phi) == 1 y selecciona e < n  */
  do
  {
    *e = 0;
    do
    {
      *e = random (MAX_NUM_RAND) % *n;
    }
    while (*e % 2 == 0);
  }
  while (gcd (*e, phi) != 1);

  /* encuentra d, La inversa de e (mod phi) */
  *d = inverse (*e, phi);

  return;
}

/*********************************************/
/*
  RSA: IMPLEAMENTACION
*/
int RSA_I(char opt)
{
  char arch_entrada[20],arch_salida[20];
  unsigned long int e = 0, d = 0, n = 0;
```

```
  unsigned long int cyphertext = 0, decrypted = 0, data = 0;
  size_t bytesRead = 0;
  FILE *input = NULL;
  FILE *output = NULL;
  printf("\nArchivo Entrada : ");scanf("%s",arch_entrada);
  printf("\nArchivo Salida  : ");scanf("%s",arch_salida);
  if ((input = fopen (arch_entrada, "rb"))== NULL)
  {
     puts("!! No se puede abrir el archivo !!");
     return 0;
  }
  if ((output = fopen (arch_salida, "wb"))== NULL)
  {
     puts ("!! No se puede crear el archivo !!");
     return 0;
  }
  if(opt == 'E' || opt == 'e')
  {
     printf("\nClave e: ");scanf("%ul",&e);
     printf("\nClave n: ");scanf("%ul",&n);
     printf("\n\nCifrando ..");
     while (( bytesRead = fread (&data, 1, 1, input) ))
     {
        /* IMPORTANTE: data < n para RSA si no falla! */
        assert (data < n);
        /* Cifrado C = M^e mod n */
        cyphertext = exponentiate (data, e, n);
        fwrite (&cyphertext, sizeof(int), 1, output);
     }
     printf("\n\n!! Cifrado Terminado !!");
  }
  else if(opt == 'D' || opt == 'd')
  {
     printf("\nClave d: ");scanf("%ul",&d);
     printf("\nClave n: ");scanf("%ul",&n);
     printf("\n\nDescifrando ...");
     while (( bytesRead = fread (&data,sizeof(int), 1, input) ))
     {
        /* IMPORTANTE: data < n para RSA si no falla! */
        assert (data < n);
        /* Descifrado  data = C^d mod n */
        decrypted = exponentiate (data, d, n);
        fwrite (&decrypted,1, 1, output);
     }
     printf("\n\n!! Descifrado Terminado !!");
  }
  fclose (input);
  fclose (output);
  return 0;
}

void Generar_Claves(void)
{
  unsigned long int e = 0, d = 0, n = 0;
  /* Generara algunas claves publica y privadas  */
  findKeys (&e, &d, &n);
  fprintf (stderr, "Clave Publica = %lu\nClave Privada = %lu\nLongitud = %lu\n",e, d, n);
};

void Alg_RSA(char ps_Opt);
void Alg_RSA(char ps_Opt)
{
   clrscr();
   puts("          TECNICAS CRIPTOGRAFIAS");
   puts("");
   puts("            CIFRADO  RSA");
```

```
   puts("");
   if(ps_Opt=='C' || ps_Opt=='c' )Generar_Claves();
   else RSA_I(ps_Opt);
   puts("");
   puts("");puts("");puts("");
   puts("        <Presione cualquier tecla para continuar>");
   getch();
}
```

8.- MD5.

```
/*************************************************************************/
/*  TESIS- TECNICAS CRIPTOGRAFICAS                                       */
/*  AUTOR: Guillermo Gomez Villa                                         */
/*  FECHA: Mayo 2004                                                     */
/*************************************************************************/

/************************ md5  ************************************/

typedef unsigned char md5_byte_t; /* 8-bit byte */
typedef unsigned int md5_word_t; /* 32-bit word */

/* Define los estados de el Algortimo MD5 . */
typedef struct md5_state_s {
    md5_word_t count[2];    /* Tamaño del mensake en bits,  */
    md5_word_t abcd[4];     /* buffer del resumen*/
    md5_byte_t buf[64];     /* bloque de acumulacion */
} md5_state_t;

#ifdef __cplusplus
extern "C"
{
#endif

/* Inicialiaza el algoritmo */
#ifdef P1
void md5_init(P1(md5_state_t *pms));
#else
void md5_init(md5_state_t *pms);
#endif

/* Agrega una cadena a el mensaje. */
#ifdef P3
void md5_append(P3(md5_state_t *pms, const md5_byte_t *data, int nbytes));
#else
void md5_append(md5_state_t *pms, const md5_byte_t *data, int nbytes);
#endif

/* Finaliza el mensake y retorna el resumen. */
#ifdef P2
void md5_finish(P2(md5_state_t *pms, md5_byte_t digest[16]));
#else
void md5_finish(md5_state_t *pms, md5_byte_t digest[16]);
#endif

#ifdef __cplusplus
}
#endif

void MD5_I(void)
{
    char s[100];
    char *test[7] = {
```

```
    "", /*d41d8cd98f00b204e9800998ecf8427e*/
    "945399884.61923487334tuvga", /*0cc175b9c0f1b6a831c399e269772661*/
    "abc", /*900150983cd24fb0d6963f7d28e17f72*/
    "message digest", /*f96b697d7cb7938d525a2f31aaf161d0*/
    "abcdefghijklmnopqrstuvwxyz", /*c3fcd3d76192e4007dfb496cca67e13b*/
    "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789",
                /*d174ab98d277d9f5a5611c2c9f419d9f*/
    "12345678901234567890123456789012345678901234567890123456789012345678901234567890"
/*57edf4a22be3c955ac49da2e2107b67a*/
    };
    int i,j,di;
    clrscr();
    puts("          TECNICAS CRIPTOGRAFIAS");
    puts("");
    puts("             RESUMENES MD5");
    puts("");
    printf("\nResumenes :\n");
    for (j = 0; j < 3; ++j)
    {
      printf("\nTEXTO[%i]:",j);scanf("%s",s);
      strcpy(test[1],s);
      for (i = 1; i < 2; ++i)
      {
        md5_state_t state;
        md5_byte_t digest[16];
        md5_init(&state);
        md5_append(&state, (const md5_byte_t *)test[i], strlen(test[i]));
        md5_finish(&state, digest);
        printf("MD5 (\"%s\") =\n              ", test[i]);
        for (di = 0; di < 16; ++di)
            printf("%02x", digest[di]);
      }
    }
    printf("\n\n");
    puts("     <Presione cualquier tecla para continuar>");
    getch();
    return ;
}

#define T1 0xd76aa478
#define T2 0xe8c7b756
#define T3 0x242070db
#define T4 0xc1bdceee
#define T5 0xf57c0faf
#define T6 0x4787c62a
#define T7 0xa8304613
#define T8 0xfd469501
#define T9 0x698098d8
#define T10 0x8b44f7af
#define T11 0xffff5bb1
#define T12 0x895cd7be
#define T13 0x6b901122
#define T14 0xfd987193
#define T15 0xa679438e
#define T16 0x49b40821
#define T17 0xf61e2562
#define T18 0xc040b340
#define T19 0x265e5a51
#define T20 0xe9b6c7aa
#define T21 0xd62f105d
#define T22 0x02441453
#define T23 0xd8a1e681
#define T24 0xe7d3fbc8
#define T25 0x21e1cde6
#define T26 0xc33707d6
#define T27 0xf4d50d87
```

```
#define T28 0x455a14ed
#define T29 0xa9e3e905
#define T30 0xfcefa3f8
#define T31 0x676f02d9
#define T32 0x8d2a4c8a
#define T33 0xfffa3942
#define T34 0x8771f681
#define T35 0x6d9d6122
#define T36 0xfde5380c
#define T37 0xa4beea44
#define T38 0x4bdecfa9
#define T39 0xf6bb4b60
#define T40 0xbebfbc70
#define T41 0x289b7ec6
#define T42 0xeaa127fa
#define T43 0xd4ef3085
#define T44 0x04881d05
#define T45 0xd9d4d039
#define T46 0xe6db99e5
#define T47 0x1fa27cf8
#define T48 0xc4ac5665
#define T49 0xf4292244
#define T50 0x432aff97
#define T51 0xab9423a7
#define T52 0xfc93a039
#define T53 0x655b59c3
#define T54 0x8f0ccc92
#define T55 0xffeff47d
#define T56 0x85845dd1
#define T57 0x6fa87e4f
#define T58 0xfe2ce6e0
#define T59 0xa3014314
#define T60 0x4e0811a1
#define T61 0xf7537e82
#define T62 0xbd3af235
#define T63 0x2ad7d2bb
#define T64 0xeb86d391

static void md5_process(md5_state_t *pms, const md5_byte_t *data /*[64]*/)
{
    md5_word_t
    a = pms->abcd[0], b = pms->abcd[1],
    c = pms->abcd[2], d = pms->abcd[3];
    md5_word_t t;

#ifndef ARCH_IS_BIG_ENDIAN
# define ARCH_IS_BIG_ENDIAN 1
#endif
#if ARCH_IS_BIG_ENDIAN

    md5_word_t X[16];
    const md5_byte_t *xp = data;
    int i;

    for (i = 0; i < 16; ++i, xp += 4)
    X[i] = xp[0] + (xp[1] << 8) + (xp[2] << 16) + (xp[3] << 24);

#else
    md5_word_t xbuf[16];
    const md5_word_t *X;

    if (!((data - (const md5_byte_t *)0) & 3)) {
    /* data are properly aligned */
    X = (const md5_word_t *)data;
    } else {
    /* not aligned */
```

```
    memcpy(xbuf, data, 64);
    X = xbuf;
    }
#endif

#define ROTATE_LEFT(x, n) (((x) << (n)) | ((x) >> (32 - (n))))

    /* Ronda 1. */
    /*  [abcd k s i] denota la operacion
       a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */
#define F(x, y, z) (((x) & (y)) | (~(x) & (z)))
#define SET(a, b, c, d, k, s, Ti)\
  t = a + F(b,c,d) + X[k] + Ti;\
  a = ROTATE_LEFT(t, s) + b
  /* Realiza las siguientes 16 operaciones. */
    SET(a, b, c, d,  0,  7,  T1);
    SET(d, a, b, c,  1, 12,  T2);
    SET(c, d, a, b,  2, 17,  T3);
    SET(b, c, d, a,  3, 22,  T4);
    SET(a, b, c, d,  4,  7,  T5);
    SET(d, a, b, c,  5, 12,  T6);
    SET(c, d, a, b,  6, 17,  T7);
    SET(b, c, d, a,  7, 22,  T8);
    SET(a, b, c, d,  8,  7,  T9);
    SET(d, a, b, c,  9, 12, T10);
    SET(c, d, a, b, 10, 17, T11);
    SET(b, c, d, a, 11, 22, T12);
    SET(a, b, c, d, 12,  7, T13);
    SET(d, a, b, c, 13, 12, T14);
    SET(c, d, a, b, 14, 17, T15);
    SET(b, c, d, a, 15, 22, T16);
#undef SET

     /* Ronda 2. */
     /* [abcd k s i] denota la operacion
          a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */
#define G(x, y, z) (((x) & (z)) | ((y) & ~(z)))
#define SET(a, b, c, d, k, s, Ti)\
  t = a + G(b,c,d) + X[k] + Ti;\
  a = ROTATE_LEFT(t, s) + b
 /* Realiza las siguientes 16 operaciones. */
    SET(a, b, c, d,  1,  5, T17);
    SET(d, a, b, c,  6,  9, T18);
    SET(c, d, a, b, 11, 14, T19);
    SET(b, c, d, a,  0, 20, T20);
    SET(a, b, c, d,  5,  5, T21);
    SET(d, a, b, c, 10,  9, T22);
    SET(c, d, a, b, 15, 14, T23);
    SET(b, c, d, a,  4, 20, T24);
    SET(a, b, c, d,  9,  5, T25);
    SET(d, a, b, c, 14,  9, T26);
    SET(c, d, a, b,  3, 14, T27);
    SET(b, c, d, a,  8, 20, T28);
    SET(a, b, c, d, 13,  5, T29);
    SET(d, a, b, c,  2,  9, T30);
    SET(c, d, a, b,  7, 14, T31);
    SET(b, c, d, a, 12, 20, T32);
#undef SET

     /* Ronda 4. */
     /* [abcd k s t] denota la operacion
          a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */
#define H(x, y, z) ((x) ^ (y) ^ (z))
#define SET(a, b, c, d, k, s, Ti)\
  t = a + H(b,c,d) + X[k] + Ti;\
  a = ROTATE_LEFT(t, s) + b
```

```
 /* Realiza las siguientes 16 operaciones. */
    SET(a, b, c, d,  5,  4, T33);
    SET(d, a, b, c,  8, 11, T34);
    SET(c, d, a, b, 11, 16, T35);
    SET(b, c, d, a, 14, 23, T36);
    SET(a, b, c, d,  1,  4, T37);
    SET(d, a, b, c,  4, 11, T38);
    SET(c, d, a, b,  7, 16, T39);
    SET(b, c, d, a, 10, 23, T40);
    SET(a, b, c, d, 13,  4, T41);
    SET(d, a, b, c,  0, 11, T42);
    SET(c, d, a, b,  3, 16, T43);
    SET(b, c, d, a,  6, 23, T44);
    SET(a, b, c, d,  9,  4, T45);
    SET(d, a, b, c, 12, 11, T46);
    SET(c, d, a, b, 15, 16, T47);
    SET(b, c, d, a,  2, 23, T48);
#undef SET

     /* Ronda 5. */
     /* [abcd k s t] denota la operacion
          a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */
#define I(x, y, z) ((y) ^ ((x) | ~(z)))
#define SET(a, b, c, d, k, s, Ti)\
  t = a + I(b,c,d) + X[k] + Ti;\
  a = ROTATE_LEFT(t, s) + b
 /* Realiza las siguientes 16 operaciones. */
    SET(a, b, c, d,  0,  6, T49);
    SET(d, a, b, c,  7, 10, T50);
    SET(c, d, a, b, 14, 15, T51);
    SET(b, c, d, a,  5, 21, T52);
    SET(a, b, c, d, 12,  6, T53);
    SET(d, a, b, c,  3, 10, T54);
    SET(c, d, a, b, 10, 15, T55);
    SET(b, c, d, a,  1, 21, T56);
    SET(a, b, c, d,  8,  6, T57);
    SET(d, a, b, c, 15, 10, T58);
    SET(c, d, a, b,  6, 15, T59);
    SET(b, c, d, a, 13, 21, T60);
    SET(a, b, c, d,  4,  6, T61);
    SET(d, a, b, c, 11, 10, T62);
    SET(c, d, a, b,  2, 15, T63);
    SET(b, c, d, a,  9, 21, T64);
#undef SET

     /* Aqui se realizan las siguientes adiciones. (Que es 1 incremento de
          cada uno de los 4 registros por el valor que tenia el bloque
        el cual a sido iniciado.) */
    pms->abcd[0] += a;
    pms->abcd[1] += b;
    pms->abcd[2] += c;
    pms->abcd[3] += d;
}

void md5_init(md5_state_t *pms)
{
    pms->count[0] = pms->count[1] = 0;
    pms->abcd[0] = 0x67452301;
    pms->abcd[1] = 0xefcdab89;
    pms->abcd[2] = 0x98badcfe;
    pms->abcd[3] = 0x10325476;
}

void md5_append(md5_state_t *pms, const md5_byte_t *data, int nbytes)
{
    const md5_byte_t *p = data;
```

```
    int left = nbytes;
    int offset = (pms->count[0] >> 3) & 63;
    md5_word_t nbits = (md5_word_t)(nbytes << 3);

    if (nbytes <= 0)
    return;

    /* Actualiza el tamaño del mensaje. */
    pms->count[1] += nbytes >> 29;
    pms->count[0] += nbits;
    if (pms->count[0] < nbits)
    pms->count[1]++;

    /* El proceso incializaza parcialmente el bloque. */
    if (offset) {
    int copy = (offset + nbytes > 64 ? 64 - offset : nbytes);

    memcpy(pms->buf + offset, p, copy);
    if (offset + copy < 64)
        return;
    p += copy;
    left -= copy;
    md5_process(pms, pms->buf);
    }

    /* Procesa todo el bloque. */
    for (; left >= 64; p += 64, left -= 64)
    md5_process(pms, p);

    /* Procesa un final bloque parcial. */
    if (left)
    memcpy(pms->buf, p, left);
}

void md5_finish(md5_state_t *pms, md5_byte_t digest[16])
{
    static const md5_byte_t pad[64] = {
    0x80, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
    };
    md5_byte_t data[8];
    int i;

    /* Salva el tamaño antes de rellenarlo. */
    for (i = 0; i < 8; ++i)
    data[i] = (md5_byte_t)(pms->count[i >> 2] >> ((i & 3) << 3));
    /* Rellena a 56 bytes mod 64. */
    md5_append(pms, pad, ((55 - (pms->count[0] >> 3)) & 63) + 1);
    /* Agrega el tamaño. */
    md5_append(pms, data, 8);
    for (i = 0; i < 16; ++i)
    digest[i] = (md5_byte_t)(pms->abcd[i >> 2] >> ((i & 3) << 3));
}
/*****************************************************************************/
```

9.- SSL.

```
*****************************************************************/
/*  TESIS- TECNICAS CRIPTOGRAFICAS                                */
/*  AUTOR: Guillermo Gomez Villa                                  */
```

```
/*  FECHA: Mayo 2004                                                        */
/*****************************************************************************/

/*************** ssl.h *****************/
#include <openssl/rsa.h>
#include <openssl/pem.h>
#include <openssl/x509.h>
#include <openssl/err.h>
#include <openssl/evp.h>
#include <openssl/objects.h>
#include <openssl/ssl.h>
#define MIN_CLAVE 6        /* Tamaño minimo de la clave */
#define MAX_CLAVE 12       /* Tamaño maximo de la clave */
#define STDIN     0
#define STDOUT    1

EVP_PKEY * ReadPublicKey(const char *certfile);
EVP_PKEY *ReadPrivateKey(const char *keyfile);

EVP_PKEY * ReadPublicKey(const char *certfile)
{
  FILE *fp = fopen (certfile, "r");
  X509 *x509;
  EVP_PKEY *pkey;

  if (!fp)
     return NULL;

  x509 = PEM_read_X509(fp, NULL, 0, NULL);

  if (x509 == NULL)
  {
     ERR_print_errors_fp (stderr);
     return NULL;
  }

  fclose (fp);

  pkey=X509_extract_key(x509);

  X509_free(x509);

  return pkey;
}

EVP_PKEY *ReadPrivateKey(const char *keyfile)
{
    FILE *fp = fopen(keyfile, "r");
    EVP_PKEY *pkey;
    SSLeay_add_all_algorithms();
    ERR_load_PEM_strings();
    EVP_set_pw_prompt("Introduzca la Clave del Cifrado Simetrico : ");
    ERR_load_crypto_strings();
    if (!fp)
        return NULL;
    pkey = PEM_read_PrivateKey(fp, NULL, 0, NULL);
    fclose (fp);
    if (pkey == NULL) return pkey;
    return pkey;
}

long filesize(FILE *stream)
{
   long curpos, length;
   fseek(stream, 0L, SEEK_END);
   length = ftell(stream);
```

```
   return length;
}

int SSL_encripta_RSA(void)
{
    FILE *fp_entrada=NULL,*fp_salida=NULL;
    char arch_entrada[40]="",arch_salida[40]="",arch_certificado[40]="";
    char ct[210] = "";
    char *buf;
    EVP_PKEY *pubKey;
    int len,len2;
    long pos_file,pos_file_ant,tmn_arch,ult_avanc;
    printf("Archivo a Cifrar con RSA: ");gets(arch_entrada);
    printf("Archivo de Salida Cifrado : ");gets(arch_salida);
    printf("Archivo de Certificado (sin extension): ");gets(arch_certificado);
    strcat(arch_certificado,".pem");
    if((fp_entrada=fopen(arch_entrada,"rb"))==NULL)
    {
       puts("      !! No se puede leer el archivo de entrada !! ");
       return 0;
    }
    if((fp_salida=fopen(arch_salida,"wb"))==NULL)
    {
       puts("      !! No se puede crear en archivo de salida !! ");
       return 0;
    }
    ERR_load_crypto_strings();
    pubKey = ReadPublicKey(arch_certificado);
    if(!pubKey)
    {
       fprintf(stderr,"\n  !!  Error, no se puede cargar la clave publica !!");
       return 0;
    }
    buf = malloc(EVP_PKEY_size(pubKey));
    fseek(fp_entrada, 0L, SEEK_SET);
    puts("");
    puts("  Cifrando ...");
    while((fread(ct,sizeof(char)*(200),1,fp_entrada))>0)
    {
       len = RSA_public_encrypt(strlen(ct), ct, buf, pubKey->pkey.rsa,RSA_PKCS1_PADDING);
       if (len != EVP_PKEY_size(pubKey))
       {
           fprintf(stderr,"\n !! Error, fallo del proceso de cifrado !!\n");
           return 0;
       }
       fwrite(buf,sizeof(char)*len,1,fp_salida);
       pos_file_ant=pos_file;
       pos_file=ftell(fp_entrada);
    }
    tmn_arch=filesize(fp_entrada);
    fseek(fp_entrada,pos_file , SEEK_SET);
    ult_avanc=tmn_arch-pos_file;
    if(ult_avanc>0)
    {
      if(fread(ct,sizeof(char)*ult_avanc,1,fp_entrada)>0)
      {
         len     =     RSA_public_encrypt(sizeof(char)*ult_avanc,     ct,     buf,     pubKey-
>pkey.rsa,RSA_PKCS1_PADDING);
         if (len != EVP_PKEY_size(pubKey))
         {
             fprintf(stderr,"\n !! Error, fallo del proceso de cifrado !!\n");
             return 0;
         }
         fwrite(buf,sizeof(char)*len,1,fp_salida);
      }
    }
```

```
    puts("");
    puts("  !! Cifrado Terminado con Exito !!");
    fclose(fp_entrada);
    fclose(fp_salida);
    EVP_PKEY_free(pubKey);
    free(buf);
    fclose(fp_entrada);
    fclose(fp_salida);
    return 0;
}

int SSL_desencripta_RSA(void)
{
    FILE *fp_entrada=NULL,*fp_salida=NULL;
    char arch_entrada_sal[40]="",arch_salida_sal[40]="",arch_clavepriv[40]="";
    char *buf2;
    char buf3[300];
    EVP_PKEY *privKey;
    int len,len2;
    long pos_file,pos_file_ant,tmn_arch,ult_avanc;
    ERR_load_crypto_strings();
    printf("Archivo a Cifrado con RSA : ");gets(arch_entrada_sal);
    printf("Archivo de Salida Descifrado : ");gets(arch_salida_sal);
    printf("Archivo de Clave privada (sin extension): ");gets(arch_clavepriv);
    strcat(arch_clavepriv,".pem");
    /****************** DESCIFRAR ****************************/
    if((fp_entrada=fopen(arch_entrada_sal,"rb"))==NULL)
    {
       puts("      !! No se puede leer el archivo de entrada !! ");
       return 0;
    }
    if((fp_salida=fopen(arch_salida_sal,"wb"))==NULL)
    {
       puts("      !! No se puede crear en archivo de salida !! ");
       return 0;
    }
    privKey = ReadPrivateKey(arch_clavepriv);
    if (!privKey)
    {
        puts("      !! No se puede cargar la clave privada !! ");
        puts("          !! Verifique clave simetrica !! ");
        return 0;
    }
    fseek(fp_entrada, 0L, SEEK_SET);
    buf2 = malloc(EVP_PKEY_size(privKey));
    puts("");
    puts("   Descifrando ...");
    puts("");
    pos_file=0;
    while(fread(buf3,sizeof(char)*EVP_PKEY_size(privKey),1,fp_entrada)>0)
    {
       len2=RSA_private_decrypt(EVP_PKEY_size(privKey),      buf3,      buf2,      privKey-
>pkey.rsa,RSA_PKCS1_PADDING);
       if(len2 < 0)
       {
          puts("");
          puts(" !!! Error en el Descifrado !!!");
          puts("");
          return 0;
       }
       fwrite(buf2,sizeof(char)*(len2),1,fp_salida);
       pos_file_ant=pos_file;
       pos_file=ftell(fp_entrada);
    }
    puts("  !! Descifrado Terminado con Exito !!");
    EVP_PKEY_free(privKey);
```

```
    fclose(fp_entrada);
    fclose(fp_salida);
    return 0;
}

int genera_firma(void)
{
  char arch_firma[40]="",arch_datos[40]="";
  int err;
  int sig_len;
  unsigned char sig_buf [4096];
  static char keyfile[40]  = "";
  static char data[100]    = "";
  EVP_MD_CTX     md_ctx;
  EVP_PKEY *      pkey;
  FILE *          fp, *fp_firma,*fp_datos;
  X509 *    x509;
  printf("Archivo a Generar su Firma: ");gets(arch_datos);
  printf("Archivo de Clave Privada para Generar la Firma(sin extension): ");gets(keyfile);
  strcat(keyfile,".pem");
  strcpy(arch_firma,arch_datos);
  strcat(arch_firma,".sig");
  SSLeay_add_all_algorithms();
  ERR_load_PEM_strings();
  EVP_set_pw_prompt("Introduzca la Clave del Cifrado Simetrico : ");
  ERR_load_crypto_strings();
  /* Lee la clave privada */

  if((fp = fopen (keyfile, "r"))==NULL)
  {
    puts("");
    puts("   !! No se puede abrir el archivo de clave privada !!");
    return 0;
  }
  if((pkey = PEM_read_PrivateKey(fp, NULL, NULL, NULL))==NULL)
  {
    puts("");
    puts("   !! Error al cargar el archivo de la clave  !!");
    puts("         !! Verifique clave simetrica  !!");
    fclose (fp);
    return 0;
  }
  fclose (fp);

  /* Realiza la firma digital*/

  if((fp_datos=fopen(arch_datos,"rb"))==NULL)
  {
    puts("");
    puts("   !! Error al cargar el archivo de la clave  !!");
    puts("         !! Verifique clave simetrica  !!");
    return 0;
  }
  fseek(fp_datos, SEEK_SET, 0);
  EVP_SignInit   (&md_ctx, EVP_sha1());
  while(fread(data,sizeof(char)*99,1,fp_datos)>0)
  {
    EVP_SignUpdate (&md_ctx, data, strlen(data));
  }
  sig_len = sizeof(sig_buf);
  err = EVP_SignFinal (&md_ctx, sig_buf, &sig_len, pkey);
  if (err != 1)
  {
    //ERR_print_errors_fp(stderr);
    puts("");
    puts("  !! Error al generar la firma digital !!");
```

```
    puts("");
    return 0;
  }
  fclose (fp_datos);
//  printf("%s",sig_buf);
  EVP_PKEY_free (pkey);
  if((fp_firma=fopen(arch_firma,"wb"))==NULL)
  {
    puts("");
    puts("  !! Error al crear el archivo de firma !!");
    puts("");
    return 0;
  }
  fwrite(sig_buf,sizeof(char)*4096,1,fp_firma);
  fclose(fp_firma);
  puts("");
  puts("      !! Generacion de la Firma Exitosa !!");
  return(0);
}

int firma_verifica(void)
{
  char arch_firma[40]="",arch_datos[40]="";
  int err;
  int sig_len;
  unsigned char sig_buf [4096];
  static char certfile[40] = "";
  static char keyfile[40]  = "";
  static char data[100]    = "";
  EVP_MD_CTX     md_ctx;
  EVP_PKEY *      pkey;
  FILE *          fp, *fp_firma,*fp_datos;
  X509 *    x509;

  printf("Archivo a Verificar su Firma: ");gets(arch_datos);
  printf("Archivo de Certificado (sin extension): ");gets(certfile);
  strcpy(arch_firma,arch_datos);
  strcat(arch_firma,".sig");
  strcat(certfile,".pem");
  SSLeay_add_all_algorithms();
  ERR_load_PEM_strings();
  EVP_set_pw_prompt("Introduzca la Clave del Cifrado Simetrico : ");
  ERR_load_crypto_strings();

  if((fp_firma=fopen(arch_firma,"rb"))==NULL)
  {
    puts("");
    puts("  !! Error al Abrir la Firma Digital !!");
    puts("");
    return 0;
  }
  fread(sig_buf,sizeof(char)*4096,1,fp_firma);
//  sig_len=strlen(sig_buf);
  sig_len=256;
  close(fp_firma);

  /* Obtiene la clave publica */

  if((fp = fopen (certfile, "r"))==NULL)
  {
    puts("");
    puts("  !! Error al cargar el archivo del certificado !!");
    puts("");
    fclose (fp);
    return 0;
  }
```

```
  x509 = PEM_read_X509(fp, NULL, NULL, NULL);
  fclose (fp);
  if (x509 == NULL)
  {
//    ERR_print_errors_fp (stderr);
    puts("");
    puts("  !! Error al leer el certificado !!");
    puts("");
    return 0;
  }

  /* Obtiene la clave */
  if((pkey=X509_get_pubkey(x509))==NULL)
  {
    puts("");
    puts(" !! Error al obtener la clave publica del certificado !!");
    puts("");
    return 0;
  }

  /* Verifica la firma */
  if((fp_datos=fopen(arch_datos,"rb"))==NULL)
  {
    puts("");
    puts("   !! Error al cargar el archivo de datos  !!");
    return 0;
  }
  fseek(fp_datos, SEEK_SET, 0);
  EVP_VerifyInit   (&md_ctx, EVP_sha1());
  while(fread(data,sizeof(char)*99,1,fp_datos)>0)
  {
     EVP_VerifyUpdate (&md_ctx, data, strlen((char*)data));
  }
  err = EVP_VerifyFinal (&md_ctx, sig_buf, sig_len, pkey);
  EVP_PKEY_free (pkey);
  fclose (fp_datos);
  if (err != 1)
  {
    // ERR_print_errors_fp (stderr);
    puts("");
    puts(" !! Verificacion de la firma fallida !!");
    puts("");
    return 0;
  }
  puts("");
  puts(" !! Verificacion de la firma Exitosa !!");
  return(0);
}

int certificado_x509(void)
{
   char arch_cert[40]={""},arch_claversa[40]={""},clave_simetrica[40]={""};
   char cadena[100],dominio[2],estado[30],organizacion[50],
        departamento[50],nombreper[50],email[30];
   FILE * fp_cert;
   BIO  *fp_clave;
   X509 * x=NULL;
   EVP_PKEY * clave=NULL;
   RSA * rsa;
   X509_NAME * nombre=NULL;
   int Nserie;
   int Nrsa;
   int Ndias;
   printf("Nombre de Archivo del Certificado (sin extension) : ");gets(arch_cert);
   printf("Nombre de Archivo de la Clave Privada RSA (sin extension) :
");gets(arch_claversa);
```

```
// Abrimos el fichero para escritura del certificado
strcat(&arch_cert,".pem");
strcat(&arch_claversa,".pem");
SSLeay_add_all_algorithms();
ERR_load_PEM_strings();
EVP_set_pw_prompt("Introduzca la Clave del Cifrado Simetrico : ");
if ((fp_cert =fopen(arch_cert,"w")) == NULL)
{
    puts("");
    perror("ERROR al escribir el archivo del certificado");
    return 0;
}
// Abrimos el fichero para escritura de la clave privada
fp_clave=BIO_new(BIO_s_file());
if (BIO_read_filename(fp_clave,arch_claversa) <= 0)
{
    puts("");
    perror("ERROR al abrir el archivo de la clave privada");
    return 0;
}
// Toma de datos de los parametros
Nserie = 8765675865;
Nrsa = 512;
Ndias = 15;
// Iniciación de variables y estructuras del programa

x = X509_new();
clave = EVP_PKEY_new();

// Generación del par de llaves RSA

if((rsa=(RSA *)PEM_read_bio_RSAPrivateKey(fp_clave,NULL,NULL,NULL))!=NULL)
{
  puts("Clave cargada. ");
}
else
{
   puts("");
   puts("  !! Error en la carga de la clave privada RSA !!! . ");
   puts("             !! Verifique Clave !!! . ");
   return 0;
}
// Asignación del par RSA a la estructura de clave privada EVP_PKEY

EVP_PKEY_assign_RSA(clave,rsa);

// Relleno de datos del certificado

X509_set_version(x,3);
ASN1_INTEGER_set(X509_get_serialNumber(x),Nserie);
X509_gmtime_adj(X509_get_notBefore(x),0);
X509_gmtime_adj(X509_get_notAfter(x),(long)60*60*24*Ndias);
X509_set_pubkey(x,clave);

nombre=X509_get_subject_name(x);
printf("\n               Informacion del certificado.         ");
gets(dominio);
printf("\n           Dominio (Ejemplo MX) : ");gets(dominio);
printf(  "       Estado (Ejemplo Morelos) : ");gets(estado);
printf(  "    Organizacion (Ejemplo UNAM) : ");gets(organizacion);
printf(  " Departamento (Ejemplo Computo) : ");gets(departamento);
printf(  "     Nombre(Ejemplo Juan Perez) : ");gets(nombreper);
printf(  "   Email (Ejemplo juan@unam.mx) : ");gets(email);

X509_NAME_add_entry_by_txt(nombre,"C",MBSTRING_ASC, dominio, -1, -1, 0);
X509_NAME_add_entry_by_txt(nombre,"ST",MBSTRING_ASC, estado, -1, -1, 0);
```

```
   X509_NAME_add_entry_by_txt(nombre,"L",MBSTRING_ASC, "Cs", -1, -1, 0);
   X509_NAME_add_entry_by_txt(nombre,"O",MBSTRING_ASC, organizacion, -1, -1, 0);
   X509_NAME_add_entry_by_txt(nombre,"OU",MBSTRING_ASC, departamento, -1, -1, 0);
   X509_NAME_add_entry_by_txt(nombre,"CN",MBSTRING_ASC, nombreper, -1, -1, 0);
   X509_NAME_add_entry_by_txt(nombre,"Email",MBSTRING_ASC, email, -1, -1, 0);

   X509_set_issuer_name(x,nombre);

   // Firmamos el certificado que hemos creado con nuestra propia clave

   X509_sign(x,clave,EVP_md5());

   // Podemos imprimir por pantalla el resultado del certificado y la clave

   RSA_print_fp(stdout,clave->pkey.rsa,0);
   X509_print_fp(stdout,x);

   // Escribimos el certificado y la clave privada en los ficheros especificados

   PEM_write_X509(fp_cert,x);

   // Cerramos archivos y liberamos de la memoria las estructuras usadas

   RSA_free(rsa);
//   EVP_PKEY_free(clave);
   X509_free(x);
   fclose(fp_cert);
   BIO_free(fp_clave);
   puts("");puts("      !! Certificado creado Exitosamente !!");
   return 0;
}


int mostrar_archivo_ascci(char *archivo)
{
  FILE *fp;
  char caracter[2]={""};
  printf("\n");
  if ((fp = fopen (archivo, "r"))== NULL)
  {
     puts("!! No se puede abrir el archivo !!");
     return 1;
  }
  fseek(fp, SEEK_SET, 0);
  while (fread(caracter,sizeof(char),1,fp)!=NULL)
  {
     printf("%s",caracter);
  }
  return 0;
}

int SSL_genera_claves_rsa(void);
int SSL_genera_claves_rsa(void)
{
   char arch_priv[30]={""},arch_pub[30]={""},clave[16]={""};
   int t_clave=0;
   FILE *bp_priv,*bp_pub;
   RSA *x;
   EVP_CIPHER *algor;
   ERR_load_crypto_strings();
   printf("Nombre del Archivo para Clave Privada (sin extencion) : ");gets(arch_priv);
   printf("\nNombre del Archivo para Clave Publica (sin extencion) : ");gets(arch_pub);
   printf("\nClave de Cifrado Simetrico Triple Des para la Clave Privada : ");gets(clave);
   /* Agregar la extencion PEM */
   strcat(arch_priv,".pem");
   strcat(arch_pub,".pem");
```

```
   t_clave=strlen(clave);
   /* Valida Tamaño de la clave */
   if(t_clave<MIN_CLAVE ||t_clave>MAX_CLAVE)
   {
      puts("");
      printf("ERROR:   El   tamaño   de   la   clave   debe   de   ser   de   %i   a   %i
caracteres.",MIN_CLAVE,MAX_CLAVE);
      return 0;
   }
   if ((bp_priv=fopen(arch_priv,"w")) == NULL)
   {
      puts("");
      perror("ERROR: Al crear el archivo de la clave privada");
      return 0;
   }
   if ((bp_pub=fopen(arch_pub,"w")) == NULL)
   {
      puts("");
      perror("ERROR: Al crear el archivo de la clave publica");
      return 0;
   }
   printf("\nGenerando Claves RSA...\n");
   x=RSA_generate_key(2048,3,NULL,NULL);
   printf("\nClaves RSA de 2048 bits Generadas.\n");
   puts("");
   algor=EVP_des_ede_cbc();
   if (PEM_write_RSAPrivateKey(bp_priv,x,algor,clave,t_clave,NULL,NULL) > 0)
   {
      fclose(bp_priv);
      mostrar_archivo_ascci(arch_priv);
      printf("\nClave Privada RSA Guardada en %s\n",arch_priv);
   }
   else ERR_print_errors_fp(stderr);
   if (PEM_write_RSAPublicKey(bp_pub,x) > 0)
   {
      fclose(bp_pub);
      mostrar_archivo_ascci(arch_pub);
      printf("\nClave Publica RSA Guardada en %s\n",arch_pub);
   }
   else ERR_print_errors_fp(stderr);
   RSA_free(x);
   return 0;
}

void SSL_rsa(void)
{
   clrscr();
   puts("          TECNICAS CRIPTOGRAFIAS");
   puts("");
   puts("                  OPENSSL");
   puts("                    PEM");
   puts("           (PRIVACY ENHACED MAIL)");
   puts("          GENERACION DE CLAVES RSA");
   puts("");
   SSL_genera_claves_rsa();
   puts("");
   puts("");puts("");puts("");
   puts("        <Presione cualquier tecla para continuar>");
   getch();

}

void SSL_certificados(void)
{
   clrscr();
   puts("          TECNICAS CRIPTOGRAFIAS");
```

```
   puts("");
   puts("                  OPENSSL");
   puts("                    ");
   puts("     GENERACION DE CERTIFICADO AUTOFIRMADO");
   puts("");
   certificado_x509();
   puts("");
   puts("");puts("");puts("");
   puts("        <Presione cualquier tecla para continuar>");
   getch();
}

void SSL_firma(void)
{
   clrscr();
   puts("          TECNICAS CRIPTOGRAFIAS");
   puts("");
   puts("                  OPENSSL");
   puts("                    ");
   puts("        GENERACION DE FIRMA DIGITAL ");
   puts("               USANDO SHA1");
   puts("");
   genera_firma();
   puts("");
   puts("");puts("");puts("");
   puts("        <Presione cualquier tecla para continuar>");
   getch();
}

void SSL_validar_firma(void)
{
   clrscr();
   puts("          TECNICAS CRIPTOGRAFIAS");
   puts("");
   puts("                  OPENSSL");
   puts("                    ");
   puts("         VALIDANDO FIRMA DIGITAL ");
   puts("               USANDO SHA1");
   puts("");
   firma_verifica();
   puts("");
   puts("");puts("");puts("");
   puts("        <Presione cualquier tecla para continuar>");
   getch();
}

void SSL_RSA_Encriptar(void)
{
   clrscr();
   puts("          TECNICAS CRIPTOGRAFIAS");
   puts("");
   puts("                  OPENSSL");
   puts("                    ");
   puts("              CIFRAR CON RSA ");
   puts("                ");
   puts("");
   SSL_encripta_RSA();
   puts("");
   puts("");puts("");puts("");
   puts("        <Presione cualquier tecla para continuar>");
   getch();
}

void SSL_RSA_Desencriptar(void)
{
   clrscr();
```

```
   puts("          TECNICAS CRIPTOGRAFIAS");
   puts("");
   puts("                 OPENSSL");
   puts("                   ");
   puts("              DESCIFRAR CON RSA ");
   puts("                ");
   puts("");
   SSL_desencripta_RSA();
   puts("");
   puts("");puts("");puts("");
   puts("        <Presione cualquier tecla para continuar>");
   getch();
}
```

10.- Técnicas Criptográficas.

```
/*************************************************************************/
/*  TESIS- TECNICAS CRIPTOGRAFICAS                                       */
/*  AUTOR: Guillermo Gomez Villa                                         */
/*  FECHA: Mayo 2004                                                     */
/*************************************************************************/

/************** Tecnicascripto.c ***********************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <conio.h>  // ELIMIRAR PARA ESTANDARIZAR -NO COMPATIBLE UNIX-
#include <mem.h>
#include <alloc.h>
#include <assert.h>
#include <math.h>

#define  VERDADERO 1
#define  FALSO 0
#define  ALPHA 26

#include "Vigenere.h"
#include "Transposicion.h"
#include "SXOR.h"
#include "Sustitucion.h"
#include "Rsa.h"
#include "Rijndael.h"
#include "MD5.h"
#include "DES.h"
#include "ssl.h"

int gi_TipoMenu;
int gb_Salir;

void Menu(int pi_TipoMenu)
{
   clrscr();
   puts("          TECNICAS CRIPTOGRAFIAS");
   puts("");
   switch (pi_TipoMenu)
   {
      case 0:  /* MENU PRINCIPAL */
            puts("");puts("");
//            puts("       1.- BASES CRIPTOGRAFICAS");
            puts("       1.- ALGORITMOS SIMETRICOS");
            puts("       2.- ALGORITMOS ASIMETRICOS");
            puts("       3.- APLICACIONES CRIPTOGRAFICAS");
```

```
      break;
case 1:  /* MENU BASES CRIPTOGRAFICAS*/
      puts("           BASES CRIPTOGRAFICAS");
      puts("");
      puts("       1.- ENTROPIA");
      puts("       2.- NUMEROS PRIMOS");
      puts("");
      puts("");
      break;

case 2:  /* MENU CIFRADOS SIMETRICOS */
      puts("          ALGORITMOS SIMETRICOS");
      puts("");
      puts("       1.- SUSTITUCION");
      puts("       2.- TRANSPOSICION");
      puts("       3.- SIMPLE XOR");
      puts("       4.- DES");
      puts("       5.- RIJNDAEL");
      break;
case 3:  /* MENU CIFRADOS ASIMETRICOS */
      puts("         ALGORITMOS ASIMETRICOS");
      puts("");
      puts("");
      puts("          1.- RSA");
      puts("");
      puts(""); puts("");
      break;
case 4:  /* MENU APLICACIONES */
      puts("        APLICACIONES CRIPTOGRAFICAS");
      puts("");
      puts("       1.- MD5 (RESUMEN)");
      puts("       2.- OPENSSL ");
      puts(""); puts("");
      break;
case 21:
      puts("         CIFRADO POR SUSTITUCION");
      puts("");
      puts("        1 - SUBSTITUCION SIMPLE");
      puts("        2 - VIGENERE");
      puts(""); puts("");
      break;
case 211:
      puts("    CIFRADO POR SUSTITUCION SIMPLE");
      puts("");
      puts("        E - Cifrar");
      puts("        D - Descifrar");
      puts(""); puts("");
      break;
case 212:
      puts("    CIFRADO POR SUSTITUCION VIGENERE");
      puts("");
      puts("        E - Cifrar");
      puts("        D - Descifrar");
      puts(""); puts("");
      break;
case 22:
      puts("       CIFRADO POR TRANSPOSICION");
      puts("");
      puts("        E - Cifrar");
      puts("        D - Descifrar");
      puts(""); puts("");
      break;
case 23:
      puts("      CIFRADO SIMPLE XOR");
      puts("");
      puts("        E - Cifrar");
```

```
            puts("        D - Descifrar");
            puts(""); puts("");
            break;
      case 24:
            puts("           CIFRADO DES");
            puts("");
            puts("           E - Cifrar");
            puts("           D - Descifrar");
            puts(""); puts("");
            break;
      case 25:
            puts("        CIFRADO RIJNDAEL");
            puts("");
            puts("          E - Cifrar");
            puts("          D - Descifrar");
            puts(""); puts("");
            break;
      case 31:
            puts("            CIFRADO RSA");
            puts("");
            puts("          C - Generar Clave Privada y Publica");
            puts("          E - Cifrar");
            puts("          D - Descifrar");
            puts(""); puts("");
            break;
      case 42:
            puts("               OPENSSL");
            puts("");
            puts("          1 - Claves RSA");
            puts("          2 - Certificados Autofirmados");
            puts("          3 - Firmar");
            puts("          4 - Validar Firma");
            puts("          5 - Encriptar/Desencriptar");
            puts(""); puts("");
            break;
      case 425:
            puts("               OPENSSL");
            puts("              CIFRADO RSA");
            puts("");
            puts("            E - Cifrar");
            puts("            D - Descifrar");
            puts(""); puts("");
            break;
      default:
            puts("Opcion de menu no definida !!");
/*
            gi_TipoMenu=1;
            Menu(gi_TipoMenu);
*/
            break;
   }
   puts("");puts("");puts("");puts("");puts("");puts("");
   puts("      S - SALIR                P - PRINCIPAL");
}

void Procesa_Opcion(int pi_TipoMenu,int pi_OpcionMenu)
{
   switch (pi_TipoMenu)
   {
       case 0:
            switch (pi_OpcionMenu)
            {
               case '1':
                     gi_TipoMenu=2;
                     break;
               case '2':
```

```
                    gi_TipoMenu=3;
                    break;
               case '3':
                    gi_TipoMenu=4;
                    break;
/*
               case '4':
                    gi_TipoMenu=4;
                    break;
*/
            }
            break;
         case 2:
            switch (pi_OpcionMenu)
            {
               case '1':
                    gi_TipoMenu=21;
                    break;
               case '2':
                    gi_TipoMenu=22;
                    break;
               case '3':
                    gi_TipoMenu=23;
                    break;
               case '4':
                    gi_TipoMenu=24;
                    break;
               case '5':
                    gi_TipoMenu=25;
                    break;
            }
            break;
         case 3:
            switch (pi_OpcionMenu)
            {
               case '1':
               gi_TipoMenu=31;
               break;
            }
            break;
         case 4:
            switch (pi_OpcionMenu)
            {
               case '1':
                    MD5_I();
                    break;
               case '2':
                    gi_TipoMenu=42;
                    break;
            }
            break;
         case 21:
            switch (pi_OpcionMenu)
            {
               case '1':
                    gi_TipoMenu=211;
                    break;
               case '2':
                    gi_TipoMenu=212;
                    break;
            }
            break;
         case 211:
            if (pi_OpcionMenu=='E' || pi_OpcionMenu=='e' ||
                pi_OpcionMenu=='d' || pi_OpcionMenu=='D')
                Sustitucion(pi_OpcionMenu);
```

```
            break;
        case 212:
            if (pi_OpcionMenu=='E' || pi_OpcionMenu=='e' ||
                pi_OpcionMenu=='d' || pi_OpcionMenu=='D')
                Vigenere(pi_OpcionMenu);
            break;
        case 22:
            if (pi_OpcionMenu=='E' || pi_OpcionMenu=='e' ||
                pi_OpcionMenu=='d' || pi_OpcionMenu=='D')
                Transposicion(pi_OpcionMenu);
            break;
        case 23:
            if (pi_OpcionMenu=='E' || pi_OpcionMenu=='e' ||
                pi_OpcionMenu=='d' || pi_OpcionMenu=='D')
                Simple_XOR(pi_OpcionMenu);
            break;
        case 24:
            if (pi_OpcionMenu=='E' || pi_OpcionMenu=='e' ||
                pi_OpcionMenu=='d' || pi_OpcionMenu=='D')
                Alg_Des(pi_OpcionMenu);
            break;
        case 25:
            if (pi_OpcionMenu=='E' || pi_OpcionMenu=='e' ||
                pi_OpcionMenu=='d' || pi_OpcionMenu=='D')
                Alg_Rijndael(pi_OpcionMenu);
            break;
        case 31:
            if (pi_OpcionMenu=='E' || pi_OpcionMenu=='e' ||
                pi_OpcionMenu=='D' || pi_OpcionMenu=='d' ||
                pi_OpcionMenu=='C' || pi_OpcionMenu=='c')
                Alg_RSA(pi_OpcionMenu);
            break;
        case 42:
            switch (pi_OpcionMenu)
            {
               case '1':
                   SSL_rsa();
                   break;
               case '2':
                   SSL_certificados();
                   break;
               case '3':
                   SSL_firma();
                   break;
               case '4':
                   SSL_validar_firma();
                   break;
               case '5':
                   gi_TipoMenu=425;
                   break;
            }
        case 425:
            if (pi_OpcionMenu=='E' || pi_OpcionMenu=='e')
                SSL_RSA_Encriptar();
            else if (pi_OpcionMenu=='d' || pi_OpcionMenu=='D')
                SSL_RSA_Desencriptar();
            break;
   }
}
int main(int argc, char **argv)
{
   char lc_OpcionMenu;
   gb_Salir=FALSO;
   gi_TipoMenu=0;
   do
   {
```

```
      Menu(gi_TipoMenu);
      lc_OpcionMenu=getch();
      if(lc_OpcionMenu=='S' || lc_OpcionMenu=='s')gb_Salir = VERDADERO;
      else if (lc_OpcionMenu=='P' || lc_OpcionMenu=='p') gi_TipoMenu=0;
      else Procesa_Opcion(gi_TipoMenu,lc_OpcionMenu);
   }while (!gb_Salir);
   return 0;
}
```

E.- Diccionario de Términos.

Alfabeto	Es la agrupación de símbolos con un orden determinado utilizado en el lenguaje que sirve como sistema de comunicación.
Alfabeto Cifrado	Es la agrupación de símbolos posibles generados a partir de un proceso de cifrado alimentado por un alfabeto.
Algoritmo	Es una secuencia finita de instrucciones, cada una de las cuales tiene un significado preciso y puede ejecutarse con una cantidad finita de esfuerzo en un tiempo finito. Ha de tener las siguientes características: legible, correcto, modular, eficiente, estructurado, no ambiguo y a ser posible se ha de desarrollar en el menor tiempo posible.
Algoritmo Simétrico	Algoritmo criptográfico que usa una misma clave para cifrar y para descifrar mensajes. Las dos partes que se comunican han de ponerse de acuerdo de antemano sobre la clave a usar. Una vez ambas tienen acceso a esta clave, el remitente cifra un mensaje usándola, lo envía al destinatario, y éste lo descifra con la misma.
Algoritmo Asimétrico	Algoritmo criptográfico que utiliza un par de claves de cifrado, una clave pública y una privada. La pública se usa para encriptar o verificar mensajes, y la privada para descifrar y firmar mensajes.
Anagrama	Transposición de las letras de una palabra o sentencia, de la que resulta otra palabra o sentencia distinta.
Anillo	Sea A un conjunto, y sean + y · dos operaciones binarias. Se dirá que el trío $(A,+,\cdot)$ es un anillo si se cumplen las siguientes propiedades: $(A,+)$ es un grupo conmutativo, esto es, verifica: • $a+(b+c)=(a+b)+c$ para todos los elementos de A (asociatividad). • Existe un elemento, 0, tal que $0+a=a+0=a$ para todo a de A (elemento neutro). • Todo elemento, a, de A tiene inverso, -a, en el anillo, donde $-a$ es inverso de a si $a+(-a)=(-a)+a=0$ (elemento inverso). • $a+b=b+a$ para todos los elementos de A (conmutatividad). $(A,\cdot)$ verifica: • $a\cdot(b\cdot c) = (a\cdot b)\cdot c$ para todos los elementos de A (asociatividad).

	• $a\cdot(b+c)=a\cdot b+a\cdot c$ y $(b+c)\cdot a=b\cdot a+c\cdot a$ para todos los elementos del anillo (propiedad distributiva con respecto a +). • A + y $a\cdot$ se les denomina la suma y el producto, respectivamente, del anillo A. Asimismo, al neutro de la suma suele denominársele cero del anillo. Tipos de anillos: • Anillo conmutativo: aquel en el que el producto es conmutativo, esto es, $a\cdot b=b\cdot a$ para todos a y b. • Anillo unitario: aquel que posee un elemento unitario y además, éste es distinto del neutro de la suma. • Anillo de división: es el anillo en el cual todo elemento, a excepción del neutro de la suma, tiene inverso. • Anillo con leyes de simplificación: aquel en el que se cumplen las leyes de simplificación. Si un anillo no tiene divisores del cero, se cumplen las leyes de simplificación, y el recíproco también es cierto. • Dominio de integridad: si un anillo es conmutativo, unitario y no posee divisores del cero, es un dominio de integridad. • Cuerpo: se trata de un anillo de división conmutativo.
Asíntota o Asintótico	Las asíntotas son rectas a las cuales una función se aproxima indefinidamente, cuando x o $f(x)$ tienden al infinito.
Ataque	Intento de quebrantar la seguridad de un criptosistema utilizando diversas técnicas de criptoanálisis.
Autenticar	Fortalecimiento de seguridad planeado en un infraestructura que provee autenticidad del sistema e integridad de datos pero no confiabilidad.
Bit	Es la unidad mínima de información empleada en informática. Representa un uno o un cero (abierto o cerrado, cierto o falso).
Biyectiva	Sea f una función de A en B, f es una función biyectiva, si y sólo si f es sobreyectiva e inyectiva a la vez. Si cada elemento de B es imagen de un solo elemento de A, diremos que la función es Inyectiva. En cambio, la función es Sobreyectiva cuando todo elemento de B es imagen de, al menos, un elemento de A. Cuando se cumplen simultáneamente las dos condiciones tenemos una

	función biyectiva.
Bloque	El criptografía un bloque es un conjunto de bits tomado de una secuencia ordenada de información. Ejemplo "1010010101110101" bloque de 16 bits.
Booleano	Operación cuyos dos posibles resultados son cierto o falso.
Byte	Se describe como la unidad básica de almacenamiento de información, siendo equivalente a ocho bits
Ca o Autoridad de Certificación	Autoridad de Certificación, es una entidad o empresa calificada y autorizada para emitir y avalar certificados digitales.
Cardinalidad	El cardinal indica el número o cantidad de los elementos constitutivos de un conjunto.
Certeza	Absoluta seguridad de ocurrencia de un suceso.
Certificado Digital	Documento electrónico vinculado a la clave de un individuo o entidad pública que ilustra atributos del portador de la clave, según verificación realizada por un tercero confiable o Autoridad de Certificación.
Cifrar o Encriptar	Acto de cifrar o encriptar mediante un algoritmo criptográfico para transformar la información de una manera ininteligible o criptograma.
Cifrador	Sistema hardware o software que incorpora un algoritmo para realizar el cifrado. Podemos distinguir 2 tipos: El cifrador por bloques, que es aquel que divide el mensaje en bloques de igual tamaño que posteriormente cifra con la misma clave y los cifradores de flujo, que cifran el mensaje carácter a carácter utilizando para cada carácter un elemento distinto de la clave.
Cifradores en Cascada	Emplean una pequeña clave para cada vuelta.
Cifradores Producto	Generan las subclaves empleadas a partir de una única clave principal mediante una serie de funciones concretas. Los cifradores de esta categoría son más seguros que los cifradores en cascada.
Clave	Uno de todos los posibles valores que pueden aplicarse al texto plano en combinación con un algoritmo de encriptación para producir texto en clave, o viceversa.
Clave Empaquetada	Clave de cifrado simétrico previamente cifrada usando un algoritmo asimétrico.
Claves Débiles	Conjunto de claves cuya combinación con el algoritmo criptográfico no produce un criptograma eficiente.
Coeficiente	Factor constante que multiplica una expresión, situado generalmente a su izquierda.

Congruencia	Expresión algébrica que manifiesta la igualdad de los restos de las divisiones de dos números congruentes por su módulo y que suele representarse con tres rayas horizontales (≡) puestas entre dichos números.
Criptoanálisis	El criptoanálisis es la ciencia opuesta a la criptografía (quizás no es muy afortunado hablar de ciencias opuestas, sino más bien de ciencias complementarias), ya que intenta romper esos sistemas, demostrando su vulnerabilidad, trata de descifrar los criptogramas.
Criptoanalista	Persona que trabaja en la ruptura de códigos y textos cifrados para recuperar la información de forma ilegítima.
Criptografía	Del griego *kryptos* (ocultar) y *grafos* (escribir), literalmente escritura oculta, la criptografía es la arte y ciencia de cifrar y descifrar datos utilizando las matemáticas. Haciendo posible intercambiar datos de manera que sólo puedan ser leídos por las personas o entidades a quien van dirigidos. La finalidad de la criptografía es en primer lugar, garantizar el secreto en la comunicación entre dos entidades, en segundo lugar, asegurar que la información que se envía es auténtica en un doble que el remitente sea realmente quien dice ser; y por último impedir que el contenido del mensaje enviado (habitualmente denominado criptograma) sea modificado en su tránsito.
Criptograma	Documento obtenido al cifrar un texto en claro. El alfabeto del criptograma no tiene por qué ser el mismo alfabeto que el del texto en claro.
Criptología	Es la conjunción de criptografía y criptoanálisis.
Criptoseguro	Algoritmo o método criptográfico que es irrompible.
Criptosistema	Mejor conocido como algoritmo criptográfico.
Datagrama	Entidad de datos auto contenida e independiente que transporta información suficiente para ser encaminada desde la computadora origen a la computadora destino sin tener que depender de que se haya producido anteriormente tráfico alguno entre ambos y la red de transporte.
Descifrar o Desencriptar	Acto de descifrar o desencriptar mediante un algoritmo criptográfico para transformar un criptograma en texto claro.
Difusión	Diluye la redundancia del texto plano repartiéndola a lo largo de todo el texto cifrado. El mecanismo más elemental para llevar a cabo una difusión es la transposición (que consiste en cambiar de sitio elementos individuales del texto plano).
Directorio	En criptografía un directorio es un repositorio de claves públicas de libre acceso usadas para validar por medio de certificados digitales una conexión segura.

Entropía	Medida de la incertidumbre existente ante un conjunto de mensajes, de los cuales se va a recibir uno solo.
Equiprobable	Conjunto de sucesos con igual probabilidad.
Espacio de Claves	Conjunto de claves que existen disponibles en el criptosistema.
Extranet	Red cooperativa que usa la tecnología de Internet para ligar actividades comerciales con proveedores, clientes u otros negocios. Las partes cooperativas o el público en general pueden acceder a la información.
Firma Digital	Firma electrónica inalterable que autentifica un emisor de mensajes y simultáneamente garantiza la integridad del mensaje.
Flujo, Cifrado en Flujo	Proceso de cifrado en el que cada carácter o bit de información de cifra de manera independiente al anterior.
Fuerza Bruta, Ataque	Método de criptoanálisis cuyo objetivo es probar uno de todos los posibles valores del espacio de claves para romper un criptosistema, también se conoce como método de búsqueda exhaustiva.
Grupo	En matemáticas, un grupo es un conjunto, con una operación binaria, que satisface ciertos axiomas. Sea una estructura formada por un conjunto, X, sobre cuyos elementos se ha definido una operación, *, o ley interna: • $X \times X \rightarrow X$ • $(x,y) \rightarrow x*y$ Si la operación verifica las siguientes propiedades, entonces se dice que la estructura es un Grupo con respecto a la operación *. • Asociativa: para todo *x, y, z* verifica :$x*(y*z) = (x*y)*z$ • Existencia del Elemento Neutro: x*1 = 1*x = x para todo x de X, siendo 1 el elemento neutro. • Existencia de Inverso (en caso de que la operación se denote aditivamente y no multiplicativamente, el término que se usa es Opuesto y el elemento neutro se denota 0): Para todo elemento x de X, existe otro elemento y de X, tal que: $x*y = y*x = 1$
Hash	Función unidireccional que produce un resumen del mensaje que no puede revertirse para reproducir el mensaje original.
Invariante	Magnitud o expresión matemática que no cambia de valor al sufrir determinadas transformaciones.
Intratable	Problema matemático sin solución.

Incertidumbre	Ausencia de certeza.
Inversa	Véase Inversa de anillo o de grupo.
Iteración	Acto de repetir.
Modulo	Residuo entero de una división
Monotónico	El tiempo siempre avanza hacia delante.
Número Primo	El conjunto de los números primos es un subconjunto de los números naturales que engloba a todos los elementos de este conjunto que son divisibles exactamente tan sólo por dos números naturales, el numero uno y si mismo. El Teorema fundamental de la Aritmética establece que cualquier entero positivo puede representarse siempre como un producto de números primos, y esta representación (factorización) es única.
PEM	Standard para encriptación de mensaje y autentificación de emisores de mensajes.
Permutar	Intercambiar de posición los elementos de un conjunto.
PGP	(Pretty Good Privacy), protocolo de aplicación y formato de datos disponible en una amplia variedad de sistemas operativos, usados para intercambiar mensajes de mails y archivos encriptados y autenticados.
Protocolo	Conjunto de reglas y pasos a seguir por las partes activas de una comunicación.
Pseudoaleatorio	Con causa, pero aparentemente impredecible, o impredecible en la práctica debido a la propagación de errores iniciales propia de un sistema caótico, pero predecible en forma de probabilidad
Redundancia	Cierta repetición de la información contenida en un mensaje, que permite, a pesar de la pérdida de una parte de este, reconstruir su contenido.
Reseña	Nota o resumen que se toma de los rasgos distintivos de mensaje para su autenticación y no modificación.
Residuo	Resto de la sustracción y de la división.
Ronda	Serie de pasos que se desarrollan ordenadamente durante un tiempo limitado.
Rotor	Parte giratoria de una máquina eléctrica o mecánica.
SSL	(Secure Sockets Layer), nivel de seguridad que se encuentra entre los niveles de aplicación y transporte. SSL protege de manera transparente los protocolos a nivel aplicación (como HTTP, para el cual fue originalmente concebido) y los datos, con poco esfuerzo por parte del diseñador de la aplicación.
Sobre Digital	Conjunción de la firma digital, la clave simétrica empaquetada y el texto cifrado.
Subclave	Parte de una clave que es el resultado de una transformación previa de la clave original para su uso en un algoritmo criptográfico.

Sustitución	Reemplazo de los elementos de un alfabeto por otro alfabeto, proceso muy usado en algoritmos simétricos de cifrado.
Texto Claro	Documento original que se desea proteger mediante el uso de técnicas criptográficas. Al conjunto de todos estos textos se le denota como M.
Transposición	Proceso que consiste en alterar el orden normal de los elementos de un alfabeto.
Variable Aleatoria	En estadística y teoría de probabilidad una variable aleatoria se define como el resultado numérico de un experimento aleatorio.
X.509	Formato de certificado usado para probar la identidad y la titularidad de clave pública que se basa en un sistema de confianza jerárquica.

F.- Bibliografía

- Pino Caballero Gil, Seguridad Informática. Técnicas Criptográficas, 1ra Edición, Editorial AlfaOmega, 1997.
- Fúster Sabater Amparo, de la Guía Martínez Dolores, Montoya Viniti Fausto, Técnicas Criptográficas de protección de datos,2da Edición, Editorial AlfaOmega, 2001.
- Schneier Bruce, Applied Cryptography Protocols, Algorithms, and Source Code in C, 2da Edición, Editorial John Wiley & Son, Inc, EUA, 2000.
- Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone, Handbook of Applied Cryptography, 1ra Edicion, Editorial CRC Press, EUA, 1996.
- Nash Andrew, Duane William, Joseph Celia, PKI, Infraestructura de Claves Públicas, 1ra Edición, Editorial Osborne McGraw-Hill, 2002.
- Ferguson Niels, Schneier Bruce, Practical Cryptography, 1ra Edición, Editorial John Wiley & Son, Inc, EUA, 2002.
- Welshenbach Michael, Cryptography in C and C++, 1ra Edición, Editorial Apress, 2001.
- Mediavilla Mauriz Manuel, Seguridad en Unix, 1ra Edición, Editorial AlfaOmega, 1998.
- G. Brassard, P. Bratley, Fundamentos de Algoritmia, 1ra Edición, Editorial Prentice Hall, 1997.
- Menecez, P. van Oorshot, S. Vanstone, HandBook of Applied Cryptography, 1ra Edición, Editorial Press, EUA, 1996.
- Chandra Pravir, Messier Matt, Viega John, Network Security with OpenSSL 1ra Edición, Editorial O`Reilly, EUA, 2002.

Libros Electrónicos.

- Lucena López Manuel José, Criptografía y Seguridad en computadores, 2da Edición, Universidad de Jaén, España, 1999.
- País Suárez Francisco, Estudio del Algoritmo de Cifrado Rijndael, Universidad de la Coruña, España, 2002.
- Ramío Aguirre Jorge, Libro Electrónico de Seguridad Informática y Criptografía, Universidad Politécnica de Madrid, España, 2004.
- Angel Angel José de Jesús, Criptografía para Principiantes, 1ra Versión, España, 2001.

Referencias Electrónicas.

- Lucena López Manuel José, Pagina Personal de Manuel J. Lucena, [en linea], Universidad de Jaen, [España], 2002-2003 Ultima actualización 12 Marzo de 2004 [Citado 15 de marzo de 2004], Texto de Español, Disponible en Internet en http://wwwdi.ujaen.es/~mlucena/
- Lucena López Manuel José, Criptografía y Números Primos, [en línea], Kriptopolis [España], 22 de junio de 2003, [Citado 2 Febrero 2004],Texto de Español, Disponible en Internet en http://www.kriptopolis.com/more.php?id=78_0_1_12_M2
- Sale Tony, The 1944 Bletchley Park Cryptographic Dictionary, [en línea], The National Archives and Records Administration, [EUA] , 20 de Julio de 1944, Ultima actualización 2001 [Citado 10 de Febrero 2004], Texto en Ingles, Disponible en Internet en http://www.codesandciphers.org.uk/documents/cryptdict/Cryptix.htm
- Sale Tony, Codes And Ciphers in the Second World War, [en línea], Big Oxford Computer Co Ltd, [EUA], [Citado 10 de febrero 2004], Texto en Ingles, Disponible en Internet en http://www.codesandciphers.org.uk/, Elementos: reconstruction of Enigma decipherment
- Schneider Bruce, Risk, Complexity, and Network Security, [en línea], Counterpane Internet Security, Inc, [EUA], Abril 2001, [Citado 20/03/2004], Documento en PDF -Texto en ingles, Disponible en Internet en http://www.counterpane.com/presentation1.pdf
- Schneider Bruce, Network Monitoring and Security, [en línea], Counterpane Internet Security, Inc, [EUA], Junio 2002, [Citado 20/03/2004], Documento en PDF-Texto en ingles, Disponible en Internet en http://www.counterpane.com/presentation3.pdf
- M. Jason Hinek (Very) Large RSA Private Exponent Vulnerabilities, [en línea], Universidad de Waterloo, [Canadá], 2 Febrero de 2004, [Citado 15/03/2004], Documento en PDF-Texto en ingles, disponible en Internet http://www.cacr.math.uwaterloo.ca/techreports/2004/cacr2004-02.pdf
- Universidad de Waterloo Center for applied Cryptographic Research, [en línea], Universidad de Waterloo, [Canadá],1998, [Citado 15/03/2004], Texto en Ingles, Disponible en Internet en http://www.cacr.math.uwaterloo.ca/
- Departament of Electrical and Computer Engineering Worcester Polytechnic Institute, Cryptographic & Information Security , [en linear], Departament of Electrical and Computer Engineering Worcester Polytechnic Institute, [EUA], 2004, [Citadel 16/03/2004], Texto en Ingles, Disponible en Internet en http://www.ece.wpi.edu/Research/Crypt/, Elemento Research.

- National Security Agency(NSA), Introduction to NSA/CSS, [en línea], National Security Agency(NSA), [EUA], 2004 ,Ultima actualización Enero 2004, [Citado 17/03/2004], Texto en Ingles, Disponible en Internet en http://www.nsa.gov/about/
- National Security Agency(NSA), Frequently Asked Questions (NSA), [en línea], National Security Agency(NSA), [EUA], 2004, Ultima actualización Enero 2004, [Citado 17/03/2004], Texto en Ingles, Disponible en Internet en http://www.nsa.gov/about/about00007.cfm, Elementos: About NSA, Information Assurance.
- ViaCorp, How electronic encryption works and how it will change your business, [en línea], ViaCorp.inc, [Australia], 1997, Ultima actualización Enero 2004, [Citado 17/03/2004], Texto en Ingles, Disponible en Internet en http://www.viacorp.com/crypto.html#laws
- Claude Elwood Shannon, Communication Theory of Secrecy Systems, [en línea], UCLA Computer Science Departament, [EUA], 1949, [Citado 18/02/2004], Texto en Ingles, Disponible en Internet en http://www.cs.ucla.edu/%7Ejkong/research/security/shannon1949/shannon1949.htm, También disponible en pdf en http://www.cs.ucla.edu/%7Ejkong/research/security/shannon1949.pdf
- Claude Elwood Shannon, A Mathematical Theory of Communication, [en linear], The bell System Technical Journal, [EUA], Octubre 1948, [Citado 18/02/2004], Documento en PDF - Texto en ingles, Disponible en Internet en http://cm.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf
- Bletchley Park Group, Cryptography, [en línea], Bletchley Park Group, [EUA], Fecha de publicación desconocida, Ultima actualización Febrero 2004, [Citado 09/03/2004], Documento en PDF-Texto en ingles, Disponible en Internet en http://www.bletchleypark.net/cryptology/ ,Documentos: Perfect_Secrecy.pdf, Perfect_Secrecy.pdf, Perfect_Secrecy.pdf, Perfect_Secrecy.pdf, Symmetric_Block_Ciphers.pdf, Hash_Functions.pdf
- Bletchley Park Group, Cryptography, [en línea], Bletchley Park Group, [EUA], Fecha de publicación desconocida, Ultima actualización Febrero 2004, [Citado 09/03/2004], Texto en Ingles, Disponible en Internet en http://www.bletchleypark.net/crypt/, Elementos: Basic Cryptography, Feistel Structure, Data Encryption Standard (DES), Cipher Modes, Advanced Encryption Standard (AES), RC4, Random Numbers, Diffie-Hellman, RSA, Public/Private Key Cryptography
- Varios autores, Códigos y utilerías de Algoritmos Criptográficos, [en línea], ftp.funet.fi [Finlandia], Junio 1995, Ultima actualización Febrero 2004, [Citado 25/03/2004], Códigos fuente en lenguaje C ANSI, Disponible en Internet vía ftp en ftp://ftp.funet.fi/pub/crypt/cryptography/, Elementos: asymmetric y symmetric

- Atreya Mohan, Digital Signatures & Digital Envelopes, [en línea], RSA Security Inc., [EUA], Febrero 2002, Ultima actualización Febrero 2002, [Citado 02/02/2004], Documento en PDF-Texto en ingles, Disponible en Internet en http://www.rsasecurity.com/products/bsafe/overview/Article5-SignEnv.pdf#xml=http://www.rsasecurity.com/programs/texis.exe/webinator/search/xml.txt?query=faq&pr=default_new&order=r&cq=&id=40d956761d
- RSA Security Inc., RSA Laboratories Frequently Asked Questions About Today's Version 4.1, [en línea], RSA Laboratories, [EUA], Mayo 2000, Ultima actualización Enero 2004, [Citado 02/02/2004], Texto en Ingles, Disponible en Internet en http://www.rsasecurity.com/rsalabs/node.asp?id=2152, También disponible en pdf vía ftp en ftp://ftp.rsasecurity.com/pub/labsfaq/rsalabs_faq41.pdf
- Bureau of Industry and Security U.S. Departament of Commerce, Commercials Encryption Export Controls, [en linear], Bureau of Industry and Security U.S. Departament of Commerce , [EUA], 1996, Ultimo actualización 6 Junio de 2002, [Citado 22/02/2004], Texto en Ingles , Disponible en Internet en http://www.bxa.doc.gov/Encryption/
- Electronic Frontier Foundation, Privacy, Security, Crypto, & Surveillance, [en línea], Electronic Frontier Foundation, [EUA], 1997, Ultima actualización 20 de Marzo de 2004, [Citado 20/03/2004], Texto en Ingles, Disponible en Internet en http://www.eff.org/Privacy/
- Cryptography World, The Cryptography Guide, [en línea], Cryptography World, [EUA], 2003, Ultima actualización Enero 2004, [Citado 16/03/2004], Texto en Ingles, Disponible en Internet en http://www.cryptographyworld.com/, Elementos: Algorithms, Key Management
- Deley David W., Computer Generated Random Numbers, [en línea], [EUA], 1991 [Citado 17/03/2004], Texto en Ingles, Disponible en Internet en http://world.std.com/~franl/crypto/random-numbers.html
- Miller Jim, Answers to Frequently Asked Questions about Electronic Money and Digital Cash (Release 2.0), [en línea], Roy Davies, [Inglaterra], 2002, Ultima actualizacion 2 Junio de 2004, [Citado 19/03/2004], Texto en Ingles, Disponible en Internet en http://www.ex.ac.uk/~RDavies/arian/emoneyfaq.html
- Young Eric A., Open SSL Proyect, [en línea], GNU Group, [EUA], 1999, Ultima actualización 17 Marzo de 2004, [Citado 20/03/2004], Código Fuente de la librería Open SSL para Unix y Borland C++ 5 para windows Version 0.9.7, Disponible en Internet en http://www.openssl.org/ También disponible en código fuente vía ftp en ftp://ftp.openssl.org/source/
- Kessler Gary C., An Overview of Cryptography, [en línea], [EUA], 1998, Ultima actualización Febrero de 2004, [Citado 07/03/2004], Texto en Ingles, Disponible en Internet en http://www.garykessler.net/library/crypto.html#intro, Elementos: Types of Cryptographic Algorithms, Cryptographic Algorithms in Action

- Denning Dorothy E. y Baugh William E., Cases Involving Encryption in Crime and Terrorism , [en línea], [EUA], Julio 1997, Ultima actualización 10 Octubre 1997, [Citado 22/03/2004], Texto en Ingles, Disponible en Internet en http://www.cs.georgetown.edu/~denning/crypto/cases.html
- Computer Security Resource Center, Cryptographic Tool Kit, [en línea], National Institute of Standards and Technology (NIST), [EUA], 23 Enero de 2001, Ultima actualización 7 de Marzo de 2003, [Citado 20/02/2004], Texto en Ingles, Disponible en Internet en http://csrc.nist.gov/CryptoToolkit/, Elementos: Encryption, Digital Signatures, Secure Hashing, Key Management, Random Number Generation, Message Authentication, Entity Authentication, Password Usage and Generation.
- Microsoft, Cryptography, [en línea], Microsoft, [EUA], 1998, Ultima actualización Enero 2004, [Citado 20/02/2004], Texto en Ingles, Disponible en Internet en http://research.microsoft.com/crypto/
- Integrity Sciences Inc, Elliptic Curve Cryptography, [en línea], Integrity Sciences, Inc, [EUA], 1997, Ultima actualización 1998, [Citado 23/03/2004], Texto en Ingles, Disponible en Internet en http://world.std.com/~dpj/elliptic.html
- Netscape Communications Corporation, Introduction to SSL, [en línea], Netscape Communications Corporation, [EUA], 1998, Ultima actualización 2002, [Citado 19/03/2004], Texto en Ingles, Disponible en Internet en http://developer.netscape.com/docs/manuals/security/sslin/index.html
- Daemen Joan y Rijmen Vincent, The Rijndael Specification, [en linea], National Institute of Standards and Technology (NIST), [EUA], 28 Febrero de 2001, [Citado 26/02/2004], Documento en PDF-Texto en ingles, Disponible en Internet en http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael-ammended.pdf
- Mason Andrew, IPSec Overview Part One: General IPSec Standards, [en línea], Cisco Systems, [EUA], 22 Febrero 2002, [Citado 30/03/2004], Texto en Ingles, Disponible en Internet en http://www.ciscopress.com/articles/article.asp?p=25470&seqNum=1
- Gonzalo Álvarez Marañón, Criptonomicon, [en línea], [España], 1997, Ultima actualización 2003, [Citado 20/01/2004], Texto de Español, Disponible en Internet en http://www.iec.csic.es/criptonomicon/default2.html Autenticacion, seguridad, e-commerce
- CriptoZona, Criptografía de clave simétrica o única, [en línea], CriptoZona, [España], 1997, Ultima actualización 1998, [Citado 25/01/2004], Texto de Español, Disponible en Internet en http://www.dat.etsit.upm.es/~mmonjas/cripto/

- Quirantes Arturo, La seguridad de los protocolos criptográficos, [en línea], [España], 2000 Ultima actualización 2001, [Citado 27/01/2004], Texto de Español, Disponible en Internet en http://www.ugr.es/~aquiran/cripto/informes/info002.htm
- Suárez Martínez Jaime, Mundocripto: criptografía, [en línea], MundoCripto, [España], 2000, Ultima actualización Enero 2004, [Citado 28/01/2004], Texto de Español, Disponible en Internet en http://mundocripto.come.to/
- R. Beresford, Finite-Field Arithmetic, [en línea], Data Encription Page, [EUA], 1997, [Citado 05/02/2004], Texto en Ingles, Disponible en Internet en http://www.anujseth.com/ crypto/ffield.html
- El Rincón de Quevedo, Introducción a la Criptografía: Aplicaciones Criptográficas, [en línea], El Rincón de Quevedo, [España], 2000, Ultima actualización Enero 2002, [Citado 17/01/2004], Texto en Español, Disponible en Internet en http://rinconquevedo.iespana.es/ rincón Quevedo/Criptografia/aplicaciones.htm
- Esteban Martí José Ramón, Criptografía, [en línea], Asociación de Internautas, [España], 1998, Ultima actualización 2002, [Citado 18/01/2004], Texto en Español, Disponible en internet en http://seguridad.internautas.org/criptografia.php
- Sáez Moreno Germán, El azar en la criptografía, [en línea], Criptonomicon, [España], 2002, Ultima actualización 2002, [Citado 18/01/2004], Texto en Español, Disponible en internet en http://www.iec.csic.es/criptonomicon/articulos/expertos52.html
- Martínez Fernando, Qué son los certificados digitales, [en línea], Criptonomicon, [España], Mayo 2003, [Citado 18/01/2001], Texto en Español, Disponible en internet en http://www.iec.csic.es/criptonomicon/articulos/expertos51.html
- Ángel José de Jesús, Criptografía para principiantes, [en línea], SeguriData, [España], Enero 2004, [Citado 19/01/2004], Texto en Español, Disponible en Internet en http://www.htmlweb.net/seguridad/cripto_p/cripto_princ_1.html
- Criptonomicon, PKI o los cimientos de una criptografía de clave pública, [en línea], Criptonomicon, [España], Enero 2004,[Citado 30/01/2004], Texto en Español, Disponible en Internet en http://www.iec.csic.es/criptonomicon/susurros/susurros11.html
- The Free Encryption, Cryptographic Libraries and Source Code, [en línea], [EUA], 2002, Ultima actualización 2 de Junio de 2003, [Citado 10/03/2004], Disponible en Internet en http:www.thefreecountry.com/sourcecode/encryption.shtml