NOM-151-SCFI-2002: Uso de la Criptografía para la Conservación de la Información.

TESIS QUE PARA OBTENER EL TITULO DE:
INGENIERO MATEMÁTICO

PRESENTA:
JOSÉ GUADALUPE GONZÁLEZ LOZA

ASESOR:
M. en C. VERONICA BOLAÑOS GRANADOS

Índice general

Índice de figuras

Índice de cuadros

Introducción

En el presente trabajo abordo la Norma Oficial Mexicana *NOM-151-SCFI-2002, prácticas comerciales-requisitos que deben observarse para la conservación de mensajes de datos*. Esta norma o simplemente la NOM-151 o norma, como la llamaremos de aquí en adelante, tiene como objetivo dar constancia de la existencia de archivos de datos electrónicos en determinado tiempo mediante el uso de elementos criptográficos y un sustento legal al dar valor probatorio a las *Constancias Digitales* expedidas de acuerdo a esta norma. En otras palabras, si una persona pretende probar ante un juez que cierto archivo electrónico existió en cierta fecha específica, deberá contar con una Constancia Digital emitida de acuerdo a la NOM-151. La NOM-151 cuenta un un sustento legal que da certeza jurídica a cualquier acto electrónico que cuente con una Constancia Digital de acuerdo a esta norma. De ahí su importancia en el mundo electrónico.

Para el entendimiento de los elementos involucrados en la parte técnica no legal de la NOM-151, en el capítulo 1 explico diversos conceptos criptográficos tales como firmas digitales, esquemas de cifrado, funciones hash, entre otros y menciono en qué basa su seguridad la criptografía y cómo se ve afectada en el tiempo, es decir, los algoritmos criptográficos tienen necesariamente un periodo de vigencia, después del cual ya no son criptográficamente seguros. En el capítulo 2 muestro una introducción a la *Abstrac Sintax Notation One*, con la que está escrita la NOM-151, y menciono algunos elementos intrínsecamente relacionados con la norma, tales como claves criptográficas, certificados digitales, entre otros. En el capítulo 3 describo la NOM-151, también menciono el documento RFC 3161, el cual contiene una recomendación para la emisión de estampas de tiempo, precisamente para acreditar la existencia de información electrónica en cierta fecha. Una gran e importante diferencia entre esta recomendación y la NOM-151 es el sustento legal con el que cuenta la norma al ser una Norma Oficial Mexicana. En el capítulo 4 describo los pasos y elementos necesarios para obtener una Constancia Digital de acuerdo a la norma. Finalmente, en el capítulo 5, menciono las áreas de oportunidad que detecté en esta norma, relacionadas por supuesto con los aspectos criptográficos involucrados.

Capítulo 1

Conceptos Criptográficos

En este capítulo se abarcan los temas básicos sobre criptografía como ¿qué es?, ¿para qué sirve?, ¿qué se necesita? y algunos ejemplos enfocándome más en lo que se aplica para la ***Norma Oficial Mexicana NOM-151-SCFI-2002, prácticas comerciales -requisitos que deben observarse para la conservación de mensajes de datos*** (en adelante la norma) y a la vez dando una introducción muy amplia sobre las muchas aplicaciones de esta disciplina.

¿Alguna vez nos hemos preguntado si es seguro enviar información a través de la red? Comunmente no nos interesa mucho saber si alguien puede ver nuestra información y la pueda manipular de alguna forma, esto es, borrarle, cambiarle o agregarle cosas. Tal vez podemos pensar: ¿quién estaría checando el correo eléctronico de otra persona?, ya que si nosotros no lo hacemos, nos imaginamos que nadie lo hace.

Pero ¿qué hay acerca de las transacciones electrónicas? Por ejemplo, cuando quieres comprar algo por medio de internet con una tarjeta de crédito, ¿tenemos la seguridad de que nadie está viendo los datos que proporcionamos y los pueda usar para comprar cosas en nuestro nombre? y, por otra parte, ¿qué seguridad tiene el banco de que somos la persona que decimos ser?

Claro que cuando hacemos eso (comprar por internet) nos fijamos que en la barra inferior del navegador esté un candadito amarillo que significa que ese sitio es seguro. Entonces debe haber algo o alguien que se encargue de dar esa seguridad; que es, propiamente dicho, la **criptografía** que se encarga de estudiar técnicas matemáticas para el aseguramiento de información, y el **criptoanálisis** que es el estudiar qué tan buenas son esas técnicas para el resguardo de dicha información.

Pero el comprar por internet consiste en enviar información, información que alguien tiene que verificar, y en dado caso, contestar también con cierta información.

Por ejemplo, digamos que Ana le quiere enviar un mensaje a Beto por cualquier medio. Ambos quieren que si alguien ve el mensaje no pueda saber en realidad que dice si no que sólo lo entiendan ellos dos, ¿cómo?, digamos que el mensaje original, que es la idea que quiere transmitir Ana, le llamamos *texto llano* denotado por $\mathcal{M}$ de Mensaje, y que de alguna manera no lo enviará así, por lo que ya expusimos, entonces tiene que modificarlo de alguna forma y entonces quedará como *texto cifrado* denotado por $\mathcal{C}$.

Al proceso que hace Ana le llamamos *Cifrar*, denotado por $\mathcal{E}$, y al proceso que debe hacer Beto para leer de forma clara el "mensaje" $\mathcal{M}$ le llamaremos *Descifrar*, denotado por $\mathcal{D}$. Entonces en términos matemáticos, digamos que Ana aplica una función para cifrar el mensaje original y Beto tiene que aplicarle otra función para descifrar el texto cifrado y poder entenderlo. En notación matemática sería:

Ana hace:

$$E(\mathcal{M}) = \mathcal{C}$$

Beto hace:

$$D(\mathcal{C}) = \mathcal{M}$$

Ahora, si digo que la **criptografía** es el estudio de técnicas matemáticas quiere decir que no es muy reciente si no que data ya de algunos siglos como las propias matemáticas y donde era exclusivo de los grupos militares y/o diplomáticos.

Por ejemplo,

- Cifrado César (siglo I a. C.): consistía en desplazarse en el alfabeto un número de posiciones fijo para cada letra; así, por ejemplo, si a la letra A le correspondía la letra D entonces a la letra B le correspondía la letra E y así sucesivamente. No funciona muy bien (no es muy seguro) ya que conociendo qué letra se repite con más frecuencia en el alfabeto utilizado se puede encontrar el número de desplazamientos que se utilizaron en la clave (función) y así encontrar la misma.

- Cifrado de Vigenère: es una generalización del anterior sólo que en éste, la clave toma sucesivamente varios valores; se logró romper en 1863.

- Cifrado Vernam (1917): es el caso límite del cifrado de Vigenère. Emplea un alfabeto binario ya que primeramente se utilizó para comunicaciones telegráficas haciendo uso del código *Baudot* (cinco dígitos binarios por carácter alfabético). La operación aritmética es una suma módulo 2, y la clave una secuencia binaria aleatoria de la misma longitud que el texto llano. Para el mensaje original "come soon" en código ASCII se muestran los resultados en el cuadro 1.1. Para

Cuadro 1.1: Ejemplo del cifrado Vernam

mensaje:	00011	01111	01101	00101	10011	01111	01111	01110
Clave:	11011	00101	01011	00110	10110	10101	01100	10010
Criptograma:	11000	01010	00110	00011	00101	11010	00011	11100

  recuperar el mensaje original se suma nuevamente al criptograma la secuencia aleatoria, ya que adición y sustracción coinciden en la aritmética módulo 2. Entonces la clave se utiliza sólo una vez, pues, en caso contrario, sucesivos criptogramas concatenados darían lugar a un cifrado tipo Vigenère. El cifrado Vernam fue usado en la Segunda Guerra Mundial y se creyó por mucho tiempo que era seguro totalmente, pero Shannon, en 1949 dio una prueba teórica de que no era así.

1.1. Objetivo de la Criptografía

En esta sección menciono en particular los objetivos que cumple la criptografía y dejando al lector que pueda pensar en dónde se puede llevar a cabo la aplicación de la criptografía para un determinado fin que le interese al mismo lector. Creo que con la pequeña definición de cada uno de los objetivos que describo aquí es fácil de entender a que se refieren.

De los principales objetivos encontramos:

- *Confidencialidad*: consiste en mantener en secreto la información de personas que no queremos que la vean. Sinónimo de privacidad, es decir, la va a poder ver quien nosotros queramos que la vea, quien tenga acceso a dicha información.

- *Integridad de datos*: se trata de que al ver la información nos podamos dar cuenta si ésta fue alterada de alguna manera, ya sea que le hayan cambiado, aumentado o simplemente borrado algo.

- *Autenticación*: sinónimo de "identificación", consta de comprobar si la persona que escribe o manda la información es realmente quien dice ser.

- *No repudiación*: esto se refiere a que si de alguna forma sabemos que alguien realizó una operación: está firmado por él, está su nombre, su clave o algo exclusivo de dicha persona, y si dicha persona dice que ella no fue quien hizo la actividad, podamos verificar si en realidad lo hizo o no y esa persona no lo pueda negar.

1.2. Primitivas Criptográficas

Esta sección trata de las técnicas iniciales utilizadas para poder tener mensajes en secreto (o confidenciales) y, que a través del tiempo, sólo se han modificado y han permanecido ahí para usarse como base a los algoritmos más complejos. No se habla de cosas triviales o simples, son técnicas con fundamento para poder ser utilizadas en criptografía, que nos dan confianza para saber que no es fácil clonarlas, por decirlo de alguna manera. Se habla de números pseudoaleatorios, que aunque son pseudo, no quiere decir que es muy fácil obtenerlos, si no que, si quisieramos obtener números aleatorios realmente se ha comprobado que es muy difícil tanto en tiempo como en recursos. También hablo de funciones de autenticación y funciones hash, las primeras creo que está claro para qué se utilizan y las últimas sirven para resumir algo, es decir, si se quiere cifrar un mensaje, al aplicarle una función hash obtendremos un texto cifrado de longitud menor al mensaje original y es en la que me enfocaré más por ser empleada en la norma.

1.2.1. Generador de Números Pseudoaleatorios

Existen muchas situaciones en criptografía donde es importante generar números aleatorios, cadenas de bits, etc. Sin embargo, esto suele ser demasiado caro, por lo cual, en la práctica, sólo se usan Generadores de Bits Pseudo-Aleatorios (GBPA).

Un GBPA comienza con una cadena de bits aleatoria corta (una "semilla") y se expande de tal forma que "parece" una cadena de bits muy larga.

Dados k y l enteros positivos tal que $l \geq k+1$ (donde l es una función polinomial específica de k). Un (k,l)-generador-de-bits pseudoaleatorio ((k,l)-GBPA) es una función $f : (\mathbb{Z}_2)^k \to (\mathbb{Z}_2)^l$ que puede ser calculada en tiempo polinomial (como una función de k). La entrada $S_0 \in (\mathbb{Z}_2)^k$ es llamada la semilla, y la salida $(s_0) \in (\mathbb{Z}_2)^l$

es llamada *cadena de bits pseudoaleatoria*.

Entre los generadores más conocidos tenemos al generador de "congruencias lineales" y al generador "RSA".

El primero es muy popular y a la vez muy inseguro, consiste en:

Dado $M \geq 2$ entero, y dados $1 \leq a, b \leq M-1$, definimos

$$k = \log_2 M$$

y

$$k+1 \leq l \leq M-1.$$

Para una semilla s_0, donde $0 \leq s_0 \leq M-1$, definimos

$$s_i = (as_{i-1} + b) \mod M$$

para $1 \leq i \leq l$, entonces definimos

$$f(s_0) = (z_1, z_2, \ldots, z_l),$$

donde

$$z_i = s_i \mod 2,$$

$1 \leq i \leq l$, entonces f es un (k,l)-GBPA.

Ejemplo.

Podemos obtener un (5,10)-GBPA tomando $M = 31$, $a = 3$ y $b = 5$ en el generador de congruencias lineales. Si consideramos el mapeo $s \mapsto 3s + 5 \mod 31$, entonces $13 \mapsto 13$, y los otros 30 residuos son permutaciones en un ciclo de longitud 30, llamémosle 0, 5, 20, 3, 14, 16, 22, 9, 1, 8, 29, 30, 2, 11, 7, 26, 21, 6, 23, 12, 10, 4, 17, 25, 18, 28, 27, 24, 15, 19. Si la semilla es diferente de 13, entonces la semilla especifica el punto donde empieza el ciclo, y los siguientes 10 elementos, reducidos con módulo 2, forma la secuencia pseudo-aleatoria.

Las posibles (31) cadenas de bits pseudo-aleatorias producidas por este generador se muestran en el cuadro 1.2.

Cuadro 1.2: Cadenas de bits pseudo-aleatorias de (5,10)-GBPA

semilla	*secuencia*	*semilla*	*secuencia*
0	1010001101	16	0110100110
1	0100110101	17	1001011010
2	1101010001	18	0101101010
3	0001101001	19	0101000110
4	1100101101	20	1000110100
5	0100011010	21	0100011001
6	1000110010	22	1101001101
7	0101000110	23	0001100101
8	1001101010	24	1101010001
9	1010011010	25	0010110101
10	0110010110	26	1010001100
11	1010100011	27	0110101000
12	0011001011	28	1011010100
13	1111111111	29	0011010100
14	0011010011	30	0110101000
15	1010100011		

1.2.2. Funciones de Autenticación

Algún mecanismo para autenticar el mensaje o la firma digital puede ser visto de manera general que posee dos niveles fundamentalmente. En el más bajo nivel, debe existir una cierta clase de funciones que produzcan un autenticador: un valor que es usado para autenticar el mensaje. Esta función de bajo nivel es una primitiva de un protocolo de autenticación de un nivel más alto que permite al receptor verificar la autenticidad del mensaje.

Los tipos de funciones que pueden ser usadas para producir un autenticador se pueden clasificar en tres grupos:

- Cifrado del Mensaje.

  El texto cifrado del mensaje completo sirve como autenticador. El mensaje cifrado provee por sí mismo una medida de autenticación. El análisis difiere para los esquemas de cifrado convencionales y los esquemas de cifrado de clave pública que describo adelante.

- Message Authentication Code (MAC).

  Una técnica de autenticación alternativa involucra el uso de una clave secreta para generar un bloque pequeño de datos de tamaño fijo. Conocido como dígito verificador criptográfico o MAC (código de autenticación de mensaje) que es apendizado al mensaje. Esta técnica asume que dos entidades en comunicación, Ana y Beto, comparten una clave secreta k común. Cuando Ana tiene un mensaje para madar a Beto, se calcula el MAC como una función del mensaje y la clave : MAC $= C_k(M)$.
  El mensaje más el MAC son transmitidos al receptor que se desea. Ese receptor realiza el mismo cálculo sobre el mensaje recibido, usando la misma clave secreta, para generar un nuevo MAC. El MAC recibido es comparado con el MAC calculado. Si asumimos que sólo el receptor como el emisor conocen la identidad de la clave secreta, y si el MAC recibido concuerda con el MAC calculado, entonces:

  1. El receptor está seguro que el mensaje no ha sido alterado. Si un atacante altera el mensaje pero no altera el MAC, entonces el cálculo del MAC del receptor diferirá del MAC recibido.
  2. El receptor está seguro que el mensaje es del emisor que dice ser. Porque nadie más conoce la clave secreta, nadie más podría preparar un mensaje con su propio MAC.
  3. Si el mensaje incluye una secuencia numérica, entonces el receptor puede estar seguro de esta secuencia, porque un atacante no podrá alterar la secuncia exitosamente.

  Una función MAC es similar a *cifrar*, a diferencia que el algoritmo MAC no necesita ser reversible, como debe serlo la función de cifrar para *descifrar*. Resultando que, a causa de las propiedades matemáticas de las funciones de autenticación, es menos vulnerable ha ser rota que el *cifrado*.

- Función Hash.

  Una variación de los MAC son las funciones hash de un sentido. Como en los MAC, una función hash toma un mensaje M de tamaño variable como entrada y produce un hash h de tamaño fijo, a veces llamado resumen del mensaje, como salida. El hash es una función de todos los bits del mensaje y capaz de detectar un error: Un cambio en algún bit del mensaje resulta un cambio en el código hash.

1.2.3. Funciones Hash

Las funciones hash criptográficas que juegan un papel importante en la criptografía moderna y las funciones hash usadas en aplicaciones computacionales no criptogáficas, no son más que el mapeo correspondiente de un dominio muy grande a un rango (de valores) muy pequeños.

Aquí me enfoco sólo en las funciones hash (de un sentido) criptográficas que en particular son usadas para la integridad de datos y autenticación del mensaje.

Una función hash (h) mapea cadenas de bits de longitud finita arbitraria a cadenas (arreglos) de longitud fija, digamos n bits. Para un dominio D y rango R con $h : D \rightarrow R$ y $\mid D \mid \geq \mid R \mid$, la función es llamada comunmente "función mucho a uno".

Las funciones hash son usadas para integridad de datos junto con esquemas de firma digital, donde por muchas razones a un mensaje primero se le aplica la función hash, y después el valor hash, como algo representativo del mensaje, es firmado en lugar del mensaje original.

Una función hash (de un sentido), $H(M)$, opera sobre la pre-imágen de un mensaje de longitud arbitraria, M. Y regresa un valor hash, h, de longitud fija.

$$h = H(M)$$

Algunas funciones pueden tomar entradas de longitud arbitraria y regresar salidas de longitud fija, pero las funciones hash de un sentido tienen características adicionales que las hacen funciones estrictamente de un sentido:

Dado M, es fácil de calcular h.

Dado h, es muy difícil de calcular M tal que $H(M) = h$.

Dado M, es muy difícil encontrar otro mensaje $M^{'}$ tal que $H(M) = H(M^{'})$.

Si alguien maliciosamente pudiera hacer estas cosas difíciles (computacionalmente hablando), podría minar la seguridad de cada protocolo que use funciones hash de un sentido. En general, el propósito de las funciones hash de un sentido es de proveer una "huella digital" única del mensaje . Si Ana firmó M usando un algoritmo digital de firmado sobre $H(M)$, y Beto pudiera producir $M^{'}$, otro mensaje diferente de M donde $H(M) = H(M^{'})$, entonces Beto podría demandar que Ana firmó $M^{'}$.

Requerimientos de una Función Hash

Examinaré los requerimientos para una función hash que es usada para autenticar mensajes. El propósito de la función hash es producir una "huella digital" en un documento, mensaje u otro bloque de datos. Para usarla en el caso de autenticación de mensajes, una función hash H debe tener las siguientes propiedades:

1. H debe poder aplicarse a bloques de datos de cualquier tamaño.

2. H produce un valor de longitud fija como salida.

3. $H(x)$ es relativamente fácil de calcular para cualquier x.

4. Para cualquier valor (código) h, debe ser computacionalmente imposible encontrar x tal que $H(x) = h$, conocido como la propiedad de "un sólo sentido".

5. Para cualquier valor x , debe ser computacionalmente imposible encontrar $y \neq x$ con $H(y) = H(x)$. Esto es, que la función hash sea débil a resistencia a colisiones.

6. Debe ser computacionalmente imposible encontrar cualquier par (x, y) tal que $H(x) = H(y)$, conocido como fuerte a resistencia a colisiones.

Las primeras tres propiedades son requisitos para autenticar mensajes.
La cuarta propiedad es la propiedad de un sentido: es fácil generar un código dado un mensaje pero virtualmente imposible generar un mensaje dado un código. Esta propiedad es importante si la técnica de autenticación envuelve el uso de un valor secreto.
La quinta propiedad garantiza que no se puede encontrar un mensaje alternativo que nos dé el mismo valor hash de un mensaje dado .
La sexta propiedad se refiere a cuanto resiste la función hash a la clase de ataque conocida como "ataque basado en la paradoja de cumpleaños" que explico más adelante.

Funciones Hash Simples

Todas las funciones hash operan de acuerdo a los siguientes principios generales. La entrada (mensaje, archivo, etc.) es visto como una secuencia de bloques de n-bits. La entrada es procesada en un bloque, en un tiempo específico, en una manera iterativa, para producir una función hash de n-bits. Una de las funciones hash más

simples es la "bit por bit" OR-exclusiva (XOR) de cada bloque. Se puede expresar como sigue:

$$C_i = b_{i1} \oplus b_{i2} \oplus \cdots \oplus b_{im}$$

donde:
$C_i = i$-ésimo bit del código hash, $1 \leq i \leq n$.
$m =$ número de bloques de n-bits en la entrada.
$b_{ij} = i$-ésimo bit en j-ésimo bloque.
$\oplus =$ operación XOR .

Esta operación es de la siguiente manera:

1. Inicialmente el conjunto de los n-bits del hash vale 0.
2. El proceso para cada bloque de n-bits de datos es como sigue:
   - Se rota el actual valor hash un bit a la izquierda.
   - Se hace la operación XOR al bloque dentro del valor hash.

Ataque Basado en la Paradoja de Cumpleaños

Suponga que se usa un código hash de 64 bits. Uno puede pensar que es bastante seguro. Por ejemplo si un hash cifrado C es transmitido con el correspondiente mensaje no cifrado M, entonces un oponente podría necesitar encontrar un $M^{'}$ tal que $H(M^{'}) = H(M)$ para sustituir otro mensaje y engañar al receptor.
En promedio el oponente tendría que probar cerca de 2^{63} mensajes para encontrar uno que le pegue al hash del mensaje interceptado.

Yuval propone la siguiente estrategia:

1. La fuente, es preparada para "firmar" un mensaje por un apropiado MAC de m-bits y cifrando ese MAC con la clave privada de Ana.
2. El oponente genera $2^{m/2}$ variaciones sobre el mensaje, todas con esencialmente el mismo mecanismo. El oponente prepara igual número de mensajes, todas variaciones fraudulentas para ser sustituidos por el mensaje original.
3. Los dos conjuntos de mensajes son comparados para encontrar un par de mensajes que produzcan un mismo hash. La probabilidad de éxito, por la paradoja del cumpleaños, es mayor que 0.5.

4. El oponente ofrece la variación válida a Ana por firma. Esta firma puede entonces ser adherida a la variación fraudulenta para transmitirla al receptor. Porque las dos variaciones tienen el mismo hash, producirán la misma firma; el oponente está seguro del éxito aunque la clave para cifrar sea desconocida.

Algunos Algoritmos Hash

Describo el algoritmo MD5, que es el utilizado en la NOM, y algunos otros para observar las principales diferencias más adelante.

- Algoritmo MD5.

  Llamado también algoritmo de digestión de mensajes, descrito en el RFC 1321, fue desarrollado por Ron Rivest en el MIT (colaborador también de RSA). Hasta hace pocos años, cuando tanto la fuerza bruta y las preocupaciones criptoanalíticas se habían presentado, este algortimo era considerado el algoritmo hash más seguro.

  El MD5 toma como entrada un mensaje de longitud arbitraria y produce como salida un mensaje (digesto o resumido) de 128 bits. La entrada es procesada en bloques de 512 bits. El procedimiento consiste de los siguientes pasos:

  1. El mensaje es rellenado en longitud congruente a 448 modulo 512 (longitud $\equiv$ 448 mod 512). Esto es, la longitud del mensaje relleno es 64 bits menos que un múltiplo entero de 512 bits.
  2. Una representación de 64 bits del mensaje original (antes de ser rellenada) es apendizada al resultado del paso anterior. Si la longitud original es más grande que 2^{64}, entonces sólo el orden de 64 bits es usado. Así, el campo contiene la longitud del mensaje original, mod 64.
  3. Un búfer de 128 bits es usado para mantener resultados intermedios y finales de la función hash. El búfer puede ser representado por 4 registros de 32 bits (`A, B, C, D`). Estos registros son inializados en los siguientes enteros (valores hexadecimales) de 32 bits:

     ```
     A = 67452301
     B = EFCDAB89
     C = 98BCADFE
     D = 10325476
     ```

4. El corazón del algoritmo es una función de compresión que consiste de cuatro vueltas de procesamiento; las cuatro vueltas tienen una estructura similar, pero cada una usa diferentes funciones lógicas, referido como `F, G, H, I` en la especificación.

   Cada vuelta toma como entrada el bloque actual de 512 bits siendo procesado (Y_q) y el valor del búfer de 128 bits `ABCD` y actualizando el contenido del búfer. Cada vuelta hace uso de un cuarto de los 64 elementos de una tabla T[1 ... 64], construida de la función seno. El i-ésimo elemento de T, denotado como T[i], tiene valor igual a la parte entera de $2^{32} \times \mid \sin(i) \mid$, donde i esta en radianes. Como $\mid \sin(i) \mid$ es un número entre 0 y 1, cada elemento de T es un número que puede ser representado en 32 bits.
   La salida de la cuarta vuelta es adherida a la entrada de la primera vuelta (CV_q) para producir CV_{q+1}. La adición es hecha independientemente por cada una de las cuatro palabras en el búfer con cada una de las correspondientes palabras en CV_q, usando adición modulo 2^{32}.

5. Después todos los L bloques de 512 bits tienen que ser procesados, la salida de la L-ésima etapa es la digestión del mensaje de 128 bits.

se puede resumir el algoritmo MD5 de la siguiente manera:

$$CV_0 = IV$$

$$CV_{q+1} = SUM_{32}(CV_Q, RF_I[Y_q, RF_H[Y_q, RF_G[Y_q, RF_F[Y_q, CV_q]]]])$$

$$MD = CV_L$$

donde,

IV = valor inicial del búfer `ABCD`, definido en el paso 3
Y_q = el q-ésimo bloque de 512 bits del mensaje
L = número de bloques en el mensaje
CV_q = variable encadenada procesada con el q-ésimo bloque del mensaje
RF_x = función de vuelta usando la función lógica x
MD = valor final de la digestión del mensaje
SUM_{32} = Adición modulo 2^{32} desarrollado separadamente en cada palabra del par de entrada.

- Algoritmo MD4

  Precursor del MD5 desarrollado por el mismo diseñador. Fue originalmente publicado como RFC en 1990 y una ligera revisión publicada en el RFC 1320 en 1992, misma fecha que el MD5. Es en teoría lo mismo que MD5 pero menos complejo.

- Algoritmo SHA-1

  El algoritmo de seguridad hash (por sus siglas en inglés) fue desarrollado por el Instituto Nacional de Estándares y Tecnologías de Estados Unidos de América y publicado como Información Federal para Procesamiento de Estándares (FIPS) en 1993. Es basado en el algoritmo MD4.

  Este algoritmo toma como entrada un mensaje de longitud máxima de 2^{64} bits y produce como salida una digestión del mensaje de 160 bits. La entrada es procesada en bloques de 512 bits. El procedimiento sigue la estructura del MD5 con bloques de 512 bits pero con un hash y variable encadenada de 160 bits.

- Algoritmo RIPEMD-160

  Este algoritmo fue desarrollado bajo el proyecto RIPE (Race Integrity Primitives Evaluation) europeo por un grupo de investigadores que lanzaron parcialmente ataques exitosos sobre MD4 y MD5. Este algoritmo toma como entrada un mensaje de longitud arbitraria y produce como salida un mensaje resumido de 160 bits. La entrada se procesa en bloques de 512 bits. El procesamiento del mensaje sigue la estructura del MD5 y también el tamaño del hash y de la variable encadenada; como en el SHA-1, es de 160 bits.

- Algoritmo HAVAL

  Es una función hash de longitud variable y modificación del MD5. Este algoritmo procesa mensajes en bloques de 1024 bits (el doble que MD5). Tiene ocho variables encadenadas de 32 bits, el doble que MD5. El número de vueltas es variable, de 3 a 5 (cada una consta de 16 pasos), y puede producir un valor hash de 128, 160, 192, 224 o 256 bits.

  HAVAL remplaza a las simples funciones no lineales del MD5 con siete funciones variables no lineales superiores, cada cual satisface un estricto criterio de avalancha. Cada vuelta usa una función singular pero en todos los pasos se aplica una permutación diferente a las entradas. Este algoritmo tiene dos

rotaciones.

Existen algunas funciones más de las cuales sólo hablaremos del tamaño (número de bits) del (valor) hash que regresan y de la velocidad de cifrado con un procesador a 33 Mhz, sólo como ejemplo o para comparar entre las mismas se muestran en el cuadro 1.3.

Cuadro 1.3: Velocidad de algunas funciones hash en un 486 a 33 MHz

Algoritmo	Tamaño del hash	Velocidad de cifrado (kbs)
GOST	256	11
HAVAL (3 vueltas)	variable	168
HAVAL (4 vueltas)	variable	118
HAVAL (5 vueltas)	variable	95
MD4	128	236
MD5	128	174
N-HASH (12 vueltas)	128	29
RIPEMD	160	182
SHA	160	75
SNEFRU (4 pasos)	128	48

1.3. Esquemas de Cifrado

En esta sección examino lo que se podría llamar criptografía clásica, donde sólo se aplica dicho procedimiento a un mensaje que queremos transmitir por cualquier medio, y que si alguien lo ve, no lo pueda entender y sólo lo pueda leer el destinatario que nosotros permitamos, aplicando algún procedimiento, el cual se puede ver como una permutacion de caractéres, por ejemplo, a la letra A asignarle la letra C, a la letra B asignarle la letra D, etc. Pero que deben de seguir cierta estructura y, de acuerdo a eso, deben tener un nombre en específico, aquí hablo del "cifrado de cambio" y del "cifrado de sustitución" por ejemplo.

Un *criptosistema* es una quintupla $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \epsilon, \mathcal{D})$ donde se satisfacen las siguientes condiciones:

1. $\mathcal{P}$ es un conjunto finito de todos los posibles mensajes (sin cifrar).

2. $\mathcal{C}$ es un conjunto finito de los posibles textos cifrados (mensajes cifrados).
3. $\mathcal{K}$ es un conjunto finito de las posibles claves.
4. ϵ es un conjunto de las posibles funciones que sirven para cifrar los mensajes.
5. $\mathcal{D}$ es un conjunto de las posibles funciones que sirven para descifrar los mensajes.
6. Para cada $K \in \mathcal{K}$ hay una regla de *cifrado* $e_K \in \epsilon$ y su correspondiente regla de *descifrado* $d_K \in \mathcal{D}$. Cada $e_K : \mathcal{P} \to \mathcal{C}$ y $d_K : \mathcal{C} \to \mathcal{P}$ son funciones tales que $d_K(e_K(x)) = x$ para todo mensaje $x \in \mathcal{P}$.

Por ejemplo,

- "cifrado de cambio".

  Dado $\mathcal{P} = \mathcal{C} = \mathcal{K} = \mathbb{Z}_{26}$, para $0 \leq K \leq 25$, se define

  $$e_K(x) = x + K \mod 27$$

  y

  $$d_K(y) = y - K \mod 27$$

  $(x, y \in \mathbb{Z}_{27})$.
  Es definido sobre $\mathbb{Z}_{27}$ porque el alfabeto en español tiene 27 letras (sin contar la ch y la rr), pero se podría definir sobre cualquier $\mathbb{Z}_m$ para algún $\mod m$.

  Nota Para el caso particular donde $K = 3$, el criptosistema es el ya mencionado "cifrado César".

  Podemos usar el "cifrado de cambio" (con módulo 27) para cifrar texto en español ordinario estableciendo una correspondencia entre los caractéres alfabéticos y los residuos módulo 27 de la siguiente manera: $A \leftrightarrow 0$, $B \leftrightarrow 1$, ..., $Z \leftrightarrow 26$.

  Ejemplo.

  Supongamos que la clave para el "cifrado de cambio" es $K = 11$, y el mensaje es

  tevereamedianoche

Primero convertimos el mensaje a una secuencia de enteros usando la correspondencia específica, y obtenemos:

$$\begin{array}{cccccccccc} 20 & 4 & 22 & 4 & 18 & 4 & 0 & 12 & 4 \\ 3 & 8 & 0 & 13 & 15 & 2 & 7 & 4 & \end{array}$$

Después, le sumamos 11 a cada valor y reducimos cada suma módulo 27:

$$\begin{array}{ccccccccc} 4 & 15 & 6 & 15 & 2 & 15 & 11 & 6 & 15 \\ 14 & 19 & 11 & 24 & 26 & 13 & 18 & 15 & \end{array}$$

Finalmente, convertimos la secuencia de enteros a caractéres alfabéticos, obteniendo el mensaje cifrado:

$$EOGOCOLGOÑLXZNRO$$

Para descifrar el mensaje cifrado, *Beto convertirá* primero dicho mensaje a una secuencia de enteros, después restará 11 de cada valor, reducirá módulo 27, y finalmente convertirá la secuencia de enteros a caractéres alfabéticos.

- "cifrado de sustitución".

  Dado $\mathcal{P} = \mathcal{C} = \mathbb{Z}_{26}$, $\mathcal{K}$ consiste de todas las posibles permutaciones de los 26 simbolos 0, 1, ..., 25, considerando que sea el alfabeto inglés, para cada permutación $\pi \in \mathcal{K}$, definimos

  $$e_\pi(x) = \pi(x),$$

  y definimos

  $$d_\pi(y) = \pi^{-1}(y)$$

  donde π^{-1} es permutar inversamente a π.

  En este esquema se puede pensar de dos formas, una sería asignarle enteros a cada letra como en el cifrado de cambio o, la otra, que es la mejor, asignarle alguna letra a las que se quieren cifrar para verlo como una permutación de caractéres alfabéticos.

  Por ejemplo,

a	b	c	d	e	f	g	h	i	j	k	l	m
X	N	Y	A	H	P	O	G	Z	Q	W	B	T

n	o	p	q	r	s	t	u	v	w	x	y	z
S	F	L	R	C	V	M	U	E	K	J	D	I

Así, e_π(a) = X, e_π(b) = N, etc. La función de descifrado es la permutación inversa. Esto se forma escribiendo la segunda línea primero, y acomodándola en orden alfabético. Quedaría como sigue:

A	B	C	D	E	F	G	H	I	J	K	L	M
d	l	r	y	v	o	h	e	z	x	w	p	t

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
b	g	f	j	q	n	m	u	s	k	a	c	i

De ahí, d_π(A) = d, d_π(B) = l, etc.

1.4. Criptografía de Clave Pública

Esta sección se trata de usar técnicas para cifrar información, bloques de bits, en las que se utilicen, como el título indica, claves. No es simplemente tomar algo (mensaje, por ejemplo) y aplicarles cierta función como en los esquemas de cifrado, se consideran claves que se escogen (crean) de cierta manera y que pueden ser procedimientos tan seguros como queramos dependiendo del tamaño de esas claves y del algoritmo. Comienzo con el procedimiento que se debe llevar a cabo al utilizar criptografía de clave pública y termino hablando muy en particular del algortimo RSA que es el que se utiliza en el caso específico de la norma.
Como ya sabemos, K es el espacio de las claves y E_e y D_d son los conjuntos de transformaciones de cifrado y descifrado respectivamente, donde nos podemos confundir en cuanto a los esquemas de cifrado (simples) que comenté en la introducción de este capítulo que más propiamente dicho es "criptografía simétrica", donde la clave (función) para descifrar es la inversa de la clave para cifrar.

Si consideramos cualquier par asociado de transformaciones cifrado/descifrado (E_e/D_d) suponiendo que cada par tiene la propiedad de que E_e es computacionalmente difícil de obtener, tomamos de manera aleatoria un texto cifrado $c \in \mathcal{C}$, para hallar el mensaje $m \in \mathcal{M}$ tal que $E_c(m) = c$.

Esta propiedad implica que si E_e es difícil de calcular para determinar la clave d correspondiente, entonces E_e es vista aquí como una función trampa de un sólo sentido con d siendo la (trampa) información necesaria para calcular la inversa y de ahí se pueda descifrar el mensaje.

Nota Esto no se parece en nada a la criptografía de clave simétrica, donde d y e son esencialmente lo mismo.

Se utiliza de la siguiente manera:

- Beto selecciona (escoge, hace) el par de claves (e, d) y manda, por cualquier canal, e (clave pública) a Ana.
- Beto mantiene la d (clave privada) en secreto (segura).
- Ana puede mandar un mensaje m a Beto aplicando la transformación de cifrado determinada por la clave pública de Beto, obteniendo el texto cifrado $C = E_e(m)$.
- Beto descifra el mensaje c aplicando la transformación inversa D_d determinada únicamente por d.

Entonces un esquema de clave pública es aquel en el cual se tiene el par de funciones de transformación (e, d), donde e es la clave pública, pues se puede enviar por cualquier canal y entonces la puede ver y tener quien quiera, y d la clave privada, que sólo tiene una sola entidad.

Para que dicho esquema sea seguro, debe ser computacionalmente difícil obtener d de e. Esto es, que si varias entidades, por ejemplo: Ana, Alejandra, Amalia, le quieren mandar un mensaje cifrado (c) a Beto basta con que hagan $E_e(m) = c$. Y si alguna de ellas destruye el mensaje m, entonces debe serle muy difícil (casi imposible) obtener m teniendo c, ya que como sabemos, $m = D_d(c)$ y d sólo la conoce Beto.

En la práctica, los puntos más importantes (diferencia entre clave pública y simétrica) son:

1. La criptografía de clave pública facilita firmas eficientes (particularmente No-repudiación) y administración de claves; y
2. La criptografía de clave simétrica es eficiente para el cifrado y algunas aplicaciones de la integridad de datos.

Aplicaciones de Criptosistemas de Clave Pública

- Cifrado/Descifrado : El emisor manda un mensaje con la clave pública del receptor.
- Firma digital: El emisor "firma" firma un mensaje con su clave privada.
- Intercambio de clave: Dos partes cooperan para intercambiar una clave (s). Esto se puede ver de diferentes formas, implicando la(s) clave(s) privada(s) de una o ambas partes.

El cuadro 1.4 muestra los principales algortimos de clave pública y sus aplicaciones.

Cuadro 1.4: Aplicaciones para criptosistemas de clave pública

Algortimo	**Cifrado/Descifrado**	Firma Digital	Intercambio de Clave
RSA	SI	SI	SI
Diffie-Hellman[1]	NO	NO	SI
DSS	NO	SI	NO

[1] Descrito en el RFC2631, en el RFC2875 y en el RFC3526

La norma se enfoca en criptografía de "clave pública" y más propiamente dicho en "RSA" que sirve tanto para *cifrar* como para *firmar* un mensaje como se menciona en el cuadro.

El algortimo RSA desarrollado por Rivest, Shamir y Adleman hace uso de una expresión con exponenciales. El texto llano es cifrado en bloques, los cuales tienen un valor binario menor que algún número n. Esto es, el tamaño del bloque debería ser menor o igual a $\log_2(n)$; en la práctica el tamaño del bloque es de 2^k bits, donde $2^k \leq n \leq 2^{k+1}$. El cifrado y el descifrado son de la siguiente forma, donde M es el bloque de texto llano y C el bloque de texto cifrado:

$$C = M^e \mod n^1$$
$$M = C^d \mod n = (M^e)^d \mod n = M^{ed} \mod n$$

Ambos, emisor y receptor, conocen el valor de n. El emisor conoce el valor de e, sólo el receptor conoce el valor de d. Aquí la clave pública es $\{e, n\}$ y la clave privada $\{d, n\}$ y el algoritmo es como sigue:

[1] $a \cong b \mod c$ se escribirá como $a = b \mod c$

- Generación de la clave.

  se selecciona p, q ambos primos
  se calcula $n = p \times q$
  se calcula $\phi(n) = (p-1)(q-1)$
  se selecciona un entero e y se calcula $\gcd(\phi(n), e) = 1; 1 < e < \phi(n)$
  se calcula $d = e^{-1} \mod \phi(n)$
  Clave pública = $\{e, n\}$
  Clave privada = $\{d, n\}$

- Cifrado.

  Texto llano (M) dede ser $M < n$
  Texto cifrado $C = M^e \mod n$

- Descifrado.

  Texto llano C
  Texto cifrado $M = C^d(\mod n)$

¿Cómo es que se establece una relación entre las funciones hash y el cifrado de clave pública?

Es posible usar un algoritmo de cifrado de clave pública en un bloque de modo encadenado como una función hash. Si tú entonces tiras la clave privada, romper el hash sería tan difícil como leer el mensaje sin esa clave privada.

Si usamos RSA, podemos hacer lo siguiente: M es el mensaje para ser resumido (con una función hash), n el producto de dos primos p y q, y e es otro número muy grande primo relativo a $(p-1)(q-1)$, entonces la función hash sería:

$$H(M) = M^e \mod n$$

Aunque una solución más fácil sería usar un primo fuerte singular como el módulo p. Entonces:

$$H(M) = M^e \mod p$$

Romper este problema es tan difícil como encontrar el logaritmo discreto de e. El problema con este algoritmo es que es demasiado lento que algunos otros algoritmos hash.

1.5. Esquemas de Firma Digital

Hasta ahora, se ha explicado como cifrar mensajes de una forma simple y de otra un poco más complicada pero que implica que es más seguro nuestro texto cifrado. Aquí hablaré de cómo firmar esos mensajes, esto es, que aquella persona que pueda (queramos) descifrar ese mensaje también sepa (verifique) que somos nosotros quien le envía dicho mensaje con una especie de firma en ese mismo mensaje y que puede ser de dos tipos. Comienzo con una pequeña introducción, definición y ejemplos de algunos esquemas de firma para terminar con el esquema de firma digital RSA, que vuelvo a repetir, es el empleado en la norma.

1.5.1. Introducción

Una firma digital es una especie de firma electrónica (establece la relación entre los datos y la identidad del firmante) que garantiza la autenticidad e integridad y la posibilidad de detectar cualquier cambio ulterior. Por lo que sus principales aplicaciones son: autenticación, integridad de datos y la no repudiación.

Un *esquema de firma digital* consiste en definir un algoritmo para la generación de la clave (***Generación***), un algoritmo para el proceso de firmado (***Firmado***) y un algoritmo de verificación (***Verificación***).

El primer paso es generar una clave privada y una clave pública.

Ya teniendo la clave privada (aquella entidad que va a firmar los mensajes) toma un mensaje y al aplicarle cierto procedimiento, función, algoritmo, se genere una firma para el mensaje de alguna manera (tomando en cuenta al mensaje o no para el firmado) y se plasme en dicho mensaje, que posteriormente se podrá comprobar si es una firma válida.

Por último, de la clave privada se obtiene una clave pública que es la llave necesaria para que alguien que quiere verificar (en este caso) si la firma es real (válida) aplique también cierto procedimiento, función, algoritmo y pueda ver si la firma es

real.

Los esquemas de firma digital se clasifican en "firma digital con apéndice" y "firma digital con recuperación del mensaje".

Para este trabajo de tesis se enfatizará en "Esquemas de Firma Digital con apéndice".

1.5.2. Notación

Cuadro 1.5: Notación para mecanismos de firma digital

$\mathcal{M}$	Conjunto de elementos llamado el espacio de mensajes
$\mathcal{M}_{\mathcal{S}}$	Conjunto de elementos llamado el espacio del proceso de firma (espacio donde se firma)
$\mathcal{S}$	Conjunto de elementos llamado el espacio de firmas
R	Función de redundancia: mapeo 1 a 1 de $\mathcal{M}$ a $\mathcal{M}_{\mathcal{S}}$
$\mathcal{M}_R$	La imagen de R
R^{-1}	La inversa de R
$\mathcal{R}$	Conjunto de elementos llamado El espacio de índices para el proceso de firma
h	Función de un sentido con dominio $\mathcal{M}$
$\mathcal{M}_h$	La imagen de h (i.e., $h : \mathcal{M} \rightarrow \mathcal{M}_h$); $\mathcal{M}_h \subseteq \mathcal{M}_{\mathcal{S}}$ llamado el espacio de valores hash

Nota: Comentarios al cuadro 1.5

i) (*mensajes*) $\mathcal{M}$ es el conjunto de elementos a los cuales un firmante puede aplicarles una firma digital.

ii) (*espacio de firmado*) $\mathcal{M}_{\mathcal{S}}$ es el conjunto de elementos a los cuales se les aplican las transformaciones de firmado (las transformaciones de firmado de un esquema de firma digital con apéndice son diferentes a las transformaciones de firmado para esquemas de firma digital con recuperación del mensaje). Las transformaciones de firma no se aplican directamente sobre $\mathcal{M}$.

iii) (*espacio de firmas*) $\mathcal{S}$ es el conjunto de elementos asociados a los mensajes en $\mathcal{M}$. Estos elementos son usados para ligar la firma al mensaje.

iv) (*espacio de índices*) $\mathcal{R}$ es usado para identificar la transformación de firmado específica.

Un esquema de firma digital (con apéndice o de recuperación de mensaje) es probabilístico si $\mid \mathcal{R} \mid > 1$, de otra manera se dice que es determinístico.

1.5.3. Esquemas de Firma Digital con Apéndice

Un esquema de firma digital con apéndice es aquel que requiere introducir al mensaje como entrada para el algoritmo de verificación.

Ejemplo,
Una entidad, llamémosle Ana, crea una clave privada para firmar los mensajes, que otras entidades con su respectiva clave pública (de Ana) verifican.

- *Generación de la clave (Generación).*

  a. Ana selecciona una clave privada definida en el espacio $\mathcal{S}_A = \{S_{A,k} : k \in \mathcal{R}\}$ de transformaciones. Cada función $\mathcal{S}_A$ es una función 1 a 1 de $\mathcal{M}_h$ a $\mathcal{S}$ y es llamada la "transformación de firmado".

  b. $\mathcal{S}_A$ define el mapeo correspondiente V_A de $\mathcal{M}_h \times \mathcal{S}$ a {1,0} tal que

$$V_A(\bar{m}, s^*) = \begin{cases} 1, & \text{si } S_{A,k}(\bar{m}) = s^* \\ 0, & \text{de otra forma} \end{cases}$$

  Para todo $\bar{m} \in \mathcal{M}_h$, $s^* \in \mathcal{S}$; aqui, $\bar{m} = h(m)$ para $m \in \mathcal{M}$. V_A es llamada "transformación de verificación" y se construye de tal forma que se pueda calcular sin conocer la clave privada del firmante.

  c. La clave pública de Ana está en V_A. La clave privada de Ana está en el conjunto $\mathcal{S}_A$

- *Creación de la firma (Firmado).* Ana hace lo siguiente:

  a. Seleccionar un elemento $k \in \mathcal{R}$.

  b. calcular $\bar{m} = h(m)$ y $s^* = S_{A,k}(\bar{m})$.

  c. La firma de Ana para m es s^*.

- *Verificación de la Firma (Verificación).* Para verificar que la firma es de Ana, otra entidad, llamada Beto, hará lo siguiente:

a. Obtener la auténtica clave pública V_A de A.

b. calcular $\bar{m} = \hbar(m)$ y $u = V_A(\bar{m}, s^*)$.

c. Aceptar la firma *si y sólo si* $u = 1$.

Esquemas de firma digital con apéndice son El Gamal, DSA y Schnorr. A continuación describo algunos de estos.

- El Gamal: El esquema de firma digital "El Gamal" es un *mecanismo de firma aleatorio*. Genera firmas digitales con apéndice en mensajes binarios de longitud arbitraria y requiere funciones *hash*: $\{0,1\}^* \to \mathbb{Z}_p$ donde p es un número primo aleatorio grande; puede ser generalizado de manera directa para trabajar sobre un campo Z_p.

  - **Generación.** Cada entidad selecciona un campo Z_p; claves públicas y privadas. Ana hace lo siguiente:
    a) Selecciona un campo Z_p apropiado de orden n con generador α.
    b) Selecciona un entero a aleatorio y secreto, $1 \leq a \leq n-1$. calcula $y = \alpha^a$ elemento de Z_p.
    c) La clave pública de Ana es (α, y), junto con una descripción de cómo se multiplican los elementos en Z_p, la clave privada de Ana es a.
  - **Firmado.** Ana firma un mensaje binario m de longitud arbitraria. Beto puede verificar esta firma usando la clave pública de Ana.
    a) Selecciona un entero k aleatorio y secreto, $1 \leq k \leq n-1$, con $\gcd(k, n) = 1$.
    b) calcula $r = \alpha^k$ elemento de Z_p.
    c) calcula $k^{-1} \mod n$.
    d) calcula $h(m)$ y $h(r)$.
    e) calcula $s = k^{-1}\{h(m) - ah(r)\} \mod n$.
    f) La firma de Ana para m es el par (r, s).
  - **Verificación.** Para verificar la firma (r, s) de Ana en m, Beto hace:
    a) Obtiene la clave pública auténtica de Ana, (α, y).
    b) calcula $h(m)$ y $h(r)$.
    c) calcula $v_1 = y^{h(r)} \cdot r^s$.
    d) calcula $v_1 = \alpha^{h(m)}$.

e) Acepta la firma si y sólo si $v_1 = v_2$

Ejemplo (El Gamal Generalizado, firmas con pequeños parámetros artificiales)

Generación. Considera el campo F_{2^5} construido del polinomio irreducible $f(x) = x^5 + x^2 + 1$ sobre F_2. Los elementos de este campo son 31 quíntuplas binarias mostradas en el cuadro 1.6, junto con 00000. El elemento α es un generador para $F_{2^5}^*$. El orden de este campo G es $n = 31$. Dado $h : \{0,1\}^* \to \mathbb{Z}_{31}$ es una función hash.

Cuadro 1.6: Elementos del campo F_{2^5}

i	α^i	i	α^i	i	α^i	i	α^i
0	00001	8	01101	16	11011	24	11110
1	00010	9	11010	17	10011	25	11001
2	00100	10	10001	18	00011	26	10111
3	01001	11	00111	19	00110	27	01011
4	10000	12	01110	20	01100	28	10110
5	00101	13	11100	21	11000	29	01001
6	01010	14	11101	22	10101	30	10010
7	10100	15	11111	23	01111		

Ana selecciona la clave privada $a = 19$ y calcula $y = \alpha^a = (00010)^{19} = (00110)$. La clave pública de Ana es $(\alpha = (00010), y = (00110))$.

Firmado. Para firmar el mensaje $m = 10110101$, Ana selecciona un entero aleatorio $k = 24$, y calcula $r = \alpha^{24} = (11110)$ y $k^{-1} \mod 31 = 22$. Después, Ana calcula $h(m) = 16$ y $h(r) = 7$ (el valor hash para este ejemplo es inventado), y $s = 22 \cdot \{16 - (9)(7)\} \mod 31 = 30$. La firma de Ana para este mensaje m es $(r = (11110), s = 30)$.

verificación. Beto calcula $h(m) = 16, h(r) = 7$,
$v_1 = y^{h(r)} r^s = (00110)^7 \cdot (11110)^{30} = (11011)$

y
$v_2 = \alpha^{h(m)} = \alpha^1 6 = (11011)$. Beto acepta la firma sólo si $v_1 = v_2$.

- DSA: Algoritmo de Firma Digital (por sus siglas en inglés) fue propuesto en 1991 por el Instituto Nacional de Estándares y Tecnología de Estados Unidos.

El algoritmo es una variante del esquema "El Gamal". El mecanismo de firma requiere una función hash $h : \{0,1\}^* \rightarrow \mathbb{Z}_q$ para algún entero q. El algoritmo es como sigue:

- **Generación.** Cada entidad crea una clave pública y su correspondiente clave privada. Ana hace lo siguiente:
  a) Selecciona un número primo q tal que $2^{159} < q < 2^{160}$.
  b) Escoge t tal que $0 \leq t \leq 8$, y selecciona un número primo p donde $2^{511+64t} < p < 2^{512+64t}$, con la propiedad que q divide a $(p-1)$.
  c) (selecciona un generador α del campo de orden q en $\mathbb{Z}_p^*$.)
    c.1) Selecciona un elemento $g \in \mathbb{Z}_p^*$ y calcula $\alpha = g^{(p-1)/q} \mod p$.
    c.2) si $\alpha = 1$ entonces ir al paso c.1.
  d) Selecciona un entero aleatorio a tal que $1 \leq a \leq q-1$.
  e) calcula $y = \alpha^a \mod p$.
  f) La clave pública de Ana es (p, q, α, y); la clave privada de Ana es a.
- **Firmado.** Ana, con su clave privada, firma un mensaje m de longitud arbitraria. Ana hace lo siguiente:
  a) Selecciona un entero aleatorio secreto k, $0 < k < q$.
  b) calcula $r = (\alpha^k \mod p) \mod q$.
  c) calcula $k^{-1} \mod q$.
  d) calcula $s = k^{-1}\{h(m) + ar\} \mod q$.
  e) La firma de Ana para m es el par (r, s).
- **Verificación.** Beto puede verificar la firma de Ana (r, s) sobre m, usando la clave pública de Ana. Beto hace lo siguiente:
  a) Obtiene la verdadera clave pública de Ana, (p, q, α, y).
  b) Verifica que $0 < r < q$ y $0 < s < q$; si no, se rechaza la firma.
  c) calcula $w = s^{-1} \mod q$ y $h(m)$.
  d) calcula $u_1 = w \cdot h(m) \mod q$ y $u_2 = rw \mod q$.
  e) calcula $v = (\alpha^{u_1} y^{u_2} \mod p) \mod q$.
  f) Acepta la firma *si y sólo si* $v = r$.

Ejemplo.

Generación. Ana selecciona el número primo $p = 124540019$ y $q = 17389$ tal que q divide a $(p-1)$; $(p-1)/q = 7162$. *Ana* escoge un elemento aleatorio

$g = 110217528 \in \mathbb{Z}_p^*$ y calcula $\alpha = g^{7162} \mod p = 10083255$. Si $\alpha \neq 1$, α es un generador para el único campo de orden q en $\mathbb{Z}_p^*$. Después, *Ana* selecciona un entero aleatorio $a = 12496$ satisfaciendo $1 \leq a \leq q-1$, y calcula $y = \alpha^a \mod p = 10083255^{12496} \mod 124540019 = 119946265$. La clave pública de Ana es $(p = 124540019, q = 17389, \alpha = 10083255, y = 119946265)$, mientras que su clave privada es $a = 12496$.

Firmado. Para firmar m, Ana selecciona un entero aleatorio $k = 9557$, y calcula

$$\begin{aligned} r &= (\alpha^k \mod p) \mod q \\ &= (10083255^{9557} \mod 124540019) \mod 17389 \\ &= 27039929 \mod 17389 \\ &= 34 \end{aligned}$$

Después *Ana* calcula $k^{-1} \mod q = 7631$, $h(m) = 5246$ (valor hash inventado para este ejemplo), y por último
$s = (7631) \cdot \{5246 + (12496)(34)\} \mod q = 13049$. La firma para m es el par $(r = 34, s = 13049)$.

Verificación. Beto calcula $w = s^{-1} \mod q = 1799$,

$$\begin{aligned} u_1 &= w \cdot h(m) \mod q \\ &= (5246)(1799) \mod 17389 \\ &= 12716 \end{aligned}$$

y

$$\begin{aligned} u_2 &= rw \mod q \\ &= (34)(1799) \mod 17389 \\ &= 8999 \end{aligned}$$

después Beto calcula

$$\begin{aligned} v &= (\alpha^{u_1}\, y^{u_2} \mod p) \mod q \\ &= (10083255^{12716} \cdot 119946265^{8999} \mod 124540019) \mod 17389 \\ &= 27039929 \mod 17389 \\ &= 34 \end{aligned}$$

Si $v = r$, Beto acepta la firma.

- Schnorr: Es otra variante del esquema de firma digital "El Gamal" y como el "DSA", esta técnica emplea un campo de orden q en $\mathbb{Z}_p^*$, donde p es un

número primo grande. El método requiere una función hash $h : \{0,1\}^* \rightarrow \mathbb{Z}_q$. El algoritmo es de la siguiente forma:

- **Generación.** La generación de la clave para el esquema de firma Schnorr es el mismo que el del esquema DSA, excepto que no hay que construir el tamaño de p y q.
- **Firmado.** Para que Ana firme un mensaje m de longitud arbitraria hace lo siguiente:
  a) Selecciona un entero k aleatorio y secreto, tal que $1 \leq k \leq q-1$.
  b) calcula $r = \alpha^k \mod p$, $e = h(m \mid\mid r)$[2], y $s = ae + k \mod q$.
  c) La firma de Ana para m es el par (s, e).
- **Verificación.** Algún Beto puede verificar la firma (s, e) sobre m usando la clave pública de Ana, para esto, hace lo siguiente:
  a) Obtiene la clave pública auténtica de Ana, (p, q, α, y).
  b) calcula $V = \alpha^s y^{-e} \mod p$ y $e' = h(m \mid\mid v)$.
  c) Acepta la firma *si y sólo si* $e' = e$.

Ejemplo.

Generación. Ana selecciona los primos $p = 129841$ y $q = 541$; aquí, $(p-1)/q = 240$. Entonces Ana selecciona un entero aleatorio $g = 26346 \in \mathbb{Z}_p^*$ y calcula $\alpha = 26346^{240} \mod p = 26$. Si $\alpha \neq 1$, α genera el único subcampo de orden 541 en $\mathbb{Z}_p^*$. Después Ana selecciona la clave privada $a = 423$ y calcula $y = 26^{423} \mod p = 115917$. La clave pública de Ana es $(p = 129841, q = 541, \alpha = 26, y = 115917)$.

Firmado. Para firmar el mensaje $m = 11101101$, Ana selecciona un número aleatorio $k = 327$ tal que $1 \leq k \leq 540$, y calcula

$$\begin{aligned} r &= \alpha^k \mod p \\ &= 26^{327} \mod 115917 \\ &= 49375 \end{aligned}$$

y
$e = h(m \mid\mid r) = 155$ (valor hash inventado para este ejemplo).

[2] $||$: concatenación. $a \mid\mid b = ab$

Finalmente Ana calcula $s = 426 \cdot 155 + 327 \mod 541 = 431$. La firma para m es $(s = 431, e = 155)$.

Verificación. Beto calcula $v = 26^{431} \cdot 115917^{-155} \mod p = 49375$ y $e' = h(m \mid\mid v) = 155$. Beto acepta la firma *si* $e = e'$.

1.5.4. Esquemas de Firma Digital con Recuperación del Mensaje

Un esquema de firma digital con recuperación del mensaje es un esquema de firma digital para el cual no se necesita el mensaje para la verificación.

- *Generación.* Ana crea una clave privada y es usada para firmar los mensajes y su correspondiente clave pública que es usada por otras entidades para verificar la firma en dichos mensajes.

  a. Ana selecciona un conjunto $\mathcal{S}_A = \{S_{A,k} : k \in \mathcal{R}\}$ de transformaciones. Cada $\mathcal{S}_A$ es una función 1 a 1 de $\mathcal{M}_s$ a $\mathcal{S}$ y es llamada la "transformación de firmado"

  b. $\mathcal{S}_A$ define su correspondiente mapeo V_A con la propiedad que $V_A \circ \mathcal{S}_{A,k}$ es el mapeo identidad sobre $\mathcal{M}_\mathcal{S}$ para todo $k \in \mathcal{R}$. V_A es llamada la transformación de verificación y es construida de tal forma que pueda ser calculada sin conocer la clave privada del firmante.

  c. La clave pública de A es V_A; La clave de A es el conjunto $\mathcal{S}_A$

- *Firmado.* Ana produce una firma $s \in \mathcal{S}$ para un mensaje $m \in \mathcal{M}$, cual puede ser después verificada por Beto. El mensaje m es recuperado de s.

  a. Selecciona un elemento $k \in \mathcal{R}$

  b. calcula $\bar{m} = R(m)$ y $s^* = S_{A,k}(\bar{m})$. (R es una función redundante)

  c. La firma de Ana es s^*; esto está disponible para cualquier entidad que desee verificar la firma.

- *Verificación.* Beto hace:

  a. Obtiene la clave pública V_A de Ana.

  b. calcula $\bar{m} = V_A(s^*)$.

c. verifica que $\bar{m} \in \mathcal{M}_R$. (si $\bar{m} \notin \mathcal{M}_R$, entonces la firma se rechaza.)

d. recupera m de $\bar{m}$ calculando $R^{-1}(\bar{m})$.

Algunos esquemas de firma digital con recuperación de mensaje son RSA, Rabin, Nyberg-Rueppel.

- RSA: El esquema de firma RSA es un esquema de firma digital determinístico. El espacio $\mathcal{M}_{\mathcal{S}}$ (espacio del proceso de firma) y $\mathcal{S}$ (espacio de la firma) son ambos $\mathbb{Z}_n$.
  El algoritmo es de la siguiente manera:

  - **Generación.** Ana crea una clave pública RSA y una clave privada correspondiente. Para eso, hace lo siguiente:
    a) Genera dos grandes números primos aleatorios distintos p y q aproximadamente del mismo tamaño.
    b) calcula $n = pq$ y $\phi = (p-1)(q-1)$.
    c) Selecciona un entero aleatorio e, $1 < e < \phi$. tal que $mcd(e, \phi) = 1$.
    d) Usa el teorema de Euclides extendido para calcular un único entero d, $1 < d < \phi$, tal que $ed \equiv 1 \mod \phi$.
    e) La clave pública de Ana es (n, e); la clave privada de Ana es d.
  - **Firmado.** Ana firma un mensaje $m \in \mathcal{M}$ de la siguiente manera:
    a) calcula $\bar{m} = R(m)$, entero en $[0, n-1]$.
    b) calcula $s = \bar{m}^d \mod n$.
    c) La firma de Ana para m es s.
  - **Verificación.** Para verificar la clave de Ana y recuperar el mensaje, Beto hace:
    a) Obtiene la clave pública de Ana (n, e).
    b) calcula $m^* = s^e \mod n$.
    c) Verifica que $m^* = \bar{m}$, si no, rechaza la firma.
    d) Recupera $m = (\bar{m})^{-1}$.

    Ejemplo. RSA: Generación de la firma con parámetros artificiales pequeños

    Generación. Ana selecciona los primos $p = 7927, q = 6997$, y calcula $n = pq = 55465219$ y $\phi = 7926 \times 6996 = 55450296$. Ana escoge $e = 5$ y resuelve $ed = 5d \equiv 1(\mod 55450296)$, obteniendo $d = 44360237$. La

clave pública de Ana es $(n = 55465219, e = 5)$.

Firmado. Por simplicidad asumimos que $\mathcal{M} = \mathbb{Z}_n$. Para firmar el mensaje $m = 31229978$, Ana calcula

$$\begin{aligned} \bar{m} &= m \\ &= 31229978 \end{aligned}$$

y calcula la firma

$$\begin{aligned} s &= \bar{m}^d \mod n \\ &= 31229978^{44360237} \mod 55465219 \\ &= 30729435 \end{aligned}$$

Verificación. Beto calcula

$$\begin{aligned} m^* &= s^e \mod n \\ &= 30729435^5 \mod 55465219 \\ &= 31229978 \end{aligned}$$

Finalmente Beto acepta la firma si $m^* = \bar{m}$, y recupera $m = (\bar{m})^{-1} = 31229978$.

Firmas RSA en la Práctica

Como en la norma lo que se usa es RSA con md5 nos indica que RSA puede usarse tanto para cifrar un mensaje como para firmarlo y de otras maneras también que describo a continuación.

i) Problema de rebloquear.

Un uso sugerido del RSA es firmar un mensaje y después cifrar dicha firma. Uno debe estar al tanto del tamaño del mod implicado cuando se implementa este procedimiento. Suponga que Ana desea firmar y después cifrar un mensaje para Beto. Ssssuponga que (n_A, e_A) y (n_B, e_B) son las claves públicas de Ana y Beto respectivamente. Si $n_A > n_B$, entonces existe la posibilidad de que el mensaje no pueda ser recuperado por Beto.

Ejemplo Sea $n_A = 8388 \times 7499 = 62894113$, $e_A = 5$, y $d_A = 37726937$; y $n_B = 55465219$, $e_B = 5$, $d_B = 44360237$. Note que $n_A > n_B$. Suponga que

$m = 1368797$ es una mensaje para ser firmado bajo la clave privada de Ana y después cifrarlo usando la clave pública de Beto. A realiza lo siguiente:

1. $s = m^{d_A} \mod n_A = 1368797^{37726937} \mod 62894113 = 59847900$
2. $c = s^{e_B} \mod n_B = 59847900^5 \mod 55465219 = 38842235$

Para recuperar el mensaje y verificar la firma, Beto tiene que hacer lo siguiente:

1. $\hat{s} = c^{d_B} \mod n_B = 38842235^{44360237} \mod 54465219 = 4382681$
2. $\hat{m} = \hat{s}^{e_A} \mod n_A = 4382681^5 \mod 62894113 = 54383568$

Observe que $m \neq \hat{m}$. La razón para esto es que s es más grande que $\mod n_B$. Aquí, la probabilidad de que el problema ocurra es de $\frac{(n_A - n_B)}{n_A} \simeq 0{,}12$.

ii) Esquema de firma digital con apéndice RSA.

Sabemos que un esquema de firma digital de recuperación de mensaje puede ser modificado y convertirse en un esquema de firma digital con apéndice. Por ejemplo, si MD5 es usado para resumir mensajes de longitud arbitraria a cadenas de 128 bits, entonces el algoritmo de firmado y verificación para esquemas de recuperación del mensaje podría ser usado para firmar estos valores hash. Si n es un $\mod$ RSA de k-bits, entonces una función de redundancia apropiada R es requerida para asignar enteros de 128 bits a enteros de k-bits.

iii) Selección del paramétro.

Desde 1996 es recomendable que como mínimo el $\mod$ de RSA para firmar sea de 768 bits. Un $\mod$ de al menos 1024 bits es recomendado para firmas que requieran más tiempo de vida o que son críticas para la seguridad total de la red. Es prudente estar consiente del progreso en la factorización de enteros, y por consiguiente estar preparado para ajustar estos parámetros .

1.6. Integridad, Confidencialidad e Imputabilidad

En esta sección trato de observar que nos dice la NOM en cuanto a las palabras: *Integridad, Confidencialidad, Imputabilidad*, en cuanto a que si tienen el mismo significado de manera común o va más allá como términos informáticos o legales y

establecer en un capítulo subsecuente si se cumplen dichas definiciones.

Como estoy hablando acerca de que estemos seguros de que la información que queremos transmitir la entienda sólo quien nosotros queramos, y en caso de que si alguien la quiere modificar en el transcurso del envío también podamos darnos cuenta de eso, no es más que hablar de Integridad de datos, Confidencialidad e Imputabilidad. Lo menciono de varias maneras, en términos comunes, en términos técnicos y en lo que nos dice la norma para hallar una relación entre los dos últimos.

¿Qué nos dice acerca de estas palabras la norma? y ¿Significan lo mismo en un sentido común, digamos en la vida cotidiana? ...

Empecemos de atrás hacia adelante:

En cuanto a integridad podemos relacionarlo a muchas cosas por ejemplo: si uno como individuo es íntegro, que es como decir que es recto, intachable, como un ejemplo a seguir, podemos relacionarlo entonces a la integridad moral, a la integridad personal, a la integridad referencial.

Ésta última es una propiedad deseable en las bases de datos. Gracias a la integridad referencial se garantiza que una entidad (fila o registro) siempre se relaciona con otras entidades válidas, es decir, que existen en la base de datos, lo que implica que en todo momento dichos datos sean correctos, sin repeticiones innecesarias, datos perdidos y relaciones mal hechas. Ejemplo: Supongamos una base de datos con las entidades: Factura y Persona. Toda factura corresponde a una persona y solamente una. Implica que en todo momento dichos datos sean correctos, sin repeticiones innecesarias, datos perdidos y relaciones mal hechas. Supongamos que una persona se identifica por su atributo CURP (Clave Única de Registro de Población). También tendrá otros atributos como el nombre y la dirección. La entidad Factura debe tener un atributo CURP_cliente que identifique a quién pertenece la factura. Por sentido común es evidente que todo valor de CURP_cliente debe corresponder con algún valor existente del atributo CURP de la entidad Persona. Ésta es la idea intuitiva de la integridad referencial.

Existe también la integridad de datos que es más o menos lo que se refiere la NOM en cuanto a esta cuestión, se refiere a la corrección y completitud de los datos en una base de datos. Cuando los contenidos de una base de datos se modifican con sentencias Insert, Delete o Update, la integridad de los datos almacenados

puede perderse de muchas maneras diferentes. Por ejemplo: Pueden añadirse datos no válidos a la base de datos, tales como un pedido que especifica un producto no existente. Pueden modificarse datos existentes tomando un valor incorrecto, como por ejemplo si se reasigna un vendedor a una oficina no existente. Los cambios en la base de datos pueden perderse debido a un error del sistema o a un fallo en el suministro de energía. Los cambios pueden ser aplicados parcialmente, como por ejemplo si se añade un pedido de un producto sin ajustar la cantidad disponible para vender. Una de las funciones importantes de un DBMS (Data Base Management System) relacional es preservar la integridad de sus datos almacenados en la mayor medida posible.

Con respecto a confidencialidad creo que está claro, es mantener de manera segura algo que sólo queremos que vean ciertas personas, es decir que se mantenga confidencial para ciertas entidades.

En cuanto a imputabilidad es poder atribuirle a alguien la responsabilidad de un hecho reprobable. Esto es, como en forma común y corriente.

Ahora veamos que nos dice la norma en cuanto a esto.

La norma se establece en conformidad con lo dispuesto en el artículo 49 del Código de Comercio, que hasta su última modificación publicada en el Diario Oficial de la Federación de fecha 6 de junio del 2006 cita como sigue:

Los comerciantes están obligados a conservar por un plazo mínimo de diez años los originales de aquellas cartas, telegramas, mensajes de datos o cualesquiera otros documentos en que se consignen contratos, convenios o compromisos que den nacimiento a derechos y obligaciones. Para efectos de la conservación o presentación de originales, en el caso de mensajes de datos, se requerirá que la información se haya mantenido íntegra e inalterada a partir del momento en que se generó por primera vez en su forma definitiva y sea accesible para su ulterior consulta. La Secretaría de Comercio y Fomento Industrial emitirá la Norma Oficial Mexicana que establezca los requisitos que deberán observarse para la conservación de mensajes de datos.
Entonces la norma tiene que permitir que los mensajes que pasen por dicho proceso cumplan con:

- *INTEGRIDAD.* Que la información se presente sin manipulación alguna durante todo el tiempo que establece dicho contrato desde la fecha en que se elaboró hasta su culminación (o mínimo 10 años) pudiendo ser accesible por cualquier entidad.

- *CONFIDENCIALIDAD.* Se entiende que la información (contrato, convenio) es confidencial para las dos partes que se relacionan con dicha información, y en dado caso, por una tercera persona que es la oficialía de partes por si hay algún conflicto, refiriéndome a conflicto como alguna situación en la que alguna de las partes diga que el convenio no es el original.

- *IMPUTABILIDAD.* Aquí entra más que nada un objetivo de la criptografía: La No Repudiación, que es, a diferencia del punto anterior, que alguien no cumpla el contrato o lo cumpla mal, entonces que se atenga a las consecuencias estipuladas en dicho contrato o convenio y no pueda decir que no sabía o que él no lo hizo mal, siendo que si el convenio o contrato se hizo entre ambas partes no tiene por qué decir que no sabía.

1.7. Nociones de Seguridad

En esta sección se da una breve explicación en cuanto a qué debemos prestar atención para saber y entender que algoritmo (en este caso, un esquema de cifrado por ejemplo) sea seguro y no es mas que estudiar cuanto tiempo se tardará en resolver un problema con ese algoritmo dependiendo de los datos de entrada que es lo que se conoce como "complejidad de algoritmos" en informática y para eso doy una pequeña definición de algoritmo, sus características, y la notación empleada para terminar con unos ejemplos y observar cómo es que existen algoritmos sencillos pero nada eficientes (lo cual resulta ser costoso).

Uno de los algoritmos más antiguos conocidos es el algoritmo de Euclides. El término algoritmo proviene del matemático Muhammad ibn Musa al-Khwarizmi, que vivió aproximadamente entre los años 780 y 850 d.C. en la actual nación Iraní. El describió la realización de operaciones elementales en el sistema de numeración decimal. De al-Khwarizmi se obtuvo la derivación algoritmo.

- ¿Qué es un algoritmo? Una definición sería: conjunto finito de reglas que dan una secuencia de operaciones para resolver todas las instancias de un problema de un tipo dado. De forma más sencilla, podemos decir que un algoritmo es un conjunto de pasos que nos permite obtener un dato (o varios, como resultado). Además debe cumplir estas condiciones:

  - Finitud: el algoritmo debe acabar tras un número finito de pasos. Es más, es casi fundamental que sea en un número razonable de pasos.

- Definibilidad: el algoritmo debe definirse de forma precisa para cada paso, es decir, hay que evitar toda ambiguedad al definir cada paso. Puesto que el lenguaje humano es impreciso, los algoritmos se expresan mediante un lenguaje formal, ya sea matemático o de programación para una computadora.
- Entrada: el algoritmo tendrá cero o más entradas, es decir, cantidad de datos dada antes de empezar el algoritmo. Estos datos pertenecen además a conjuntos específicos de objetos. Por ejemplo, pueden ser cadenas de caractéres, enteros, naturales, fraccionarios, etc. Se trata siempre de cantidades representativas del mundo real expresadas de tal forma que sean aptas para su interpretación por la computadora.
- Salida: el algoritmo tiene una o más resultados, en relación con los problemas.
- Efectividad: se entiende por esto que una máquina (o individuo) sea capaz de realizar el algoritmo de modo exacto en un lapso de tiempo finito.

- Notación O-grande

  En la teoría de complejidad de cómputo, la notación O-grande se utiliza a menudo para describir cómo el tamaño de los datos de entrada afecta el uso de s del algoritmo, de recursos de cómputo (tiempo generalmente en marcha o memoria). También se llama "notación Big Oh", notación (λ) landau, y notación asintótica. La notación O-grande también se utiliza en muchos otros campos científicos y matemáticos para proporcionar valoraciones similares.

  Una definición rigurosa de esta notación es la siguiente: Una función g pertenece a $O(f)$ si y sólo si existen las constantes c_0 y N_0 tales que: $\mid g \mid \leq \mid c_0 f \mid$, para todo $N \geq N_0$.
  En general, el tiempo de ejecución es proporcional, esto es, multiplica por una constante a alguno de los tiempos de ejecución anteriormente propuestos, además de la suma de algunos términos más pequeños. Así, un algoritmo cuyo tiempo de ejecución sea $T = 3N^2 + 6N$ se puede considerar proporcional a N^2. En este caso se diría que el algoritmo es del orden de N^2, y se escribe $O(N^2)$. Por ejemplo, los grafos definidos por matriz de adyacencia ocupan un espacio $O(N^2)$, siendo N el número de vértices de éste.

  La notación O-grande ignora los factores constantes, es decir, ignora si se hace una mejor o peor implementación del algoritmo, además de ser independiente

de los datos de entrada del algoritmo. Es decir, la utilidad de aplicar esta notación a un algoritmo es encontrar un límite superior del tiempo de ejecución, es decir, el peor caso. A veces ocurre que no hay que prestar demasiada atención a esto. Conviene diferenciar entre el peor caso y el esperado. Por ejemplo, el tiempo de ejecución del algoritmo Quick Sort (oredenamiento rápido) es un algoritmo basado en la técnica de divide y vencerás, que permite, en promedio, ordenar n elementos en un tiempo proporcional a $O(NlogN)$. Esta es la técnica de ordenamiento más rápida conocida.
A diferencia del algoritmo quick sort está el "estupid sort", que es probablemente el más sencillo de los algoritmos de ordenamiento, utilizado para reorganizar valores en un arreglo en orden ascendente o descendente. Su nombre se refiere al hecho de que su extrema sencillez repercute en su baja eficiencia, es decir, su rendimiento es pobre en términos de tiempo de ejecución.

Stupid sort nunca reitera en ordenar los datos en el mejor caso (es decir, cuando los datos ya estén en orden) con un tiempo de ejecución $O(N \times N!)$, donde N es el número de elementos en el arreglo. Adicionalmente, su forma no recursiva ordena los datos en su lugar , por lo cual no se necesita memoria extra para guardar los datos temporales.

- Clasificación de algoritmos.

  - Algoritmo determinista: en cada paso del algoritmo se determina de forma única el siguiente paso.
  - Algoritmo no determinista: deben decidir en cada paso de la ejecución entre varias alternativas y agotarlas todas antes de encontrar la solución.

    Todo algoritmo tiene una serie de características, entre otras que requiere una serie de recursos, algo que es fundamental considerar a la hora de implementarlos en una máquina. Estos recursos son principalmente:

    - El tiempo: período transcurrido entre el inicio y la finalización del algoritmo.
    - La memoria: la cantidad de espacio en disco que necesita el algoritmo para su ejecución.

    Obviamente, la capacidad y el diseño de la máquina pueden afectar al diseño del algoritmo. En general, la mayoría de los problemas tienen un parámetro de entrada que es el número de datos que hay que tratar, esto es, N. La cantidad de recursos del algoritmo es tratada como una función de N. De esta manera puede establecerse un tiempo de ejecución del algoritmo que suele ser proporcional a una de las siguientes funciones:

- 1 : Tiempo de ejecución constante. Significa que la mayoría de las instrucciones se ejecutan una vez o muy pocas.
- $\log N$: Tiempo de ejecución logarítmico. Se puede considerar como una gran constante. La base del logaritmo (en informática la más común es la base 2) cambia la constante, pero no demasiado. El programa es más lento cuanto más crezca N, pero es inapreciable, pues $\log N$ no se duplica hasta que N llegue a N^2.
- N : Tiempo de ejecución lineal. Un caso en el que N valga 40, tardará el doble que otro en que N valga 20. Un ejemplo sería un algoritmo que lee N números enteros y devuelve la media aritmética.
- $N \log N$: El tiempo de ejecución es $N \log N$. Es común encontrarlo en algoritmos como Quick Sort y otros del estilo divide y vencerás. Si N se duplica, el tiempo de ejecución es ligeramente mayor del doble.
- N^2 : Tiempo de ejecución cuadrático. Suele ser habitual cuando se tratan pares de elementos de datos, como por ejemplo un bucle anidado doble. Si N se duplica, el tiempo de ejecución aumenta cuatro veces. El peor caso de entrada del algoritmo Quick Sort se ejecuta en este tiempo.
- N^3 : Tiempo de ejecución cúbico. Como ejemplo se puede dar el de un bucle anidado triple. Si N se duplica, el tiempo de ejecución se multiplica por ocho.
- 2^N : Tiempo de ejecución exponencial. No suelen ser muy útiles en la práctica por el elevadísimo tiempo de ejecución. El problema de la mochila resuelto por un algoritmo de fuerza bruta -simple vuelta atrás- es un ejemplo. Si N se duplica, el tiempo de ejecución se eleva al cuadrado.

- Algoritmos polinomiales: aquellos que son proporcionales a N^k. Son en general factibles.
- Algoritmos exponenciales: aquellos que son proporcionales a k^N. En general son infactibles salvo un tamaño de entrada muy reducido.

- Clasificación de problemas.

  Los problemas matemáticos se pueden dividir en primera instancia en dos grupos:

  - Problemas indecidibles: aquellos que no se pueden resolver mediante un algoritmo.

- Problemas decidibles: aquellos que cuentan al menos con un algoritmo para su cómputo.

  Sin embargo, que un problema sea decidible no implica que se pueda encontrar su solución, pues muchos problemas que disponen de algoritmos para su resolución son inabordables para una computadora por el elevado número de operaciones que hay que realizar para resolverlos. Esto permite separar los problemas decidibles en dos:

- Intratables: aquellos para los que no es factible obtener su solución.
- Tratables: aquellos para los que existe al menos un algoritmo capaz de resolverlo en un tiempo razonable.

Los problemas pueden clasificarse también atendiendo a su complejidad. Aquellos problemas para los que se conoce un algoritmo polinomial que los resuelve se denominan clase P. Los algoritmos que los resuelven son deterministas. Para otros problemas, sus mejores algoritmos conocidos son no deterministas. Esta clase de problemas se denomina clase NP. Por tanto, los problemas de la clase P son un subconjunto de los de la clase NP, pues sólo cuentan con una alternativa en cada paso.

- La Clase P.

  Un problema es P si se tiene un algoritmo que lo soluciona en tiempo polinomial, por ejempolo, encontrar el *mcd* de dos o más números. Existen muchos problemas P que parecen *difíciles*. Ejemplos de problemas Clase P:

  - Primalidad (VP): Dado un entero positivo n, decidir si n es un número primo o no.
  - Progamación Lineal (PL): Dada una matriz A_{mxn}, dos vectores b_m y c_n y un entero k, decidir si existe un vector de racionales x_n tal que $A_x \leq b$ y $cx \geq k$.

- La Clase NP.

  Un problema es NP si es un problema de decisión en el cual dada la respuesta se puede verificar si es acertada o no en tiempo polinomial y sin embargo resolver el problema no es posible en tiempo polinomial. Por ejemplo, obtener los factores primos de un número en geneal no es fácil. Un ejemplo de problema NP es el de la "Búsqueda Tabú":

  - Usa una "memoria" para guiar la búsqueda.

- Algunas soluciones examinadas recientemente son "memorizadas" y se vuelven tabú (prohibidas) al tomar decisiones acerca del siguiente punto de búsqueda.
- Es determinística, aunque se le pueden agregar elementos probabilísticos.

El algoritmo de búsqueda A* (A Estrella) se clasifica dentro de los algoritmos de búsqueda. El algoritmo encuentra, siempre y cuando se cumplan unas determinadas condiciones, el camino de menor costo entre un nodo origen y uno objetivo. El algoritmo A* utiliza una función de evaluación $f = g + h$, donde h representa el valor heurístico del nodo a evaluar desde el actual, N, hasta el final, y g, el costo real del camino recorrido para llegar a dicho nodo, N. A* mantiene dos estructuras de datos auxiliares, que podemos denominar abiertos, implementado como una cola de prioridad (ordenada por el valor f de cada nodo), y cerrados, donde se guarda la información de los nodos que ya han sido visitados. En cada paso del algoritmo, se expande el nodo que esté primero en abiertos, y en caso de que no sea un nodo objetivo, calcula la f de todos sus hijos, los inserta en abiertos, y pasa el nodo evaluado a cerrados.

El algoritmo es una combinación entre búsquedas del tipo primero en anchura con primero en profundidad: mientras que h tiende a primero en profundidad, g tiende a primero en anchura. De este modo, se cambia de camino de búsqueda cada vez que existen nodos más prometedores.
La complejidad computacional del algoritmo está íntimamente relacionada con la calidad de la heurística que se utilice en el problema. En el peor caso, con una heurística de pésima calidad, la complejidad será exponencial, mientras que en el mejor caso, con una buena h, el algoritmo se ejecutará en tiempo lineal. Para que esto último suceda, se debe cumplir que

$$h(x) \leq g(y) - g(x) + h(y)$$

Donde h* es una heurística óptima para el problema, como por ejemplo, el costo real de alcanzar el objetivo.

Capítulo 2

Información Electrónica

En este capítulo se define qué es la información electrónica y si es lo mismo procesar dicha información en los diferentes sistemas operativos (mejor dicho plataformas) que existen o cuál es la diferencia. Qué debemos tomar en cuenta al trabajar nuestra información en la computadora, etc. Es importante estudiar esto porque veremos si la aplicación de la norma es sólo para ciertas plataformas o se puede decir que es multiplataforma. En la primera sección vemos que es necesario utilizar un tipo de codificación (se ve la definición también) para el caso de implementar la norma en diferentes plataformas y no estar haciendo cosas adicionales después de dicha implementación para migrar la información de una plataforma a otra, continúando con la descripción de ASN.1, formato que utiliza la norma y que apoyo, y terminar con la descripción de algunos estándares internacionales que tienen que ver con este tema: Constancias digitales, criptografía de clave pública, etc...

En la actualidad es muy difícil mantener o hacer todos nuestros documentos ya sean texto, imagen, audio, etc. físicamente ya que si queremos o necesitamos compartirlos con alguien más sería muy difícil hacerlo en poco tiempo y de manera económica. Por ejemplo, como compartir con tus familiares que viven a cientos o miles de kilómetros las fotos de alguna ocasión especial sin que se pierdan esas fotos y sin que pase mucho tiempo desde esa ocasión para que ellos puedan ver lo que pasó por medio de esas fotografías. Ahora, lo más fácil es tomar las fotos con una cámara digital y enviarlas por medio del correo electrónico, o simplemente si se tomaron con una cámara analógica escanearlas y enviarlas también por correo electrónico. Entonces ahí está lo que nos interesa: "electrónico" será entonces cualquier tipo de información pasada a un medio electrónico ya sea texto, secuencias numéricas o imagenes, manejada como bloques de bits.

A menudo los algoritmos requieren una organización bastante compleja de los datos, y es por tanto necesario un estudio previo de las estructuras de datos fundamentales. Dichas estructuras pueden implementarse de diferentes maneras, y es más, existen algoritmos para implementar dichas estructuras. El uso de estructuras de datos adecuadas pueden hacer trivial el diseño de un algoritmo, o un algoritmo muy complejo puede usar estructuras de datos muy simples.

2.1. Necesidad de una Codificación

En esta sección se consideran los puntos necesarios para poder observar, procesar y si es necesario migrar nuestra información en diferentes plataformas estudiando las normas oficiales que existen y también ver si están claras las que se utilizan en la norma.

Como sabemos, toda la información que manejan las computadoras está escrita (por decirlo de alguna manera) en ceros y unos solamente, que es como la traducción del lenguaje común al lenguaje que manejan las máquinas, y que además es muy diferente la estructura de una plataforma a otra.

Como definición formal podría escribir como sigue: La codificación de caracteres es el método que permite convertir un caracter de un lenguaje natural (alfabeto o silabario) en un símbolo en otro sistema de representación, como un número o una secuencia de pulsos eléctricos en un sistema electrónico, aplicando normas o reglas de codificación.

Existen muchas normas de codificación, algunas comunes y otras no tanto, describo algunas a continuación:

- ASCII. Por estar íntimamente ligado al octeto (y por consiguiente a los enteros que van del 0 al 127) el problema que presenta es que no puede codificar mś que 128 símbolos diferentes (128 es el número total de diferentes configuraciones que se pueden conseguir con 7 dígitos binarios o digitales 0000000, 0000001, ..., 1111111), usando el octavo dígito de cada octeto (bit o dígito de paridad) para detectar algún error de transmisión). Un conjunto de 128 símbolos es suficiente para incluir mayúsculas y minúsculas del abecedario inglés, además de cifras, puntuación, y algunos çaracteres de control"(por ejemplo, uno que instruye a una impresora que pase a la hoja siguiente), pero el ASCII no incluye ni los

caracteres acentuados ni el comienzo de interrogación que se usa en castellano, ni tantos otros símbolos (matemáticos, letras griegas, ...) que son necesarios en muchos contextos.

- ASCII Extendido. Debido a las limitaciones del ASCII se definieron varios códigos de caracteres de 8 bits, entre ellos el ASCII extendido. Sin embargo, el problema de estos códigos de 8 bits es que cada uno de ellos se define para un conjunto de lenguas con escrituras semejantes y por tanto no dan una solución unificada a la codificación de todas las lenguas del mundo. Es decir, no son suficientes 8 bits para codificar todos los alfabetos y escrituras del mundo.

- Unicode.

  Como solución a estos problemas, desde 1991 se ha acordado internacionalmente utilizar la norma Unicode, que es una gran tabla, que en la actualidad asigna un código a cada uno de los más de cincuenta mil símbolos, los cuales abarcan todos los alfabetos europeos, ideogramas chinos, japoneses, coreanos, muchas otras formas de escritura, y más de un millar de símbolos especiales.

- UTF-8. Es una norma de transmisión utilizada junto con la norma de codificación Unicode. Utilizadas en conjunto, funcionan de la siguiente manera:

  - Unicode asigna los enteros del 0 al 127 (un total de 128) a exactamente los mismos caracteres que ASCII
  - UTF-8 empaqueta cualquier entero del 0 al 127 en un octeto .ª la antigua"pero con el octavo dígito siempre en cero, ya que actualmente el bit de paridad no se utiliza mas para detección de errores
  - Además, como la tabla de Unicode es tan grande, la mayoría de sus símbolos están asignados a enteros mayores que 127 (códigos que, en consecuencia, necesitan más que 7 dígitos para su representación binaria). En todos esos casos, UTF-8 envía el comienzo de la representación binaria del código en cuestión en un primer octeto con dígito de paridad = 1
  - El receptor de este mensaje, interpreta este dígito en 1 como señal de que lo que está siendo transmitido es un código que no cabe en 7 dígitos binarios; y por tanto determina que el símbolo no lo va a conocer mientras no lea el siguiente octeto, y tal vez el que sigue. En el peor de los casos, quizás se haga necesario leer seis octetos consecutivos para determinar un código alto.

Estas normas establecen que código representa dicho caracter pero no quiere decir que si en dos plataformas distintas, dígamos UNIX y Windows, usan UTF-8, se ordenarán de la misma forma, es decir, tal vez en la primera plataforma un número de cuatro dígitos se escribe como $d_1d_2d_3d_4$ y en la segunda la forma es $d_4d_3d_2d_1$, esto es lo que se conoce normalmente como ENDIAN, que es el orden que ocupa en memoria un byte (a veces un bit) para representar cierto tipo de datos. Generalmente hablando, ENDIAN es un atributo particular de un formato de representación - cuál byte sería usado en la dirección de memoria más baja, es decir, coloca las cosas en orden de magnitud.
La mayoría de los procesadores modernos han convenido en ordenar cada bit dentro de bytes individuales (no siempre es el caso). Esto significa que el valor de un byte individual puede ser leído igual en cualquier computadora como en la que se envió.

Los enteros usualmente son almacenados como secuencia de bytes, entonces el valor codificado puede ser obtenido por una simple codificación. Los dos más comunes son:

- *little endian*: incremento en la significancia numérica con incremento en la dirección de memoria.
- *big endian*: conocido como el más grande primero, ordena los bytes del más significante al menos.

Los procesadores x86 de intel usan el formato little endian. Los procesadores motorola han usado generalmente big endian. Las powerPC (como las mac de Apple) adoptan big endian. La figura 2.1 muestra como se ordenan los bytes dependiendo del formato endian utilizado.

En este sentido es que es necesario establecer un estándar en qué norma (s) y estructura se tiene que llevar a cabo la codificación para el cifrado de mensajes. Estableciendo cómo ordenar cada bit, en qué formato guardar dicha información, etc.

2.2. Notación ASN.1

En esta sección comento de qué trata la Notación ASN.1, que es la utilizada en la NOM-151-SCFI, y si es que realmente es necesario este tipo de notación o tal vez no es suficiente en cuanto a todos los campos que requiere la norma si es que queremos que tenga la validez requerida y por qué no que sea una norma que se puede aplicar en documentos internacionales.

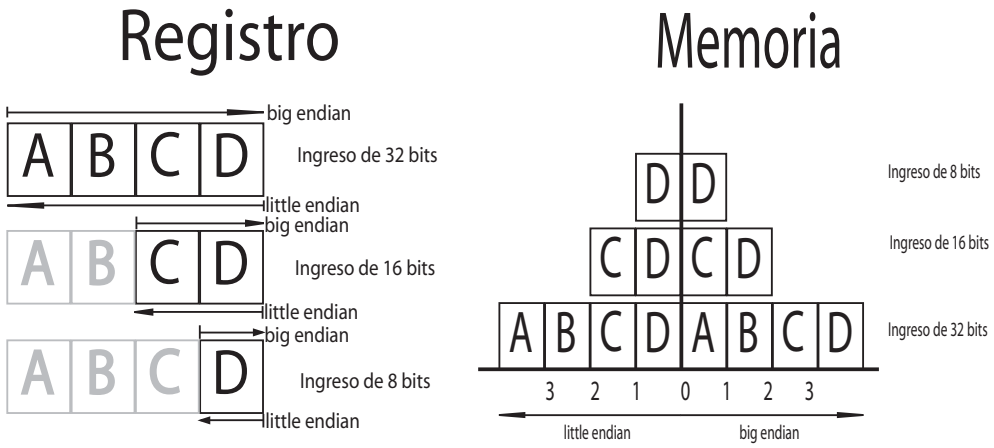

Figura 2.1: Esquema de la codificación ENDIAN

Abstract Syntax Notation One (notación de sintaxis abstracta número 1, ASN.1) es una norma para representar datos independientemente de la máquina que se esté usando y sus formas de representación internas. Es un protocolo de nivel de presentación en el modelo OSI. El protocolo SNMP usa el ASN.1 para representar sus objetos gestionables.
ASN.1 no tiene relación con ningún estándar, método de codificación, lenguaje de programación o plataforma de hardware en concreto. Se trata simplemente de un lenguaje para definir estándares. O dicho de otro modo: los estándares se escriben en ASN.1.

ASN.1 usa la notación de Backus-Naur (BNF) para describir la forma en que la información es almacenada.

Tipos de datos

Los tipos de datos se clasifican según si son simples (primitivo), compuestos (construido) o una mera etiqueta (definido).

Tipos primitivos

Los tipos primitivos son escalares, es decir, almacenan un único valor, como por ejemplo una cadena de texto o un entero. Los tipos primitivos más importantes son:

- El tipo INTEGER se usa para representar números enteros.
- El tipo OCTET STRING almacena una secuencia de bytes. De él se derivan tres subtipos:
  - DisplayString (para cadenas de caracteres ASCII)
  - OctetBitString (para cadenas de bits mayores de 32)
  - PhysAddress (para representar direcciones del nivel de enlace).
- El tipo OBJECT IDENTIFIER (OID) para representar los identificadores de los objetos, es decir, la posición de un objeto dentro del árbol de la MIB (Management Information Base).
- El tipo BOOLEAN para valores que sólo pueden ser verdadero o falso.
- El tipo NULL para representar la ausencia de valor.

Tipos construidos

Los tipos construidos son tipos compuestos. Se usan para crear arrays y tablas. Los más importantes son:

- El tipo SEQUENCE es una estructura de datos, es decir, una lista ordenada de tipos de datos diferentes. Es el tipo que se usa para almacenar una fila de una tabla.
- El tipo SEQUENCE OF es una lista ordenada de tipos de datos iguales. Es similar al tipo "SEQUENCE", excepto que todos los tipos han de ser iguales. Es el tipo usado en tablas para almacenar todas las filas.
- El tipo SET es equivalente al "SEQUENCE", pero la lista no está ordenada.
- El tipo SET OF es equivalente al "SEQUENCE OF" pero la lista no está ordenada.
- El tipo CHOICE es un tipo de datos en el que los valores son fijos y hay que elegir uno de ellos. Por ejemplo, una variable "Estado" del tipo "CHOICE" podría tomar los valores "pagado", "En reposo" o "Con errores".

Tipos definidos

Son tipos derivados de los anteriores pero con un nombre más descriptivo. Los más importantes son:

- *IpAddress* sirve para almacenar una dirección IP. Son 4 bytes y se define como "OCTET STRING (SIZE (4))".
- *Counter* representa un contador que únicamente puede incrementar su valor y que cuando llega a su valor máximo, vuelve a cero. Se define como un entero sin signo, que sólo puede tomar valores positivos y el cero.
- *Gauge* es como un indicador de nivel. Es un valor que puede incrementar o decrementar. Puede llevar asociados acciones a tomar en caso de que se superen unos umbrales. Un ejemplo es un medidor del ancho de banda medido en una determinada interfaz. Está definido como un entero de 32 bits.
- *TimeTicks* es un tipo de datos usado para medir tiempos. Indica el número de centésimas de segundo que han transcurrido desde un determinado evento temporal. Es un entero de 32 bits.

Macros de ASN.1

Podemos crear nuevos tipos de ASN.1 usando macros.

Clases de datos

Existen cuatro clases de datos, que etiquetan al resto de tipos de datos.

- Universal: Para tipos de datos generales, como "Boolean", "Integer" y "Real".
- Específica al contexto: Definidos para el contexto local en que se usan estos tipos.
- Aplicación: Definidos para la aplicación específica.
- Privada: Definidos por el usuario.

Codificación

La sintaxis de transferencia especifica cómo se codifican los distintos tipos de datos. Define la forma de codificar en el transmisor y de decodificar en el receptor los valores expresados con ASN.1.

Este estándar no define cómo se han de codificar esos datos, sino que es una sintaxis abstracta para indicar el significado de los datos. Para la codificación de los datos se usan otras normas como: BER (basic encoding rules) (BER - X.209), CER (canonical encoding rules), DER (distinguished encoding rules), PER (packed encoding rules) y XER (XML encoding rules).

Estándares

Estándares que describen la notación ASN.1:

- ITU-T Rec. X.680 — ISO/IEC 8824-1
- ITU-T Rec. X.681 — ISO/IEC 8824-2
- ITU-T Rec. X.682 — ISO/IEC 8824-3
- ITU-T Rec. X.683 — ISO/IEC 8824-4

Estándares que describen las reglas de codificación de ASN.1:

- ITU-T Rec. X.690 — ISO/IEC 8825-1 (BER, CER y DER)
- ITU-T Rec. X.691 — ISO/IEC 8825-2 (PER)
- ITU-T Rec. X.693 — ISO/IEC 8825-4 (XER)
- ITU-T Rec. X.694 — ISO/IEC 8825-5 (mapeado XSD)
- RFC 3641 (GSER)

Para el caso de la norma la definición de ASN.1 es como sigue:

```
NCI-NOM-000-SECOFI DEFINITIONS ::=
BEGIN

NombreOP ::= PrintableString
```

```
TipoOP ::= OBJECT IDENTIFIER

EmisorOP ::= IdentificadorUsuario

IdUsuarioOP ::= IdentificadorUsuario

md5                   OBJECT IDENTIFIER ::= { 1 2 840 113549 2 5}
rsaEncryption         OBJECT IDENTIFIER ::= { 1 2 840 113549 1 1 1}
md5WithRSAEncryption OBJECT IDENTIFIER ::= { 1 2 840 113549 1 1 4}
```

La primer línea de la primera parte de esta definición especifica que significan ciertos campos en las normas de SECOFI (Secretaría de Comercio y Fomento Industrial) donde los campos es lo que se encuentra del lado izquierdo del símbolo `::=` y del lado derecho qué representa. Por ejemplo, el campo `NombreOP` (nombre del operador, ya sea del sistema de información que guarda los archivos o del operador de la oficialía de partes que emite la constancia) se representa como una cadena de bits, es decir, puede ser el nombre real de la persona (operador). El campo `TipoOP` es una cadena de bits que indica que tipo de persona está emitiendo ya sea el archivo, expediente, etc. puede ser, por ejemplo, `nomIdentificacion2` que se refiere al nombre de una persona moral y el otro campo (`IdUsuarioOP`) que es con qué documento se identifica, puede ser cédula fiscal, pasaporte, etc. Estas claves se encuentran más adelante bajo el encabezado "Identificadores de objeto a utilizar para identificación de usuarios".

Identificador de objeto a utilizar para las Normas Oficiales Mexicanas

```
mex OBJECT IDENTIFIER ::= { 2 37 137 }

nom OBJECT IDENTIFIER ::= { mex 179 }
```

Creo que lo anterior queda claro. La primer línea dice que es una norma oficial mexicana y la segunda indica qué norma es.

Identificadores de objeto a utilizar para tipos de archivos, los cuales se pueden convertir a archivos parciales.

```
nomArchivos OBJECT IDENTIFIER ::= {nom 197}

nomABinario OBJECT IDENTIFIER ::= {nomArchivos 1}

nomATexto   OBJECT IDENTIFIER ::= {nomArchivos 2} -- Extensiones
nomAT-TXT   OBJECT IDENTIFIER ::= {nomATexto 1}   -- .txt
nomAT-TEX   OBJECT IDENTIFIER ::= {nomATexto 2}   -- .tex
nomAT-PS    OBJECT IDENTIFIER ::= {nomATexto 3}   -- .ps
nomAT-HTML  OBJECT IDENTIFIER ::= {nomATexto 4}   -- .htm .html

nomAAudio   OBJECT IDENTIFIER ::= {nomArchivos 3} -- Extensiones
nomAA-AU    OBJECT IDENTIFIER ::= {nomAAudio 1}   -- .au
nomAA-WAV   OBJECT IDENTIFIER ::= {nomAAudio 2}   -- .wav
nomAA-MP3   OBJECT IDENTIFIER ::= {nomAAudio 3}   -- .mp3
nomAA-RAM   OBJECT IDENTIFIER ::= {nomAAudio 4}   -- .ram

nomAVideo   OBJECT IDENTIFIER ::= {nomArchivos 4} -- Extensiones
nomAV-MPEG  OBJECT IDENTIFIER ::= {nomAVideo 1}   -- .mpg .mpeg
nomAV-DVD   OBJECT IDENTIFIER ::= {nomAVideo 2}   -- PENDIENTE
nomAV-MOV   OBJECT IDENTIFIER ::= {nomAVideo 3}   -- .mov .qt .movie
                                                  -- .moov
nomAImagen  OBJECT IDENTIFIER ::= {nomArchivos 5} -- Extensiones
nomAI-JPEG  OBJECT IDENTIFIER ::= {nomAImagen 1}  -- .jpeg .jpg
nomAI-GIF   OBJECT IDENTIFIER ::= {nomAImagen 2}  -- .gif
nomAI-BMP   OBJECT IDENTIFIER ::= {nomAImagen 3}  -- .bmp

nomAMicrosoft OBJECT IDENTIFIER ::= {nomArchivos 6}   -- Extensiones
nomAM-WORD    OBJECT IDENTIFIER ::= {nomAMicrosoft 1} -- .doc
nomAM-W6      OBJECT IDENTIFIER ::= {nomAM-WORD 1}
nomAM-W97     OBJECT IDENTIFIER ::= {nomAM-WORD 2}
nomAM-W2000   OBJECT IDENTIFIER ::= {nomAM-WORD 3}
nomAM-PPT     OBJECT IDENTIFIER ::= {nomAMicrosoft 2} -- .ppt
nomAM-EXCEL   OBJECT IDENTIFIER ::= {nomAMicrosoft 3} -- .xls
nomAM-OUTLOOK OBJECT IDENTIFIER ::= {nomAMicrosoft 4} -- .pst
nomAM-ACCESS  OBJECT IDENTIFIER ::= {nomAMicrosoft 5} -- .mdb
```

Aunque se maneja la mayoría de arhivos, se puede observar que prevalecen los de “microsoft office” y aunque en distintas plataformas a windows se puedan crear

archivos con las mismas extensiones, ¿no será un costo adicional para el usuario si no puede con facilidad manejar estas extensiones ?.

Identificadores de objeto a utilizar para identificación de usuarios y de los campos, con pocos o muchos datos, que se deben de existir en la norma con todos los datos (necesariamente) que estan aquí a menos que se especifique lo contrario (OPTIONAL).

```
nomIdentificacion OBJECT IDENTIFIER ::= {nom 373}

nomIPersonaFisica  OBJECT IDENTIFIER ::= {nomIdentificacion 1}
nomIF-NOMBRE       OBJECT IDENTIFIER ::= {nomIPersonaFisica 1}
nomIF-IFE          OBJECT IDENTIFIER ::= {nomIPersonaFisica 2}
nomIF-CURP         OBJECT IDENTIFIER ::= {nomIPersonaFisica 3}
nomIF-PASAPORTE    OBJECT IDENTIFIER ::= {nomIPersonaFisica 4}
nomIF-CEDULAFISCAL OBJECT IDENTIFIER ::= {nomIPersonaFisica 5}

nomIPersonaMoral   OBJECT IDENTIFIER ::= {nomIdentificacion 2}
nomIM-NOMBRE       OBJECT IDENTIFIER ::= {nomIPersonaMoral 1}
nomIM-CURP         OBJECT IDENTIFIER ::= {nomIPersonaMoral 2}
nomIM-CEDULAFISCAL OBJECT IDENTIFIER ::= {nomIPersonaMoral 3}

NombrePersonaFisica ::= SEQUENCE {
 nombreIdP        PrintableString,
 apellido1IdP     PrintableString,
 apellido2IdP     PrintableString
 }

IdentificadorPersona ::= SEQUENCE {
 nombreIdP        NombrePersonaFisica
 tipoIdP          OBJECT IDENTIFIER, ,
 contenidoIdP     PrintableString
}

NumeroCertificado ::= PrintableString

IdentificadorUsuario ::= SEQUENCE {
 personaFisicaMoral   OBJECT IDENTIFIER,
```

```
 nombreRazonSocialIdU CHOICE {NombrePersonaFisica, PrintableString},
 tipoIdU              OBJECT IDENTIFIER
 contenidoIdU         PrintableString, ,
 numeroCertificadoU   NumeroCertificado,
 representanteIdU     IdentificadorPersona OPTIONAL  -- Este campo es
                                          -- para el representante legal
}

AlgorithmIdentifier ::= SEQUENCE
 algorithm OBJECT IDENTIFIER,
 NULL
}

NombreConstanciaOP ::= PrintableString

FirmaUsuarioOP ::= SEQUENCE {
 algoritmoFirma  AlgorithmIdentifier,
 firma     BIT STRING
}

FirmaConstanciaOP ::= SEQUENCE {
 algoritmoFirma  AlgorithmIdentifier,
 firma           BIT STRING
}

ResumenOP ::= SEQUENCE {
 algoritmoresumen  AlgorithmIdentifier
 resumen           BIT STRING
}

Folio-UsuarioOP ::= INTEGER

ArchivoParcial ::= SEQUENCE {
 titulo  NombreOP,
 tipo   TipoOP,
 contenido  BIT STRING
}
  Entrada-al-Indice ::= SEQUENCE {
```

```
 titulo NombreOP,
 resumen      ResumenOP
}

Expediente ::= SEQUENCE {
 nombre-expediente   PrintableString,
 indice              SET OF Entrada-al-Indice,
 id-usuario          IdUsuarioOP,
 firma-usuario       FirmaUsuarioOP
}

Sello ::= SEQUENCE {
 estampa-de-tiempo    UTCTime,
 emisor               EmisorOP,
folio-usuario         Folio-UsuarioOP
}

Constancia ::= SEQUENCE {
 nombre-de-la-constancia   NombreConstanciaOP,
 expediente                Expediente,
 marca-de-tiempo           Sello,
 firma-constancia          FirmaConstanciaOP
}
END
```

2.2.1. Codificación BER

BER es el acrónimo en inglés de las Reglas básicas de codificación de ASN.1. BER se define en las recomendaciones X.209 y X.690 de ITU-T. Es un conjunto de reglas para codificar datos ASN.1 en una secuencia de octetos que se pueden transmitir a través de un vínculo de comunicaciones. Hay otros métodos de codificación de datos ASN.1, entre los que se incluyen Distinguished Encoding Rules (DER, Reglas de codificación completa), Canonical Encoding Rules (CER, Reglas de codificación canónica) y Packing Encoding Rules (PER, Reglas de codificación de empaquetamiento). Cada método de codificación tiene su aplicación, pero BER parece ser el método más utilizado y del que más se habla.

BER define lo siguiente:

- Métodos para codificar valores ASN.1.
- Reglas para decidir cuándo se utiliza un método determinado.
- El formato de determinados octetos de los datos.

2.3. Estándares PKCS

Aquí describo de qué tratan sólo algunos estándares que hablan sobre mi tema en particular: Constancias digitales. Enfocándome en estampas de tiempo, firma digital, la sintaxis utilizada, etc.

Son Estándares de Criptografía de Clave Pública que permiten la interoperatibilidad de las aplicaciones, ya que aunque los fabricantes estén de acuerdo en las técnicas básicas de la criptografía de clave pública, ésta no basta para garantizar la compatibilidad entre las implementaciones.

Estos estándares fueron desarrollados de acuerdo a los siguientes objetivos:

- Ser compatibles con PEM (PEM se describe en las RFCs 1421-1424) donde fuera posible, al menos para extender o poder compartir certificados y traducir mensajes cifrados o firmados de PEM a PKCS y viceversa.
- Extender la capacidad de PEM para ser capaz de manejar cualquier tipo de datos binarios (no sólo ASCII).
- Describir un estándar capaz de incorporarse en futuros sistemas OSI (Open Systems Interconnection, se describe en X.200). Los estándares se basan en el uso de asn_1 (Abstract Syntax Notation One, descrita en X.208) y ber (Basic Encoding Rules, descritas en X.209).

PKCS describe la sintaxis de los mensajes de una forma abstracta, dando completos detalles sobre algoritmos. Sin embargo, no especifica cómo se representan los mensajes, aunque BER es la elección lógica.

Los estándares PKCS son ofrecidos por RSA Laboratories y para nuestro caso se comentan los siguientes:

2.3.1. PKCS #1: Cryptography Standard

- Describe un método, llamado rsaEncryption, para el cifrado de datos utilizando el criptosistema de clave pública RSA. Es utilizado en la construcción de firmas digitales y envoltorios digitales, tal y como se describen en el PKCS #7.
- También describe una sintaxis para claves públicas y claves privadas RSA. La sintaxis de clave pública es idéntica a la utilizada tanto en X.509 como en PEM. Así que claves RSA X.509/PEM Pueden ser utilizadas en PKCS #1.
- También define tres algoritmos de firmado, llamados md2WithRSAEncryption, md4WithRSAEncryption, md5WithRSAEncryption

En el RFC 3447 se especifica este PKCS.

2.3.2. PKCS #6: Extended-Certificate Syntax Standard

- Describe una sintaxis para certificados con extensiones. Un certificado con extensiones consta de un certificado de clave pública X.509 y un conjunto de atributos, firmados por el emisor del certificado X.509. De esta forma los atributos y el certificado X.509 pueden ser verificados con una única operación de clave pública, pudiendo extraer si fuese necesario el certificado X.509, por ejemplo, para PEM.

  La intención de incluir un conjunto de atributos es extender el proceso de certificación más allá de la clave pública para incluir otra información sobre la entidad dada, como la dirección de e-mail.

2.3.3. PKCS # 7: Cryptographic Message Syntax Standard

- Describe una sintaxis general para aquellos datos tratados criptográficamente, como firmas digitales y envoltorios digitales. La sintaxis admite recursividad, por ejemplo, un envoltorio puede ser anidado en otro, o se puede firmar una parte previamente envuelta.
- Es compatible con PEM.
- Puede soportar varias arquitecturas para gestión de claves basadas en certificados, como la descrita para PEM en la RFC 1422. Las decisiones sobre la arquitectura quedan fuera del alcance de este PKCS.

- Los valores producidos de acuerdo a este PKCS están preparados para ser codificados mediante BER, lo cual significa que los valores serán representados por cadenas de octetos. Como es sabido, algunos sistemas de correo electrónico no están preparados para estas cadenas, PKCS #7 no da ningún método para codificar cadenas de octetos en cadenas de caracteres ASCII u otras técnicas que permitan una transmisión fiable mediante recodificación de la cadena de octetos. En el RFC 1421 se sugiere una posible solución a este problema.

  El PKCS # 7 se especifica en el RFC 2315.

2.3.4. PKCS #10: Certification Request Syntax Specification

- Describe una sintaxis para peticiones de certificados. Una petición de certificado consiste en un nombre distinguido, una clave pública, y opcionalmente un conjunto de atributos firmados por la entidad que realiza la petición. Las peticiones de certificación son enviadas a una Autoridad de Certificación, la cual transforma la petición en un certificado X.509 o en un PKCS #6.

  La intención de incluir un conjunto de atributos es doble: proporcionar información sobre la entidad y proporcionar los atributos para el PKCS #6.

  Las autoridades de certificación también pueden tener formas no electrónicas de recibir peticiones y devolver las respuestas, esto queda fuera del alcance de este PKCS.

  El PKCS #10 se especifica en el RFC 2986.

Capítulo 3

NOM-151-SCFI-2002

Este capítulo trata de lo que es la norma, para qué sirve, de qué está formada, requisitos (en cuanto a su implementación) y un breve comentario a cada uno de los campos que debe contener. Se analiza el RFC-3161 que nos da recomendaciones para crear estampas de tiempo que es lo que se utiliza en la norma por lo que se hace un análisis de las diferencias entre esta recomendación y la norma.

La Secretría de Economía establece que todos los comerciantes que elaboren contratos,convenios o compromisos que den nacimiento a derechos y obligaciones entre ellos y las personas que pacten dichos contratos, convenios o compromisos están obligados a guardar dicha información de una manera segura y permanente para porteger los intereses del consumidor y cumplir con dichos contratos en la forma que se establecieron por primera vez, y esto se hace en expedientes, certificados y/o constancias digitales con los requisitos que se establecen en esta norma.

Habrá dos entidades interactuando para dar cumplimiento a los requisitos de esta norma: un sistema de conservación de información (en la empresa que hace el contrato) y una oficialía de partes (entendiéndose como un prestador de servicios de certificación).

El proceso es "sencillo": la empresa con ayuda de un sistema de conservación (sistema donde se guarda todos los archivos) forma un archivo parcial en formato ASN.1, después se le aplica el algoritmo hash MD5 para obtener un resumen digital (o compendio según la presente norma) el cual se llama entrada al índice en la norma. Paso siguiente es formar un expediente electrónico que es como el conjunto de entradas al índice que tienen que ver a lo mejor con el mismo cliente o mismo

producto, etc. , y que tienen (los expedientes) tanto la identificación y firma del sistema de conservación. Después el sistema de conservación envía dicho expediente a un prestador de servicios de certificación, el cual creará una constancia digital firmada por él correspondiente a dicho expediente que dice que la información es cierta y/o existe y lo regresará al sistema de conservación.

3.1. Constancias Digitales

En esta sección se define qué es y cómo se forma una constancia digital así como qué debe contener según la norma.

Le llamaremos constancia digital a un mensaje de datos en formato ASN.1 hecho por algún prestador de servicios de certificación donde se establece que dicho expediente es válido para una fecha en particular, entendiéndose como válido que se encuentra íntegro (posee ciertas entradas al índice) y que además la firma (del que solicita la constancia) es verdadera, firmando y colocando un sello (estampa de tiempo) que nos dice en qué tiempo se está comprobando que existe dicho expediente de acuerdo a lo estipulado en esta norma pero sin comprobar que los índices sean correctos, o sea, solo se corrobora que posea ciertos índices pero no se comprueba que estos sean los correctos al aplicar MD5 a los archivos parciales.

- ¿CÓMO OBTIENE LA CONSTANCIA EL PRESTADOR DE SERVICIOS DE CERTIFICACIÓN ?

  Para la obtención de la constancia, el sistema de conservación deberá usar el protocolo de aplicación descrito en esta sección para enviar el expediente al prestador de servicios de certificación, quien emitirá una constancia en formato ASN.1 y la regresará al sistema de conservación, haciendo uso del mismo protocolo.

  El prestador de servicios de certificación podrá recibir, si así lo acuerda con sus clientes, medios físicos como discos duros o memorias de almacenamiento digital, conteniendo los archivos correspondientes a los expedientes.

- FORMACIÓN DE LA CONSTANCIA

  El prestador de servicios de certificación formará una constancia en formato ASN.1 que contendrá:

1. el nombre del archivo en donde está almacenada la constancia, esto es, la ruta completa de donde estuvo o está almacenado el archivo.
2. el expediente enviado por el sistema de conservación tal como se envió que incluye un encabezado (un identificador de objeto que dice que es una constancia) y el nombre de la constancia (ejemplo en el siguiente capítulo).
3. fecha y hora del momento en que se crea la constancia, esto es, la estampa de tiempo (sello según la norma) que coloca el prestador de servicios de certificación (oficialía de partes).
4. la identificación del prestador de servicios de certificación, como nombre de quien elaboró la constancia, si es persona física o moral y algún documento de identificación como IFE o pasaporte.
5. su firma digital de acuerdo a la definición correspondiente de esta Norma, es decir, también aplicando cifrado RSA con MD5.

Finalmente la constancia del prestador de servicios de certificación contiene una copia del expediente más la estampa de tiempo y la identificación (nombre y firma) del prestador de servicios de certificación que la generó.

En la figura 3.1 se ejemplifica lo que es un archivo parcial seguido de cómo se forma el expediente electrónico en la figura 3.2 y termino con el ejemplo de cómo se obtiene la constancia digital del prestador de servicios de certificación en la figura **??**.

3.2. El RFC 3161

En esta sección se describe qué es el RFC-3161 y si es que nos puede servir para hacer observaciones a la norma diferenciando cuál de los dos es mejor. Un RFC se puede ver como recomendaciones sobre cierto tema, entonces aquí se dan las recomendaciones sobre estampas de tiempo (sello) que es lo que debe contener la constancia digital según la norma.

Tal vez este término nos suene conocido como el Registro Federal de Causantes, pero en este caso no es así, sino que se trata de *Request For Comments*, que traducido es como petición de comentarios. Se trata de una propuesta oficial para nuevos protocolos de internet y por lo tanto debe contener hasta el más mínimo detalle para que en dado caso de ser aceptado pueda ser implementado fácilmente.

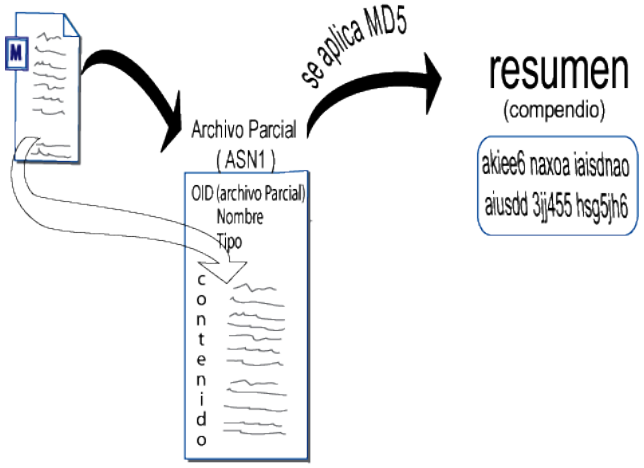

Figura 3.1: Archivo parcial según la norma

Cada RFC tiene un título y un número asignado, que no puede repertirse ni eliminarse aunque el documento quede obsoleto. Cada protocolo de los que hoy existen en Internet tiene asociado un RFC que lo define, y posiblemente otros RFCs adicionales que lo amplían. Por ejemplo el protocolo IP se detalla en el RFC-791, el FTP en el RFC-959, y el HTTP (escrito por Tim Berners-Lee, entre otros) en el RFC-2616.

Existen varias categorías, pudiendo ser informativos (cuando se trata simplemente de valorar por ejemplo la implantación de un protocolo), propuestas de estándares nuevos, o históricos (cuando quedan obsoletos por versiones más modernas del protocolo que describen). Las RFC se redactan en inglés según una estructura específica y en formato de texto ASCII. Antes de que un documento tenga la consideración de RFC, debe seguir un proceso muy estricto para asegurar su calidad y coherencia. Cuando lo consigue, prácticamente ya es un protocolo formal al que probablemente se interpondrán pocas objeciones, por lo que el sentido de su nombre como petición de cometarios ha quedado prácticamente obsoleto, dado que las críticas y sugerencias se producen en las fases anteriores. De todos modos, el nombre de RFC se mantiene por razones históricas.

Entre los organismos oficiales que proponen RFCs se encuentra el IETF (Internet Engineering Task Force), IAB (Internet Architecture Board) e IRTF (Internet

Figura 3.2: Expediente según la norma

Research Task Force).

El RFC 3161 trata del protocolo de estampas de tiempo (Time Stamp Protocol, TSP) en Internet X.509.

Este documento describe el formato que debe llevar una petición enviada a una Autoridad de Estampas de Tiempo (TSA) y de la respuesta que devuelve ésta.

También establece varios requisitos de seguridad relevantes para la operación del TSA, en lo que respecta a las peticiones y a sus respuestas.

A grandes rasgos podemos describir a este RFC de la siguiente manera:

1. Introducción.

   El servicio de estampas de tiempo prueba que un dato existió en un tiempo (fecha) en particular. Un TSA puede funcionar como servicio de una tercera persona (Trusted Third Party, TTP), aunque también pueden ser propios, es decir, una organización pudo requerir un TSA para colocar estampas de tiempo en documentos internos.

Figura 3.3: Constancia según la norma

Sirve tambien para servicios de la No-renegación [ISONR] que requieren la capacidad de establecer la existencia de datos antes de fechas específicas. Este protocolo se puede utilizar para apoyar tales servicios.

Este estándar no establece los requisitos totales de la seguridad para la operación de un TSA. Se anticipa que un TSA dará a conocer a los clientes anticipadamente las políticas que pone en ejecución para asegurar la generación exacta de la estampa de tiempo, y los clientes harán uso de los servicios de un TSA solamente si están convencidos que estas políticas resuelven sus necesidades.

2. ¿Qué puede hacer el TSA?.

   a) Utilizar una fuente digna de confianza del tiempo.

   b) Incluir un valor digno de confianza del tiempo para cada símbolo estampa de tiempo.

   c) Incluir un número entero único para cada símbolo nuevamente generado de la estampa de tiempo.

   d) Producir un símbolo estampa de tiempo sobre la recepción de una petición válida del solicitante, cuando es posible.

e) Incluir dentro de cada símbolo estampa de tiempo un identificador para indicar únicamente la política de la seguridad bajo la cual el símbolo fue creado.

f) Imprimir solamente un valor hash en la estampa de tiempo como representación del dato, es decir, una impresión de los datos asociada a una función hash resistente a colisiones identificada únicamente por un OID (Object Identifiers).

g) Para examinar el OID de la función hash resistente a colisiones y verificar que la longitud del valor del hash es consistente con el algoritmo hash utilizado.

h) No examinar la impresión horaria de cualquier manera (con excepción de comprobar su longitud, según lo especificado en el punto anterior).

i) No incluir cualquier identificación del solicitante que pide la estampa en el símbolo estampa de tiempo.

j) Firmar cada símbolo estampa de tiempo usando una llave generada exclusivamente para este propósito y tener esta característica de la llave indicada en el certificado correspondiente.

k) Para incluir información adicional en el símbolo estampa de tiempo, si es pedido por el solicitante usando el campo de las extensiones, solamente para las extensiones que son apoyadas por el TSA. Si esto no es posible, el TSA responderá con un mensaje de error.

3. Operación del TSA

   Como el primer mensaje de este mecanismo, el interesado solicita un símbolo estampa de tiempo a la TSA enviando una petición (que sea o incluya un TimeStampReq, según lo definido abajo). Como el segundo mensaje, la TSA responde enviando una respuesta (que sea o incluya un TimeStampResp, según lo definido abajo) a la entidad interesada.

   Sobre la recepción de la respuesta (que incluya o no un TimeStampResp que contenga normalmente un TimeStampToken (prueba), según lo definido abajo), la entidad interesada verificará el error del estado vuelto en la respuesta y si no hay error presente él verificará los campos contenidos en el TimeStampToken y la validez de la firma digital del TimeStampToken. En detalle, verificará cual estampa de tiempo corresponde a lo que fue solicitado. El solicitante verificará que el TimeStampToken contenga el identificador correcto del

certificado del TSA, la impresión correcta y el algoritmo OID (Object Identifiers) correcto de los datos de la función hash.

Entonces, puesto que el certificado del TSA pudo haber sido revocado, el estado del certificado Se Debe comprobar (es decir, comprobando el CRL apropiado) para verificar que el certificado sigue siendo válido.

Entonces, el cliente Debe comprobar el campo política (policy) para determinar si la política bajo la cual el símbolo fue publicado sea aceptable.

4. Identificación del TSA

   El TSA Debe firmar cada mensaje en la estampa de tiempo con una clave reservada específicamente para ese propósito. Un TSA Puede tener claves privadas distintas, es decir, para acomodar diversas políticas, diversos algoritmos, diversos tamaños dominantes privados o para aumentar el funcionamiento. El certificado correspondiente Debe contener solamente una instancia de la clave extendida en la extensión del campo de uso según lo definido en la sección [RFC2459] 4.2.1.13 con KeyPurposeID que tiene valor:

   id-kp-timeStamping

   El siguiente identificador de objeto identifica el KeyPurposeID que tiene valor id-kp-timeStamping.

```
id-kp-timeStamping OBJECT IDENTIFIER ::= {iso(1)
                identified-organization(3) dod(6)
                internet(1) security(5) mechanisms(5) pkix(7)
                kp (3) timestamping (8)}
```

5. Formato de la Petición

   Una petición de estampas de tiempo es como sigue:

```
TimeStampReq ::= SEQUENCE  {
   version                      INTEGER  { v1(1) },
   messageImprint               MessageImprint,
     --un algoritmo hash OID y su valor
     --impresion del tiempo
```

```
reqPolicy             TSAPolicyId              OPTIONAL,
nonce                 INTEGER                  OPTIONAL,
certReq               BOOLEAN                  DEFAULT FALSE,
extensions            [0] IMPLICIT Extensions  OPTIONAL  }
```

El campo "version" (actualmente v1) describe la versión de la petición del grupo fecha/hora.

El campo "messageImprint" DEBE contener el hash del dato para ser impreso en la estampa de tiempo. El hash se representa como un OCTETO. Su longitud DEBE emparejar la longitud del valor del hash para ese algoritmo (es decir, 20 octetos para SHA-1 o 16 octetos para MD5).

```
MessageImprint ::= SEQUENCE  {
     hashAlgorithm                AlgorithmIdentifier,
     hashedMessage                OCTET STRING  }
```

El algoritmo hash indicado en el campo hashAlgorithm DEBE ser un algoritmo conocido(una función hash de un sentido y resistente a colisión). La TSA DEBE comprobar si el algoritmo dado hash conocido es "suficiente" (basado en el estado actual del conocimiento en criptoanálisis y del estado actual en recursos de cómputo, por ejemplo). Si el TSA no reconoce el algoritmo hash o sabe que el algoritmo es débil (una decisión independiente de cada TSA), entonces el TSA DEBE rechazar proporcionar el símbolo estampa de tiempo regresando un pkiStatusInfo como "bad alg".

El campo reqPolicy, si está incluido, indica la política del TSA bajo la cual el TimeStampToken DEBE ser proporcionado. Se define TSAPolicyId como sigue:

```
TSAPolicyId ::= OBJECT IDENTIFIER
```

El "nonce", si está incluido, permite que el cliente verifique la puntualidad de la respuesta cuando no hay reloj local disponible. El nonce es un número al azar grande con una alta probabilidad de que el cliente lo genere solamente una vez (es decir, un número entero de 64 bits). En tal caso el mismo valor del nonce SE DEBE incluir en la respuesta, si no la respuesta será rechazada.

Si el campo del "certReq" está presente y puesto como verdadero, el certificado de clave pública del TSA que es referenciado por "ESSCertID identifier"

dentro del atributo "SigningCertificate" Se Debe proporcionar por el TSA en el campo "SignedData" en esa respuesta. Ese campo puede también contener otros certificados.

Si el campo "certReq" falta o si el campo "certReq" esta presente y puesto como falso entonces el campo "SignedData" No Debe estar presente en la respuesta.

El campo de las extensiones es una manera genérica de agregar la información adicional a una petición en el futuro. Las extensiones se definen dentro del RFC 2459. Si una extensión, crítica o no, es utilizada por un solicitante pero no reconocida por un servidor de estampas de tiempo, el servidor no publicará un símbolo y devolverá una falla (unacceptedExtension).

La petición de la estampa de tiempo no identifica al solicitante, pues esta información no es validada por el TSA. En las situaciones donde el TSA requiere la identidad de la entidad de petición, la identificación/autentificación la realiza por medios alternos como encapsulación del CMS [CMS] o autentificación de TLS [RFC2246].

6. Formato de la Respuesta

   Una respuesta de estampas de tiempo es como sigue:

```
TimeStampResp ::= SEQUENCE  {
   status                  PKIStatusInfo,
   timeStampToken          TimeStampToken     OPTIONAL  }
```

   El status (estado) se basa en la definición de status en la sección 3.2.3 [RFC2510] como sigue:

```
PKIStatusInfo ::= SEQUENCE {
   status        PKIStatus,
   statusString  PKIFreeText     OPTIONAL,
   failInfo      PKIFailureInfo  OPTIONAL  }
```

   Cuando status contiene el valor cero o uno, un TimeStampToken (símbolo estampa de tiempo) Debe estar presente. Cuando status contiene un valor con excepción de cero o de uno, un TimeStampToken No Debe estar presente. Los valores siguientes son los que Debe contener status:

```
   PKIStatus ::= INTEGER {
      granted                (0),
   -- cuando PKIStatus contiene el valor de 0 el TimeStampToken,
                               -- por requerimiento esta presente.
      grantedWithMods        (1),
   -- cuando PKIStatus contiene el valor de 1 un TimeStampToken,
                           -- con modificaciones, estara presente.
      rejection      (rechazado)        (2),
      waiting         (en espera)       (3),
      revocationWarning      (4),
-- este mensaje contiene una advertencia que es una revocacion.
      revocationNotification (5)
               -- notificacion que ha ocurrido una revocacion   }
```

Los servidores NO DEBEN producir ningún otro valor. Los clientes DEBEN generar un error si los valores que no entienden están presentes.

Cuando el TimeStampToken no está presente, failInfo indica la razón por la que la petición de la estampa de tiempo fue rechazada y puede tener uno de los siguientes valores:

```
PKIFailureInfo ::= BIT STRING {
   badAlg               (0),
     -- identificador desconocido o algoritmo no soportado
   badRequest           (2),
     -- transaccion no permitida
   badDataFormat        (5),
     -- el dato proporcionado esta en formato incorrecto
   timeNotAvailable    (14),
-- la fuente del tiempo del TSA no esta disponible
   unacceptedPolicy    (15),
     -- la politica solicitada no es soportada por el TSA
   unacceptedExtension (16),
     -- la extension solicitada no es soportada por el TSA
   addInfoNotAvailable (17)
      -- la informacion adicional solicitada no puede ser
      -- entendida o no esta disponible
```

```
systemFailure       (25)
  -- la peticion no puede ser dirigido debido a una
                              -- falla del sistema   }
```

Estos son los únicos valores que PKIFailureInfo puede soportar.

Los servidores NO DEBEN producir ningún otro valor. Los clientes DEBEN generar un error si existen valores que no entienden.

El campo statusString de PKIStatusInfo SE PUEDE utilizar para incluir texto de justificación tal como "campo del messageImprint no se ajusta a formato correctamente".

Un TimeStampToken se define como ContentInfo ([CMS]) y encapsula un dato firmado en el tipo "content".

```
TimeStampToken ::= ContentInfo
  -- contentType is id-signedData ([CMS])
  -- content is SignedData ([CMS])
```

Los campos de tipo EncapsulatedContentInfo de la construcción de SignedData tienen los siguientes significados: eContentType es un identificador del objeto que únicamente especifica el tipo content. Para un TimeStampToken se define como:

```
id-ct-TSTInfo  OBJECT IDENTIFIER ::= { iso(1) member-body(2)
us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) ct(1) 4}
```

eContent es el "content" en sí mismo, como un octeto. El eContent será un valor codificado en DER de TSTInfo.

El TimeStampToken NO DEBE contener ninguna firma con excepción de la firma del TSA. El identificador del certificado (ESSCertID) del certificado de TSA DEBE incluir como atributo del signerInfo dentro de un atributo de SigningCertificate.

```
TSTInfo ::= SEQUENCE  {
   version                    INTEGER  { v1(1) },
```

```
policy                  TSAPolicyId,
messageImprint          MessageImprint,
  -- DEBE tener el mismo valor que el campo similar
                                        -- en  TimeStampReq
serialNumber            INTEGER,
    -- numero de serie entero por usuario mayor a 160 bits
genTime                 GeneralizedTime,
accuracy                Accuracy                OPTIONAL,
ordering                BOOLEAN             DEFAULT FALSE,
nonce                   INTEGER                 OPTIONAL,
  -- DEBE estar presente si el campo similar estaba presente
                                        --  en TimeStampReq
  -- En ese caso DEBE tener el mismo valor.
tsa                    [0] GeneralName          OPTIONAL,
extensions             [1] IMPLICIT Extensions   OPTIONAL }
```

El campo de version (actualmente v1) describe la versión del símbolo estampa de tiempo (TimeStampToken).

Entre los campos opcionales, solamente el campo "nonce" DEBE ser soportado.

El campo política (policy) DEBE indicar la política del TSA bajo la cual la respuesta fue producida. Si un campo similar estaba presente en el TimeStampReq, entonces DEBE tener el mismo valor, si no un error (unacceptedPolicy) DEBE ser regresado. Esta política PUEDE incluir los tipos siguientes de información (aunque esta lista no es ciertamente exhaustiva):

- Las condiciones bajo las cuales el símbolo estampa de tiempo puede ser utilizado.
- La disponibilidad de un registro simbólico de la estampa de tiempo, para permitir verificar más adelante que la estampa de tiempo es auténtica.

El campo del serialNumber es un número entero asignado por el TSA a cada TimeStampToken. DEBE ser único para cada TimeStampToken publicado por un TSA único (es decir, el nombre y el número de serie del TSA identifican un TimeStampToken único). DEBE ser notado que la característica DEBE preservar incluso después de una interrupción (desplome) posible del servicio.

genTime es el tiempo en el cual el símbolo estampa de tiempo ha sido creado por el TSA. Se expresa como tiempo UTC (tiempo universal coordinado) para reducir la confusión con el uso del huso horario local. El UTC es una escala de tiempo, basada en el segundo (SI), definido y recomendado por el CCIR(Comité Consultivo Internacional de Radiocomunicaciones) . Un sinónimo es el tiempo "Zulú" que es utilizado por la aviación civil y representado por la letra "Z" (fonético "Zulú").

La sintaxis en ASN.1 para GeneralizedTime puede incluir las fracciones de segundos en detalle. Tal sintaxis, sin las restricciones [RFC 2459] de la sección 4.1.2.5.2, donde GeneralizedTime se limita para representar el tiempo con un fraccionamiento de un segundo, se puede utilizar aquí.

Los valores de GeneralizedTime DEBEN incluir segundos. Sin embargo, cuando no hay necesidad de tener una precisión mejor que la segunda, entonces GeneralizedTime con una precisión limitada a un segundo DEBE ser utilizado (como adentro [RFC 2459]).

La sintaxis es: `YYYYMMDDhhmmss[.s...]Z`
ejemplo: `19990609001326.34352Z`

La codificación DEBE terminar con "Z" (que significa tiempo "Zulú"). Los elementos fracción de segundos, si se presentan, DEBEN omitir arrastrar todos 0; si los elementos corresponden todos a 0, DEBEN ser omitidos enteramente, y el elemento de la coma también DEBE ser omitido.

La medianoche (GMT) será representada en la forma:

```
''YYYYMMDD000000Z''
```

donde "YYYYMMDD" representa el día que sigue a la medianoche en la petición. Aquí están algunos ejemplos de representaciones válidas:

```
"19920521000000Z"
"19920622123421Z"
"19920722132100.3Z"
```

la exactitud representa la desviación de tiempo alrededor del tiempo del UTC contenido en GeneralizedTime.

```
Accuracy ::= SEQUENCE {
      seconds        INTEGER              OPTIONAL,
      millis     [0] INTEGER  (1..999)    OPTIONAL,
      micros     [1] INTEGER  (1..999)    OPTIONAL  }
```

Si los segundos, los milisegundos o los microsegundos faltan, entonces un valor de cero Se Debe tomar para el campo que falta. La exactitud (acurancy) se puede descomponer en segundos, milisegundos (entre 1-999) y los microsegundos (1-999), expresados todos como número entero.

Las extensiones son una manera genérica de agregar información adicional en el futuro. Las extensiones se definen dentro del RFC 2459.

Los tipos de campo particulares de la extensión se pueden especificar en estándares o se pueden definir y colocar por cualquier organización o comunidad.

7. Archivo basado en este protocolo

   Un archivo que contiene una estampa de tiempo Debe contener solamente la codificación DER de un mensaje de TSA, es decir, No Debe haber información extraña del jefe o del acoplado en el archivo. Tales archivos se pueden utilizar para transportar mensajes del tipo estampa de tiempo usando por ejemplo, FTP.

   Una petición de estampas de tiempo Debe estar dentro de un archivo con extensión .tsq (time-Stamp query). Una respuesta de estampas de tiempo Debe contener en un archivo con extensión .tsr (time-Stamp response).

   Otra posibilidad es utilizar un reloj local y una ventana móvil de tiempo durante la cual el solicitante recuerda (observa) todo el hash enviado durante ese tiempo. Al recibir una respuesta, el solicitante se asegura que la época de la respuesta está dentro de la ventana del tiempo y que hay solamente una ocurrencia del valor hash en esa ventana del tiempo. Si el mismo valor hash está presente más de una vez dentro de la ventana de tiempo, el solicitante puede utilizar un nonce, o esperar hasta que la ventana de tiempo se ha movido para volverse al caso donde el mismo valor hash aparece solamente una vez durante esa ventana de tiempo.

Atributos de una estampa de tiempo usando el CMS

Una de las aplicaciones principales de estampas de tiempo es el de estampar el tiempo en una firma digital para probar que la firma digital fue creada en un tiempo

dado. Si se revoca el certificado de clave pública correspondiente a éste permite que un verificador sepa si la firma fue creada antes o después de la fecha de la revocación.

Un lugar sensible para almacenar una estampa de tiempo es en una estructura [CMS] como un atributo sin firmar.

Aquí se define un atributo de la estampa de tiempo de firma que se pueda utilizar para estampar el tiempo en una firma digital.

El identificador siguiente del objeto identifica el atributo de la firma en la estampa de tiempo:

```
id-aa-timeStampToken OBJECT IDENTIFIER ::= { iso(1) member-body(2)
us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) aa(2) 14 }
```

El valor del atributo de la firma en la estampa de tiempo es tipo ASN.1 SignatureTimeStampToken:

```
SignatureTimeStampToken::= TimeStampToken
```

El valor del campo messageImprint dentro de TimeStampToken será un hash del valor del campo signature dentro de SignerInfo para el signedData que esta siendo estampado.

Colocación de una firma en un punto particular en el tiempo.

Se presenta un ejemplo de un uso posible de este servicio de estampa de tiempo. Coloca una firma en un punto particular en el tiempo, de el cual la información de estado apropiada del certificado (CRLs- certificate revocation list) DEBE ser comprobada. Este uso se piensa para ser utilizado conjuntamente con la evidencia generada usando un mecanismo de firma digital.

Las firmas se pueden verificar solamente según una política de la no-renegación. Esta política PUEDE ser implícita o explícita (es decir, indicado en la evidencia proporcionada por el firmante). La política de la no-renegación puede especificar, entre otras cosas, el período permitido por un firmante para declarar el compromiso de una clave de firma usada para la generación de firmas digitales. Así una firma no se puede garantizar ser válida hasta la terminación de este período.

Para verificar una firma digital, puede ser utilizada la siguiente técnica:

A. La información de la estampa de tiempo necesita ser obtenida tan pronto después de que se haya producido la firma (es decir, dentro de algunos minutos u horas).

B. La validez de la firma digital se puede entonces verificar de la siguiente manera:

1. El símbolo estampa de tiempo DEBE ser verificado en sí mismo y DEBE ser verificado que se aplica a la firma del firmante.
2. SE DEBE recuperar el día/hora indicado por el TSA en el TimeStampToken.
3. SE DEBE identificar y recuperar el certificado usado por el firmante.
4. El día/hora indicado por el TSA DEBE estar dentro del período de la validez del certificado del firmante.
5. SE DEBE recuperar la información de la revocación sobre ese certificado, en el Día/hora en la estampa de tiempo.
6. Si se revoca el certificado, entonces el día/hora de la revocación será posterior que el día/hora indicado por el TSA.

   Si todas estas condiciones se cumplen, entonces la firma digital será declarada como válida.

En resumen, un TSA:

- Coloca símbolos "estampas d tiempo".
- incluye un número entero único para cada estampa de tiempo.
- Produce, cuando es posible, sobre la recepcón de una petición de estampas de tiempo.
- No examina la impresión horaria.
- Incluye información adicional, si es posible, pedida por el solicitante.
- Se identifica y firma las estampas de tiempo.
- Especifica que función hash utilizó.
- Especifica que política utilizó.
- Cuenta con un nonce (número aleatorio) que nos ayuda a verficar el tiempo cuando no hay reloj disponible.
- Proporciona el certificado de clave pública.

Ahora, veamos que se requiere en el sello (estampa de tiempo) de la constancia digital según la norma.

De acuerdo a la definición ASN.1 mostrada en el capítulo anterior, existe un objeto sello, el cual contiene los campos:

- Estampa-de-tiempo: es la fecha y hora en formato GMT o IMT.
- Emisor: es el representante del prestador de servicios de certificación.
- Folio-usuario: es un número secuencial ascendente para cada usuario registrado del prestador de servicios de certificación.

Aparte de este objeto ya sabemos que existen otros más como: emisor, usuario, una firma digital, un resumen, un folio, entrada al índice y un expediente dónde esta por demás explicar, cumplen con la función de identificar a quien pone el sello, su firma, el resumen obtenido, se utiliza MD5 por lo que no es necesario especificar que algoritmo genera la firma digital como en el protocolo de estampas de tiempo, el folio es un número asignado a los usuarios registrados del prestador de servicios de certificación.

Sabemos que los RFCs son solamente comentarios, sugerencias para poder llevar a cabo ciertos procedimientos por lo que no espero que la norma cumpla con todo lo que dice el RFC 3161 pero si que pueda servir para más cosas, como por ejemplo que me de un recibo de que le solicité una estampa de tiempo (o constate) para cierta información como lo comenta el RFC 3161.

Capítulo 4

Requisitos de la NOM-151-SCFI-2002

En esta sección se presentan los elementos necesarios para la implantación de la presente Norma Oficial Mexicana; la descripción del algoritmo de conservación de información y la definición ASN.1 de los objetos usados.

- FORMACIÓN DE ARCHIVOS PARCIALES

  Para formar un archivo parcial se crea un mensaje en formato ASN.1 que contiene

  1. El nombre del archivo del sistema de información en el que está o estuvo almacenado el contenido del archivo, se especifica la ruta completa del archivo.
  2. el tipo del archivo, es decir, si es archivo de texto, de audio, etc, con una cadena de bits de acuerdo a la definición ASN.1 de la norma, y
  3. El contenido del mismo; con el objetivo de guardar la relación lógica que existe entre estos tres elementos.

- OBTENCIÓN DE LOS COMPENDIOS O RESÚMENES DIGITALES

  Se calcula el compendio o resumen digital del archivo o archivos parciales resultado del proceso anterior, usando el algoritmo MD5 (se especifica que se usa solo este algoritmo en el Diario Oficial de la Federación con fecha 20 de marzo del 2002 en la página 6).

- INTEGRACIÓN DEL EXPEDIENTE ELECTRÓNICO

  Para conformar un expediente electrónico se creará un mensaje ASN.1 que contiene

  1. El nombre del expediente, que debe de coincidir con el nombre con el que se identifica en el sistema de información en donde está o estuvo almacenado, como en el caso de los archivos parciales, es toda la ruta del archivo.
  2. Un índice (puede ser más de uno), que contiene el nombre y el compendio (valor hash al aplicarle MD5) de cada archivo parcial que integra el expediente. Si son dos o más índices se colocan en orden después de la línea que especifica que después de la misma siguen los índices (`35 e5 /*indice*/` en el ejemplo expediente docusuario de abajo) de la siguente manera: entrada al índice, título (nombre del archivo del cual se obtuvo el compendio), resumen (literalmente, `30 21` eso se escribe en el expediente) , algoritmo resumen (también, `30 oc` literalmente se escribe en el expediente)
  3. La identificación del operador del sistema de conservación.
  4. Su firma digital de acuerdo a la definición correspondiente en la presente Norma Oficial Mexicana.

- OBTENCIÓN DE LA CONSTANCIA DEL PRESTADOR DE SERVICIOS DE CERTIFICACIÓN

  Para la obtención de la constancia el sistema de conservación deberá usar el protocolo de aplicación descrito en esta sección para enviar el expediente al prestador de servicios de certificación, quien emitirá una constancia en formato ASN.1 y la regresará al sistema de conservación, haciendo uso del mismo protocolo.

  El prestador de servicios de certificación podrá recibir, si así lo acuerda con sus clientes, medios físicos conteniendo los archivos correspondientes a los expedientes.

- FORMACIÓN DE LA CONSTANCIA

  El prestador de servicios de certificación formará una constancia en formato ASN.1 que contendrá:

1. El nombre del archivo en donde está almacenada la constancia.
2. El expediente enviado por el sistema de conservación.
3. Fecha y hora del momento en que se crea la constancia.
4. La identificación del prestador de servicios de certificación.
5. Su firma digital de acuerdo a la definición correspondiente de esta Norma Oficial Mexicana.

- MÉTODO DE VERIFICACIÓN DE AUTENTICIDAD

  La verificación de la autenticidad de una constancia se realizará por medio del uso de un sistema de verificación que lleve a cabo los pasos siguientes:

  1. Verificar la firma digital del prestador de servicios de certificación en la constancia;
  2. Verificar la firma digital del operador del sistema de conservación en el expediente contenido en la constancia.
  3. Recalcular el compendio de él o los archivos parciales y verificar que coincidan con los compendios asentados en el expediente.

En el programa ASN.1 se definen primeramente los identificadores de objeto necesarios para identificar los tipos de archivo que se podrán almacenar observando la presente Norma Oficial Mexicana; estas definiciones podrán ser objeto de revisiones periódicas para incluir nuevos formatos, luego los objetos necesarios para almacenar los archivos o mensajes de datos que serán conservados.
A lo largo del programa se definen diferentes objetos ASN.1 cuyo uso dentro del programa aclara su función.

El campo firma-usuario del objeto expediente es la firma digital de los campos nombre-expediente, índice e id-usuario concatenados en ese orden, vistos como una secuencia de bytes.

En el objeto sello, el campo estampa-de-tiempo es la fecha y hora en formato GMT o IMT con la cual se creó el sello, emisor es el representante del prestador de servicios de certificación que está creando el sello y folio-usuario es un número secuencial ascendente para cada usuario registrado del prestador de servicios de certificación. Es decir, cada usuario llevará un registro numerado consecutivamente de

cada operación que registra el prestador de servicios de certificación.

El objeto Constancia contiene un campo nombre-de-la-constancia que almacena el nombre del archivo de computadora donde se guardará dicha constancia en el sistema de información del prestador de servicios de certificación, expediente que es de tipo Expediente y es la información que se registra con el prestador de servicios de certificación con un sello emitido por ella, este sello contiene la fecha y la hora del momento en que se crea la Constancia.

El campo firma-constancia es la firma digital de los campos nombre-de-la-constancia, expediente, marca-de-tiempo concatenados en ese orden y vistos como una secuencia de bytes.

El ejemplo de codificación está organizado de la siguiente forma: primero se presentan dos archivos que se desea conservar, a continuación se construyen cada uno de los objetos ASN.1 correspondientes, (i) los archivos parciales, (ii) el expediente que está almacenado en un archivo de nombre "docusuario.ber" y (iii) la Constancia que está en el archivo "recibo.ber". Los nombres de los archivos que almacenan al expediente, constancia y archivos parciales están almacenados en los campos nombre-expediente, nombre-de-la-constancia y título respectivamente.

Enseguida se presenta el contenido de los objetos ASN.1 correspondientes. La línea ==========representa el principio y el fin del archivo respectivamente y no forma parte del archivo.

Los objetos ASN.1 que se presentan están en formato BER y se muestra un vaciado hexadecimal comentado.

Archivo mensaje.txt

==========

Archivo de texto utilizado para ejemplificar la creación de documentos de usuario y constancias de la Oficialía de Partes. Este es uno de dos archivos que se utilizarán en dicho ejemplo.

==========

Archivo mensaje1.txt

==========

Segundo archivo de texto que se utilizará en la creación de un ejemplo para mostrar un documento usuario y una constancia de la oficialía de partes.

==========

Archivo parcial arp1.ber

==========

```
30 81 d7 /*ArchivoParcial*/
   13 0b 6d 65 6e 73 61 6a 65 2e 74 78 74 /*titulo:mensaje.txt*/
   06 09 75 81 09 81 33 81 45 02 01 /*tipo:nomAT-TXT*/
   03 81 bc /*contenido*/
      00 41 72 63 68 69 76 6f 20 64 65 20 74 65 78 74 6f 20 75 74 69 6c 69
7a 61 64 6f 20 70 61 72 61 20 65 6a 65 6d 70 6c 69 66 69 63 61 72 20 6c 61
20 63 72 65 61 63 69 6f 6e 20 64 65 20 64 6f 63 75 6d 65 6e 74 6f 73 20 64
65 0a 75 73 75 61 72 69 6f 20 79 20 63 6f 6e 73 74 61 6e 63 69 61 73 20 64
65 20 6c 61 20 4f 66 69 63 69 61 6c 69 61 20 64 65 20 50 61 72 74 65 73 2e
20 45 73 74 65 20 65 73 20 75 6e 6f 20 64 65 20 64 6f 73 20 61 72 63 68 69
76 6f 73 0a 71 75 65 20 73 65 20 75 74 69 6c 69 7a 61 72 61 6e 20 65 6e 20
64 69 63 68 6f 20 65 6a 65 6d 70 6c 6f 2e 0a
```

==========

En el archivo parcial anterior (arp1.ber) la primera línea `30 81 d7` corresponde a la identificación de cómo debe ser tratado el objeto, que en este caso es un archivo parcial. En la segunda línea `13 0b 6d 65 6e 73 61 6a 65 2e 74 78 74` el primer caracter hexadecimal nos indica que es el nombre del archivo y los caractéres siguientes son el nombre como tal del archivo, por ejemplo, los ultimos 4 caractéres son `2e 74 78 74` que corresponden a .txt respectivamente. La tercer línea indica qué tipo es el contenido y en la cuarta línea se indica que a partir de ahí comienza, en este caso, el texto. De los caractéres subsecuentes, el primero indica que ahí comienza el texto y los demás significan un caracter del archivo de texto “mensaje.txt” que es-

ta arriba, por ejemplo, el caracter 2e significa punto (.) y el caracter 0a salto de línea.

Archivo parcial arp2.ber

==========

```
30 81 b3 /*ArchivoParcial*/
  13 0c 6d 65 6e 73 61 6a 65 31 2e 74 78 74 /*titulo:mensaje1.txt*/
  06 09 75 81 09 81 33 81 45 04 01 /*tipo:nomAV-MPEG*/
  03 81 97 /*contenido*/
     00 53 65 67 75 6e 64 6f 20 61 72 63 68 69 76 6f 20 64 65 20 74 65 78
74 6f 20 71 75 65 20 73 65 20 75 74 69 6c 69 7a 61 72 61 20 65 6e 20 6c 61
20 63 72 65 61 63 69 6f 6e 20 64 65 20 75 6e 20 65 6a 65 70 6d 6c 6f 73 0a
70 61 72 61 20 6d 6f 73 74 72 61 72 20 75 6e 20 64 6f 63 75 6d 65 6e 74 6f
20 75 73 75 61 72 69 6f 20 79 20 75 6e 61 20 63 6f 6e 73 74 61 6e 63 69 61
20 64 65 20 6c 61 20 6f 66 69 63 69 61 6c 69 61 20 64 65 20 70 61 72 74 65
73 2e 0a
```

==========

Expediente docusuario.ber

El expediente en formato BER correspondiente a los archivos parciales que aparecen arriba es:

==========

```
30 82 01 4b /*expediente electronico-usuario:*/
   13 0f 64 6f 63 75 6d 65 6e 74 6f 20 23 20 34 35 36 /*nombre-expe-
diente electronico:documento # 456*/
  31 5e /*indice:*/
     30 2d /*Entrada-al-Indice:*/
        13 08 61 72 70 31 2e 62 65 72 /*titulo:arp1.ber*/
        30 21 /*resumen:*/
           30 0c /*algoritmoresumen:*/
              06 08 2a 86 48 86 f7 0d 02 05 /*Identificador de Obje-
to: md5*/
```

```
            05 00 /*NULL*/
         03 11 00 23 e7 4a 8a be d5 60 dd ec 07 5c 66 44 29 71 c2
                                                  -- /*resumen*/
   30 2d /*Entrada-al-Indice:*/
       13 08 61 72 70 32 2e 62 65 72 /*titulo:arp2.ber*/
      30 21 /*resumen:*/
         30 0c /*algoritmoresumen:*/
            06 08 2a 86 48 86 f7 0d 02 05 /*Identificador de Obje-
to: md5*/
            05 00 /*NULL*/
         03 11 00 8c c0 81 b0 ce 66 e9 b7 90 5a 96 05 e8 38 13 20
                                                  -- /*resumen*/
  30 64 /*id-usuario*/
    06 08 75 81 09 81 33 82 75 01 /*peronaMoralFisica: nomIPersona-
Fisica*/
    30 1c /*nombreRazonSocialIdU:*/
       13 08 52 61 79 6d 75 6e 64 6f /*Raymundo*/
       13 07 50 65 72 61 6c 74 61 /*Peralta*/
       13 07 48 65 72 72 65 72 61 /*Herrera*/
    06 09 75 81 09 81 33 82 75 01 03 /*tipopIdU: nomIF-CURP*/
    13 2f 41 71 75 69 20 76 61 20 6c 61 20 43 6c 61 76 65 20 55 6e 69
63 61 20 64 65 20 52 65 67 69 73 74 72 6f 20 64 65 20 50 6f 62 6c 61 63 69
6f 6e /*contenidoIdU:Aqui va la Clave Unica de Registro de Poblacion*/
       30 72 /*firma-usuario:*/
          30 0d /*algoritmoFirma:*/
             06 09 2a 86 48 86 f7 0d 01 01 04 /*Identificador de
Objeto:md5WithRSAEncryption*/
             05 00 /*NULL*/
          03 61 /*firma:*/
             00 6f 06 26 71 0e 7a 2a 55 33 f2 e1 cc 1b 44 de 3a 40 e9 b3
0d 87 ee 32 5d 90 5b 7c b2 29 72 56 d8 57 88 6d e4 37 c2 7b 95 2f 32 f8 72
15 87 ce 95 71 39 66 3c b2 d7 25 76 08 15 49 07 cf 2c 87 04 87 f5 f3 d6 31
c3 d0 13 16 1b 26 fc f2 6b 73 63 2c 37 e1 ce d6 0a a7 b4 30 57 df 96 c5 6d
30 98
```

==========

Constancia recibo.ber

Finalmente la constancia del prestador de servicios de certificación contiene una copia del expediente (desde la línea más la estampa de tiempo y la identificación del prestador de servicios de certificación que la generó

==========

```
30 82 02 f0 /*Constancia de la Oficialia de Partes*/
  13 0a 72 65 63 69 62 6f 2e 62 65 72 /*nombre-de-la-constancia:re-
cibo.ber*/
  30 82 01 4b /*expediente electronico-usuario:*/
     13 0f 64 6f 63 75 6d 65 6e 74 6f 20 23 20 34 35 36 /*nombre-
expediente-electronico:documento # 456*/
     31 5e /*indice:*/
        30 2d /*Entrada-al-Indice:*/
           13 08 61 72 70 31 2e 62 65 72 /*titulo:arp1.ber*/
           30 21 /*resumen:*/
              30 0c /*algoritmoresumen:*/
                 06 08 2a 86 48 86 f7 0d 02 05 /*Identificador de
Objeto: md5*/
                 05 00 /*NULL*/
              03 11 00 23 e7 4a 8a be d5 60 dd ec 07 5c 66 44 29 71 c2
/*resumen*/
        30 2d /*Entrada-al-Indice:*/
           13 08 61 72 70 32 2e 62 65 72 /*titulo:arp2.ber*/
           30 21 /*resumen:*/
              30 0c /*algoritmoresumen:*/
                 06 08 2a 86 48 86 f7 0d 02 05 /*Identificador de
Objeto: md5*/
                 05 00 /*NULL*/
               03 11 00 8c c0 81 b0 ce 66 e9 b7 90 5a 96 05 e8 38 13 20
/*resumen*/
     30 64 /*id-usuario*/
        06 08 75 81 09 81 33 82 75 01 /*peronaMoralFisica: nomIPer-
sonaFisica*/
        30 1c /*nombreRazonSocialIdU:*/
           13 08 52 61 79 6d 75 6e 64 6f /*Raymundo*/
           13 07 50 65 72 61 6c 74 61 /*Peralta*/
           13 07 48 65 72 72 65 72 61 /*Herrera*/
```

```
        06 09 75 81 09 81 33 82 75 01 03 /*tipopIdU: nomIF-CURP*/
        13 2f 41 71 75 69 20 76 61 20 6c 61 20 43 6c 61 76 65 20 55 6e 69
63 61 20 64 65 20 52 65 67 69 73 74 72 6f 20 64 65 20 50 6f 62 6c 61 63 69
6f 6e /*contenidoIdU:Aqui va la Clave Unica de Registro de Pobla-
cion*/
        30 72 /*firma-usuario:*/
           30 0d /*algoritmoFirma:*/
              06 09 2a 86 48 86 f7 0d 01 01 04 /*Identificador de
Objeto: md5WithRSAEncryption*/
              05 00 /*NULL*/
           03 61 /*firma:*/
              00 6f 06 26 71 0e 7a 2a 55 33 f2 e1 cc 1b 44 de 3a 40 e9 b3
0d 87 ee 32 5d 90 5b 7c b2 29 72 56 d8 57 88 6d e4 37 c2 7b 95 2f 32 f8 72
15 87 ce 95 71 39 66 3c b2 d7 25 76 08 15 49 07 cf 2c 87 04 87 f5 f3 d6 31
c3 d0 13 16 1b 26 fc f2 6b 73 63 2c 37 e1 ce d6 0a a7 b4 30 57 df 96 c5 6d
30 98
     30 81 fc /*marca-de-tiempo:*/
        17 0d 30 31 30 34 33 30 31 33 32 33 32 32 5a
/*estampa-de-tiempo:010430132322Z */
        30 81 e7 /*emisor:*/
           06 08 75 81 09 81 33 82 75 02 /*personaMoralFisica:
nomIPersonaMoral*/
           13 1c 4f 66 69 63 69 61 6c 69 61 20 64 65 20 50 61 72 74 65 73
           20 4e 75 6d 65 72 6f 20 31 /*nombreRazonSocialIdU:Oficia-
lia de Partes Numero 1*/
           06 09 75 81 09 81 33 82 75 02 03 /*tipoIdU: nomIM-CEDU-
LAFISCAL*/
           13 2a 41 71 75 69 20 76 61 20 6c 61 20 63 65 64 75 6c 61 20 64
65 20 69 64 65 6e 74 69 66 69 63 61 63 69 6f 6e 20 66 69 73 63 61 6c
/*contenidoIdU:
Aqui va la cedula de identificacion fiscal*/
           30 81 85 /*representanteIdU:*/
              30 4c /*nombreIdP:*/
                 13 11 4e 6f 6d 62 72 65 20 64 65 6c 20 65 6d 69 73 6f 72
/*nombreIdP: Nombre del emisor*/
                 13 1a 50 72 69 6d 65 72 20 41 70 65 6c 6c 69 64 6f 20 64
65 6c 20 65 6d 69 73 6f 72 /*apellido1IdP:Primer Apellido del emisor*/
                 13 1b 53 65 67 75 6e 64 6f 20 41 70 65 6c 6c 69 64 6f 20
```

```
64 65 6c 20 65 6d 69 73 6f 72 /*apellido2IdP:Segundo Apellido del
emisor*/
             06 09 75 81 09 81 33 82 75 01 02 /*tipoIdP: nomIF-IFE*/
             13 2a 4e 75 6d 65 72 6f 20 64 65 20 6c 61 20 43 72 65 64 65
6e 63 69 61 6c 20 64 65 20 45 6c 65 63 74 6f 72 20 64 65 6c 20 49 46 45
/*contenidoIdP:
Numero de la Credencial de Elector del IFE*/
        02 01 01 /*folio-usuario:1*/
     30 81 93 /*firma-constancia:*/
        30 0d /*algoritmoFirma:*/
           06 09 2a 86 48 86 f7 0d 01 01 04 /*Identificador de Ob-
jeto: md5WithRSAEncryption*/
           05 00 /*NULL*/
         03 81 81 /*firma:*/
           00 94 c1 94 4a 8c 32 59 5d 5f b8 2c f8 6c fc f4 d7 b0 1f 24 81
b9 ad ba 2d db 7e c8 43 f4 25 5e cf d6 40 a9 2e f8 d0 02 59 1a b2 99 95 76
5e 56 ee f6 e8 4b ee 0b 45 3d 3f 50 86 12 f4 74 f4 17 59 2f e5 45 d2 d9 d6
d6 ec f7 e6 58 54 f8 da c2 8e a8 6b 9f d3 0f e1 cd 87 de 2d 38 85 ee 56 cd
03 53 c9 c6 49 f3 36 b3 a6 d9 03 3a d6 e7 16 db 6d 82 89 54 93 8d 92 f9 2b
5f 63 10 1e e6 bb 94 78

==============================================
```

Capítulo 5

Áreas de Oportunidad

En este capítulo hablo de los puntos en los cuales la NOM-151-SCFI-2002 podría ser más versátil con la misma seguridad que tiene o incluso volverse más segura. Cuestiones en las cuales se puede mejorar o implementar para que muchas más entidades (empresas, personas) puedan hacer lo que establece la norma y no sólo las empresas que ya tienen la autorización por parte de la Secretaría de Economía dando paso, a que si hay empresas especializadas en diversos esquemas criptográficos puedan utilizar dichos esquemas para la implantación de esta norma y no encontrarse con dificultades al implementarla sólo como está establecida (recordemos que sólo utiliza RSA con MD5). Además, de que se puedan dar diferentes servicios como simplemente firmado de documentos y/o validación de la firma de documentos para el interés de algún tercero (llamémosle cliente).

Recordando, se toma un archivo y se coloca (como contenido) en un archivo parcial en formato ASN.1, los cuales sirven para formar una entrada al índice (aplicando la función hash, es decir el valor hash del archivo parcial es esta entrada al índice) y que se utilizan para formar los expedientes junto con otros campos claro. Estos expedientes son los que se envían a la oficialía de partes para que se forme la constancia digital. Vea figuras 3.1, 3.2, 3.3.

El escenario que establece la norma es como sigue:

- Una empresa (llamémosle usuario) crea sus archivos parciales y con estos uno o varios expedientes los cuales traen la identificación del sistema de información que los creó y su firma.

- Se envía dicho (s) expediente (s) a la oficialía de partes para que emita una constancia, recordemos, que sólo valida la firma, es decir, no le interesa comprobar los resúmenes de los archivos parciales contenidos en los expedientes.

Y, como ya dije en el capítulo 3, siguiendo los comentarios para un prestador de servicios de certificación (Autoridad de Estampas de Tiempo -TSA) se pueden crear muchos escenarios más (como el colocar una estampa de tiempo en una firma digital para verificar, cuando un certificado de clave pública sea revocado, si la firma se creó antes o después de dicha revocación o como un acuse de recibo que explico adelante).

De acuerdo a esto, menciono los puntos en los que se puede mejorar, o simplemente se puede utilizar, la norma a continuación:

5.1. Algoritmos de Digestión

En el texto de la norma en la página 6 en el párrafo concerniente a la Obtención de los Compendios o Resúmenes Digitales dice textualmente "Se calcula el copendio o resumen digital del *archivo* o *archivos parciales* resultado del proceso anterior [Formación de Archivos Parciales] usando el algoritmo `MD5`". Esto es, la norma especifica que los expedientes se conforman de una o varias entradas al índice (valor hash de un archivo parcial) y otros campos a partir de los contratos aplicando solo el algoritmo hash `MD5`, pero habiendo muchos más, por qué limitarnos a sólo usar ese, entonces puedo proponer que sea posible usar muchas más funciones hash como RIPEMD-160, HAVAL o la familia de funciones SHA (SHA-1, SHA-256, SHA-512), que tienen al menos la misma seguridad y que en la norma se tiene que especificar que algoritmo se utiliza, es decir, se da el OID para MD5 en la definición ASN.1 de la norma y en el ejemplo se tiene que especificar ese OID y entonces se puede (podría) ser otro algoritmo hash.

Es importante mencionar que bajo el supuesto de que el ataque de la paradoja del cumpleaños es la forma óptima de atacar al algoritmo `MD5`, su tamaño de salida (16 bytes) le da una menor vigencia en comparación con un algoritmo de resumen (bajo el mismo supuesto de ataque) cuya salida sea de 20 bytes, como por ejemplo `SHA-1`. Otra debilidad del algoritmo de resumen `MD5` es que se han encontrado una forma de generar cierto tipo especíico de colisiones, por lo que ya no debería de usarse este algorimo de resumen.

5.2. Vigencia de Algoritmos

La norma establece que al menos deben de durar 10 años las constancias emitidas por la oficialía de partes lo que quiere decir que tal vez existan constancias con una vigencia eterna, o sin vigencia (que serán válidas siempre), pero, dado al

poder de computo que crece rápidamente no se puede garantizar que los algoritmos criptográficos duren para siempre y mucho menos MD5 que es el único utilizado en la norma (especificado en la página 6 del Diario Oficial de la Federación). Además los certificados que se utilizan para verificar las firmas, al menos en nuestro país, tienen una vigencia muy corta, por ejemplo, los certificados que emite el SAT (Servicio de Admnistración Tributaria) duran solo 2 años.

De acuerdo al artículo publicado en junio 30 del 2004 por Arjen K. Lenstra se dan los valores para RSA en cuanto a la longitud de la clave y el año en que todavía se puede considerar seguro de forma conservadora y de forma optimista. Presento dichos valores en la tabla 5.1.

Cuadro 5.1: Seguridad en las claves

longitud de la clave en bits	año hasta el cual es seguro (conservativamente)	año hasta el cual es seguro (optimistamente)
1024	2006	2006
1280	2014	2017
1536	2020	2025
2048	2030	2040
3072	2046	2065
4096	2060	2085
8192	2100	2142

5.3. Acuse de Recibo

Las constancias digitales, según la norma, sirven para certificar que cierta información existió en cierto tiempo en particular y lo que puedo proponer es que también sirvan como especie de acuse de recibo. Supongamos que existen dos entidades Ana y Beto celebran un convenio en el cual Ana tiene que entregarle a Beto cierta información en cierta fecha, entonces, algo para lo lo que podría servir (utilizar sus servicios) la oficialía de partes es que emita un acuse de recibo de que la información se la entrego Ana a Beto y el proceso sería como sigue:

- Ana le envía a Beto cierta información electrónica a Beto.
- Beto recibe la información de Ana y solicita una constancia a la oficialía de partes de que recibió información de Ana.

- Beto recibe constancia de la oficialía de partes y se la envía a Ana para que ella pueda respaldarse en caso de que Beto diga que no ha recibido la información.

Por ejemplo, si yo para presentar mi exámen profesional me piden que entregue original y copia del certificado de estudios y cómo eso lo detienen en la subdirección académica, ¿nos podríamos imaginar que pasaría si alguien perdiera esos papeles y yo sólo tuviera una hoja con sello y firma de recibido?, sería mejor si tuviera una constancia digital (como acuse de recibo) que tuviera tanto una copia de los dos documentos como el sello y firma de quien me los recibió, ¿o no?

5.4. Refrendos de la Constancia

Según lo expuesto en el segundo punto, no sabemos qué vigencia pueden tener los algoritmos criptográficos (en específico MD5) pero con seguridad no serán resistentes a colisiones por diez años y, en este sentido, la constancia no puede ser válida diez años. Por lo que se tendría que hacer una revisión períodica tanto del algoritmo involucrado como de la constancia. Es decir, se crea una constancia con MD5 y se le da una vigencia de 3 años (hablando optimistamente). Durante ese tiempo se tendría que verificar que MD5 sigue siendo seguro y calcular qué tiempo es que seguiría siendo seguro para que cuando termine el lapso de vida de la constancia se le pueda poner un refrendo (del tiempo aproximado calculado) y siga siendo válida dicha constancia.

Capítulo 6

Conclusiones

En este trabajo presenté la teoría necesaria para entender los elementos criptográficos involucrados en el proceso de la obtención de una constancia digital de acuerdo a la NOM-151. Mostré las partes técnicas de la norma y puntualicé algunas áreas de oportunidad en donde se puede reforzar la norma en futuras revisiones de la misma.

Es importante tener en cuenta que para alcanzar verdaderamente el objetivo de la NOM-151, se deben implementar mecanismos que permitan prolongar la vigencia de las Constancias Digitales emitidas de acuerdo a la norma, pues de entrada las Constancias Digitales están primeramente limitadas por la vigencia de los Certificados Digitales necesarios para la verificación de las firmas digitales involucradas, sin mencionar la vida útil de los algoritmos subyacientes debido al poder de cómputo, el cual crece continuamente.

Enumero mis conclusiones a continuación:

1. Es necesario el estudio de teoría criptográfica para entender el aspecto técnico involucrado en el tiempo de vigencia de las constancias, así como de los parámetros necesarios en los elementos que conforman las direfentes partes de una constancia digital (funciones hash para los archivos parciales, tamaños de claves para la vigencia de las firmas digitales, etc.) según la NOM-151.

2. Puede haber constancias digitales mejor implementadas que como lo indica la norma pero ésta tiene sustento legal, por lo que, aunque tengamos una constancia con instancias criptográficas más confiables, tiene mayor valor jurídico la implementada según la NOM-151.

3. Deben existir refrendos a la constancia porque las instancias criptográficas empleadas no garantizan que la constancia sea segura por lo menos 10 años que es lo que establece la norma.

4. La constancia digital de acuerdo a la norma, no contempla a tres o más actores que podrían estar involucrados en un "Acuse de Recibo" (emisor de la información, receptor de la misma, emisor de constancias). Esto con el fin de que no sólo se dé sustento legal a que tenemos cierta información (en determinada fecha) sino que también se pueda respaldar (constatar) que se recibió (entregamos) dicha información en determinado tiempo (fecha y hora).

Bibliografía

[ALFJ04] Fuster Amparo, Dolores Luis, Montoya Fausto, and Muñoz Jaime. *Técnicas Criptográficas De Protección De Datos*. Ra-Ma, 3 edition, 2004.

[Bon99] Dan Boneh. Twenty years of attacks on the rsa cryptosystem. *In Notices of the American Mathematical Society*, 1(2):203–213, 1999.

[BSW06] Albrecht Beutelspacher, Jörg Schwenk, and Klaus-Dieter Wolenstetter. *Moderne Verahren der Kryptographie, von RSA zu Zero-Knowledge*. Vieweg, 2006.

[Buc04] Johannes Buchmann. *Introduction to Cryptography*. Undergraduate Texts in Mathematics. Springer-Verlag, second edition edition, 2004.

[CDL+00] Stefania Cavallar, Bruce Dodson, Arjen K. Lenstra, Walter M. Lioen, Peter L. Montgomery, Brian Murphy, Herman te Riele, Karen Aardal, Jeff Gilchrist, Gerard Guillerm, Paul C. Leyland, Joel Marchand, Francois Morain, Alec Muffett, Chris Putnam, Craig Putnam, and Paul Zimmermann. Factorization of a 512-bit RSA modulus. In *Theory and Application of Cryptographic Techniques*, pages 1–18, 2000.

[Coh96] Henri Cohen. *A Course in Computational Algebraic Number Theory*. Graduate Texts in Mathematics. Springer-Verlag, 1996.

[dE02] Secretaría de Economía. Nom-151-scfi-2002, prácticas comerciales-requisitos que deben observarse para la conservación de mensajes de datos. http://www.economia.gob.mx/work/normas/noms/2002/151scfi.pdf, 2002.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–65, November 1976.

[Dub00] Olivier Dubuisson. *ASN.1 - Communication between heterogeneous systems*. Morgan Kaufmann, 2000.

[Eas] D. Eastlake. Rfc 3174 - us secure hash algorithm 1 (sha1). `http://www.faqs.org/rfcs/rfc3174.html`.

[Eng03] Michael Englbrecht. *Entwicklung sicherer Software*. Spektrum, 2003.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. W. H. Freeman and company, 1979.

[GMR88] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. In *Siam Journal on Computing*, pages 281–308, 1988.

[Gol99] Oded Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*. Springer-Verlag, 1999.

[GP04] Álvarez Maranon Gonzalo and Pérez García Pedro Pablo. *Seguridad Informática Para Empresas Y Particulares*. McGraw â Hill, 2004.

[JA98] Pastor Jose and Sarasa Miguel Angel. *Criptografía Digital*. Prensas Universitarias De Zaragoza, 1998.

[Knu97] Donald E. Knuth. *The Art of Computer Programming*, volume 2. Addison-Wesley, 1997.

[Laba] RSA Laboratories. Public-key cryptography standards (pkcs). `http://www.rsasecurity.com/rsalabs/pkcs/`.

[Labb] RSA Laboratories. Rsa-based cryptographic schemes. `http://www.rsa.com/rsalabs/node.asp?id=2146`.

[LV01] A.K. Lenstra and E.R. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology*, 14(4):255–293, 2001.

[MOV96] Alfred J. Menezes, Paul C. Van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[odlF02] Diario oficial de la Federación. Norma oficial mexicaca scfi 2000, 2002. `http://www.economia.gob.mx/work/normas/noms/2002/151scfi.pdf`.

[Riv] R. Rivest. Rfc 1321 - the md5 message-digest algorithm. `http://www.faqs.org/rfcs/rfc1321.html`.

[RSA78] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[Sch] J. Schaad. Rfc 3537 - wrapping a hashed message authentication code (hmac) key with a triple-data encryption standard (des) key or an advanced encryption standard (aes) key. `http://www.faqs.org/rfcs/rfc3537.html`.

[Sch96] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, 1996.

[Sch05] Jörg Schwenk. *Sicherheit und Kryptographie im Internet*. Vieweg, 2005.

[Sta03] William Stallings. *Cryptography and network security essentials*. Prentice Hall, 3 edition, 2003.

[WFLY04] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu. Collisions for hash functions md4, md5, haval-128 and ripemd. Cryptology ePrint Archive, Report 2004/199, 2004. `http://eprint.iacr.org/`.