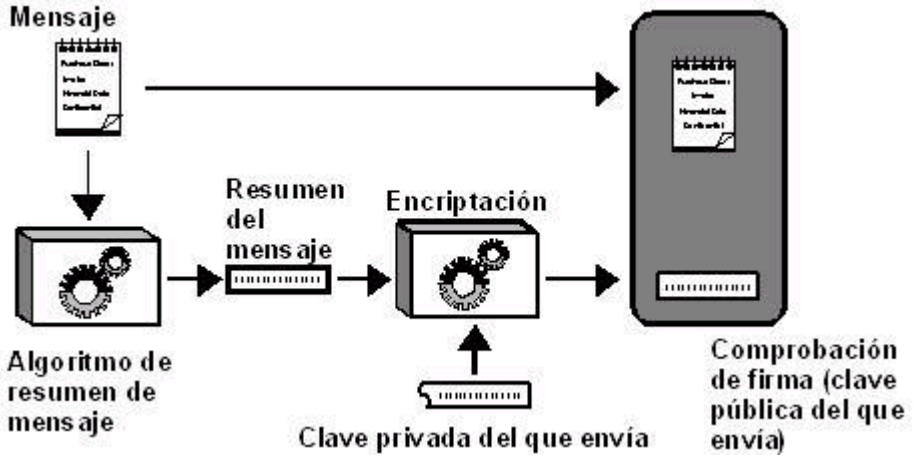
C A P Í T U L O V

VALIDACIÓN DE IDENTIFICACIÓN



5.1 Protocolos de Validación de Identificación

5.2 Autentificación de Mensajes

5.2.1 Algoritmo MD5 *(Message Digest 5)*

5.2.2 Algoritmo SHA *(Secure Hash Algorithm)*

5.3 Firmas Digitales

5.3.1 DSS (Digital Signature Standard)

5.4 Sistemas de Identificación Biométrica

5.1 PROTOCOLOS DE VALIDACIÓN DE IDENTIFICACIÓN

La validación de identificación *(autentificación)* es la técnica mediante la cual un proceso comprueba que su compañero de comunicación es quien se supone que es y no un impostor. La validación de identificación se encarga del asunto de comprobar si realmente nos estamos comunicando con otro proceso específico, es decir, el proceso *A* está seguro de que está 'hablando' con el proceso *B*, y *B* también está seguro de que está 'hablando' con *A* [LIB 001].

AUTENTIFICACIÓN BASADA EN CLAVE SECRETA COMPARTIDA

Para este protocolo de validación de identificación, suponemos que los procesos *A* y *B* ya comparten una clave secreta K_{AB}. En la *figura 5.1* se muestra el protocolo de *reto-respuesta*, se denomina así, ya que una parte envía un número aleatorio a la otra, este lo transforma de una manera especial y le devuelve el resultado.

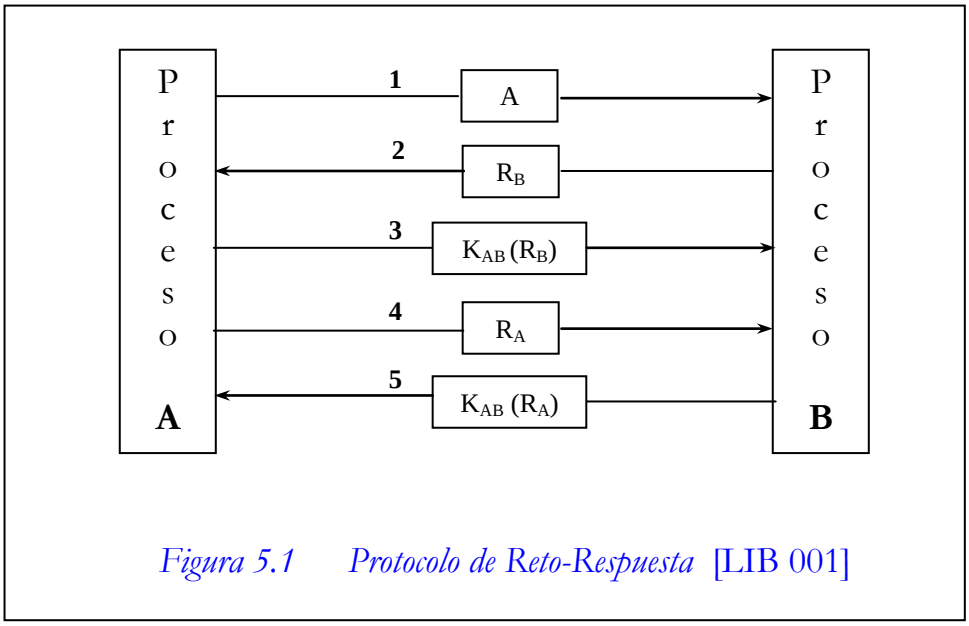


Figura 5.1 Protocolo de Reto-Respuesta [LIB 001]

En el mensaje 1, el proceso *A* envía su identidad al proceso *B*. *B* no tiene manera de saber si este mensaje viene de *A* o de un intruso *Z*, por lo que escoge un reto, un número aleatorio grande R_B y lo envía a *A* como mensaje 2 en texto normal. *A* cifra el mensaje con la clave secreta que comparte con *B* y envía el texto cifrado $K_{AB}(R_B)$ en el mensaje 3. Cuando *B* recibe este mensaje, sabe que vino de *A* porque *Z* no conoce la clave secreta K_{AB} y por lo tanto él no pudo haber generado. En este punto *B* está seguro de que está 'hablando' con *A*, pero *A* no está seguro de nada todavía. *A* selecciona un número aleatorio grande R_A y le envía a *B* como texto normal en el mensaje 4. Cuando *B* responde con $K_{AB}(R_A)$ en el mensaje 5, *A* está seguro que está 'hablando' con *B*.

Ahora, si desean establecer una clave de sesión, *A* puede seleccionar un K_S y enviársela a *B* cifrada con la clave secreta compartida K_{AB}.

INTERCAMBIO DE CLAVES DIFFIE-HELLMAN

En noviembre de 1976 W. Diffie y M. E. Hellman inventaron un método para el intercambio de claves secretas en canales inseguros [WWW 009]. Fue el primer algoritmo de clave pública desarrollado y, hoy en día, sigue siendo muy utilizado para el intercambio de claves. El procedimiento está basado en un sistema de claves asimétrico en el que cada comunicante posee dos claves, una de ellas secreta y la otra pública. Este método no sólo resolvió el problema de intercambio de claves, sino que además fue el origen de la denominada criptografía de clave pública, cuya aplicación tiene importantes implicaciones en el ámbito de la seguridad de las comunicaciones digitales.

El método, propuesto para el intercambio de claves secretas a través de canales inseguros, está basado en la existencia de *funciones de una sola dirección*. Una función de una sola dirección es aquella cuyo cálculo directo es viable,

pero el cálculo de la función inversa tiene tanta complejidad, que resulta verdaderamente imposible resolverlo. Por ejemplo, si ***y*** es una función de este tipo, el cálculo de $y = f(x)$ es sencillo, pero el cálculo de $x = f^{-1}(y)$ es tan complejo que no se puede resolver con los conocimientos matemáticos actuales.

Está muy extendido en sistemas de Internet con confidencialidad de clave simétrica tales como: red privada virtual (VPN), protocolo SSL, etc. La seguridad del algoritmo depende de la dificultad del cálculo de un logaritmo discreto y su funcionamiento se muestra en la *figura 5.2*.

La generación de claves públicas es como sigue:

- Se busca un número primo grande *p*, donde *(p-1)/2* también es primo.
- Se busca *g* raíz primitiva de *p*. Para ser raíz primitiva debe cumplir que:

  $g \bmod p$, $g^2 \bmod p$, $g^3 \bmod p$, ..., $g^{p-1} \bmod p$ son números diferentes.
- *g*, *p* son claves públicas.

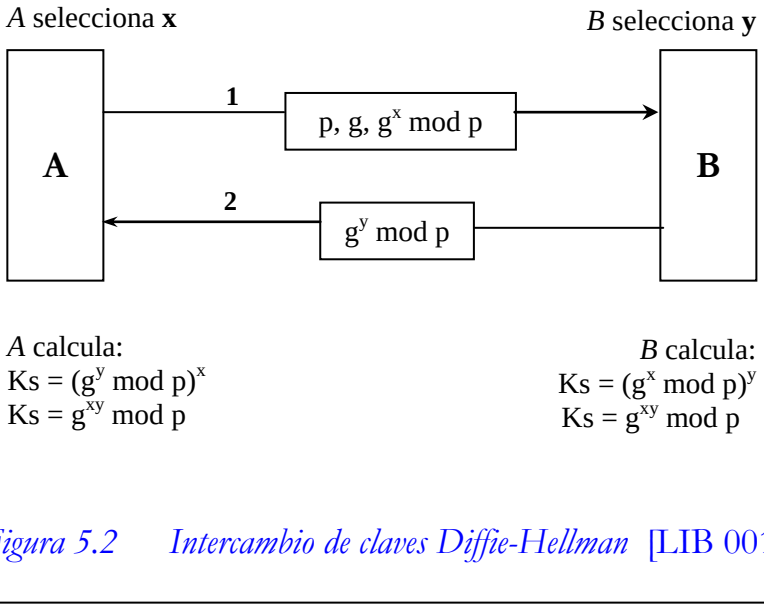


Figura 5.2 Intercambio de claves Diffie-Hellman [LIB 001]

El proceso *A* selecciona un número grande (digamos de 512 bits), llamémoslo ***x*** y lo mantiene en secreto. De igual manera lo hace *B* y selecciona el número ***y***. Ahora *A* envía a *B* el mensaje 1 que contiene la tripleta (p, g, g^x mod p). *B* responde enviando el mensaje 2 que contiene g^y mod p. *A* toma el número que le envía *B* y lo eleva a la potencia ***x*** obteniendo la clave de sesión Ks = g^{xy} mod p. Por la otra parte, *B* toma el tercer número que le envía *A* y lo eleva a la potencia ***y*** obteniendo también la clave de sesión Ks = g^{xy} mod p. Observe que las *Ks* calculadas por *A* y *B* son iguales y esto se debe a las propiedades de la aritmética modular.

Utilizar siempre la misma clave secreta compartida para muchas transmisiones tiene dos problemas:

1. Cuanto más criptogramas cifrados con la misma clave se tiene, más fácil es romper un sistema.
2. Si un criptoanalista descubre la clave, podrá descifrar todas las transmisiones siguientes sin que los implicados sean conscientes.

Por lo tanto es aconsejable cambiar de clave a menudo. A éstas claves se les llaman claves de sesión y son utilizadas durante una única sesión.

A continuación se muestra *un ejemplo* sencillo del intercambio de claves de Diffie-Hellman*:*

- Seleccionamos p=7, y como (p-1)/2 = (7-1)/2 = 3 también es primo, entonces p es válido.
- Seleccionamos g=3, puesto que:

  g^1 mod q, g^2 mod q, ..., g^{p-1} mod p son números diferentes.

  3^1 mod 7 ≠ 3^2 mod 7 ≠ 3^3 mod 7 ≠ 3^4 mod 7 ≠ 3^5 mod 7 ≠ 3^6 mod 7

  3 ≠ 2 ≠ 6 ≠ 4 ≠ 5 ≠ 1

Si *A* escoge ***x***=6 y *B* escoge ***y***=8 entonces el mensaje que *A* envía a *B* sería la tripleta (p, g, g^x mod p) = (7, 3, 3^6 mod 7) = (7, 3, 1) y el mensaje que *B* envía a *A* sería g^y mod p = 3^8 mod 7 = 2. En este punto, *A* procede a calcular $(g^y \text{ mod } p)^x = 2^6 \equiv 1 \text{ (mod 7)}$. Por su parte, *B* procede a calcular la expresión $(g^x \text{ mod } p)^y = 1^8 \equiv 1 \text{ (mod 7)}$. Ahora, tanto *A* y *B* han determinado independientemente que la clave secreta Ks=1.

A pesar de la elegancia del algoritmo, el protocolo de intercambio de clave de Diffie-Hellman es vulnerable al *ataque activo del hombre en medio (figura 5.3)* porque no autentifica a los comunicantes, es decir, cuando *B* recibe la tripleta (7, 3, 1), cómo sabe que es de *A* y no de *Z*. Afortunadamente esto puede resolverse utilizando la firma digital y los certificados de autentificación, como lo hacen otros protocolos variantes de este.

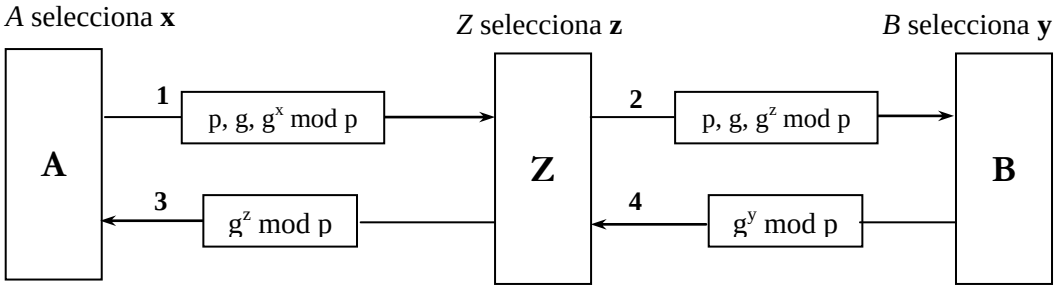


Figura 5.3 Ataque del hombre en medio [LIB 001]

VALIDACIÓN DE IDENTIFICACIÓN USANDO KERBEROS

Kerberos es un personaje de la mitología griega que por ser quien cuidaba las puertas del infierno, representa seguridad. Se podría decir que como servicio

de autentificación, ahora cuida las puertas de la red, impidiendo el ingreso a personas indeseadas.

Kerberos es un Protocolo de Validación de Identificación usado en muchos sistemas reales, tales como Unix, Linux y Windows 2000. Desarrollado en el MIT por Miller y Neuman en el contexto del Proyecto Athena en 1987 [WWW 015]. Se diseño para permitir a los usuarios de estaciones de trabajo el acceso a los recursos de la red de una manera segura, es decir, es una aplicación utilizada para asegurar la autentificación en comunicaciones Cliente/Servidor. Se encuentra en la Capa de Aplicación del Modelo de Referencia OSI[21] y trabaja sobre UDP[22] ya que requiere utilizar su propio protocolo para la asignación de secuencia y control de flujo. Actualmente se encuentra en la versión 5 y está definido en la RFC 1510.

Este protocolo usa fuertemente la criptografía simétrica, por tanto un cliente puede demostrar su identidad a un servidor (y viceversa) a través de una conexión de red insegura. Después de que un cliente y un servidor han usado Kerberos para verificar su identidad, ellos pueden encriptar todas sus comunicaciones para asegurar privacidad e integridad de datos.

En lugar de construir protocolos de autenticación en cada servidor, Kerberos provee un servidor de autenticación centralizado, cuya función es autenticar usuarios frente a servidores y servidores frente a usuarios.

COMPONENTES DE KERBEROS

Kerberos comprende tres servidores:

- Servidor de Autenticación (AS: Authentication Server): Verifica a los usuarios durante el establecimiento de la sesión inicial.

[21] Interconexión a Sistemas Abiertos
[22] User Datagram Protocol (Protocolo de Datagrama de Usuario)

- Servidor de Otorgamiento de Tickets (TGS: Ticket Granting Server): Emite Tickets de identidad comprobada. Este es el servicio que permitirá al cliente autenticarse con los servicios de la red.
- Servidor Final: es quien en realidad hace el trabajo solicitado, por ejemplo: servidor Web, FTP, correo, impresión, archivos, etc.

El TGS es lógicamente distinto del AS. A veces se refiere a ambos como KDC (Centro de Distribución de Claves).

Cada usuario y cada servidor final tendrán una clave, y Kerberos tiene una base de datos que las contendrá a todas. En el caso de ser de un usuario, su clave será derivada de su contraseña y estará encriptada, mientras que en el caso del servidor, la clave se generará aleatoriamente. Los servicios de red que requieren autenticación y los usuarios que requieran estos servicios, se deben registrar con Kerberos. Las claves privadas se negocian cuando se registran.

Como Kerberos sabe todas las claves privadas, puede crear mensajes que convenzan a un servidor de que un usuario es realmente quien dice ser y viceversa. La otra función de Kerberos es generar las claves de sesión, que serán compartidas entre un cliente y un servidor, y nadie más. La clave de sesión podrá ser usada para encriptar mensajes que serán intercambiados entre ambas partes. El almacenamiento de la base de datos y la generación de claves, se lleva a cabo en el Servidor de Autenticación (AS).

Hay dos tipos de credenciales que se utilizan en el modelo de autenticación de Kerberos: Los Tickets y Autenticadores. Aunque ambos se basan en encriptado de clave privada, se encriptan con claves diferentes. El *ticket* se usa para pasarle al servidor final la identidad de la persona para la que fue emitido. El *autenticador* es una prueba de que el ticket fue creado para el

usuario y no fue robado; contiene información que al ser comparada contra la que está en el ticket prueba que el usuario que lo presenta es el mismo al que le fue emitido.

Notación:

C = cliente

S = servidor

Addr = dirección de red del cliente

K_x = Clave privada de x

$K_{x, y}$ = Clave de sesión de x e y

{info}K_x = Información encriptada con la clave privada de x

$T_{x, y}$ = Ticket de x para usar y

A_x = Autenticador para x

Conexión Inicial

Cuando el usuario ingresa a una estación de trabajo, lo único que puede probar su identidad es su contraseña. Al usuario se le pide el nombre de usuario el cual consta de tres componentes:

1. El nombre del usuario
2. La Instancia.- Se usa para determinar los privilegios, puede ser root o admin. En caso de un usuario normal se obvia.
3. Dominio.- Es el nombre de una entidad administrativa que mantiene datos de autenticación

Los nombres de usuario se forman como nombre.instancia@dominio, por ejemplo: Ivan.root@fica.utn.edu.ec ; Francisco@fica.utn.edu.ec

Una vez obtenido, se envía una solicitud al AS que consiste en el nombre del usuario y el nombre de un servicio especial llamado *ticket-granting service*, el cual se encuentra en el TGS. El AS chequea la información sobre el cliente. Si sabe quién es, genera una clave de sesión aleatoria que se usará luego entre el cliente y el TGS. Luego crea un ticket para el TGS que se llama *ticket-granting ticket* (TGT) y que deberá ser presentado al TGS cada vez que se solicite un servicio.

$$TGT = \{T_{C,TGS}\}K_{TGS} = \{C, TGS, Addr, TimeStamp, LifeTime, K_{C,TGS}\}K_{TGS}$$

El TGT, al igual que cualquier otro ticket tiene 6 componentes:

1. Nombre del cliente
2. Nombre del servidor (en este caso TGS)
3. Dirección de red
4. Sello de tiempo (TimeStamp)
5. Tiempo de vida (LifeTime)
6. Clave de sesión

Esta información se encripta con la clave privada del TGS, que sólo conocen el TGS y el AS. El AS envía el ticket al cliente, junto con una copia de la clave de sesión. La respuesta que recibe este, está encriptada con la clave privada del cliente, que se deriva de la contraseña del usuario y que conocen sólo el AS y el cliente.

Una vez que la respuesta ha sido recibida por el cliente, se le pide la contraseña al usuario. La contraseña se convierte a una clave DES y se usa para desencriptar la respuesta del AS. El ticket y la clave de sesión se guardan para usar en el futuro, mientras que la contraseña del usuario y la clave DES se borran de la memoria. Así, si el cliente posee el ticket es porque el usuario conocía la contraseña correcta y por tanto debe ser quien dice ser.

Un ticket sirve para un solo servidor y para un solo cliente pero una vez emitido, puede ser utilizado muchas veces por el cliente para tener acceso a ese servidor, hasta que el ticket expira.

Solicitud de Tickets

Como un ticket sirve para un solo servidor, es necesario obtener un nuevo ticket para cada servicio que el usuario quiera utilizar. Los tickets para los distintos servidores se pueden obtener a través del TGS. La idea del TGS surge para evitar que el usuario deba ingresar su contraseña más de una vez.
Cuando un programa requiere un ticket que no ha sido solicitado aún, envía una solicitud al TGS. La solicitud contiene el nombre del servidor para el que se solicita el ticket, el TGT y un autenticador. A diferencia del ticket, el autenticador sólo se puede utilizar una vez. De lo contrario, sería posible robar el ticket y el autenticador juntos y hacer el "replay" de todas formas. Se debe generar uno nuevo cada vez que el cliente quiere utilizar un servicio, pero esto no presenta un problema porque como el cliente tiene toda la información necesaria, puede construir el autenticador por sí mismo.

Un autenticador contiene el nombre del cliente, la dirección IP de la estación de trabajo, y el tiempo actual, y está encriptado con la clave de sesión que es parte del ticket; en este caso, la que comparten el cliente y el TGS.

El TGS chequea el autenticador y el TGT. Si son válidos genera una nueva clave de sesión aleatoria a ser usada entre el cliente y el nuevo servidor. Luego construye un ticket para el nuevo servidor con el nombre del cliente, el nombre del servidor, el tiempo actual, la dirección IP del cliente y la nueva clave de sesión del servicio.

Luego, el TGS envía el ticket junto con la clave de sesión nuevamente al cliente. Esta vez, sin embargo, la respuesta está encriptada con la clave de sesión que había en el TGT. De este modo, no hay necesidad para el usuario de volver a ingresar su contraseña una vez más.

Solicitud de Servicios

Cuando el cliente tiene el ticket para el servidor deseado lo envía al servidor junto con el autenticador. Una vez que el servidor recibió el ticket y el autenticador, desencripta el ticket con su clave privada. Si éste no expiró usa la clave de sesión que se encuentra dentro para desencriptar el autenticador y compara la información del ticket con la del autenticador. Si todo está bien, permite el acceso porque después de este intercambio, el servidor está seguro que de acuerdo a Kerberos, el usuario es quien dice ser.

Se asume que los relojes están sincronizados. Si el tiempo de la solicitud está muy lejos en el futuro o en el pasado, el servidor considera que la solicitud es un intento de hacer un "replay" de una solicitud anterior. Además, el servidor mantiene un registro de todas las solicitudes pasadas con sellos de tiempo aún válidos y si recibe una solicitud con el mismo ticket y el mismo sello de tiempo que una anterior, la descarta.

Síntesis del Funcionamiento de Kerberos

a. Solicitud de un ticket de acceso
b. Ticket + clave de sesión
c. Solicitud de un ticket de acceso al servicio
d. Ticket + clave de sesión

e. Solicitud de servicio

f. Autenticador del servidor

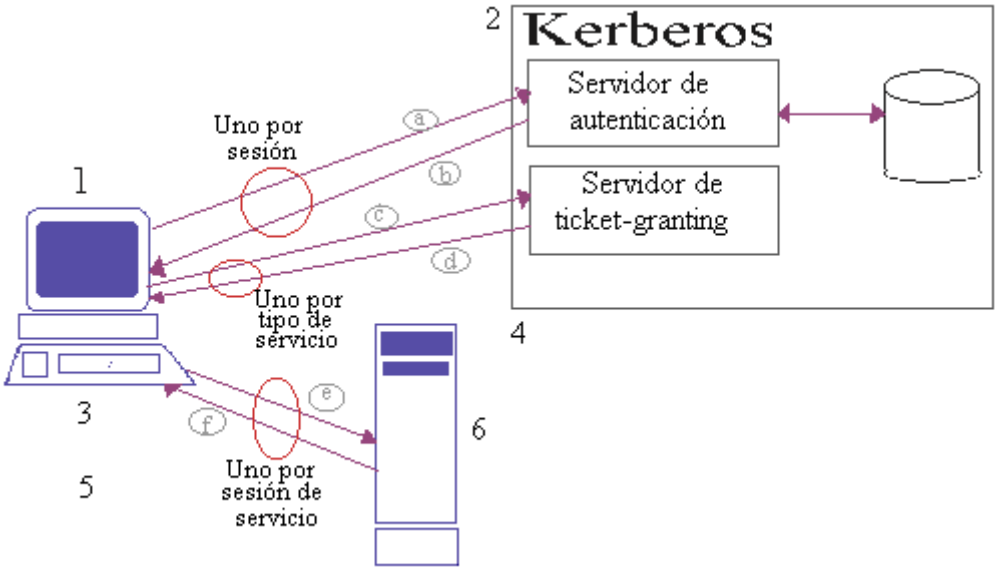


Figura 5.4 Funcionamiento de Kerberos [WWW 010]

1. Un usuario desde una workstation requiere un servicio.
2. AS verifica el correcto acceso del usuario a la Base de Datos, crea un ticket y una clave de sesión. Los resultados son encriptados usando la clave derivada del password del usuario.
3. La workstation solicita el password al usuario y la utiliza para desencriptar el mensaje, luego envía al TGS el ticket y el autenticador que contienen el nombre de usuario, la dirección de red y el tiempo de vida del ticket.
4. El TGS desencripta el ticket y el autenticador, verifica la solicitud y crea un ticket para ser enviado al servidor.

5. La workstation envía el ticket y el autenticador al servidor.
6. El servidor verifica que el ticket y el autenticador coincidan, luego permite el acceso al servicio.

Limitaciones de Kerberos

- Kerberos no tiene efectividad frente al ataque de diccionario. Si el usuario escoge una contraseña pobre, un atacante que la consiga tratando de adivinarla puede hacerse pasar por él.
- Kerberos requiere un camino seguro para ingresar las contraseñas. Si la comunicación entre el usuario y el programa de autenticación inicial puede ser monitoreada, entonces el atacante puede llegar a obtener suficiente información para hacerse pasar por el usuario.
- No hay un lugar seguro donde guardar las claves de sesión. De hecho, el lugar donde se guardan puede ser accedido por el root. Así es que un intruso que logre crackear el mecanismo de protección de la computadora local podrá robar las claves de sesión.
- El protocolo liga los tickets a las direcciones de red y esto es un problema en hosts con más de una dirección IP.
- Tiempo de vida de un ticket. La elección del tiempo de vida de los tickets no es trivial. Si se elige un tiempo de vida para los tickets muy largo, y un usuario desprevenido olvida "desloguearse" de una máquina, otra persona puede tomar su lugar. Por otro lado, si el tiempo de vida de los tickets es muy corto, el usuario va a ser molestado cada cierto tiempo para que ingrese nuevamente su contraseña.
- Manejo de proxies. Todavía no está claro cómo permitir autenticación mediante proxies, es decir que un servidor utilice servicios de otros servidores en nombre de un usuario autenticado.

5.2 AUTENTIFICACIÓN DE MENSAJES

Las técnicas de encriptación protegen de las agresiones pasivas, es decir, de las escuchas, pero no protege de las agresiones activas como la falsificación de los datos. La protección contra estas agresiones activas se conoce como autentificación de mensajes y los llevan a cabo los algoritmos de hashing (también llamados fingerprint, checksum o digesto de mensaje).

Un mensaje, fichero u otra colección de datos se dice que está autentificada cuando son genuinos y vienen del origen pretendido. Los dos aspectos importantes son verificar que el contenido del mensaje no haya sido alterado durante el transcurso y que el origen del mensaje es auténtico.

En la *figura 5.5* se muestra una forma de autentificar mensajes utilizando encriptado simétrico.

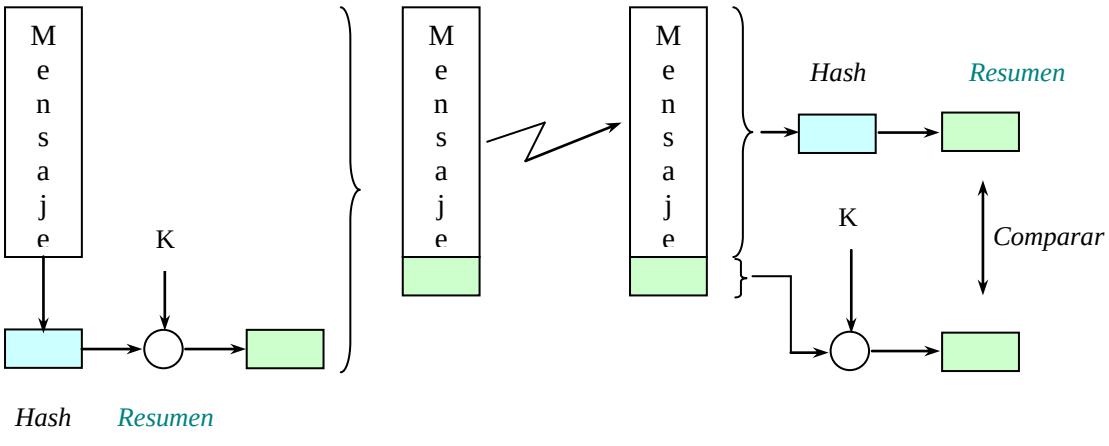


Figura 5.5 Autentificación de Mensajes [LIB 002]

Las funciones hash (H) deben cumplir con una serie de requisitos:

1. H puede ser aplicada a bloques de datos de cualquier tamaño.
2. H produce una salida de longitud fija.
3. H(x) es relativamente fácil de calcular para un *x* dado.
4. Para un resumen de mensaje *m* dado, es imposible computacionalmente encontrar un *x* tal que H(x)=m.
5. Para un bloque dado *x*, es imposible computacionalmente encontrar un $y \neq x$ con H(y)=H(x).
6. Es imposible computacionalmente encontrar una pareja (x, y) tal que H(x)=H(y)

Las tres primeras propiedades son requisito para la aplicación práctica de una función hash. La cuarta propiedad es la propiedad de *un solo sentido,* es decir, es fácil generar un código dado un mensaje, pero virtualmente imposible generar un mensaje dado un código. La quinta propiedad garantiza que no se puede encontrar un mensaje alternativo que produzca el mismo valor de un mensaje dado. La sexta propiedad protege de un ataque sofisticado conocido como ataque de cumpleaños[23].

Se ha propuesto una variedad de funciones hash. Las más sobresalientes y de mayor uso son: El MD5 y el SHA.

5.2.1 ALGORITMO MD5 (Message Digest 5)

El algoritmo de compendio de mensaje MD5, especificado en el RFC 1321, fue desarrollado por Ron Rivest en el MIT. El algoritmo toma como entrada un mensaje de longitud arbitraria y produce un extracto o resumen del mensaje de 128 bits. La entrada se procesa en bloques de 512 bits. En la

[23] Ver [LIB 001] para una discusión sobre Ataque de Cumpleaños .

figura 5.6 se muestra el procesamiento general de un mensaje para producir un resumen. Este procesamiento consta de 5 pasos que se detallan a continuación:

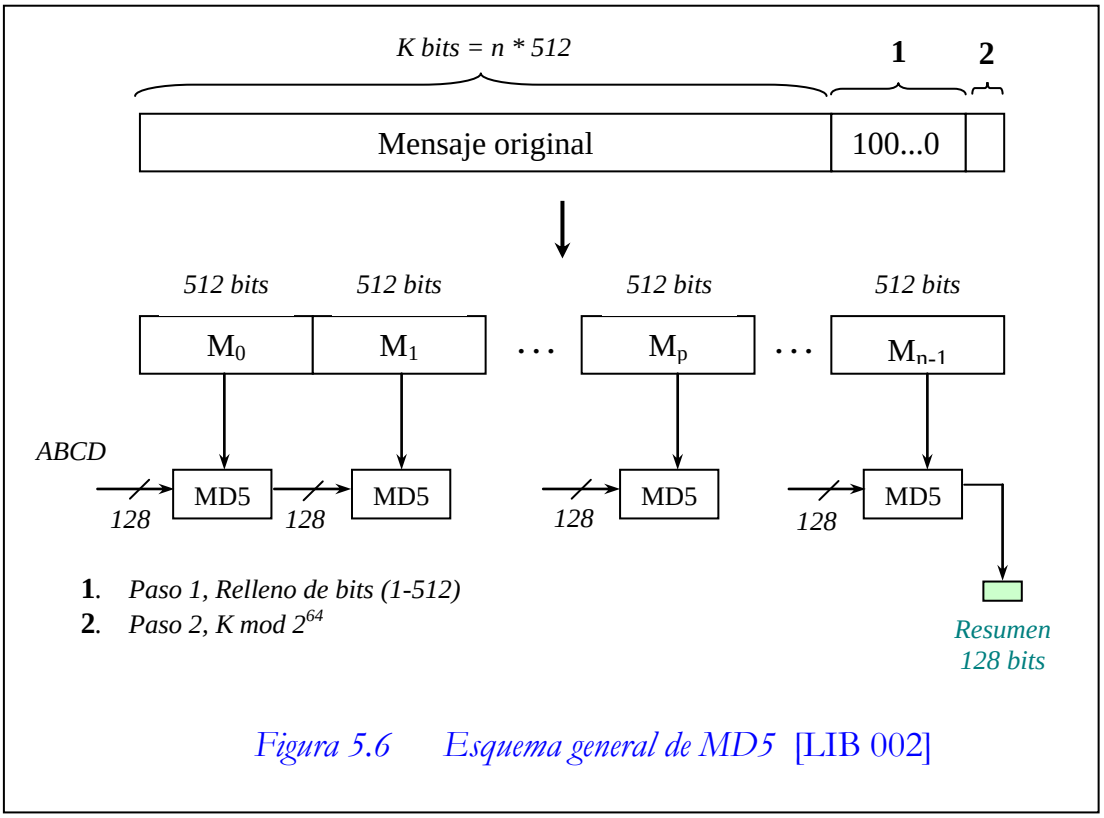


Figura 5.6 Esquema general de MD5 [LIB 002]

1. Adición de bits de relleno

Se incorporan al mensaje *b* bits de relleno para que su longitud en bits sea congruente con 448 módulo 512, es decir, longitud = 448 mod 512. Siempre se incorpora el relleno, incluso si el mensaje tiene ya la longitud deseada. Así *b* está en el rango de 1 a 512. De esos *b* bits, el primer bit es 1 y el resto son 0.

2. Adición de la longitud

Al resultado del paso 1 se le añade una representación de 64 bits de la longitud en bits del mensaje original (antes de añadir los *b* bits de relleno).

Si la longitud original del mensaje es mayor que 2^{64}, entonces solamente se usan los 64 bits menos significativos. De esta manera, el campo contiene la longitud del mensaje original módulo 2^{64}.

La salida de los dos pasos anteriores produce un mensaje cuya longitud es múltiplo de 512 y es igual a n*512 bits. En la figura, el mensaje ampliado se representa como una secuencia de bloques de 512 bits denotados como $M_0, M_1, ...,M_{n-1}$.

3. Inicializar la memoria temporal MD

Se utiliza una memoria temporal de 128 bits para almacenar los resultados intermedios. La memoria temporal se representa con cuatro registros de 32 bits cada uno y son: A, B, C y D. Estos registros se inicializan a los siguientes valores hexadecimales:

A	=	0	1	2	3	4	5	6	7
B	=	8	9	*A*	*B*	*C*	*D*	*E*	*F*
C	=	*F*	*E*	*D*	*C*	*B*	*A*	9	8
D	=	7	6	5	4	3	2	1	0

4. Procesar el mensaje en bloques de 512 bits

Como se muestra en la *figura 5.7*, el algoritmo es un módulo que consta de cuatro segmentos de procesamiento. Cada segmento tiene una estructura similar pero cada una utiliza una función lógica primitiva diferente conocidas como ***F***, ***G***, ***H*** e ***I*** . Cada función primitiva tienen como entrada tres palabras de 32 bits y como salida una palabra de 32 bits, y se definen a continuación:

$$\mathbf{F(}X, Y, Z\mathbf{)} = (X \mathbf{.} Y) \mathbf{+} (X\mathbf{'} \mathbf{.} Z)$$

$$\mathbf{G(}X, Y, Z\mathbf{)} = (X \mathbf{.} Z) \mathbf{+} (Y \mathbf{.} Z\mathbf{'})$$

$$\mathbf{H(}X, Y, Z\mathbf{)} = X \oplus Y \oplus Z$$

$$\mathbf{I(}X, Y, Z\mathbf{)} = Y \oplus (X \mathbf{+} Z\mathbf{'})$$

donde los símbolos (**.**, **+**, **'**,⊕) representan los operadores lógicos (and, or, not, xor) respectivamente.

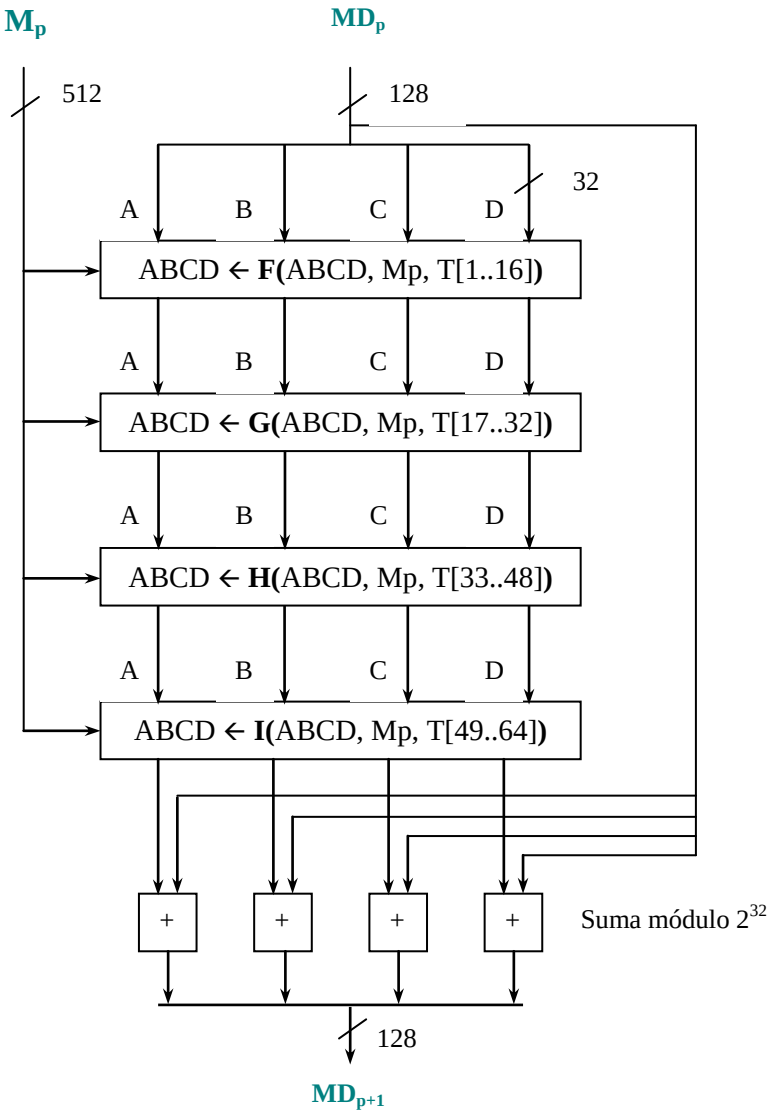


Figura 5.7 Procesamiento MD5 de un bloque de 512 bits [LIB 002]

Observe que cada segmento toma como entrada el bloque de 512 bits que se está procesando (M_p) y el valor de la memoria temporal de 128 bits

ABCD que se actualiza al final de cada segmento. Además, cada segmento también hace uso de un cuarto de una tabla de 64 elementos **T**[1 ... 64] construida a partir de la función seno. El i-ésimo elemento de T, denotado como T[i] tiene un valor igual a la parte entera de $2^{32} * abs(sen(i))$, donde *i* se expresa en radianes. Como *abs(sen(i))* es un número comprendido entre 0 y 1, cada elemento de T es un entero que se puede representar con 32 bits.

La salida de los cuatro segmentos se suma a MD_p para producir MD_{p+1}. Esta suma se hace independientemente para cada una de las palabras almacenadas en la memoria temporal con cada palabra correspondiente en MD_p, utilizando suma módulo 2^{32}.

5. Salida

Después de que los *n* bloques de 512 bits se han procesado, el resumen del mensaje de 128 bits es la salida de la etapa n-ésima.

En los últimos tiempos el algoritmo MD5 ha mostrado ciertas debilidades, aunque sin implicaciones prácticas reales, por lo que se sigue considerando en la actualidad un algoritmo seguro, si bien su uso tiende a disminuir.

5.2.2 ALGORITMO SHA (Secure Hash Algorithm)

El algoritmo de compendio de mensaje SHA fue inventado en 1994 por la Agencia de Seguridad Nacional (NSA). Esto ha llevado a mucha desconfianza hacia este algoritmo, pero han sido analizados por expertos y no se han encontrado "puertas traseras".

SHA genera un hash de 160 bits de longitud, por lo que parece ser un algoritmo mucho más seguro que MD5. No se conoce ningún ataque criptográfico contra SHA, por lo que la única manera de atacarlo sería con un ataque de fuerza bruta. Hay dos formas de ataques de fuerza bruta contra las funciones hash. Uno consiste en, dado un hash $h=H(M)$, encontrar un mensaje M' tal que $H(M) = H(M')$. El otro ataque es denominado "ataque de cumpleaños", que consiste en buscar dos mensajes aleatorios, M y M', que produzcan el mismo hash (lo que se denomina una colisión).

El proceso para calcular el hash comienza rellenando el mensaje para hacer que tenga una longitud múltiplo de 512. Este rellenado es como el de MD5: se añade un 1, después los ceros necesarios hasta que sólo falten 64 bits para que el mensaje sea múltiplo de 512, y por último se añade la longitud del mensaje antes del rellenado en formato de 64 bits.

El algoritmo es similar a MD5, y se inicializa igual que éste, solo que con cinco registros de 32 bits en lugar de cuatro: Estos registros se inicializan a los siguientes valores hexadecimales:

A	=	6	7	4	5	2	3	0	1
B	=	*E*	*F*	*C*	*D*	*A*	*B*	8	9
C	=	9	8	*B*	*A*	*D*	*C*	*F*	*E*
D	=	1	0	3	2	5	4	7	6
E	=	*C*	3	*D*	2	*E*	1	*F*	0

Comienza ahora el bucle principal, que va tomando bloques de 512 bits del mensaje hasta que se acaban. En primer lugar, se copian las variables en otras que representaremos como *a*, *b, c*, *d* y *e*. Así mismo, el bloque de 512 bits pasa de estar formado por 16 palabras de 32 bits ($M_0...M_{15}$) a estarlo por 80 palabras de 32 bits ($W_0...W_{79}$) aplicando las siguientes operaciones:

$w_t = m_t$ para t = 0 ... 15

$w_t = (w_{t-3} \oplus w_{t-8} \oplus w_{t-14} \oplus w_{t-16}) <<< 1$ para t = 16 ... 79

La operación <<< *n* es un desplazamiento circular a la izquierda de n bits.

Cada bucle está formado por cuatro rondas con 20 operaciones cada una. Cada operación es una función no lineal, utilizando cada una tres de las variables como entrada, y son:

G(X, Y, Z**)** = (X **.** Y) **+** (X**'** **.** Z)	para 0 ≤ t ≤ 19
H(X, Y, Z**)** = X ⊕ Y ⊕ Z	para 20 ≤ t ≤ 39 y 60 ≤ t ≤ 79
I(X, Y, Z**)** = (X **.** Y) **+** (X **.** Z) **+** (Y **.** Z)	para 40 ≤ t ≤ 59

donde los símbolos (**.**, **+**, **'**,⊕) representan los operadores lógicos (and, or, not, xor) respectivamente.

La operación ***G*** se emplea en la primera ronda, la ***H*** en la segunda y en la cuarta, y la ***I*** en la tercera. Además se emplean cuatro constantes, una para cada ronda:

K_0 = 5A827999	para 0 ≤ t ≤ 19
K_1 = 6ED9EBA1	para 20 ≤ t ≤ 39
K_2 = 8F1BBCDC	para 40 ≤ t ≤ 59
K_3 = CA62C1D6	para 60 ≤ t ≤ 79

Con todo esto, si ***i*** es el número de la ronda y W_t el subbloque t-ésimo, el bucle principal es el siguiente:

```
FOR  t = 0  TO 79
        i = t div 20            /*  devuelve el cociente  */
        Tmp = (a <<< 5) + F_t(b, c, d) + e + w_t + K_i
        e = d
        d = c
        c = b <<< 30
        b = a
        a = Tmp
```

siendo F_t la función ***G***, ***H*** o ***I*** según el valor de ***t***. Después de todas las operaciones, *a*, *b*, *c*, *d* y *e* son sumadas respectivamente a *A*, *B*, *C*, *D* y *E*, y el algoritmo continúa con el siguiente bloque de 512 bits. La salida final es la concatenación de *A*, *B*, *C*, *D* y *E*.

La NSA introdujo el desplazamiento a la izquierda para corregir una debilidad del algoritmo, pero no ha dado explicaciones sobre la naturaleza de esta debilidad.

5.3 FIRMAS DIGITALES

Las firmas digitales son un reemplazo a las firmas manuscritas con tinta y papel, por lo tanto, la firma digital tiene el mismo valor legal que una firma holográfica tradicional (en los países que poseen una ley de firma digital). Básicamente lo que pretende es *"firmar"* el mensaje que se va a enviar a otra parte de modo que:

1. El receptor pueda verificar la identidad proclamada del transmisor.
2. El transmisor no pueda repudiar después el contenido del mensaje.

3. El receptor no haya podido confeccionar el mensaje, él mismo.

Las firmas digitales son generadas utilizando un algoritmo de clave pública. Su procedimiento se muestra en la *figura 5.8*.

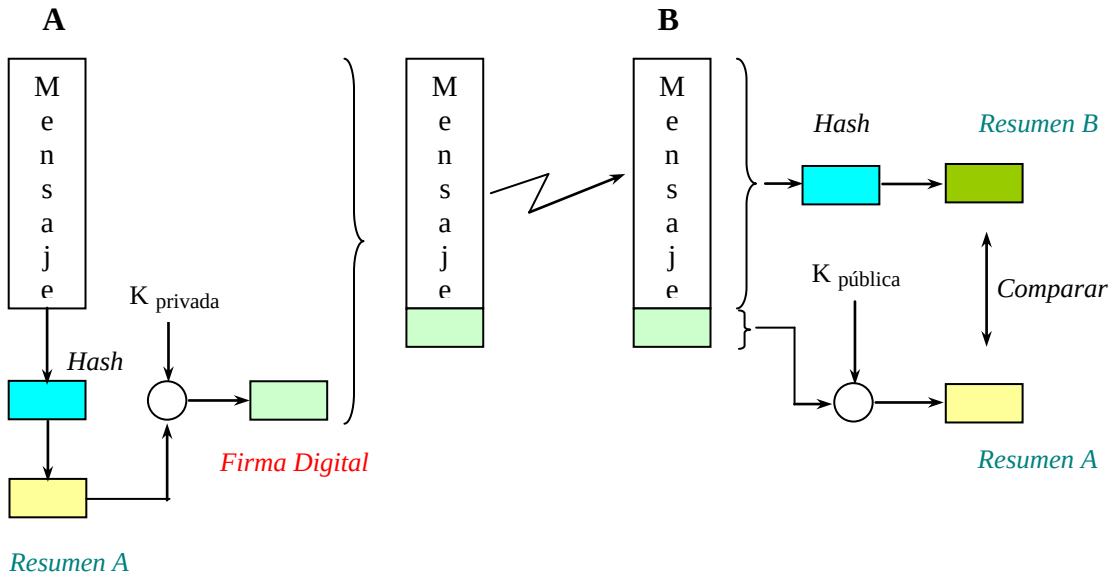


Figura 5.8 Firma Digital [LIB 002]

El procedimiento que se utiliza es el siguiente:

1. *A* aplica una función hash H al mensaje M, esto es H(M).
2. *A* cifra el resumen H(M) con su clave privada Kp, esto es, Ekp(H(M)). Este resultado es su firma digital.
3. *A* envía el mensaje M junto con el resumen firmado (firma digital) a *B*.

La verificación de la firma digital es como sigue:

4. *B* genera un resumen del mensaje M recibido de *A*, usando la misma función hash.

5. *B* descifra con la clave pública de *A* el resumen firmado (firma digital de *A*).
6. Si los resultados obtenidos en los pasos 4 y 5 coinciden, entonces *B* puede tener la certeza de que el mensaje no ha sido modificado en su transcurso por la red y que el mensaje ha sido remitido, efectivamente, por *A*.

De esta forma se ofrecen conjuntamente los servicios de ***no repudio***, ya que nadie excepto *A* podría haber firmado el mensaje con su clave privada, y de ***autentificación***, ya que si el mensaje viene firmado por *A*, podemos estar seguros de su identidad, dado que sólo ella ha podido firmarlo. En último lugar, mediante la firma digital se garantiza la ***integridad*** del mensaje, ya que en caso de ser modificado, resultaría imposible hacerlo de forma tal que se generase la misma función de resumen que había sido firmada.

5.3.1 DSS (DIGITAL SIGNATURE STANDARD)

El DSS es un sistema de firma digital adoptado como estándar por el NIST[24]. Utiliza la función Hash **SHA** para la generación del compendio del mensaje y el algoritmo asimétrico **DSA** para generar y verificar la firma digital.

DSA (DIGITAL SIGNATURE ALGORITHM)

El Algoritmo de Firma Digital (DSA) es apropiado para aplicaciones que requieren una digital firma en lugar de la firma escrita. La firma digital DSA consiste en un par de números grandes representados en una computadora como dos cadenas de dígitos binarios. La firma digital se computa usando un grupo de reglas (es decir, el DSA) y un grupo de parámetros tal que la

identidad del signatario y la integridad de los datos puedan verificarse. El DSA proporciona la capacidad para generar y verificar las firmas.
La comprobación de la firma digital hace uso de una llave pública que corresponde a la llave privada. Cualquiera puede verificar la firma de un usuario empleando la llave pública de ese usuario. La generación de la firma sólo puede realizarse por el poseedor de la llave privada del usuario.

El DSA puede ser usado en el correo electrónico, transferencia de fondos electrónicos, intercambio de datos electrónicos, distribución de software, almacenamiento de datos, y otras aplicaciones que requieren convicción de integridad y autenticación de los datos.

PARÁMETROS DSA

El DSA hace uso de los siguientes parámetros:

1. ***p*** = un módulo primo, donde $2^{L-1} < p < 2^{L}$ para $512 =< L =< 1024$ y L es múltiplo de 64
2. ***q*** = un divisor primo de p - 1, donde $2^{159} < q < 2^{160}$
3. ***g*** = $h^{(p-1)/q} \pmod p$, donde ***h*** es cualquier entero con $1 < h < p - 1$ tal que $h^{(p-1)/q} \pmod p > 1$
4. ***x*** = un número entero aleatorio con $0 < x < q$
5. ***y*** = $g^{x} \pmod p$
6. ***k*** = un número entero aleatorio con $0 < k < q$

Los enteros ***p***, ***q***, y ***g*** son públicos y pueden ser común a un grupo de usuarios. Las llaves privadas y públicas de un usuario son ***x*** y ***y***, respectivamente. Ellos son normalmente fijos para un periodo de tiempo.

[24] National Institute of Standards and Technology (Instituto Nacional de Estándares y Tecnología)

Los parámetros ***x*** y ***k*** sólo se usan para la generación de la firma, y deben ser confidenciales. El parámetro ***k*** debe generarse para cada firma.

GENERACIÓN DE LA FIRMA DIGITAL

La firma digital de un mensaje ***M*** consiste en el par de números ***r*** y ***s*** de 160 bits cada uno, calculados de acuerdo a las siguientes ecuaciones:

$$r = (g^k \bmod p) \bmod q$$

$$s = (k^{-1}(SHA(M) + xr)) \bmod q.$$

Donde k^{-1} es el inverso multiplicativo de k (mod q), es decir, $(k^{-1}\ k) \bmod q = 1$ y $0 < k^{-1} < q$. El valor de SHA(M) es una cadena de 160-bit.

Como una opción, se puede verificar: Si $r = 0$ ó $s = 0$, un nuevo valor de ***k*** deberá ser generado y la firma deberá ser recalculada.

La firma se transmite junto con el mensaje ***M*** al verificador, tal como se muestra en la *figura 5.9*.

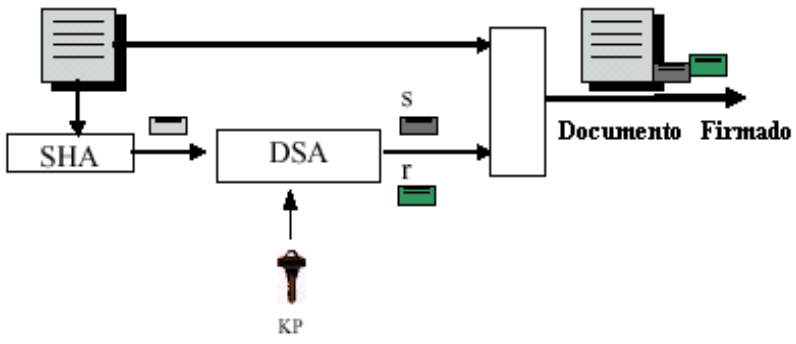


Figura 5.9 Generación de Firma DSA

VERIFICACIÓN DE LA FIRMA DIGITAL

Antes de verificar la firma de un mensaje firmado, los parámetros ***p***, ***q***, ***g*** y la llave pública del remitente deben estar disponibles al verificador de la firma de una manera autenticada.

Los parámetros ***M'***, ***r'*** y ***s'*** son las versiones recibidas de ***M***, ***r***, y ***s***, respectivamente, y el parámetro ***y*** es la clave pública del signatario. El verificador primero chequea que $0 < r' < q$ y $0 < s' < q$; si cualquier condición es falsa, la firma se rechazará. Si se cumple las dos condiciones, se procede a verificar la firma de la siguiente manera:

$w = (s')^{-1} \bmod q$

$u1 = ((SHA(M')w) \bmod q$

$u2 = ((r')w) \bmod q$

$v = (((g)^{ul} (y)^{u2}) \bmod p) \bmod q.$

Si v = r', entonces el verificador puede tener la certeza que el mensaje recibido es el original y no ha sido alterado en el transcurso.

Si v ≠ r', entonces el mensaje ha sido modificado, o ha sido firmado por un impostor. El mensaje es considerado inválido.

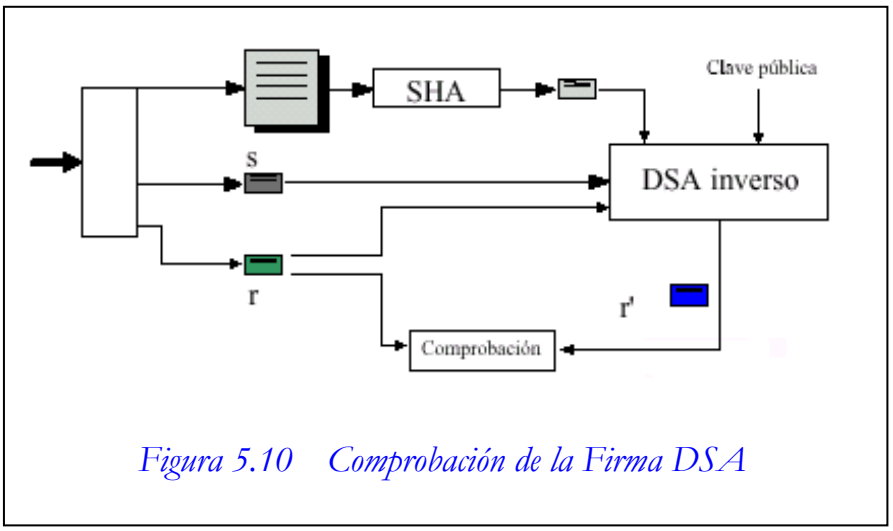


Figura 5.10 Comprobación de la Firma DSA

Un ejemplo didáctico sería:

Parámetros DSA

Suponemos que la función de resumen genera el valor SHA(M)=8

1. Seleccionamos p = 23
2. Escogemos q = 11, puesto que q es un factor de (p-1)
3. Escogemos h = 5, puesto que cumple:
   - $1 < h < (p - 1) \quad \Rightarrow \quad 1 < 5 < (23 - 1)$
   - $h^{(p-1)/q} \pmod p > 1 \quad \Rightarrow \quad 5^2 \pmod{23} > 1 \quad \Rightarrow \quad 2 > 1$
4. $g = h^{(p-1)/q} \pmod p = 2$
5. Escogemos x = 3, puesto que: $0 < x < q \Rightarrow 0 < 3 < 11$
6. $Y = g^x \pmod p$

   $Y = 2^3 \pmod{23}$

   $Y = 8$
7. Escogemos k = 6, puesto que: $0 < k < q \Rightarrow 0 < 6 < 11$

Generación de la Firma Digital

$r = (g^k \bmod p) \bmod q$

$r = (2^6 \bmod 23) \bmod 11$

$r = 18 \bmod 11$

r = 7

$k * k^{-1} = 1 \pmod q$

$6 * k^{-1} = 1 \pmod{11}$

$k^{-1} = 2$

$s = (k^{-1} (SHA(M) + xr)) \bmod q$

$s = (2\ (8 + 3 * 7))\ \text{mod}\ 1$

$s = (2 * 29)\ \text{mod}\ 11$

$s = 3$

Verificación de la Firma Digital

Como $0 < r < q$ y $0 < s < q$ entonces procedemos a calcular los siguientes valores:

$s * s^{-1} = 1\ (\text{mod}\ q)$
$3 * s^{-1} = 1\ (\text{mod}\ 11)$
$s^{-1} = 4$

$w = s^{-1}\ \text{mod}\ q$

$w = 4\ \text{mod}\ 11$

$w = 4$

$u1 = (SHA(M)w)\ \text{mod}\ q$

$u1 = (8 * 4)\ \text{mod}\ 11$

$u1 = 10$

$u2 = (r * w)\ \text{mod}\ q$

$u2 = (7 * 4)\ \text{mod}\ 11$

$u2 = 6$

$v = ((g^{u1} * y^{u2})\ \text{mod}\ p)\ \text{mod}\ q$

$v = ((2^{10} * 8^{6})\ \text{mod}\ 23)\ \text{mod}\ 11$

$v = ((12 * 13)\ \text{mod}\ 23)\ \text{mod}\ 11$

$v = (156\ \text{mod}\ 23)\ \text{mod}\ 11$

$v = 18\ \text{mod}\ 11$

v = 7

Puesto que v = r, el documento se considera auténtico.

5.4 SISTEMAS DE IDENTIFICACIÓN BIOMÉTRICA

La identificación por medio de sistemas biométricos es una de las tecnologías más prometedoras e inquietantes y se perfilan como la futura llave que nos abrirá todas las puertas. Nuestras características físicas, únicas y distintas de las de cualquier otro ser humano, tales como las huellas dactilares, geografía de la mano, reconocimiento facial, del iris, de la voz o del ADN se convertirán en los nuevos passwords de entrada a múltiples sistemas, desde el acceso a cuentas bancarias, vehículos, áreas laborales y archivos informáticos hasta, ¿por qué no?, a nuestro propio domicilio.

Existen varios sistemas para verificar la identidad de un individuo, sin embargo la biometría se considera como el método más apropiado, ya que ciertos rasgos de cada persona, son inherentes a ella y sólo a ella. La biometría permite una autentificación segura, al contrario que el empleo de contraseñas o tarjetas, que pueden ser robadas y utilizadas por personas no autorizadas.

La identificación biométrica se puede efectuar a partir de diferentes elementos, tales como:

- Huellas dactilares.
- Dibujo papilar de la mano o de los dedos.

- Forma o rasgos de la cara.
- Características de la voz.
- Dibujo del sistema venoso de la retina o fondo del ojo.
- Informaciones genéticas.
- Dinámica de la firma, etc.

Los Sistemas biométricos son sistemas automatizados capaces de: capturar una muestra biométrica de una persona; extractar datos biométricos de la muestra; comparar estos datos con los datos de uno o más patrones referenciales almacenados en una base de datos; decidir el nivel de correspondencia de la comparación; indicar si la identificación o verificación de identidad se ha realizado.

Los sistemas biométricos se componen de hardware y software; el primero captura la característica concreta del individuo y el segundo interpreta la información y determina su aceptabilidad o rechazo, todo en función de los datos que han sido almacenados por medio de un registro inicial de la característica biométrica que mida el dispositivo en cuestión.

Ese registro inicial o toma de muestra es lo que determina la eficacia del sistema. En el caso de las huellas dactilares, un usuario coloca el dedo (por lo general el índice) en un sensor que hace la lectura digital de su huella, después, el programa guardará la información como un modelo; la próxima vez que ese usuario intente acceder al sistema deberá repetir la operación y el software verificará que los datos corresponden con el modelo, como el de la *figura 5.11*. El mismo principio rige para la identificación por el iris/retina, el rostro, la mano completa, etc.

Figura 5.11 Lectura Digital de la Mano

Las tasas de exactitud en la verificación dependen en gran medida de dos factores:

1. El cambio que se puede producir en las personas, debido a accidentes o al envejecimiento.
2. Las condiciones ambientales, como la humedad en el aire, suciedad y sudor, en especial en la lectura que implique el uso de las manos.

Cada sistema biométrico utiliza una cierta clase de interfaz, un sensor o mecanismo de captura determinado y un software específico.

La identificación por geometría de la mano o huellas digitales, la más extendida, crea una imagen digital tridimensional, que es capturada, calibrada y guardada en un archivo.

Para la identificación por el ojo existen dos sistemas: topografía del iris, identificando en pocos segundos más de 4.000 puntos, y topografía de la

retina, midiendo con luz infrarroja de baja intensidad 320 puntos predefinidos en el diagrama de las venas.

El reconocimiento facial compara las características faciales con una imagen previamente escaneada, lo mismo que la identificación por voz con un patrón pregrabado, que analiza la presión del aire y las vibraciones sobre la laringe.

La identificación por firma mide el tiempo, la presión, la velocidad, el ángulo de colocación del lápiz y la velocidad de las curvas, todo a través de un lápiz óptico con el que la persona firma en un soporte específico.

Con toda seguridad, la tecnología también llegará a desarrollar un sistema de identificación automática por el ADN y, de hecho, la viabilidad de sistemas basados en el análisis del ADN es una de las líneas de investigación abiertas en la actualidad.