Mecanismos de protección

Xavier Perramon

Índice

Introducción 5

Objetivos 6

1. Conceptos básicos de criptografía 7
1.1 Criptograía de clave simétrica 9
1.1.1 Algoritmos de cifrado en flujo 11
1.1.2 Algoritmos de cifrado en bloque 13
1.1.3 Uso de los algoritmos de clave simétrica 16
1.1.4 Funciones *hash* seguras 19
1.2 Criptografía de clave pública 22
1.2.1 Algoritmos de clave pública 22
1.2.2 Uso de la criptografía de clave pública 24
1.3 Infraestructura de clave pública (PKI) 26
1.3.1 Certificados de clave pública 26
1.3.2 Cadenas de certificados y jerarquías de certificación 29
1.3.3 Listas de revocación de certificados (CRL) 30

2. Sistemas de autenticación 32
2.1 Autenticación de mensaje 32
2.1.1 Códigos de autenticación de mensaje (MAC) 33
2.1.2 Firmas digitales 33
2.2 Autenticación de entidad 34
2.2.1 Contraseñas 35
2.2.2 Protocolos de reto-respuesta 43

3. Protección del nivel de red: IPsec 50
3.1 La arquitectura IPsec 51
3.2 El protocolo AH 53
3.3 El protocolo ESP 54
3.4 Modos de uso de los protocolos IPsec 55

4. Protección del nivel de transporte: SSL/TLS/WTLS 58
4.1 Características del protocolo SSL/TLS 59
4.2 El transporte seguro SSL/TLS 61
4.2.1 El protocolo de registros SSL/TLS 62
4.2.2 El protocolo de negociación SSL/TLS 63
4.3 Ataques contra el protocolo SSL/TLS 68
4.4 Aplicaciones que utilizan SSL/TLS 70

5. Redes privadas virtuales (VPN) 71

5.1 Definición y tipos de VPN 71
5.2 Configuraciones y protocolos utilizados en VPN 72

Resumen 75

Actividades 77

Ejercicios de autoevaluación 78

Soluciones 81

Glosario 83

Bibliografía 86

Introducción

Para proteger las redes de comunicaciones, la **criptografía** es la herramienta que nos permite evitar que alguien intercepte, manipule o falsifique los datos transmitidos. Dedicaremos la primera parte de este módulo a introducir los conceptos de criptografía necesarios para entender cómo se aplica a la protección de las comunicaciones.

La finalidad básica de la criptografía es el envío de información secreta. Si aplicamos una transformación, conocida como **cifrado**, a la información que queremos mantener en secreto, aunque un adversario consiga ver qué datos estamos enviando le serán completamente ininteligibles. Sólo el destinatario legítimo será capaz de realizar la transformación inversa y recuperar los datos originales.

Pero más allá de mantener la información en secreto, existen otros servicios que pueden ser igualmente necesarios, como, por ejemplo, la **autenticación**. Debemos evitar, de esta forma, que después de tomar todas las medidas necesarias para que sólo el destinatario final pueda leer la información, resulte que este destinatario sea un impostor que haya conseguido hacerse pasar por el auténtico destinatario. En la segunda parte del módulo veremos algunos sistemas para garantizar la autenticidad en las comunicaciones, la mayoría de ellas basadas en técnicas criptográficas.

En el resto de este módulo didáctico estudiaremos ejemplos de protocolos de comunicación que, aplicado los mecanismos anteriores, permiten proteger la información que se transmite entre ordenadores. Esta protección se puede obtener en distintos niveles de la arquitectura de comunicaciones. A **nivel red**, el mecanismo principal en un entorno de interconexión basado en IP es el conjunto de protocolos conocido como **IPsec**.

Alternativamente, se puede implementar la protección a **nivel de transporte**, aprovechando así la infraestructura IP existente, principalmente los encaminadores o *routers*. Como ejemplo de protección a nivel de transporte veremos la familia de protocolos **SSL/TLS/WTLS**.

Para finalizar el módulo, introduciremos la tecnología de **redes privadas virtuales** o **VPN**, que permite utilizar una red pública ampliamente extendida como es Internet para comunicaciones seguras, como si fuera una red privada dedicada.

Objetivos

Los conceptos presentados en el presente módulo didáctico deben permitir al estudiante alcanzar los siguientes objetivos:

1. Saber qué funciones nos ofrece la criptografía, tanto las técnicas de clave simétrica como las de clave pública.

2. Conocer los distintos algoritmos de cifrado, integridad y autenticación disponibles y sus posibles usos.

3. Conocer el uso de los certificados X.509 y las listas de revocación, su estructura y la utilidad de los distintos campos.

4. Reconocer la necesidad de los sistemas de autentificación, qué técnicas concretas utilizan, y cómo estas técnicas permiten contrarrestar los intentos de suplantación.

5. Comprender las posibilidades de protección de los protocolos de comunicación a distintos niveles, y en particular, el nivel de red y el de transporte.

6. Conocer los protocolos que forman la arquitectura IPsec, y qué protecciones ofrece cada uno de ellos.

7. Entender el mecanismo general de funcionamiento de los protocolos de transporte seguro SSL/TLS, y cómo estos protocolos pueden ser utilizados por otros niveles superiores, como HTTP o TELNET.

8. Introducir la tecnología de las redes privadas virtuales, y cómo se pueden usar para conectar intranets de forma segura mediante una red de acceso público, como es el caso de Internet.

1. Conceptos básicos de criptografía

A lo largo de la historia se han diseñado distintas técnicas para ocultar el significado de la información que no interesa que sea conocida por extraños. Algunas de ellas ya se usaban en tiempos de la antigua Grecia o del Imperio romano: por ejemplo, se atribuye a Julio César la invención de un cifrado para enviar mensajes cifrados que no pudieran ser interpretados por el enemigo.

Criptografía

Los términos **criptografía**, **criptología**, etc. provienen de la raíz griega *kryptós*, que significa "escondido".

La **criptografía** estudia, desde un punto de vista matemático, los métodos de protección de la información. Por otro lado, el **criptoanálisis** estudia las posibles técnicas utilizadas para contrarrestar los métodos criptográficos, y es de gran utilidad para ayudar a que estos sean más robustos y difíciles de atacar. El conjunto formado por estas dos disciplinas, criptografía y criptoanálisis, se conoce como **criptología**.

Cuando la protección que queremos obtener consiste en garantizar el secreto de la información, es decir, la **confidencialidad**, utilizamos el método criptográfico conocido como **cifrado**.

Uso de cifrado

El hecho de usar técnicas de cifrado parte de la idea que es muy costoso intentar evitar que un intruso intercepte la información. Enviar mensajes cifrados es más fácil, ya que no podrá interpretar la información que contienen.

Si M es el mensaje que queremos proteger o **texto en claro**, cifrarlo consiste en aplicarle un **algoritmo de cifrado** f, que lo transforma en otro mensaje que llamaremos **texto cifrado**, C. Esto lo podemos expresar como:

$$C = f(M)$$

Para que este cifrado sea útil, debe existir otra transformación o **algoritmo de descifrado** f^{-1}, que permita recuperar el mensaje original a partir del texto cifrado:

$$M = f^{-1}(C)$$

El cifrado de César

Por ejemplo, el "cifrado de César" que hemos mencionado anteriormente consistía en sustituir cada letra del mensaje por la que hay tres posiciones más adelante en el alfabeto (volviendo a empezar por la letra A si llegamos a la Z). De este modo, si aplicamos este algoritmo de cifrado al texto en claro "ALEA JACTA EST" (y utilizando el alfabeto latino actual, porque en tiempos del César no existían letras como la "W"), obtenemos el texto cifrado "DOHD MDFWD HVW". El descifrado en este caso es muy simple: sólo es necesario sustituir cada letra por la que hay tres posiciones antes en el alfabeto.

Un esquema como el del cifrado de César tiene el inconveniente de que, si el enemigo descubre cuál es el algoritmo de cifrado (y a partir de aquí deduce el

algoritmo inverso), será capaz de interpretar todos los mensajes cifrados que capture. Entonces se requeriría instruir a todos los "oficiales de comunicaciones" del ejército para que aprendieran un nuevo algoritmo, lo cual podría resultar complejo. En lugar de ello, lo que se hace en la actualidad es utilizar como algoritmo una función con un parámetro llamado **clave**.

En este caso podemos hablar de una función de cifrado e con una **clave de cifrado** k, y una función de descifrado d con una **clave de descifrado** x:

$$\begin{aligned} C &= e(k,M) \\ M &= d(x,C) = d(x,e(k,M)) \end{aligned}$$

De este modo, una solución al problema del espía que llega a conocer cómo descifrar los mensajes podría ser seguir utilizando el mismo algoritmo, pero con una clave distinta.

Ejemplo de uso de una clave

El algoritmo de Julio César se puede generalizar definiendo una clave k que indique cuántas posiciones hay que avanzar cada letra en el alfabeto. El cifrado de César utiliza $k = 3$. Para descifrar se puede utilizar el mismo algoritmo pero con la clave invertida ($d \equiv e, x = -k$).

Una premisa fundamental en la criptografía moderna es la **suposición de Kerckhoffs**, que establece que los algoritmos deben ser conocidos públicamente y su seguridad solo depende de la clave. En lugar de intentar ocultar el funcionamiento de los algoritmos, es mucho más seguro y efectivo mantener en secreto solamente las claves.

Seguridad por ocultismo

A lo largo de la historia ha habido casos que han demostrado la peligrosidad de basar la protección en mantener los algoritmos en secreto (lo que se conoce como "seguridad por ocultismo"). Si el algoritmo es conocido por muchos, es más fácil que se detecten debilidades o vulnerabilidades y se puedan corregir rápidamente. Si no, un experto podría deducir el algoritmo por ingeniería inversa, y terminar descubriendo que tiene puntos débiles por donde se puede ataca, como sucedió con el algoritmo A5/1 de la telefonía móvil GSM.

Un algoritmo se considera **seguro** si a un adversario le es imposible obtener el texto en claro M aun conociendo el algoritmo e y el texto cifrado C. Es decir, es imposible descifrar el mensaje sin saber cuál es la clave de descifrado. La palabra "imposible", debe tormarse en consideración con distintos matices. Un algoritmo criptográfico es **computacionalmente seguro** si, aplicando el mejor método conocido, la cantidad de recursos necesarios (tiempo de cálculo, número de procesadores, etc.) para descifrar el mensaje sin conocer la clave es mucho más grande (unos cuantos órdenes de magnitud) de lo que está al alcance de cualquier persona. En el límite, un algoritmo es **incondicionalmente seguro** si no se puede invertir ni con recursos infinitos. Los algoritmos que se utilizan en la práctica son (o intentan ser) computacionalmente seguros.

La acción de intentar descifrar mensajes sin conocer la clave de descifrado se conoce como "ataque". Si el ataque tiene éxito, se suele decir coloquialmente que se ha conseguido "romper" el algoritmo. Existen dos formas de llevar a cabo un ataque:

- Mediante el **criptoanálisis**, es decir, estudiando matemáticamente la forma de deducir el texto en claro a partir del texto cifrado.
- Aplicando la **fuerza bruta**, es decir, probando uno a uno todos los valores posibles de la clave de descifrado x hasta encontrar uno que produzca un texto en claro con sentido.

Ataques al cifrado de César

Continuando con el ejemplo del algoritmo del César generalizado (con clave), un ataque criptoanalítico podría consistir en el análisis de las propiedades estadísticas del texto cifrado. Las letras cifradas que más se repiten probablemente corresponderán a vocales o a las consonantes más frecuentes, las combinaciones más repetidas de dos (o tres) letras seguidas probablemente corresponden a los dígrafos (o trígrafos) que normalmente aparecen más veces en un texto.

Por otro lado, el ataque por fuerza bruta consistiría en intentar el descifrado con cada uno de los 25 posibles valores de la clave ($1 \leq x \leq 25$, si el alfabeto tiene 26 letras) y mirar cuál de ellos da un resultado inteligible. En este caso, la cantidad necesaria de recursos es tan modesta que incluso se puede realizar el ataque a mano. Por lo tanto, este cifrado seria un ejemplo de algoritmo inseguro o débil.

A continuación veremos las características de los principales sistemas criptográficos utilizados en la protección de las comunicaciones. A partir de ahora, consideraremos los mensajes en claro, los mensajes cifrados y las claves como secuencias de bits.

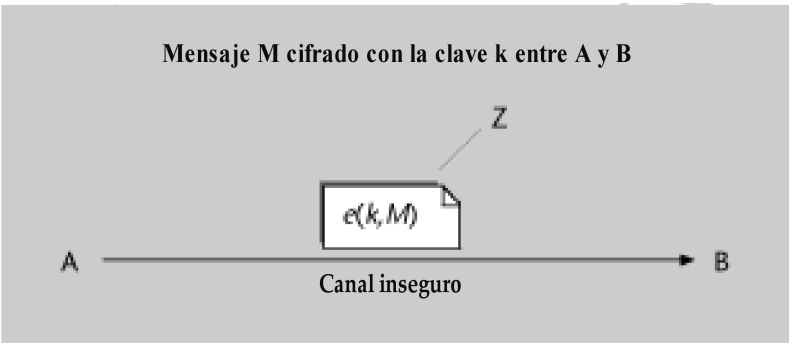
1.1. Criptograía de clave simétrica

Los sistemas criptográficos **de clave simétrica** se caracterizan porque la clave de descifrado x es idéntica a la clave de cifrado k, o bien se puede deducir directamente a partir de ésta.

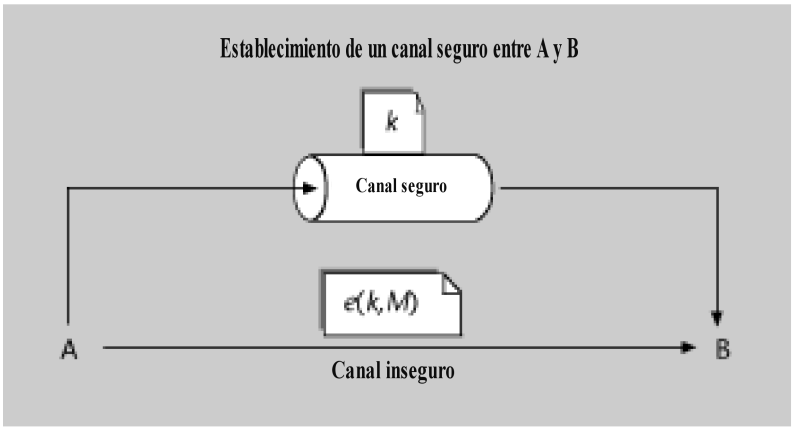
Para simplificar, supondremos que en este tipo de criptosistemas la clave de descifrado es igual a la de cifrado: $x = k$ (si no, siempre podemos considerar que en el algoritmo de descifrado el primer paso es calcular la clave x a partir de k). Es por esto que estas técnicas criptográficas se denominan de *clave simétrica*, o a veces también de **clave compartida**. Así, tenemos:

$$\begin{aligned} C &= e(k,M) \\ M &= d(k,C) = d(k,e(k,M)) \end{aligned}$$

La seguridad del sistema recae pues en mantener en secreto la clave k. Cuando los participantes en una comunicación quieren intercambiarse mensajes confidenciales, tienen que escoger un clave secreta y usarla para cifrar los mensajes. Entonces, pueden enviar estos mensajes por cualquier canal de comunicación, con la confianza que, aun que el canal sea inseguro y susceptible de ser inspeccionado por terceros, ningún espía Z será capaz de interpretarlos.



Si el sistema es de clave compartida, es necesario que el valor de la clave secreta *k* que usan *A* y *B* sea el mismo. ¿Cómo podemos asegurar que esto sea así? Está claro que no pueden mandar la clave escogida mediante el canal de comunicación de que disponen, porque la hipótesis inicial es que este canal es inseguro y todo el mundo podría descubrir la información que se transmite a través del mismo. Una posible solución consiste en utilizar un canal aparte, que se pueda considerar suficientemente seguro:



Esta solución, sin embargo, presenta algunos inconvenientes. Por un lado se supone que el canal seguro no será de uso tan ágil como el canal inseguro (si lo fuera, sería mucho mejor enviar todos los mensajes confidenciales sin cifrar por el canal seguro y olvidarnos del canal inseguro). Por tanto, puede ser difícil ir cambiando la clave. Y por otro lado, este esquema no es suficientemente general: puede ser que tengamos que enviar información cifrada a alguien con quien no podemos contactar de ningún otro modo. Como veremos más adelante, estos problemas relacionados con el **intercambio de claves** se solucionan con la criptografía de clave pública.

Canales seguros

Podrían ser ejemplos de "canales seguros" el correo tradicional (no electrónico) o un servicio de mensajería "física", una conversación telefónica, o cara a cara, etc.

A continuación repasaremos las características básicas de los principales algoritmos criptográficos de clave simétrica, que agruparemos en dos categorías: algoritmos de cifrado en flujo y algoritmos de cifrado en bloque.

1.1.1. Algoritmos de cifrado en flujo

El funcionamiento de una cifrado en flujo consiste en la combinación de un texto en claro M con un texto de cifrado S que se obtiene a partir de la clave simétrica k. Para descifrar, sólo se requiere realizar la operación inversa con el texto cifrado y el mismo texto de cifrado S.

La operación de combinación que se utiliza normalmente es la suma, y la operación inversa por tanto es la resta. Si el texto está formado por caracteres, este algoritmo sería como una cifra de César en que la clave va cambiando de un carácter a otro. La clave que corresponde cada vez viene dada por el texto de cifrado S (llamado *keystream* en inglés).

Si consideramos el texto formado por bits, la suma y la resta son equivalentes. En efecto, cuando se aplican bit a bit, ambas son idénticas a la operación lógica "O exclusiva", denotada con el operador XOR (*eXclusive OR*) o el símbolo $\oplus$. Así pues:

$$\begin{aligned} C &= M \oplus S(k) \\ M &= C \oplus S(k) \end{aligned}$$

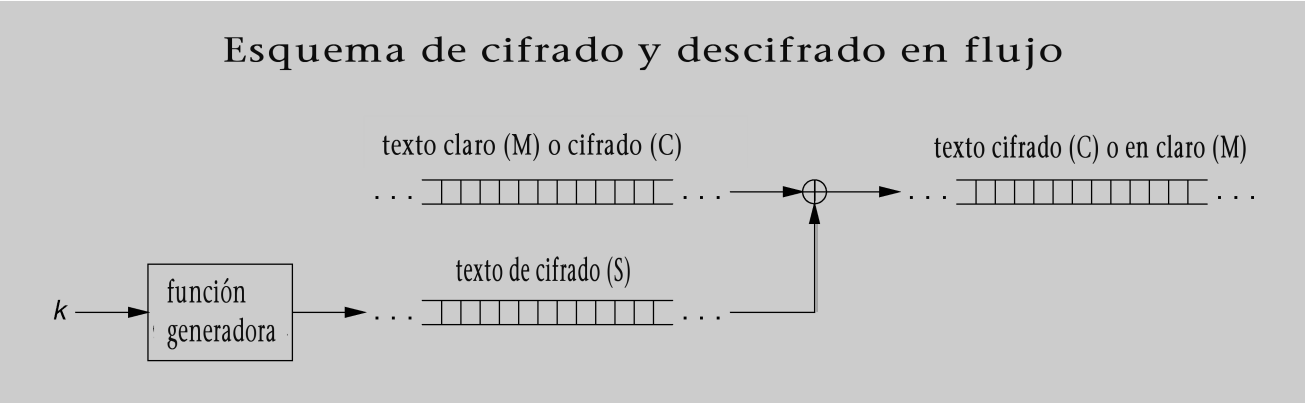
Suma y resta de bits

Cuando trabajamos con aritmética binaria o aritmética modulo 2, se cumple que:

$$\begin{array}{ll} 0+0=0 & 0-0=0 \\ 0+1=1 & 0-1=1 \\ 1+0=1 & 1-0=1 \\ 1+1=0 & 1-1=0 \end{array}$$

$$\begin{array}{c} 0 \oplus 0 = 0 \\ 0 \oplus 1 = 1 \\ 1 \oplus 0 = 1 \\ 1 \oplus 1 = 0 \end{array}$$

En los esquemas de cifrado en flujo, el texto en claro M puede ser de cualquier longitud, y el texto de cifrado S ha de ser como mínimo igual de largo. De hecho, no es necesario disponer del mensaje entero antes de empezar a cifrarlo o descifrarlo, ya que se puede implementar el algoritmo para que trabaje con un "flujo de datos" que se va generando a partir de la clave (el texto de cifrado). De ahí procede el nombre de este tipo de algoritmos. La siguiente figura ilustra el mecanismo básico de su implementación:



Existen distintas formas de obtener el texto de cifrado S en función de la clave k:

- Si se escoge una secuencia k más corta que el mensaje M, una posibilidad

sería repetirla cíclicamente tantas veces como sea necesario para ir sumándola al texto en claro.

El inconveniente de este método es que se puede romper fácilmente, sobre todo cuanto más corta sea la clave (en el caso mínimo, el algoritmo sería equivalente al cifrado de César).

- En el otro extremo, se podría tomar directamente $S(k) = k$. Esto quiere decir que la propia clave debe ser tan larga como el mensaje que hay que cifrar. Este es el principio del conocido **cifrado de Vernam**. Si k es una secuencia totalmente aleatoria que no se repite cíclicamente, estamos ante de un ejemplo de cifrado incondicionalmente seguro, tal como lo hemos definido al inicio de este módulo. Este método de cifrado se llama en inglés *one-time pad* ("cuaderno de un sol uso").

  El problema es en este caso que el receptor tiene que disponer de la misma secuencia aleatoria para poder realizar el descifrado, y si le tiene que llegar a través de un canal seguro, la pregunta es inmediata: ¿por qué no enviar el mensaje confidencial M, que es igual de largo que la clave k, directamente por el mismo canal seguro? Es evidente, pues, que este algoritmo es muy seguro pero no es demasiado práctico.

Uso del cifrado de Vernam

En ocasiones las comunicaciones entre portaaviones y los aviones utilizan el cifrado de Vernam. En este caso, se aprovecha que en un momento dado (antes del despegue) tanto el avión como el portaaviones están en el mismo sitio, con lo cual, intercambiarse, por ejemplo, un disco duro de 20 GB con una secuencia aleatoria no es ningún problema. Posteriormente, cuando el avión despega puede establecer una comunicación segura con el portaaviones utilizando un cifrado de Vernam con la clave aleatoria que ambas partes comparten.

- Lo que en la práctica se utiliza son funciones que generan **secuencias pseudoaleatorias** a partir de una **semilla** (un número que actúa como parámetro del generador), y lo que se intercambia como clave secreta k es solamente esta semilla.

  Las secuencias pseudoaleatorias reciben este nombre porque intentan parecer aleatorias pero, obviamente, son generadas algorítmicamente. En cada paso el algoritmo se encontrará en un determinado estado, que vendrá dado por sus variables internas. Dado que las variables serán finitas, habrá un número máximo de posibles estados distintos. Esto significa que al cabo de un cierto período, los datos generados se volverán a repetir. Para que el algoritmo sea seguro, interesa que el período de repetición sea cuanto más largo mejor (con relación al mensaje que hay que cifrar), con el fin de dificultar el criptoanálisis. Las secuencias pseudoaleatorias también deben tener otras propiedades estadísticas equivalentes a las de las secuencias aleatorias puras.

Funciones pseudoaleatorias

Son ejemplos de funciones pseudoaleatorias las basadas en registros de desplazamiento realimentados (*feedback shift registers* o FSR). El valor inicial del registro es la semilla. Para ir obteniendo cada bit pseudoaleatorio se desplazan todos los bits del registro una posición y se toma el que sale fuera del registro. El bit que queda libre al otro extremo se llena con un valor que es función del resto de bits.

Cifrado síncrono y asíncrono

Si el texto de cifrado S depende exclusivamente de la clave k, se dice que el cifrado es *síncrono*. Este cifrado tiene el problema de que, si por algún error de transmisión, se pierden bits (o llegan repetidos), el receptor se desincronizará y sumará bits del texto S con bits del texto cifrado C que no corresponden, con lo cual el texto descifrado a partir de entonces será incorrecto.

Esto se puede evitar con el cifrado asíncrono (o auto-sincronizante), en el cual el texto S se calcula a partir de la clave k y el mismo texto cifrado C. Es decir, en lugar de realimentarse con sus propios bits de estado, el generador se realimenta con los últimos n bits cifrados transmitidos. De este modo, si se pierden m bits consecutivos en la comunicación, el error afectará como máximo al descifrado de $m+n$ bits del mensaje original.

Ejemplos de algoritmos de cifrado en flujo

Los algoritmos de cifrado en flujo actualmente en uso tienen la propiedad que son poco costosos de implementar. Las implementaciones en hardware son relativamente simples y, por lo tanto, eficientes en su rendimiento (en términos de bits cifrados por segundo). Pero también las implementaciones en software pueden ser muy eficientes.

Las características del cifrado en flujo lo hacen apropiado para entornos en los que se necesite un rendimiento alto y los recursos (capacidad de cálculo, consumo de energía) sean limitados. Para ello se suelen utilizar en comunicaciones móviles: redes locales sin hilos, telefonía móvil, etc.

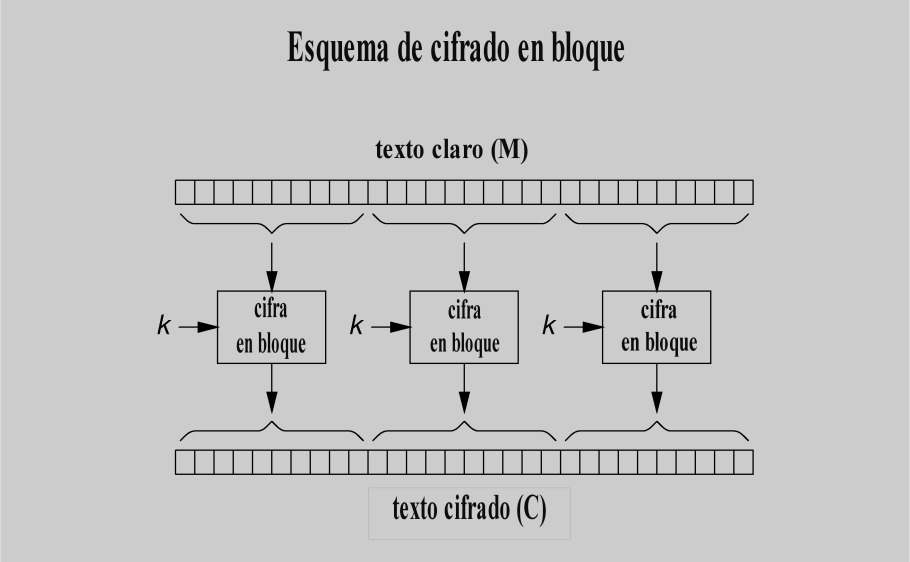
Un ejemplo de algoritmo de cifrado en flujo es el **RC4 (Ron's Code 4)**. Fue diseñado por Ronald Rivest en 1987 y publicado en Internet por un remitente anónimo en 1994. Es el algoritmo de cifrado en flujo mas utilizado en muchas aplicaciones gracias a su simplicidad y velocidad. Por ejemplo, el sistema de protección WEP (*Wired Equivalent Privacy*) que incorpora el estándar IEEE 802.11 para tecnología LAN inalámbrica utiliza este criptosistema de cifrado en flujo.

1.1.2. Algoritmos de cifrado en bloque

En una cifra de bloque, el algoritmo de cifrado o descifrado se aplica separadamente a bloques de entrada de longitud fija b, y para cada uno de ellos el resultado es un bloque de la misma longitud.

Para cifrar un texto en claro de L bits debemos dividirlo en bloques de b bits cada uno y cifrar estos bloques uno a uno. Si L no es múltiple de b, se pueden agregar bits adicionales hasta llegar a un número lleno de bloques, pero luego puede ser necesario indicar de alguna forma cuántos bits había realmente en el mensaje original. El descifrado también se debe realizar bloque a bloque.

La siguiente figura muestra el esquema básico del cifrado de bloque:



Muchos de los algoritmos del cifrado de bloque se basan en la combinación de dos operaciones básicas: sustitución y transposición.

- La **sustitución** consiste en traducir cada grupo de bits de la entrada a otro, de acuerdo con una permutación determinada.

  El cifrado del César sería un ejemplo simple de sustitución, en el que cada grupo de bits correspondería a una letra. De hecho, se trata de un caso particular de **sustitución alfabética**. En el caso más general, las letras del texto cifrado no tienen por qué estar a una distancia constante de las letras del texto en claro (la k del algoritmo, tal como la hemos definido). La clave se puede expresar como la secuencia correlativa de letras que corresponden a la A, la B, la C, etc. Por ejemplo:

  A B C D E F G H Y J K L M N O P Q R S T U V W X Y Z
  Clave: Q W E R T Y U Y O P A S D F G H J K L Z X C V B N M

  Texto en claro: A L E A J A C T A E S T
  Texto cifrado: Q S T Q P Q E Z Q T L Z

Clave de la sustitución alfabética

Está claro que la clave ha de ser una **permutación** del alfabeto, es decir, no puede haber letras repetidas ni faltar ninguna. Si no, la transformación no sería invertible en general.

- La **transposición** consiste en reordenar la información del texto en claro según un patrón determinado. Un ejemplo podría ser la formación grupos de cinco letras, incluidos los espacios en blanco, y rescribir cada grupo $(1,2,3,4,5)$ en el orden $(3,1,5,4,2)$:

  Texto en claro: A L E A J A C T A E S T
  Texto cifrado: E A A L C J A T A S T E

  La transposición por si sola no dificulta extraordinariamente el criptoanálisis, pero puede combinarse con otras operaciones para añadir complejidad a los algoritmos de cifrado.

El **producto de cifras**, o combinación en cascada de distintas transforma-

ciones criptográficas es una técnica muy efectiva para implementar algoritmos bastante seguros de forma sencilla. Por ejemplo, muchos algoritmos de cifrado de bloque se basan en una serie de iteraciones de productos sustitución–transposición.

Confusión y difusión

Dos propiedades deseables en un algoritmo criptográfic son la *confusión*, que consiste en esconder la relación entre la clave y las propiedades estadísticas del texto cifrado, y la *difusión*, que propaga la redundancia del texto en claro a lo largo del texto cifrado para que no sea fácilmente reconocible.

La confusión consigue que, cambiando un solo bit de la clave, cambien muchos bits del texto cifrado, y la difusión implica que el cambio de un solo bit del texto en claro afecte también a muchos bits del texto cifrado.

En un bucle de productos de cifrados básicos, la sustitución contribuye a la confusión, mientras que la transposición contribuye a la difusión. La combinación de estas transformaciones simples, repetidas diversas veces, provoca que los cambios en la entrada se propaguen por toda la salida por efecto "alud".

Ejemplos de algoritmos de cifrado en bloque

DES (Data Encryption Standard). Durante muchos años ha sido el algoritmo más estudiado y, a la vez, el más utilizado. Desarrollado por IBM durante los años 70, fue adoptado en 1977 por el NBS norteamericano (nombre que tenía entonces el actual NIST) como estándar para el cifrado de datos.

NBS y NIST©

NBS era la sigla de *National Bureau of Standards*, y NIST es la sigla de *National Institute of Standards and Technology*.

El algoritmo admite una clave de 64 bits, pero sólo 7 de cada 8 intervienen en el cifrado, de modo que la longitud efectiva de la clave es de 56 bits. Los bloques de texto a los que se les aplica el DES tienen que ser de 64 bits cada uno.

Bits adicionales de la clave DES

Un posible uso de los bits de la clave DES que no influ en en el algoritmo es su utilización como bits de paridad.

La parte central del algoritmo consiste en dividir la entrada en grupos de bits, hacer una sustitución distinta sobre cada grupo y, a continuación una transposición de todos los bits. Esta transformación se repite dieciséis veces: en cada iteración, la entrada es una transposición distinta de los bits de la clave sumada bit a bit (XOR) con la salida de la iteración anterior. Tal como está diseñado el algoritmo, el descifrado se realiza igual que el cifrado pero realizando las transposiciones de la clave en el orden inverso (empezando por la última).

Triple DES. Aunque a lo largo de los años el algoritmo DES se ha mostrado muy resistente al criptoanálisis, su principal problema es actualmente la vulnerabilidad a los ataques de fuerza bruta, a causa de la longitud de la clave, de sólo 56 bits. Aunque en los años 70 era muy costoso realizar una búsqueda entre las 2^{56} combinaciones posibles, la tecnología actual permite romper el algoritmo en un tiempo cada vez más corto.

Los retos DES

En la dirección `www.rsasecurity.com/rsalabs/challenges/` se puede encontrar información sobre los "retos DES", que demuestra que en 1999 ya era posible romper una clave DES en menos de 24 horas.

Por este motivo, en 1999 el NIST cambió el algoritmo DES por el "Triple DES" como estándar oficial, mientras no estuviera disponible el nuevo están-

dar AES. El Triple DES, como su nombre indica, consiste en aplicar el DES tres veces consecutivas. Esto se puede realizar con tres claves (k_1, k_2, k_3), o bien con sólo dos (k_1, k_2, y otra vez k_1). La longitud total de la clave con la segunda opción es de 112 bits (dos claves de 56 bits), que hoy ya se considera suficientemente segura; la primera opción proporciona más seguridad, pero a costa de utilizar una clave total de 168 bits (3 claves de 56 bits), que puede ser un poco más difícil de gestionar e intercambiar.

Repetición del DES

La operación de aplicar el cifrado DES con una clave, y el resultado de volverlo a cifrar con otra clave, no es equivalente a un solo cifrado DES (no hay ninguna clave única que dé el mismo resultado que dan las otras dos juntas). Si no fuera de este modo, la repetición del DES no sería más segura que el DES simple.

Para conseguir que el sistema sea adaptable al estándar antiguo, en el Triple DES se aplica una secuencia cifrado-descifrado-cifrado (E-D-E) en lugar de tres cifrados:

$$\begin{aligned} C &= e(k_3, d(k_2, e(k_1, M))) \\ \text{o bien:} \quad C &= e(k_1, d(k_2, e(k_1, M))) \end{aligned}$$

Así, tomando $k_2 = k_1$ tenemos un sistema equivalente al DES simple.

AES (Advanced Encryption Standard). Dado que el estándar DES empezaba a quedarse anticuado, a causa sobretodo de la longitud tan corta de sus claves, y el Triple DES no es excesivamente eficiente cuando se implementa con software, en 1997 el NIST convocó a la comunidad criptográfica a presentar propuestas para un nuevo estándar, el AES, que sustituyera al DES. De los quince algoritmos candidatos que se aceptaron, se escogieron cinco como finalistas, y en octubre de 2000 se dio a conocer el ganador: el algoritmo Rijndael, propuesto por los criptógrafos belgas Joan Daemen y Vincent Rijmen.

El Rijndael puede trabajar en bloques de 128, 192 o 256 bits (aunque el estándar AES sólo prevé los de 128), y la longitud de la clave también puede ser de 128, 192 o 256 bits. Dependiendo de esta última longitud, el número de iteraciones del algoritmo es 10, 12 ó 14, respectivamente. Cada iteración incluye una sustitución fija byte a byte, una transposición, una transformación consistente en desplazamientos de bits y XORs, y una suma binaria (XOR) con bits obtenidos a partir de la clave.

1.1.3. Uso de los algoritmos de clave simétrica

Cuando se utiliza el cifrado simétrico para proteger las comunicaciones, se puede escoger el algoritmo que sea más apropiado a las necesidades de cada aplicación: normalmente, a más seguridad menos velocidad de cifrado, y viceversa.

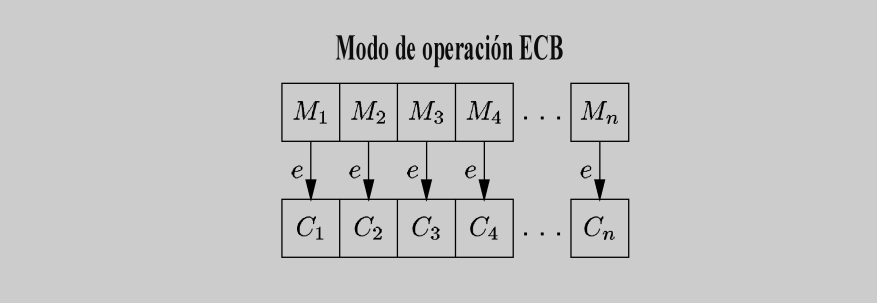
Un aspecto que hay que tener en cuenta es que, aunque el cifrado puede conseguir que un atacante no descubra directamente los datos transmitidos, en ocasiones es posible que se pueda deducir información indirectamente. Por ejemplo, en un protocolo que utilice mensajes con una cabecera fija, la aparición de los mismos datos cifrados varias veces en una transmisión puede indicar dónde empiezan los mensajes.

Esto pasa con el cifrado en flujo si su periodo no es lo suficientemente largo, pero en un cifrado en bloque, si dos bloques de texto en claro son iguales y se utiliza la misma clave, los bloques cifrados también serán iguales. Para contrarrestar esta propiedad, se pueden aplicar distintos **modos de operación** al cifrado en bloque.

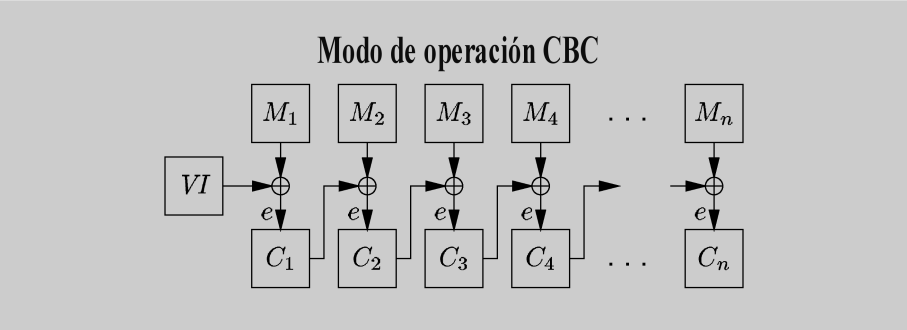
- El modo ECB (*Electronic Codebook*) es el más simple, y consiste en dividir el texto en bloques y cifrar cada uno de ellos de forma independiente. Este modo parte del problema de dar bloques iguales cuando en la entrada existen bloques iguales.

Modo ECB

El nombre del modo ECB (*Electronic Codebook*) da la idea que se puede considerar como una simple sustitución bloque a bloque, de acuerdo con un código o diccionario (con muchas entradas, sin duda) que viene dado por la clave.



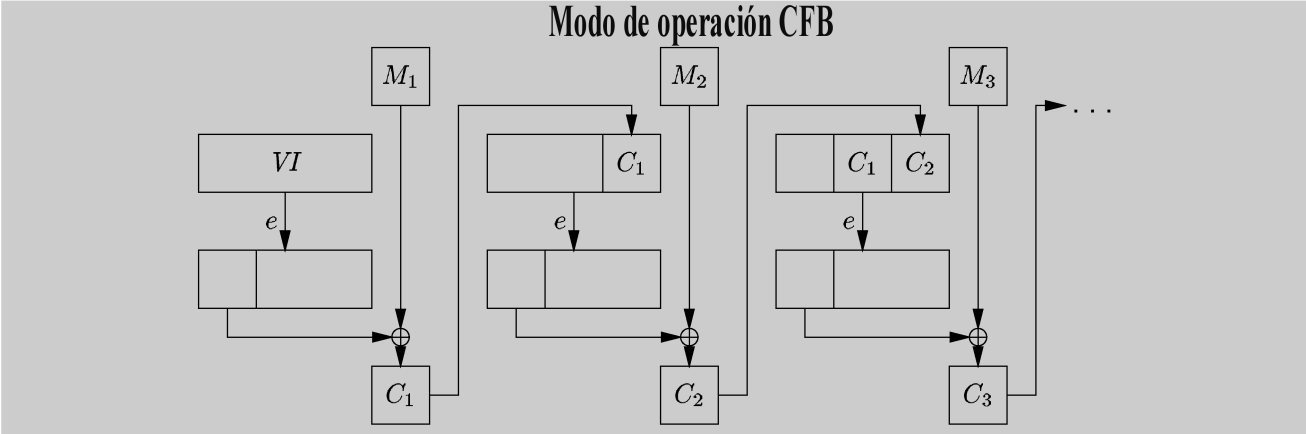
- En el modo CBC (*Cipher Block Chaining*), se suma a cada bloque de texto en claro, antes de cifrarlo, (bit a bit, con XOR) el bloque cifrado anterior. Al primer bloque se le suma un **vector de inicialización** (VI), que es un conjunto de bits aleatorios de la misma longitud que un bloque. Escogiendo vectores distintos cada vez, aun que el texto en claro sea el mismo, los datos cifrados serán distintos. El receptor debe conocer el valor del vector antes de empezar a descifrar, pero es necesario falta guardar este valor en secreto, sino que normalmente se transmite como cabecera del texto cifrado.



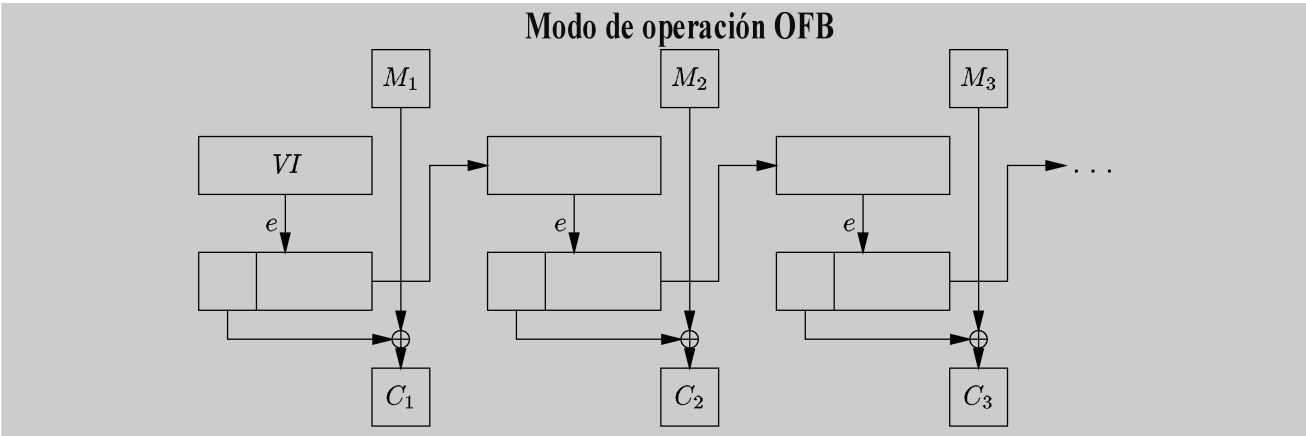
- En el modo CFB (*Cipher Feedback*), el algoritmo de cifrado no se aplica directamente al texto en claro sino a un vector auxiliar (inicialmente igual al VI). Del resultado del cifrado se toman n bits que se suman a n bits del texto en claro para obtener n bits de texto cifrado. Estos bits cifrados se utilizan también para actualizar el vector auxiliar. El número n de bits generados en cada iteración puede ser menor o igual que la longitud de bloque b. Tomando como ejemplo $n = 8$, tenemos un cifrado que genera un byte cada vez sin que sea necesario esperar a tener un bloque entero para poderlo descifrar.

Modo CFB como cifrado en flujo

Es fácil ver que el modo CFB (y también el OFB) se puede considerar como un cifrado en flujo que utiliza como función generador un cifrado en bloque.



- El modo OFB (*Output Feedback*) opera como el CFB pero en lugar de actualizar el vector auxiliar con el texto cifrado, se actualiza con el resultado obtenido del algoritmo de cifrado. La propiedad que distingue este modo de los demás consiste en que un error en la recuperación de un bit cifrado afecta solamente al descifrado de este bit.



- A partir de los modos anteriores se pueden definir varias variantes. Por ejemplo, el modo CTR (*Counter*) es como el OFB, pero el vector auxiliar no se realimenta con el cifrado anterior sino que simplemente es un contador que se va incrementando.

Existe otra técnica para evitar que textos de entrada iguales produzcan textos cifrados iguales, que se puede aplicar también a cifrados que no utilizan vector de inicialización (incluido el cifrado en flujo). Esta técnica consiste en la modificación de la clave secreta con bits aleatorios antes de usarla en el algoritmo de cifrado (o en el de descifrado). Como estos bits aleatorios sirven para dar un "sabor" distinto a la clave, se les suele llamar **bits de sal**. Igual que el vector de inicialización, los bits de sal se envían en claro antes que el texto cifrado.

1.1.4. Funciones *hash* seguras

Aparte de cifrar datos, existen algoritmos basados en técnicas criptográficas que se usan para garantizar la autenticidad de los mensajes. Un tipo de algoritmos de estas características son las llamadas **funciones *hash* seguras**, también conocidas como funciones de **resumen de mensaje** (*message digest*, en inglés).

En general, podemos decir que una función *hash* nos permite obtener una cadena de bits de longitud fija, relativamente corta, a partir de un mensaje de longitud arbitraria:

$$H = h(M)$$

Para mensajes M iguales, la función h debe dar resúmenes H iguales. Pero si dos mensajes dan el mismo resumen H no deben ser necesariamente iguales. Esto es así porque sólo existe un conjunto limitado de posibles valores H, ya que su longitud es fija, y en cambio puede haber muchos más mensajes M (si la longitud puede ser cualquiera, habrá infinitos).

Para poderla aplicar en un sistema de autenticación, la función h debe ser una función *hash* segura.

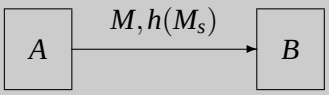
Secreto de los algoritmos

Observad que las funciones *hash* son conocidas, puesto que todo el mundo debe poder calcular los resúmenes del mismo modo.

Se entiende que una función *hash* o de resumen es **segura** si cumple las siguientes condiciones:

- Es **unidireccional**, es decir, si tenemos $H = h(M)$ es computacionalmente inviable encontrar M a partir del resumen H.
- Es **resistente a colisiones**, es decir, dado un mensaje M cualquiera es computacionalmente inviable encontrar un mensaje $M' \neq M$ tal que $h(M') = h(M)$.

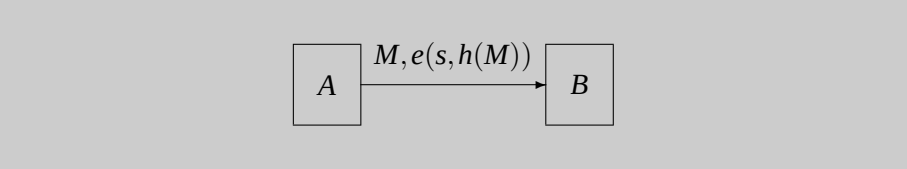
Estas propiedades permiten el uso de las funciones *hash* seguras para dar un servicio de autenticidad basado en una clave secreta s compartida entre dos partes A y B. Aprovechando la unidireccionalidad, cuando A quiere mandar un mensaje M a B puede preparar otro mensaje M_s: por ejemplo, concatenando el original con la clave: $M_s = (M, s)$. Entonces manda a B el mensaje M y el resumen del mensaje M_s:



Para comprobar la autenticidad del mensaje recibido, B verifica que el resumen

corresponda efectivamente a M_s. Si es así, quiere decir que lo ha generado alguien que conoce la clave secreta s (que debería ser A), y también que nadie ha modificado el mensaje.

Otra técnica consistiría en calcular el resumen del mensaje M y cifrarlo utilizando s como clave de cifrado:



Para verificar la autenticidad se debe recuperar el resumen enviado, descifrándolo con la clave secreta s, y compararlo con el resumen del mensaje M. Un atacante que quisiera modificar el mensaje sin conocer la clave podría intentar sustituirlo por otro que diera el mismo resumen, con lo cual B no detectaría la falsificación. Pero si la función de resumen es resistente a colisiones, esto le sería imposible al atacante.

Autenticidad y confidencialidad

Cifrar solamente el resumen, en lugar del mensaje entero, es más eficiente porqué hay que cifrar menos bits. Esto, evidentemente, suponiendo que solamente se precise autenticidad, y no confidencialidad. Si también interesa que el mensaje sea confidencial, entonces sí es necesario cifrarlo entero.

Para dificultar los ataques contra las funciones de resumen, por un lado los algoritmos tienen que definir una relación compleja entre los bits de entrada y cada bit de salida. Por otro lado, los ataques por fuerza bruta se contrarrestan alargando lo suficiente la longitud del resumen. Por ejemplo, los algoritmos usados actualmente generan resúmenes de 128 ó 160 bits. Esto quiere decir que un atacante podría tener que probar del orden de 2^{128} o 2^{160} mensajes de entrada para encontrar una colisión (es decir, un mensaje distinto que diera el mismo resumen).

Pero existe otro tipo de ataque más ventajoso para el atacante, llamado **ataque del cumpleaños**. Un ataque de este tipo parte de la suposición de que el atacante puede escoger el mensaje que será autenticado. La víctima lee el mensaje y, si lo acepta, lo autentica con su clave secreta. Pero el atacante ha presentado este mensaje porque ha encontrado otro que da el mismo resumen y, por lo tanto, puede hacer creer al destinatario que el mensaje auténtico es este otro. Y esto se puede conseguir realizando una búsqueda por fuerza bruta con muchas menos operaciones: del orden de 2^{64} ó 2^{80}, si el resumen es de 128 ó 160 bits, respectivamente.

Resistencia fuerte a las colisiones

La resistencia de los algoritmos de resumen a las colisiones, tal como la hemos definido, a veces recibe el nombre de "resistencia débil", mientras que la propiedad de ser resistente a ataques de cumpleaños se conoce como "resistencia fuerte".

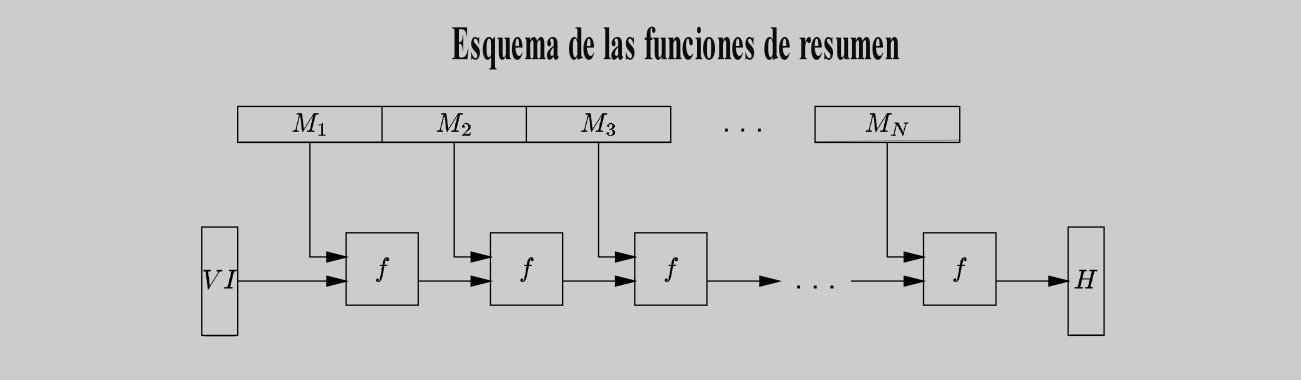
Paradoja del cumpleaños

El nombre de este tipo de ataque viene de un problema clásico de probabilidades conocido como la "paradoja del cumpleaños". El problema consiste en encontrar el número mínimo de personas que debe haber en un grupo para que la probabilidad de que como mínimo dos de ellas celebren su aniversario el mismo día sea superior al 50%. Una respuesta intuitiva puede ser que la solución sea del orden de 200, pero este resultado no es correcto. Podría ser correcto si se quisiera obtener el número de personas necesarias para obtener un 50% de probabilidad de coincidencia con *una* persona determinada. Si permitimos que la coincidencia sea entre *cualquier* pareja de personas, la solución es un número mucho menor: 23.

La conclusión es que si una función de resumen puede dar N valores distintos, porque la probabilidad de encontrar dos mensajes con el mismo resumen sea del 50% el nombre de mensajes que hay que probar es del orden de $\sqrt{N}$.

Ejemplos de funciones *hash* seguras

El esquema de la mayoría de funciones *hash* usadas actualmente es parecido al de los algoritmos de cifrado de bloque: el mensaje de entrada se divide en bloques de la misma longitud, y a cada uno se le aplica una serie de operaciones junto con el resultado obtenido en el bloque anterior. El resultado que queda después de procesar el último bloque es el resumen del mensaje.



El objetivo de estos algoritmos es que cada bit de salida dependa de todos los bits de entrada. Esto se consigue con diferentes iteraciones de operaciones que "mezclan" los bits entre ellos, de forma parecida a cómo la sucesión de transposiciones en los cifrados de bloque provoca un "efecto alud" que garantiza la difusión de los bits.

Hasta hace poco, el algoritmo de *hash* más usado era el **MD5 (Message Digest 5)**. Pero como el resumen que da es de sólo 128 bits, y aparte se han encontrado otras formas de generar colisiones parciales en el algoritmo, actualmente se recomienda utilizar algoritmos más seguros, como el **SHA-1 (Secure Hash Algorithm-1)**. El algoritmo SHA-1, publicado el 1995 en un estándar del NIST (como revisión de un algoritmo anterior llamado simplemente SHA), da resúmenes de 160 bits. El año 2002 el NIST publicó variantes de este algoritmo que generan resúmenes de 256, 384 y 512 bits.

Longitud del resumen MD5

Como la longitud del resumen MD5 es de 128 bits, el número de operaciones para un ataque de aniversario es del orden de 2^{64}. Comparad esta magnitud con la de un ataque por fuerza bruta contra el DES (menos de 2^{56} operaciones), de la que no está demasiado lejos.

1.2. Criptografía de clave pública

1.2.1. Algoritmos de clave pública

Como hemos visto en el subapartado anterior, uno de los problemas de la criptografía de clave simétrica es el de la distribución de las claves. Este problema se puede solucionar si utilizamos **algoritmos de clave pública**, también llamados **de clave asimétrica**.

> En un algoritmo criptográfico de clave pública se utilizan claves distintas para el cifrado y el descifrado. Una de ellas, la **clave pública**, se puede obtener fácilmente a partir de la otra, la **clave privada**, pero por el contrario es computacionalmente de muy difícil obtención la clave privada a partir de la clave pública.

Los algoritmos de clave pública típicos permiten cifrar con la clave pública (k_{pub}) y descifrar con la clave privada (k_{pr}):

$$\begin{aligned} C &= e(k_{\text{pub}}, M) \\ M &= d(k_{\text{pr}}, C) \end{aligned}$$

Pero también puede haber algoritmos que permitan cifrar con la clave privada y descifrar con la pública (más adelante veremos cómo se puede utilizar esta propiedad):

$$\begin{aligned} C &= e(k_{\text{pr}}, M) \\ M &= d(k_{\text{pub}}, C) \end{aligned}$$

Los algoritmos de clave pública se basan en problemas matemáticos "fáciles" de plantear a partir de la solución, pero "difíciles" de resolver. En este contexto, se entiende que un problema es fácil si el tiempo para resolverlo, en funciones de la longitud n de los datos, se puede expresar en forma polinómica como, por ejemplo n^2+2n (en teoría de la complejidad, se dice que estos problemas son de la "clase P"). Si el tiempo de resolución crece más rápidamente, como por ejemplo con 2^n, el problema se considera difícil. Así, se puede escoger un valor de n tal que el planteamiento sea viable pero la resolución sea computacionalmente intratable.

Adaptación de los problemas difíciles

Si los adelantos de la tecnología reducen el tiempo de resolución, se puede aumentar la longitud n, con lo cual se necesitarán unas cuantas operaciones más para el planteamiento, pero la complejidad de la solución crecerá exponencialmente.

Un ejemplo de problema fácil de plantear pero difícil de resolver es el de los logaritmos discretos. Si trabajamos con aritmética módulo m, resulta fácil calcular esta expresión:

$$y = b^x \bmod m$$

El valor x se llama logaritmo discreto de y en base b módulo m. Escogiendo convenientemente b y m, puede ser difícil calcular el logaritmo discreto

de cualquier y. Una posibilidad es ir probando todos los valores de x: si m es un número de n bits, el tiempo para encontrar la solución aumenta proporcionalmente a 2^n. Hay otros métodos más eficientes para calcular logaritmos discretos, pero el mejor algoritmo conocido también necesita más tiempo del que se puede expresar polinómicamente.

Ejemplo de operaciones módulo *m*

Por ejemplo, para obtener 14^{11} mod 19 podemos multiplicar 11 veces el número 14, dividir el resultado entre 19 y tomar el residuo de la división, que es igual a 13. Pero también se puede aprovechar que el exponente 11 es 1011 en binario ($11 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$), y entonces $14^{11} = 14^8 \cdot 14^2 \cdot 14^1$, para obtener el resultado con menos multiplicaciones:

$$\begin{array}{lllllll}
14^1 = 14 & & & \equiv 14 & \pmod{19} & \rightarrow & 14 \\
14^2 = 14^1 \times 14^1 & \equiv 14 \times 14 & = 196 & \equiv 6 & \pmod{19} & \rightarrow & 6 \\
14^4 = 14^2 \times 14^2 & \equiv 6 \times 6 & = 36 & \equiv 17 & \pmod{19} & & \\
14^8 = 14^4 \times 14^4 & \equiv 17 \times 17 & = 289 & \equiv 4 & \pmod{19} & \rightarrow & 4 \\
\hline
 & & & & & & 336 \equiv 13 \pmod{19}
\end{array}$$

Así, sabemos que $\log_{14} 13 = 11 \pmod{19}$. Pero si hubiéramos calculado el logaritmo de cualquier otro número y tendríamos que ir probando uno a uno los exponentes hasta encontrar uno que dé como resultado y. Y si en lugar de tratarse de números de 4 o 5 bits como estos fueran números de más de 1000 bits, el problema sería intratable.

Así pues, los algoritmos de clave pública han de ser diseñados de manera que sea inviable calcular la clave privada a partir de la pública, y lógicamente también ha de ser inviable invertir-los sin saber la clave privada, pero el cifrado y el descifrado se han de poder realizar en un tiempo relativamente corto.

A la práctica, los algoritmos utilizados permiten cifrar y descifrar fácilmente, pero todos ellos son considerablemente más lentos que los equivalentes con criptografía simétrica. Por eso, la criptografía de clave pública se suele utilizar solos en los problemas que la criptografía simétrica no puede resolver: el intercambio de claves y la autenticación con no repudio (firmas digitales).

Velocidad de la criptografía de clave pública

El cifrado y descifrado con algoritmos de clave pública puede llegar a ser dos o tres ordenes de magnitud más lentos que con criptografía simétrica.

- Los mecanismos de**intercambio de claves** permiten que dos partes se pongan de acuerdo en les claves simétricas que utilizaran para comunicar-se, sin que un tercer que esté escuchando el diálogo pueda deducir cuales son estas claves.

  Por ejemplo, A puede escoger una clave simétrica k, cifrar-la con la clave pública de B, y enviar el resultado a B. Luego B descifrará con su clave privada el valor recibido, y sabrá cual es la clave k que ha escogido A. El resto de la comunicación irá cifrada con un algoritmo simétrico (mucho más rápido), y utilizará esta clave k. Los atacantes, al no conocer la clave privada de B, no podrán deducir el valor de k.

- La**autenticación** basada en clave pública se puede utilizar si el algoritmo permite utilizar las claves a la inversa: la clave privada para cifrar y la clave

pública para descifrar. Si A envía un mensaje cifrado con su clave privada, todo el mundo podrá descifrarlo con la clave pública de A, y al mismo tiempo todo el mundo sabrá que el mensaje sólo lo puede haber generado quien conozca la clave privada asociada (que debería ser A). Ésta es la base de las **firmas digitales**.

Ejemplos de algoritmos de clave pública

Intercambio de claves Diffie-Hellman. Es un mecanismo que permite que dos partes se pongan de acuerdo de forma segura sobre una clave secreta. El algoritmo se basa en la dificultad de calcular logaritmos discretos.

RSA. Es el algoritmo más utilizado en la historia de la criptografía de clave pública. Su nombre procede de las iniciales de quienes lo diseñaron en 1977: Ronald Rivest, Adi Shamir y Leonard Adleman. La clave pública está formada por un número n, calculado como producto de dos factores primos muy grandes ($n = p \cdot q$) y un exponente e. La clave privada es otro exponente d calculado a partir de p, q y e, de tal forma que el cifrado y el descifrado se puede realizar de la siguiente forma:

$$\begin{aligned} \text{Cifrado:} \quad C &= M^e \bmod n \\ \text{Descifrado:} \quad M &= C^d \bmod n \end{aligned}$$

Como se puede ver, la clave pública y la privada son intercambiables: si se usa cualquiera de ellas para cifrar, se deberá utilizar la otra para descifrar.

Valores usados en el RSA

Actualmente el problema de factorizar números de 512 bits es muy complejo, aunque abordable si se dispone de suficientes recursos. Por lo tanto, se recomienda utilizar claves públicas con un valor n a partir de 1.024 bits. Como exponente público e normalmente se utilizan valores simples como 3 ó 65.537 ($2^{16}+1$) porque hacen más rápido el cifrado.

La fortaleza del algoritmo RSA se basa, por un lado, en la dificultad de obtener M a partir de C sin conocer d (problema del logaritmo discreto), y por otro lado, en la dificultad de obtener p y q (y, por tanto, d) a partir de n (problema de la factorización de números grandes, que es otro de los problemas considerados difíciles).

ElGamal. Otro esquema de cifrado, en este caso, basado en el problema de Diffie-Hellman.

DSA (Digital Signature Algorithm). Publicado por el NIST en distintas versiones del *Digital Signature Standard* (DSS), la primera de ellas en 1991. Es una variante del algoritmo de firma ElGamal, que a su vez sigue el esquema de cifrado de ElGamal. El DSA no es un algoritmo de cifrado, sino que sólo permite generar firmas y verificarlas.

1.2.2. Uso de la criptografía de clave pública

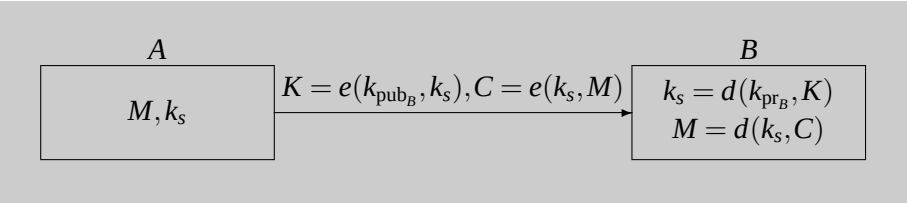
Hemos visto antes que las principales aplicaciones de la criptografía de clave pública son el intercambio de claves para proporcionar confidencialidad y la firma digital para proporcionar autenticidad y no repudio.

Sobre digital

Como veremos más adelante, esta técnica para proporcionar confidencialidad con criptografía de clave pública se suele llamar "sobre digital" en el contexto del correo electrónico segur.

- El problema de la confidencialidad entre dos partes que sólo disponen de

un canal inseguro para comunicarse se resuelve con la criptografía de clave pública. Cuando A quiere mandar un mensaje secreto M a B, no hace falta cifrar todo el mensaje con un algoritmo de clave pública (esto podría resultar muy lento), sino que se escoge una clave simétrica k_s, en ocasiones llamada **clave de sesión** o **clave de transporte**, y se cifra el mensaje con un algoritmo simétrico usando esta clave. Lo único que hace falta cifrar con la clave pública de B (k_{pub_B}) es la clave de sesión. En recepción, B utiliza su clave privada (k_{pr_B}) para recuperar la clave de sesión k_s, y entonces ya puede descifrar el mensaje cifrado.

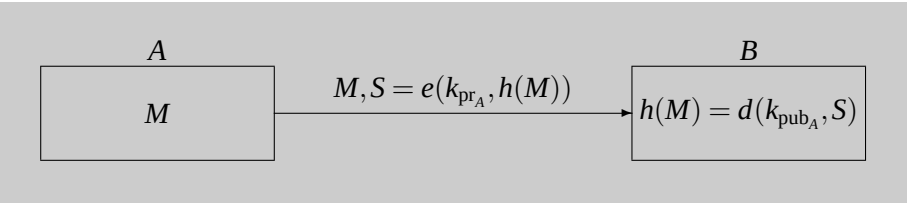


Dado que la clave de sesión es un mensaje relativamente corto (por ejemplo, si es una clave DES sólo tendrá 56 bits), un atacante podría intentar romper el cifrado por fuerza bruta, pero no intentando descifrar el mensaje con los posibles valores de la clave privada k_{pr_B}, sino cifrando los posibles valores de la clave de sesión k_s con la clave pública k_{pub_B}. En el caso de una clave de sesión DES, independientemente del número de bits de la clave pública, el atacante solamente necesitaría un esfuerza del orden de 2^{56} operaciones.

Para evitar este tipos de ataque, la información que se cifra realmente con la clave pública no es directamente el valor secreto (en este caso k_s), sino que a este valor se le añade una cadena más o menos larga de bits aleatorios. El receptor solamente tiene que descartar estos bits aleatorios del resultado que obtenga del descifrado.

- Una **firma digital** es básicamente un mensaje cifrado con la clave privada del firmante. Pero, por cuestiones de eficiencia, lo que se cifra no es directamente el mensaje a firmar, sino solamente su resumen calculado con una función *hash* segura.

  Cuando A quiera mandar un mensaje firmado, tendrá que obtener su resumen y cifrarlo con la clave privada k_{pr_A}. Para verificar la firma, el receptor tiene que descifrarla con la clave pública k_{pub_A} y comparar el resultado con el resumen del mensaje: si son iguales, quiere decir que el mensaje lo ha generado A y nadie lo ha modificado. Dado que se supone que la función de resumen es resistente a colisiones, un atacante no podrá modificar el mensaje sin que la firma deje de ser válida.



1.3. Infraestructura de clave pública (PKI)

Como hemos visto hasta ahora, la criptografía de clave pública permite resolver el problema del intercambio de claves, utilizando las claves públicas de los participantes. Pero se platea otro problema: si alguien afirma ser A y su clave pública es k_{pub}, ¿cómo podemos saber que realmente k_{pub} es la clave pública de A? Porque es perfectamente posible que un atacante Z genere su par de claves (k'_{pr}, k'_{pub}) y afirme "yo soy A, y mi clave pública es k'_{pub}".

Una posible solución a este problema es que exista una entidad de confianza que nos asegure que, efectivamente, las claves públicas pertenecen a sus supuestos propietarios. Esta entidad puede firmar un documento que afirme "la clave pública de A es k_{pub_A}", y publicarlo para que todos los usuarios lo sepan. Este tipo de documento se llama **certificado de clave pública** o **certificado digital**, y es la base de lo que se conoce como **infraestructura de clave pública**.

PKI

La sigla PKI corresponde al nombre en inglés de la infraestructura de clave pública (*Public Key Infrastructure*).

1.3.1. Certificados de clave pública

Un certificado de clave pública o certificado digital consta de tres partes básicas:

- Una identificación de usuario como, por ejemplo, su nombre.
- El valor de la clave pública de este usuario.
- La firma de las dos partes anteriores.

Si el autor de la firma es alguien en quien confiamos, el certificado nos sirve como garantía de que la clave pública pertenece al usuario que figura identificado en el certificado. Quien firma el certificado puede ser una autoridad que se responsabilice de verificar fehacientemente la autenticidad de las claves públicas. En este caso, se dice que el certificado ha sido generado por una **autoridad de certificación** (CA).

CA

CA es la sigla inglesa de *Certification Authority*.

Puede haber distintos formatos de certificados, pero el más usado es el de los **certificados X.509**, especificado en la definición del **servicio de directorio X.500**.

El directorio X.500

La especificación del directorio X.500 está publicada en la Serie de Recomendaciones ITU-T X.500, una de las cuales es la Recomendación X.509.

El directorio X.500 permite almacenar y recuperar información, expresada como **atributos**, de un conjunto de **objetos**. Los objetos X.500 pueden representar, por ejemplo, países, ciudades, o bien empresas, universidades (en general, organizaciones), departamentos, facultades (en general, unidades organizativas), personas, etc. Todos estos objetos están organizados jerárquicamente en forma de árbol (en cada nodo del árbol existe un objeto) y, dentro de cada ni-

vel, los objetos se identifican mediante un **atributo distintivo**. A nivel global, cada objeto se identifica con un **nombre distintivo** (DN), que no es más que la concatenación de los atributos distintivos que hay entre la raíz del árbol y el objeto en cuestión. El sistema de nombres es, pues, parecido al DNS de Internet, con la diferencia que los componentes de un nombre DNS son simples cadenas de caracteres, y los de un DN X.500 son atributos, cada uno con un tipo y un valor.

DN

DN es la sigla inglesa de *Distinguished Name*.

Algunos ejemplos de tipos de atributos que se pueden usar como atributos distintivos en un DN son: *countryName* (habitualmente denotado con la abreviatura "`c`"), *stateOrProvinceName* ("`st`"), *localityName* ("`l`"), *organizationName* ("`o`"), *organizationalUnitName* ("`ou`"), *commonName* ("`cn`"), *surname* ("`sn`"), etc. Un ejemplo de DN es "`c=ES`, `st=Barcelona`, `l=Barcelona`, `o=Universitat Oberta de Catalunya`, `ou=SI`, `cn=cv.uoc.edu`".

X.500 define un protocolo de acceso al servicio que permite realizar operaciones de consulta, y también operaciones de modificación de la información de los objetos. Estas últimas operaciones normalmente sólo están permitidas a ciertos usuarios autorizados, y por tanto hacen falta mecanismos de autenticación de los usuarios, y estos mecanismos están definidos en la Recomendación X.509. Hay un mecanismo básico que utiliza contraseñas, y un mecanismo más avanzado que utiliza certificados.

A continuación podéis ver la estructura de un certificado X.509, con sus campos y subcampos. En la notación usada aquí, "rep." significa que el campo se puede repetir una o más veces, y "opc." significa que el campo es opcional (y por tanto "opc. rep." significa que el campo se puede repetir cero o más veces).

Campo	Tipo
`toBeSigned`	
`version` (opc.)	entero
`serialNumber`	entero
`signature`	
`algorithm`	identificador único
`parameters`	(depende del algoritmo)
`issuer`	DN
`validity`	
`notBefore`	fecha y hora
`notAfter`	fecha y hora
`subject`	DN
`subjectPublicKeyInfo`	
`algorithm`	
`algorithm`	identificador único
`parameters`	(depende del algoritmo)
`subjectPublicKey`	cadena de bits
`issuerUniqueIdentifier` (opc.)	cadena de bits
`subjectUniqueIdentifier` (opc.)	cadena de bits
`extensions` (opc. rep.)	
`extnId`	identificador único
`critical` (opc.)	booleano
`extnValue`	(depende de la extensión)
`algorithmIdentifier`	
`algorithm`	identificador único
`parameters`	(depende del algoritmo)
`encrypted`	cadena de bits

El cuerpo del certificado ("`toBeSigned`" en la tabla anterior) está formado por los siguientes campos:

- El campo `version` es el número de versión del formato: puede ser 1, 2 ó 3. Aunque es un campo opcional, es obligatorio si la versión es 2 ó 3 (por tanto, la ausencia de este campo indica que la versión es 1).
- El campo `serialNumber` es el número de serie del certificado, y sirve para distinguirlo de todos los demás que haya generado la misma CA.
- El campo `signature` indica el algoritmo utilizado por la CA para formar el certificado.
- El campo `issuer` es el nombre distintivo (DN) del firmante o **emisor** del certificado, es decir, de la CA que lo ha generado.
- El campo `validity` indica el período de validez del certificado, entre un instante inicial y un instante de caducidad. Cuando hablemos más adelante de las listas de revocación, veremos la utilidad de la fecha de caducidad en

Números de serie

Los números de serie de los certificados generados por una CA no tienen por qué ser consecutivos: es suficiente que cada uno sea distinto a todos los anteriores.

los certificados.

- El campo `subject` es el nombre distintivo (DN) del **sujeto** a nombre del cual esta emitido el certificado, es decir, el titular de la clave pública que figura en el certificado.
- En el campo `subjectPublicKeyInfo` se encuentre la clave pública del sujeto: a que algoritmo corresponde, y su valor.
- Los campos `issuerUniqueIdentifier` y `subjectUniqueIdentifier` se introdujeron en la versión 2, pero no se suelen utilizar porqué con el campo `extensions` son innecesarios.
- El campo `extensions` se introdujo en la versión 3, y es una lista de atributos adicionales del certificado. El subcampo `extnId` indica de que tipo de extensión se trata. El subcampo `critical` indica si la extensión es crítica o no: si lo es, las aplicaciones que no reconozcan el tipo de extensión han de considerar el certificado como no válido (este subcampo es opcional porqué por defecto las extensiones no son críticas). El subcampo `extnValue` es el valor concreto de la extensión.

Tipo de extensiones

Hay un cierto número de extensiones estándares, pero se pueden definir más de nuevas, según las necesidades de las aplicaciones, mientras cada una se identifique con un valor inambiguo del subcampo `extnId`.

Después del cuerpo hay el campo `algorithmIdentifier` (que en realidad es redundante con el campo `signature`), y finalmente hay el campo `encrypted` que es la firma del certificado, es decir, el resultado de cifrar con la clave privada de la CA el resumen (*hash*) del cuerpo del certificado.

Para calcular el resumen, el cuerpo del certificado se tiene que representar como una secuencia de bytes mediante una codificación no ambigua, que garantice que cualquiera que quiera verificar la firma pueda reconstruir exactamente la misma secuencia de bytes. Las reglas para obtener esta codificación se llaman DER, y forman parte de la notación ASN.1, que es la notación en que está definido el formato de los certificados X.509.

ASN.1 y DER

ASN.1 es la sigla de *Abstract Syntax Notation One*, y DER es la sigla de *Distinguished Encoding Rules*.

1.3.2. Cadenas de certificados y jerarquías de certificación

Un certificado nos soluciona el problema de la autenticidad de la clave pública si está firmado por una CA en la cual confiamos, Pero que pasa si nos comunicamos con un usuario que tiene un certificado emitido por una CA que no conocemos?

Existe la posibilidad que una CA tenga un certificado que garantice la autenticidad de su clave pública, firmado por otra CA. Esta otra CA puede que si que la conozcamos, o puede que a su vez tenga un certificado firmado por una tercera CA, y así sucesivamente. De esta forma, se puede establecer una jerarquía de autoridades de certificación, donde las CA de nivel más bajo emiten los certificados de usuario, y las CA de cada nivel son certificadas por una de nivel superior.

En un mundo ideal, podría haber una CA raíz que estuviera en la parte superior de la jerarquía y tuviera la máxima autoridad. En la práctica, esta CA raíz global no existe, y seguramente no existirá nunca (sería difícil que fuera aceptada por todo el mundo). En lugar de haber un sólo árbol que incluya a todas las CA del mundo, lo que existe en la realidad son árboles independientes (algunos , posiblemente con una sola CA), cada uno con su CA raíz.

Para que podamos verificar la autenticidad de su clave pública, un usuario nos puede enviar su certificado, más el certificado de la CA que lo ha emitido, más el de la CA que ha emitido este otro certificado, etc. hasta llegar al certificado de una CA raíz. Esto es lo que se denomina una **cadena de certificados**. Los certificados de las CA raíz tienen la propiedad de ser auto-firmados, es decir, están firmados por ellas mismas.

Un posible tipo de extensión de los certificados X.509 es `basicConstraints`, y un campo de su valor indica si el certificado es de CA (se puede usar su clave para emitir otros certificados) o no. En el caso de que lo sea, otro subcampo (`pathLenConstraint`) permite indicar el número máximo de niveles de jerarquía que se sitúan por debajo de esta CA.

1.3.3. Listas de revocación de certificados (CRL)

La Recomendación X.509, además de definir el formato de los certificados, también define otra estructura llamada **lista de revocación de certificados** o CRL. Una lista de este tipo sirve para publicar los certificados que han dejado de ser válidos antes de su fecha de caducidad. Los motivos pueden ser diversos: se ha emitido otro certificado que sustituye al revocado, ha cambiado el DN del titular (por ejemplo, ha dejado de trabajar en la empresa en la que estaba), le han robado su clave privada, etc.

CRL

CRL es la sigla de *Certificate Revocation List*.

De este modo, si queremos asegurarnos completamente de la validez de un certificado, no basta con verificar su firma, sino que debemos obtener la versión actual de la CRL (publicada por la CA que emitió el certificado) y comprobar que el certificado no aparece en esta lista.

Una CA normalmente actualizará su CRL de forma periódica, añadiendo cada vez los certificados que hayan sido revocados. Cuando llegue la fecha de caducidad que constaba en el certificado, ya no será necesario volver a incluirlo en la CRL. Esto permite que las CRL no crezcan indefinidamente.

Éstos son los campos de una CRL:

Campo	Tipo
`toBeSigned`	
`version` (opc.)	entero
`signature`	
`algorithm`	identificador único
`parameters`	(depende del algoritmo)
`issuer`	DN
`thisUpdate`	fecha y hora
`nextUpdate` (opc.)	fecha y hora
`revokedCertificates` (opc. rep.)	
`userCertificate`	entero
`revocationDate`	fecha y hora
`crlEntryExtensions` (opc. rep.)	
`extnId`	identificador único
`critical` (opc.)	booleano
`extnValue`	(depende de la extensión)
`crlExtensions` (opc. rep.)	
`extnId`	identificador único
`critical` (opc.)	booleano
`extnValue`	(depende de la extensión)
`algorithmIdentifier`	
`algorithm`	identificador único
`parameters`	(depende del algoritmo)
`encrypted`	cadena de bits

El campo `issuer` identifica la CA que emite la CRL, y en el campo `revokedCertificates` se encuentra la lista de los certificados revocados (cada uno identificado por su número de serie). A las extensiones de cada elemento de la lista puede haber, por ejemplo, el motivo de la revocación.

2. Sistemas de autenticación

Uno de los servicios de seguridad que se requiere en muchas aplicaciones es el de la **autenticación**. Este servicio permite garantizar que nadie ha falsificado la comunicación.

Podemos distinguir dos tipos de autenticación:

- La **autenticación de mensaje** o **autenticación de origen de datos** permite confirmar que el originador *A* de un mensaje es auténtico, es decir, que el mensaje no ha sido generado por un tercero *Z* que quiere hacer creer que lo ha generado *A*.

  Como efecto adicional, la autenticación de mensaje proporciona implícitamente el servicio de **integridad de datos**, que permite confirmar que nadie ha modificado un mensaje enviado por *A*.
- La **autenticación de entidad** permite confirmar la identidad de un participante *A* en una comunicación, es decir, que no se trata de un tercero *Z* que dice ser *A*.

A continuación veremos cómo se puede conseguir cada uno de estos dos tipos de autenticación.

2.1. Autenticación de mensaje

Existen dos grupos de técnicas para proporcionar autenticación de mensaje:

- Los **códigos de autenticación de mensaje** o **MAC**, basados en claves simétricas.
- Las **firmas digitales**, que se basan en la criptografía de clave pública.

MAC

MAC es la sigla de *Message Authentication Code*.

2.1.1. Códigos de autenticación de mensaje (MAC)

Un **código de autenticación de mensaje** o **MAC** se obtiene con un algoritmo a que tiene dos entradas: un mensaje M de longitud arbitraria, y una clave secreta k compartida por el originador y el destinatario del mensaje. Como resultado da un código $C_{\mathrm{MAC}} = a(k,M)$ de longitud fija. El algoritmo MAC debe garantizar que sea computacionalmente inviable encontrar un mensaje $M' \neq M$ que de el mismo código que M, y también obtener el código de un mensaje cualquiera sin conocer la clave.

Las propiedades de los algoritmos MAC hacen que dado un par (M, C_{MAC}) un atacante no pueda obtener otro par (M', C_{MAC}), ni en general cualquier par (M', C'_{MAC}). Por tanto, el código MAC sirve como prueba de autenticidad del mensaje.

Un posible algoritmo MAC consiste en aplicar al mensaje M un cifrado en bloque en modo CBC con la clave k, y tomar el último bloque cifrado como código C_{MAC}. Normalmente, pero, los algoritmos MAC usados actualmente, suelen estar basados en una función *hash*. Por ejemplo, la técnica de calcular el resumen a partir de la concatenación del mensaje y la clave, o la de calcular el resumen del mensaje y cifrarlo con la clave, podrían servir como algoritmos MAC. Para mejorar la seguridad contra ciertos ataques, muchos protocolos usan una técnica de autenticación de mensajes un poco más sofisticada, conocida como HMAC.

Código HMAC

Para calcular el código HMAC, al mensaje se le añade como prefijo una cadena de bits derivada de la clave, se calcula su *hash*, al resultado se le prefija otra cadena de bits derivada de la clave, y se vuelve a calcular el *hash*. (Aun que se tenga que llamar dos veces la función *hash*, la segunda será mucho más rápida puesto que los datos a resumir serán más cortos.)

2.1.2. Firmas digitales

Los códigos MAC, dado que se basan en una clave secreta, sólo tienen significado para quienes conozcan dicha clave. Si A envía mensajes a B autenticados con una clave compartida, sólo B podrá verificar la autenticidad de estos mensajes.

Por otro lado, en caso de un conflicto en que A denegase la autoría de un mensaje autenticado, B no podría demostrar delante de un tercero imparcial (por ejemplo, un árbitro o un juez) que el mensaje lo generó A. Revelar la clave secreta no sería prueba suficiente ya que, por el hecho de ser conocida por las dos partes, siempre habría la posibilidad que el mensaje en disputa y el su código de autenticación los hubiera generado B.

En cambio, si A autentica los mensajes adjuntándoles la firma digital calculada con su clave privada, todo el mundo podrá verificarlos con su clave pública. Esta técnica de autenticación proporciona, como efecto adicional, el servicio

de **no repudio**. Esto quiere decir que un destinatario *B* puede demostrar fehacientemente ante un tercero que un mensaje ha sido generado por *A*.

Como hemos visto en el subapartado 1.2, los algoritmos de firma digital usados normalmente se basan en el cálculo de un *hash* y en un cifrado mediante una clave privada. Son ejemplos de algoritmos de firma el RSA, el ElGamal, y el estándar DSA (*Digital Signature Algorithm*).

2.2. Autenticación de entidad

La autenticación de entidad se utiliza cuando en una comunicación una de las partes quiere asegurarse de la identidad de la otra. Normalmente, esta autenticación es un requisito para permitir el acceso a un recurso restringido, como, por ejemplo, una cuenta de usuario en un ordenador, dinero en efectivo en un cajero automático, acceso físico a una habitación, etc.

En general, las técnicas utilizadas para la identificación de un usuario *A* pueden estar basadas en:

- Algo que *A sabe* como, por ejemplo, una contraseña o una clave privada.
- Algo que *A tiene* como, por ejemplo, una tarjeta con banda magnética o con chip.
- Algo que *A es* o, dicho de otro modo, alguna propiedad inherente a *A* como, por ejemplo, sus características biométricas.

Técnicas biométricas

Las técnicas biométricas hacen uso de características fisiológicas humanas (como la huella dactilar, el iris, la retina, la cara o la mano) o características del comportamiento humano (como el habla, la firma manual o la pulsación de teclas).

La biometría es una disciplina relativamente moderna que es posible que tenga una cierta implantación en un futuro próximo. Aun así, en este módulo nos centraremos en las técnicas basadas en el intercambio de información por medios electrónicos.

Una diferencia entre la autenticación de mensaje y la autenticación de entidad es que la primera puede ser intemporal (es posible verificar la autenticidad de un documento firmado, por ejemplo, diez años atrás) mientras que la segunda normalmente se realiza en tiempo real. Esto quiere decir que para la autenticación de entidad se puede llevar a cabo un protocolo interactivo, en el que ambas partes se intercambien mensajes hasta que la identidad en cuestión quede confirmada.

A continuación veremos dos grupos de técnicas que se pueden utilizar para la autenticación de entidad:

- Las basadas en **contraseñas** (o *passwords*, en inglés), también llamadas *técnicas de autenticación débil*.
- Las basadas en **protocolos de reto-respuesta** (*challenge-response*, en in-

glés), también llamadas técnicas de autenticación fuerte.

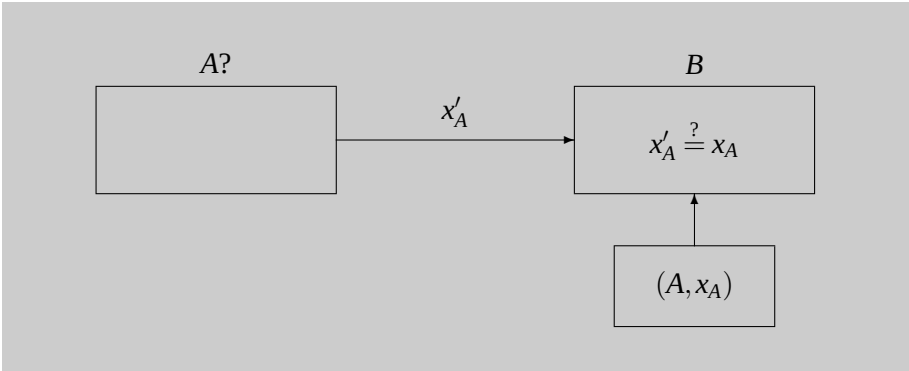
2.2.1. Contraseñas

La idea básica de la autenticación basada en contraseñas es que el usuario A manda su identidad (su identificador de usuario, su nombre de *login*, etc.) seguida de una contraseña secreta x_A (una palabra o combinación de caracteres que el usuario pueda memorizar). El verificador B comprueba que la contraseña sea válida, y si lo es da por buena la identidad de A.

Existen distintas maneras de llevar a cabo esta autenticación, y también hay maneras diversas de intentar atacarla. A continuación veremos algunas variantes de la autenticación con contraseñas que intentan evitar determinados tipos de ataques.

Lista de contraseñas en claro

La manera más simple de comprobar una contraseña es que el verificador tenga una lista de las contraseñas asociadas a los usuarios, es decir, una lista de pares (A, x_A). Cuando A envía su contraseña, el verificador compara directamente el valor recibido x'_A con el que figura en la lista, x_A.



Si se trata de las contraseñas correspondientes a usuarios de un sistema informático, es evidente que no pueden estar guardadas en un fichero de acceso público: es necesario que la lista esté protegida contra lectura. Pero si alguien encuentra la manera de saltarse esta protección (cosa que en ocasiones pasa en los sistemas multiusuario), automáticamente tendrá acceso a las contraseñas de todos los usuarios.

A demás, en estos sistemas normalmente hay un usuario administrador o "super-usuario" que tiene acceso a todos los ficheros, y por tanto a las contraseñas de los usuarios. Un super-usuario que fuera mal intencionado podría hacer un mal uso de este privilegio. O un super-usuario que tuviera un momento de descuido podría dar pie a que otro usuario se aprovechara.

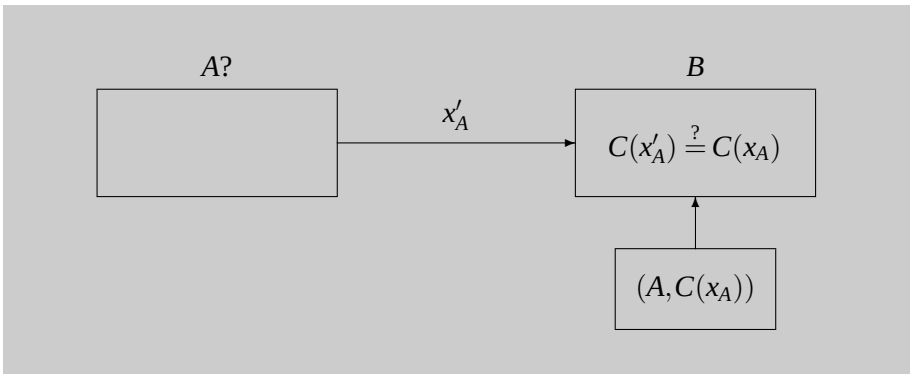
Lista de contraseñas codificadas

Una segunda opción consiste en que en la lista de contraseñas, en lugar de estar guardadas éstas en claro por pares (A, x_A), cada una esté codificada con alguna transformación C de manera que no se pueda deducir su valor real. De este modo, la lista debe contener pares $(A, C(x_A))$.

Esta codificación C podría ser, por ejemplo, un cifrado, pero entonces sería preciso tomar las precauciones oportunas con la clave de descifrado (quien consiguiera acceder a esta clave podría obtener todas las contraseñas). Lo más habitual es que la codificación consista en la aplicación de una función unidireccional como ,

por ejemplo, una función *hash*.

Para la verificación, B debe calcular el valor transformado $C(x'_A)$ a partir de la contraseña recibida, y compararlo con el que consta en la lista.



Codificación de contraseñas en Unix

Un caso muy conocido es el de la gestión de las contraseñas en el sistema operativo Unix, ya que la mayoría de las variantes de Unix usadas actualmente utilizan técnicas muy parecidas, basadas en las versiones originales de este sistema.

En Unix existe un fichero en el que se guardan las contraseñas de los usuarios codificadas con una función unidireccional. Pero en lugar de una función *hash*, lo que se utiliza es un cifrado simétrico: el texto en claro es fijo (una cadena de bits todos iguales a 0), como clave de cifrado se utiliza la contraseña, y lo que se guarda en la lista es el texto cifrado. De esta forma se aprovecha la propiedad que tienen los algoritmos criptográficos que no permiten deducir la clave a partir del texto cifrado ni que se conozca el texto en claro.

Dado que la codificación de las contraseñas es unidireccional, nadie (ni tan siquiera el superusuario) puede saber cuál es la contraseña de cada usuario aunque tenga acceso a la lista. Esto haría innecesaria la protección contra lectura del fichero en el que hay la lista.

Pero, aunque la codificación no sea reversible, puede ser vulnerable a otro tipo de ataque: la fuerza bruta. Si el atacante conoce la contraseña codificada $C(x_A)$

y conoce el algoritmo de codificación C, puede probar de codificar posibles contraseñas hasta encontrar un resultado que coincida con $C(x_A)$.

Existe un hecho que favorece al atacante en este caso: si los usuarios pueden escoger sus contraseñas, normalmente no usarán combinaciones arbitrarias de caracteres sino palabras fáciles de recordar. Por tanto, el espacio de búsqueda se reduce considerablemente. El atacante puede, pues, limitarse a probar palabras de un diccionario o combinaciones derivadas a partir de estas palabras. Por este motivo a este tipo de ataque se lo conoce como **ataque de diccionario**.

Espacio de contraseñas

Si consideramos las posibles combinaciones de, por ejemplo, 8 caracteres (suponiendo cada carácter comprendido entre los códigos ASCII 32 y 126, es decir, 95 caracteres distintos), existen 95^8 (aprox. $6.6 \cdot 10^{15}$). En cambio, hay estudios que indican que una gran parte de las contraseñas que escogen los usuarios se pueden encontrar en un diccionario de 150000 palabras, un espacio 10^{10} veces menor.

Programas "rompe-contraseñas"

Desde hace años existen los programas "rompe-contraseñas", o *"password crackers"* en inglés. A un programa de este tipo se le da una lista de contraseñas codificadas y un diccionario, y va probando cada una de las palabras, tanto directamente como con diferentes variaciones: todo minúsculas, todo mayúsculas, la primera letra mayúscula, con las letras al revés, añadiendo una cifra numérica, añadiendo dos, cambiando ciertas letras por cifras, etc. También puede probar combinaciones con datos adicionales como los identificadores de los usuarios, sus nombres, apellido, etc.

Es sorprendente la cantidad de contraseñas que estos programas pueden llegar a encontrar en sólo unas pocas horas.

Técnicas para dificultar los ataques de diccionario

A continuación veremos algunas posibles técnicas para dificultar los ataques de diccionario.

1) Ocultar la lista de contraseñas codificadas

Una primera solución es restringir el acceso a la lista de contraseñas, protegiéndola contra lectura. Aun que en la lista aparezcan las contraseñas codificadas, el acceso a esta lista por parte de un atacante le permite realizar cómodamente un ataque de diccionario.

Lista de contraseñas codificadas en Unix

En las versiones antiguas del sistema Unix, las contraseñas codificadas estaban en el fichero `/etc/passwd`. Este mismo fichero se aprovechaba para guardar otros datos sobre la cuenta de cada usuario: su directorio inicial, su *shell*, el nombre del usuario, etc. Dado que esta información es de acceso público, el fichero `/etc/passwd` tenia permiso de lectura para todos los usuarios.

En las versiones modernas de Unix continua existiendo un fichero `/etc/passwd` de lectura pública con información sobre las cuentas de los usuarios, pero las contraseñas codificadas ya no se guardan en este fichero sino en otro, `/etc/shadow`, sobre el cual ningún usuario (excepto el super-usuario) tiene permiso de lectura.

Si el atacante no tiene acceso a la lista de contraseñas, ya no podrá realizar un ataque "fuera de línea" (*off-line*), es decir, un ataque realizado en un ordenador escogido por el atacante a su conveniencia (por ejemplo, un or-

denador potente, donde nadie pueda ver que está haciendo, etc.). No le quedará más remedio que realizar el ataque "en línea" (*on-line*), es decir, directamente con el verificador real (por ejemplo, el programa `login` del sistema que quiere atacar), enviándole contraseñas para ver si las acepta o no.

Si el verificador ve que alguien está realizando pruebas de autenticación fallidas, esto es una señal de que puede tratarse de un ataque en línea. Entonces puede tomar ciertas medidas para contrarrestar este ataque. Por ejemplo, se puede conseguir que si se recibe un cierto número de contraseñas inválidas consecutivas, el recurso asociado quede bloqueado.

Protección contra ataques en las tarjetas con PIN

El mecanismo del bloqueo se utiliza típicamente en los métodos de autenticación basados en dispositivos físicos, como las tarjetas bancarias o los módulos SIM (*Subscriber Identity Module*) de los teléfonos móviles. Estos dispositivos requieren que el usuario introduzca un PIN (*Personal Identification Number*), que hace las funciones de contraseña.

Por motivos históricos y de conveniencia de los usuarios, este PIN es un número de longitud corta (por ejemplo de cuatro cifras). En este caso, para evitar los ataques de fuerza bruta (que sólo necesitarían 10.000 intentos como máximo), el dispositivo se bloquea, por ejemplo, al tercer intento equivocado.

La protección que proporciona el bloqueo puede ser un inconveniente para el usuario legítimo que no tiene ninguna culpa de que alguien haya intentando descubrir su contraseña. Para evitarle este inconveniente, una posibilidad es que cada usuario tenga asociada otra contraseña de desbloqueo, mucho más difícil de adivinar (por ejemplo, un PIN de ocho cifras para desbloquear el PIN de cuatro cifras).

Otra posibilidad consiste en permitir múltiples intentos de autenticación, pero en lugar de responder inmediatamente con un mensaje de "contraseña incorrecta", demorar esta respuesta con un tiempo de retraso, que irá creciendo a medida que se vayan enviando más contraseñas erróneas por parte del mismo usuario. Esto ralentizaría tanto un ataque que lo haría inviable a partir de unos pocos intentos. Y cuando el usuario legítimo utilice su contraseña correcta, el tiempo de respuesta será el normal.

2) Reglas para evitar contraseñas fáciles

La solución de ocultar la lista de contraseñas codificadas da una seguridad parecida a la de la lista de contraseñas en claro. Si alguien descubre la lista (saltándose la protección contra lectura), puede realizar sin más problemas un ataque de diccionario.

Otro modo de dificultar este ataque es obligar a los usuarios a escoger contraseñas que cumplan unas determinadas reglas para que no sean fáciles de adivinar. Por ejemplo:

- Que la contraseña tenga una longitud mínima.

- Que no sea todo letras ni todo números.
- Que las letras no coincidan con ninguna palabra de diccionario ni con combinaciones triviales de las palabras (como escribir las letras al revés, etc.).
- Que la contraseña no se derive del identificador del usuario, de su nombre, apellido, etc.
- Etc.

Con estas reglas se consigue que el espacio de contraseñas donde hacer la búsqueda sea más grande del que es habitual, y el ataque de diccionario necesite más tiempo.

Por otro lado, también es cierto que como más tiempo mantenga un usuario su contraseña sin cambiar, más grande será la probabilidad que un atacante consiga descubrirla. Por esto es recomendable que los usuarios renueven las contraseñas periódicamente. Hay sistemas que implantan una política de cambio forzado de contraseñas. A partir de un cierto período (por ejemplo, 90 días) la contraseña se considera caducada y el usuario la tiene que cambiar por una de nueva.

Caducidad de las contraseñas

Un tiempo de caducidad de las contraseñas demasiado corto también podría ser contraproducente, porque si los usuarios deben pensar muchas contraseñas en poco tiempo, pueden acabar olvidando cual era la última, o anotándola en un papel, con lo cual la contraseña es mucho más vulnerable.

3) Añadir complejidad a la codificación de las contraseñas

Otra solución para dificultar los ataques de diccionario es ralentizarlos haciendo que cada contraseña cueste más de codificar. Por ejemplo, si el algoritmo de codificación no es directamente una función *hash* sino un bucle de N llamadas a la función *hash*, probar cada contraseña cuesta N veces más.

El valor N se puede escoger tan grande como se quiera, pero teniendo en cuenta que si es demasiado grande los usuarios legítimos también podrán notar que la autenticación normal es más lenta.

Complejidad de la codificación de las contraseñas en Unix

La solución que se adoptó en Unix fue repetir N veces el cifrado para obtener la contraseña codificada, y se escogió el valor $N = 25$.

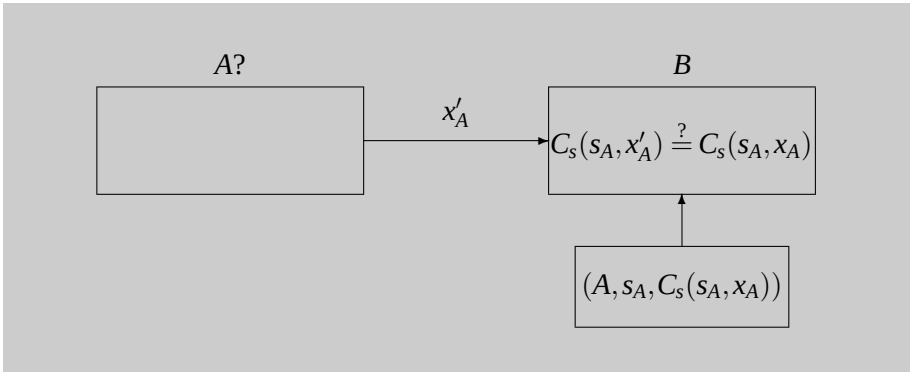
Por tanto, aun que el atacante disponga de una implementación rápida del algoritmo de cifrado, el tiempo esperado para romper una contraseña se multiplica por 25.

4) Añadir bits de sal a la codificación de las contraseñas

En el subapartado 1.1 hemos visto que, para variar el resultado del cifrado aun que se utilice la misma clave con el mismo texto en claro, hay la posibilidad de introducir los llamados "bits de sal" para modificar la clave. De la misma manera, se puede definir un algoritmo de codificación C_s que, a más de la contraseña, tenga como entrada unos bits de sal.

De este modo, cada vez que un usuario A cambie su contraseña x_A, se generan aleatoriamente los bits de sal s_A y se guardan estos bits juntamente con la contraseña codificada $C_s(s_A, x_A)$. A la hora de verificar, se vuelve a

calcular este valor a partir de los mismos bits de sal y se compara con el valor guardado.



Esta técnica no complica la verificación de una contraseña ni un ataque de diccionario contra esta contraseña específica. Pero si el ataque de diccionario se realiza sobre distintas contraseñas a la vez (por ejemplo, sobre la lista entera de todos los usuarios, como hacen normalmente los programas rompe-contraseñas), el tiempo de cálculo se multiplica por el número total de contraseñas, suponiendo que todas estén codificadas con bits de sal distintos.

Esto es así porque, sin bits de sal, se puede tomar cada palabra del diccionario, codificarla, y comparar el resultado con todas las contraseñas codificadas. En cambio, si hay bits de sal, es preciso realizar una codificación separada para cada contraseña, cada una con sus bits de sal correspondientes.

Con los bits de sal también se evita una variante más eficiente del ataque de diccionario, en la cual el atacante no va probando una a una las palabras, sino que utiliza una lista preconstruida con las palabras del diccionario ya codificadas. Si, por ejemplo, el diccionario contiene 150.000 palabras, cuando no se utilizan bits de sal el atacante sólo tiene que buscar la contraseña codificada en una lista de 150.000 palabras codificadas. Pero si se utilizan 12 bits de sal, el tamaño de la lista se multiplicaría por más de 4.000 ($2^{12} = 4.096$), y el atacante debería trabajar con una lista de más de seiscientos millones de palabras codificadas.

Otra ventaja de los bits de sal es que si, por casualidad, dos usuarios eligen la misma contraseña, los valores codificados de sus contraseñas serán distintos (si los bits de sal también lo son), y aunque vieran la lista de contraseñas, no sabrían que ambos tienen la misma.

Bits de sal en las contraseñas en Unix

En Unix se utilizan precisamente 12 bits aleatorios de sal (que se obtienen a partir de los bits de menos peso de la hora en el momento en que el usuario se cambia la contraseña). Esto significa que hay 4.096 valores posibles de estos bits. Una lista completa de contraseñas codificadas a partir de un diccionario de 150.000 palabras tendría que contener, pues, más de seiscientos millones de entradas. Esto podría ser una cantidad descomunal en la época en que se desarrolló el sistema Unix, pero actualmente una lista de estas dimensiones cabe perfectamente en el disco duro de

cualquier ordenador personal.

Por otro lado, el algoritmo de cifrado que se utiliza en Unix para obtener la codificación de las contraseñas es el DES. Com hemos visto antes, para codificar una contraseña se aplica un cifrado a un texto formado por todos los bits a 0, y se usa la contraseña como clave de cifrado (esto explica porque las versiones estándar de Unix sólo utilizan contraseñas de hasta 8 caracteres). En este caso, los bits de sal no se utilizan para modificar la clave, sino que sirven para permutar ciertos bits de los resultados intermedios que se obtienen en cada iteración del algoritmo DES.

Con esto se consigue otro objetivo: que un atacante no pueda utilizar directamente una implementación eficiente del algoritmo DES (las más eficientes son los chips DES, es decir, las implementaciones con hardware), sino que tenga que introducir modificaciones (si se trata de un chip DES, es preciso ciertos recursos que no siempre estan al alcance de cualquiera).

5) Uso de *passphrases*

La propiedad que aprovechan los ataques de diccionario es que el conjunto de contraseñas que utilizan normalmente los usuarios es un subconjunto muy pequeño de todo el espacio de contraseñas posibles.

Para ampliar este espacio se puede hacer que el usuario no utilice palabras relativamente cortas, de unos 8 caracteres, sino frases más largas, llamadas *"passphrases"*. La codificación de estas *passphrases* puede continuar siendo una función unidireccional, como por ejemplo una función *hash* (con la misma longitud que en el caso de las contraseñas). La diferencia es que, si antes la lista de contraseñas codificadas contenía valores de entre un total de unos 150000 distintos, con *passphrases* contendrá muchísimos más valores posibles, y la búsqueda exhaustiva del ataque de diccionario será mucho más larga.

Contraseñas de un solo uso

Todos los esquemas de autenticación con contraseñas que hemos visto hasta ahora, a parte de los ataques de diccionario, son vulnerables a los **ataques de repetición** o de *"replay"*. Para llevar a cabo este ataque, hace falta interceptar la comunicación entre A y B durante una autenticación, por ejemplo utilizando un *sniffer*, y ver qual es el valor x_A enviado. Entonces el atacante Z sólo tiene que realizar una autenticación delante B enviándole este mismo valor x_A, y B se pensará que se trata del auténtico usuario A.

Sniffers

Ver los *sniffers* en el módulo didáctico "Vulnerabilidades en redes TCP/IP".

Este tipo de ataque se evita con los protocolos de autenticación fuerte que veremos a continuación. Pero existe otra técnica que, aun que se puede considerar basada en contraseñas, tiene algunas propiedades de la autenticación fuerte, entre ellas la resistencia a los ataques de repetición. Esta técnica es la de las llamadas **contraseñas de un solo uso** (*"one-time passwords"*).

Como su nombre indica, las contraseñas de un solo uso son contraseñas que sólo son válidas una vez, y a la vez siguiente es preciso utilizar una contraseña distinta. Así, aun que el atacante vea que valor se está enviando para la auten-

ticación, no le servirá de nada porqué ya no lo podrá volver a utilizar.

Existen distintas posibilidades para implementar la autenticación con contraseñas de un solo uso. Por ejemplo, podemos considerar las siguientes:

- Lista de contraseñas compartida: A y B se ponen de acuerdo, de manera segura (es decir, no a través de un canal que pueda ser interceptado), en una lista de N contraseñas. Si se trata de una lista ordenada, cada vez que A se tenga que autenticar utilizará la siguiente contraseña de la lista. Alternativamente, las contraseñas pueden tener asociado un identificador: en cada autenticación B escoge una contraseña de las que aún no se hayan utilizado, envía a A su identificador, y A debe responder con la contraseña correspondiente.
- Contraseñas basadas en una función unidireccional (normalmente una función *hash*): A escoge un valor secreto x_0 y utiliza una función unidireccional h para calcular los siguientes valores:

$$\begin{aligned} x_1 &= h(x_0) \\ x_2 &= h(x_1) = h^2(x_0) \\ &\vdots \\ x_N &= h(x_{N-1}) = h^N(x_0) \end{aligned}$$

  y envía el valor x_N a B (B debe asegurarse de que ha recibido este valor del usuario A auténtico).

  Entonces A inicializa un contador y a N. Cada vez que se tenga que autenticar, A enviará el valor x_{y-1}, y B aplicará la función unidireccional a este valor, lo comparará con el que tiene guardado, y comprobará si $h(x_{y-1}) = x_y$. Si se cumple esta igualdad, la contraseña es válida. Para la siguiente vez, B sustituye el valor que tenía guardado, x_y, por el que acaba de recibir, x_{y-1}, y A actualiza el contador y disminuyéndolo en 1.

  Dado que la función h es unidireccional, nadie que vea el valor x_{y-1} sabrá cuál es el valor que corresponderá la próxima vez (x_{y-2}), ya que A obtuvo x_{y-1} como $h(x_{y-2})$, y hacer el cálculo inverso (es decir, obtener x_{y-2} a partir de x_{y-1}) debe ser computacionalmente inviable.

La técnica de las contraseñas basadas en una función *hash* es más eficiente que la de la lista compartida, porque B sólo debe recordar el último valor x_y recibido.

Implementaciones de las contraseñas de un solo uso

Existen varias implementaciones de contraseñas basadas en una función *hash* como, por ejemplo, **S/KEY** o **OPIE**. Con estos programas, cuando el usuario A está conectado de manera segura (por ejemplo localmente, es decir, frente a la consola) al sistema servidor con el que se quiere autenticar, puede generar una lista de N contraseñas, imprimirla, y llevarla encima cada vez que deba realizar una autenticación desde un sistema remoto. Para facilitar la introducción de las contraseñas, el programa convierte los bits en una secuencia de palabras cortas. Por ejemplo:

```
...
90: BAND RISE LOWE OVEN ADEN CURB
91: JUTE WONT MEEK GIFT OWL PEG
92: KNOB QUOD PAW SEAM FEUD LANE
...
```

La traducción se hace a partir de una lista de 2048 palabras de hasta 4 caracteres, y a cada una se le asigna una combinación de 11 bits.

El usuario *A* no necesita recordar el valor del contador *y*, ya que este valor se guarda en el servidor. A cada autenticación, el servidor envía el valor actual del contador, el usuario lo utiliza para consultar su lista, y responde con la contraseña correspondiente. Para la siguiente vez, el servidor actualizará el contador decrementándolo en 1.

2.2.2. Protocolos de reto-respuesta

El problema que tienen los esquemas de autenticación basados en contraseñas es que cada vez que se quiere realizar la autenticación se tiene que enviar el mismo valor al verificador (excepto en las contraseñas de un solo uso, como acabamos de ver). Cualquier atacante que consiga interceptar este valor fijo podrá suplantar la identidad del usuario a quien corresponda la contraseña.

Hay otro grupo de mecanismos donde el valor que se envía para la autenticación no es fijo, sino que depende de otro, generado por el verificador. Este último valor se llama **reto**, y se debe enviar al usuario *A* como primer paso para su autenticación. Entonces *A*, haciendo uso de una clave secreta, calcula una **respuesta** a partir de este reto, y lo envía al verificador *B*. Por este motivo, estos mecanismos de autenticación reciben el nombre de **protocolos de reto-respuesta**.

El algoritmo para calcular la respuesta debe garantizar que no se pueda obtener sin saber la clave secreta. Esto permite al verificador confirmar que la respuesta sólo ha podido enviarla *A*. Si se utiliza un reto distinto cada vez, un atacante no podrá sacar provecho de la información que descubra interceptando la comunicación.

Dependiendo del protocolo, el verificador puede generar los retos de diversas maneras:

- Secuencialmente: en este caso el reto es simplemente un número que se va incrementando cada vez (lo más normal es incrementarlo de uno en uno), y que por tanto no se repetiré nunca.
- Aleatoriamente: el reto puede ser generado con un algoritmo pseudoaleatorio, pero la propiedad que tiene en este caso es que no es predecible para los atacantes.
- Cronológicamente: el reto se obtiene a partir de la fecha y hora actuales (con la precisión que sea adecuada para el protocolo). Este tipo de reto también se llama **marca de hora** o *"timestamp"*. El receptor, es decir

quien quiere validar la identidad, puede utilizar la marca de hora para saber si se trata de una nueva autenticación o si alguien quiere reutilizar los mensajes de otra autenticación para intentar un ataque de repetición.

La ventaja de los retos cronológicos es que, para obtenerlos, sólo es necesario un reloj, que seguramente estará disponible en cualquier sistema, mientras que con los secuenciales y los aleatorios es preciso mantener información de estado (el número de secuencia o la entrada para calcular el siguiente número pseudoaleatorio) para evitar repeticiones. El inconveniente es que es precisa una cierta sincronía entre los relojes. Si se utilizan técnicas como un servidor de tiempo que da la hora actual, también es preciso verificar que la hora obtenida sea auténtica, es decir, que un atacante no nos está engañando haciéndonos creer que es la hora que le interesa.

Sincronía de relojes

Si el reto se obtiene a partir de un reloj, dependiendo del protocolo puede ser necesario que emisor y receptor tengan los relojes sincronizados, o bien que se permita una cierta tolerancia entre la hora que ha enviado el emisor y la que indica el reloj del receptor. La tolerancia tiene que ser tal que permita diferencias entre los relojes pero que no de tiempo a un tercero a realizar un ataque de repetición.

Dispositivos par el cálculo de las respuestas

El cálculo de la respuesta se puede hacer con algún software diseñado a tal efecto, o también se puede utilizar un dispositivo, parecido a una pequeña calculadora, que guarda la clave secreta. El usuario introduce el reto mediante el teclado de esta calculadora, y en la pantalla aparece la respuesta.

Alternativamente, si el reto es cronológico, el dispositivo no tiene teclado sino que va mostrando por la pantalla las respuestas en función de la hora actual (por ejemplo, cada minuto), en sincronía aproximada con el reloj del verificador.

Para más seguridad, el dispositivo también puede estar protegido con un PIN.

Los protocolos de reto-respuesta se pueden clasificar en dos grupos:

- Los basados en técnicas simétricas, en las cuales la clave secreta es compartida por el usuario A y el verificador B.
- Los basados en técnicas de clave pública, en las cuales A utiliza una clave privada para calcular la respuesta.

En la descripción de los protocolos que veremos a continuación, utilizaremos la notación $\{a, b, \ldots\}$ para representar un mensaje que contiene los componentes $a, b, \ldots$, y si alguno de estos componentes es opcional, lo representaremos entre corchetes como, por ejemplo, $\{a, [b]\}$.

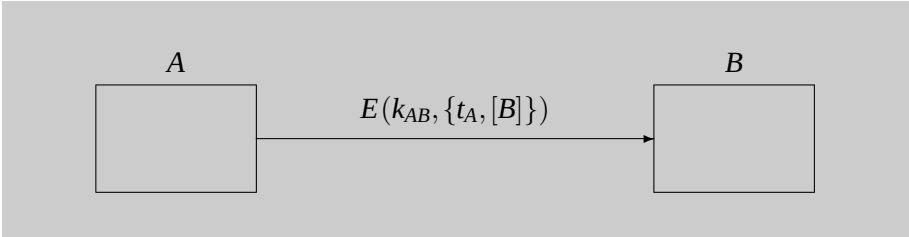
Protocolos de reto-respuesta con clave simétrica

Si el protocolo de reto-respuesta se basa en una clave k_{AB} compartida por A y B, existen distintas formas de obtener esta clave. Por ejemplo, la pueden haber acordado directamente A y B de forma segura, o bien la pueden solicitar a un servidor de claves centralizado.

1) Autenticación con marca de tiempo

El protocolo más simple es el que utiliza como reto implícito la hora ac-

tual (no es preciso el envío del reto). El usuario A obtiene una marca de tiempo t_A y la manda al verificador B cifrada con la clave compartida.



El verificador descifra el mensaje recibido y comprueba si la marca de tiempo es aceptable, es decir, si está dentro la tolerancia de sincronía y no esta repetida.

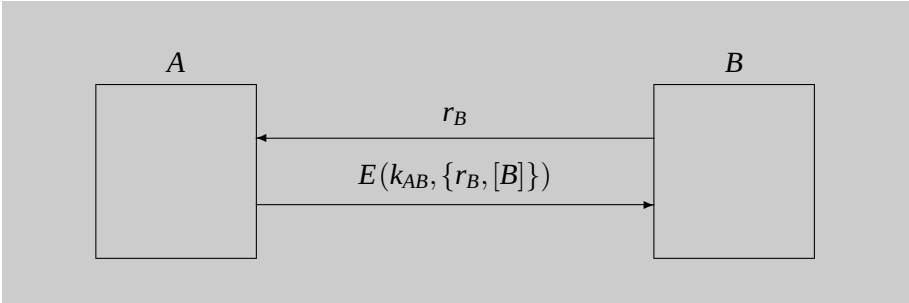
Si se utiliza la misma clave en los dos sentidos (de A a B y de B a A), el hecho de incluir la identidad del verificador B evita que, en un proceso de autenticación mutua, un atacante intente hacerse pasar por B delante de A repitiendo el mensaje en sentido contrario.

Detección de repeticiones

Para saber si se esta reutilizando una marca de tiempo, el verificador necesita recordar cuales se han utilizado ya, pero sólo las que estén dentro del intervalo de tolerancia. Pasado este intervalo ya se puede borrar la marca de la lista.

2) Autenticación con números aleatorios

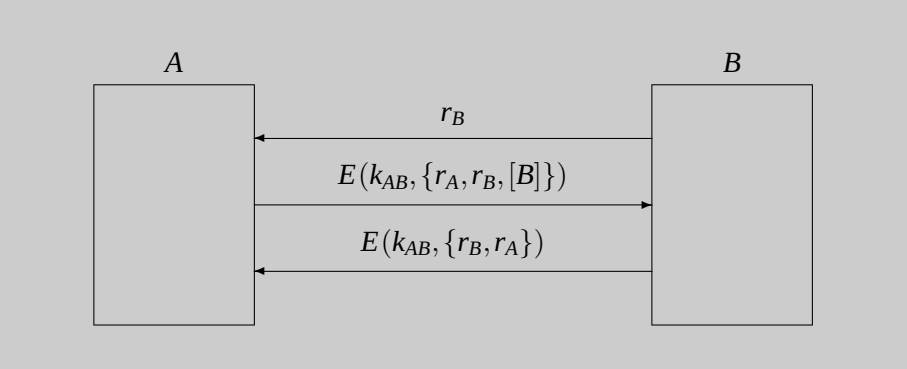
En este caso es preciso enviar explícitamente el reto, que consiste en un número aleatorio r_B generado por B. La respuesta es el reto cifrado con la clave compartida.



El verificador descifra el mensaje y comprueba que el número aleatorio que contiene es igual al reto. El hecho de incluir la identidad del verificador B evita que el mensaje se pueda utilizar en sentido contrario si alguna vez A genera como reto el mismo número r_B.

3) Autenticación mutua con números aleatorios

Con tres mensajes, el protocolo anterior se puede convertir en un protocolo de autenticación mutua, es decir, de A delante B y de B delante A. En este caso, A debe generar otro número aleatorio r_A, que actuará como reto en sentido inverso, y incluirlo en su respuesta.



Cuando B descifra la respuesta, además de comprobar que el valor r_B sea el esperado, también obtiene r_A, que deberá utilizar para enviar su respuesta a A. Entonces A comprueba que los dos números aleatorios r_A y r_B tengan los valores correctos.

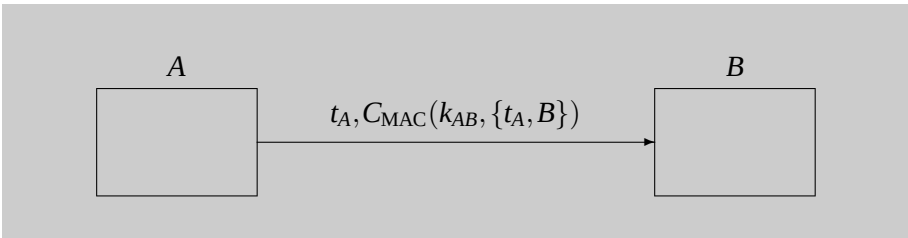
Como sucedía anteriormente, la inclusión del identificador de B en el segundo mensaje evita ataques de repetición en sentido contrario.

4) Autenticación con función unidireccional

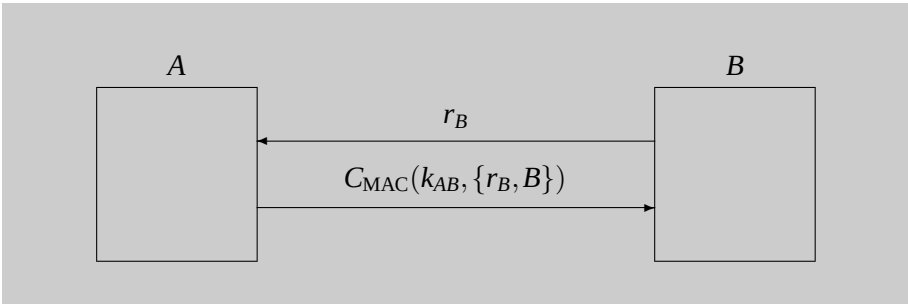
Los protocolos anteriores hacen uso de un algoritmo de cifrado simétrico, pero también es posible utilizar funciones unidireccionales con clave secreta como, por ejemplo, los algoritmos de MAC. Para la verificación, en lugar de descifrar es preciso comprobar que el MAC sea correcto y, por tanto, los valores necesarios para calcularlo se han de enviar en claro.

A continuación se muestran las variantes de los protocolos anteriores cuando se utiliza un algoritmo de MAC en lugar de un algoritmo de cifrado.

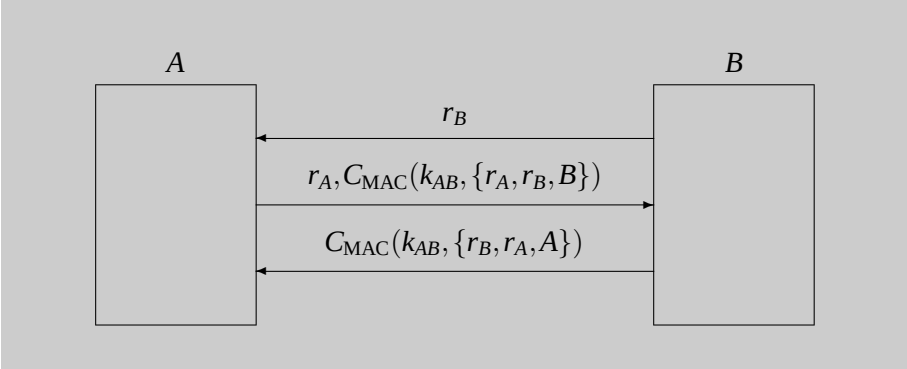
- Autenticación con marca de tiempo:



- Autenticación con números aleatorios:



- Autenticación mutua con números aleatorios:



En este caso, se debe añadir el identificador del destinatario A en el cálculo del último mensaje para evitar ataques de repetición en sentido contrario, ya que ahora los atacantes pueden ver el valor r_A.

Protocolos de reto-respuesta con clave pública

Hay dos formas de utilizar las técnicas de clave pública en los protocolos de reto-respuesta:

- El reto se manda cifrado con clave pública y la respuesta es el reto descifrado con la correspondiente clave privada.
- El reto se envía en claro y la respuesta es la firma del reto.

Dado que el usuario A tiene que utilizar si clave privada sobre un mensaje que le han enviado, ha de tomar ciertas precauciones para evitar que la otra parte haga un uso ilegítimo de la respuesta. Esto lo veremos en cada uno de los dos tipos de protocolos de reto-respuesta con clave pública.

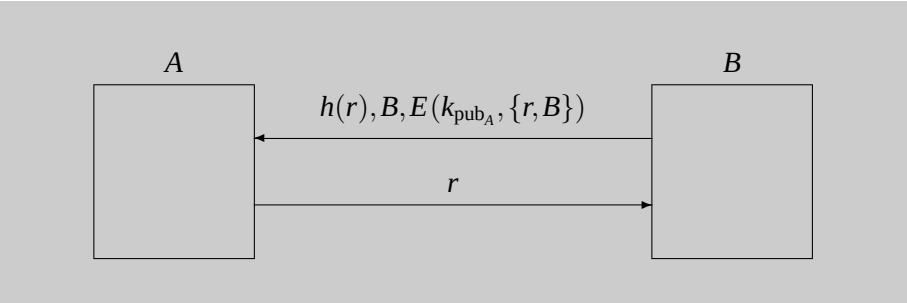
Claves para aplicaciones distintas

Para evitar los abusos que se pueden cometer contra una clave pública, es muy recomendable utilizar claves públicas distintas para cada aplicación, por ejemplo una clave para recibir mensajes confidenciales y otra para la autenticación.

1) Descifrado del reto

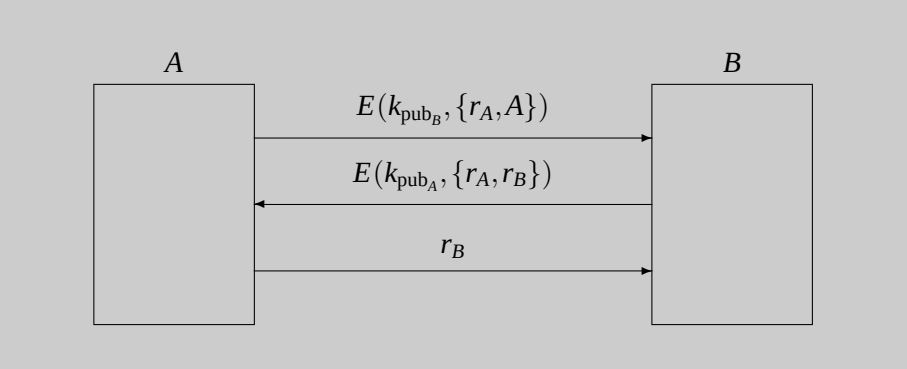
Si A recibe un reto cifrado con su clave pública, antes de enviar la respuesta debe asegurarse que quien ha enviado el reto conoce su valor. Sino, un atacante podría enviarle a A un reto cifrado, haciendo creer que lo ha generado aleatoriamente, pero que en realidad lo ha extraído de otro mensaje confidencial que alguien había enviado a A cifrado con su clave pública. Si A responde a este reto, está dando al atacante el mensaje confidencial descifrado.

Una posibilidad es que A reciba, juntamente con el reto r cifrado, un resumen o *hash* de este reto.



Cuando A recibe el mensaje, utiliza su clave privada para obtener r y B, y comprueba que estos valores sean correctos. Si el *hash* del reto r coincide con el valor recibido $h(r)$, esto significa que B conoce este reto. Por tanto, se puede enviar a B como respuesta el valor descifrado. Si no, no se le envía nada, porque puede tratarse de un defraudador que quiera obtener ilegítimamente este valor descifrado.

Otra posibilidad es que A pueda escoger una parte del reto. Este es el principio del llamado *protocolo modificado de clave pública* de Needham-Schroeder, que además proporciona autenticación mutua.



Para A, el reto está formado por r_A y r_B: el hecho de que la primera parte la haya generado A evita que el reto haya sido elegido maliciosamente por B. Para B, el reto incluye el identificador de A, con lo cual tampoco puede ser un mensaje cifrado que un tercero había enviado a B y que A quiere descifrar ilegítimamente.

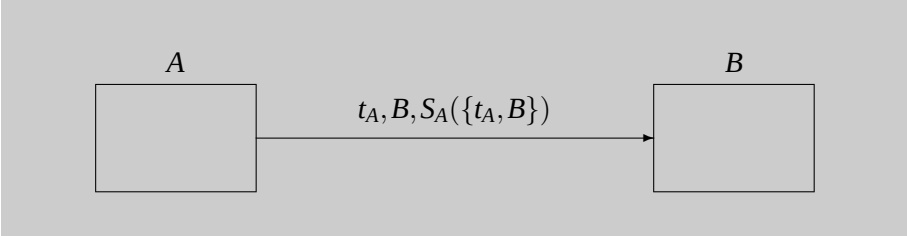
2) Firma del reto

La Recomendación X.509 no sólo especifica los formatos de certificados y listas de revocación que hemos visto en el subapartado 1.3, sino que también define unos protocolos de autenticación fuerte basados en firmas digitales que utilizan marcas de tiempo y números aleatorios. Aquí mostraremos otros protocolos, que son los equivalentes a los que hemos visto anteriormente basados en claves simétricas.

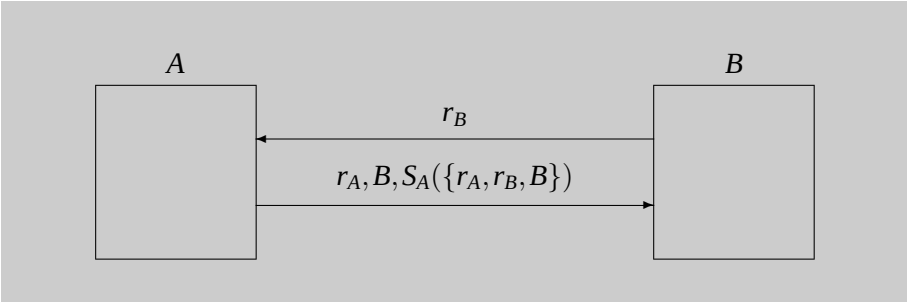
Como en el caso del descifrado con clave privada, A debe prestar atención con lo que firma: nunca debe firmar directamente un reto que le hayan enviado, sino que en la firma siempre tiene que intervenir como mínimo una parte del texto que haya escogido el propio A.

En la descripción de estos protocolos, $S_A(M)$ quiere decir la firma digital del mensaje M con la clave privada de A.

- Autenticación con marca de tiempo:

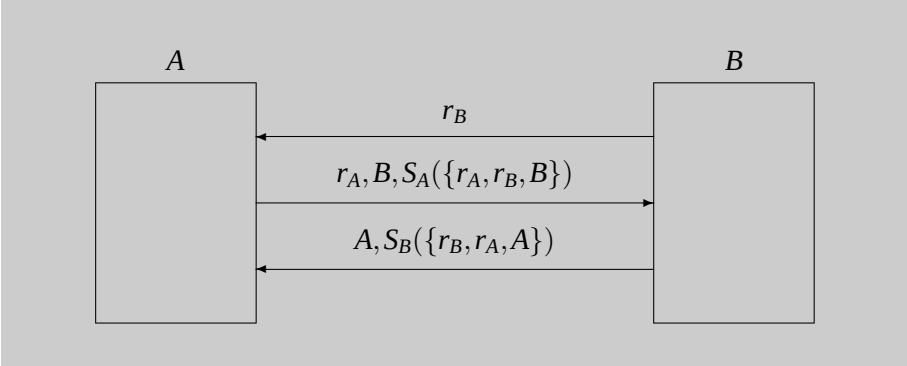


- Autenticación con números aleatorios:



Aquí la inclusión del valor r_A evita que B haya escogido r_B maliciosamente para hacer firmar a A un mensaje sin que se de cuenta.

- Autenticación mutua con números aleatorios:



3. Protección del nivel de red: IPsec

En los apartados anteriores hemos visto los mecanismos básicos de protección, que proporcionan servicios como la confidencialidad o la autenticación.

En el momento de aplicar estos mecanismos a las redes de computadores, existen diversas opciones en cuanto al nivel de las comunicaciones donde se introduzcan les funciones de seguridad.

- La protección a **nivel de red** garantiza que los datos que se envíen a los protocolos de nivel superior, como TCP o UDP, se transmitirán protegidos. El inconveniente es que puede ser necesario adaptar la infraestructura de la red y, en particular los encaminadores (*routers*), para que entiendan las extensiones que es preciso añadir al protocolo de red (IP) para proporcionar esta seguridad.
- La protección a **nivel de transporte**, por su lado, tiene la ventaja de que sólo se precisa adaptar las implementaciones de los protocolos (TCP, UDP, etc.) que haya en los nodos extremos de la comunicación, que normalmente están incorporadas al sistema operativo o en librerías especializadas. En este caso, pues, sólo sería necesario un cambio en el software.
- La protección a **nivel de aplicación** puede responder mejor a las necesidades de ciertos protocolos. Un ejemplo concreto es el del correo electrónico, en el que interesa proteger los datos de aplicación, es decir, los mensajes de correo, más que los paquetes a nivel de transporte o de red. Esto es así porque un mensaje es vulnerable a ataques de acceso ilegítimo o de falsificación no sólo cuando se está transmitiendo por la red sino también cuando está almacenado en el buzón del destinatario.

En esta sección veremos la arquitectura IPsec, diseñada para proteger el protocolo de red usado en Internet, es decir, el protocolo IP. En la sección siguiente veremos mecanismos para proteger las comunicaciones a nivel de transporte, y en el módulo de aplicaciones seguras veremos ejemplos de protección de los protocolos a nivel de aplicación.

3.1. La arquitectura IPsec

La **arquitectura IPsec** (RFC 2401) añade servicios de seguridad al protocolo IP (versión 4 y versión 6), que pueden ser usados por los protocolos de niveles superiores (TCP, UDP, ICMP, etc.).

IPsec se basa en el uso de una serie de protocolos seguros, de los cuales hay dos que proporcionan la mayor parte de los servicios:

- El **protocolo AH** (*Authentication Header*, RFC 2402) ofrece el servicio de autenticación de origen de los datagramas IP (incluyendo la cabecera y los datos de los datagramas).
- El **protocolo ESP** (*Encapsulating Security Payload*, RFC 2406) puede ofrecer el servicio de confidencialidad, el de autenticación de origen de los datos de los datagramas IP (sin incluir la cabecera), o los dos a la vez.

Opcionalmente, cada uno de estos dos protocolos también puede proporcionar otro servicio, el de protección contra repetición de datagramas.

Para la autenticación y la confidencialidad es necesario utilizar unas determinadas claves, correspondientes a los algoritmos criptográficos que se apliquen. Una posibilidad es configurar estas claves de forma manual en los nodos IPsec. Normalmente, pero, se utilizaran otros protocolos para la **distribución de claves**, como pueden ser:

- ISAKMP (*Internet Security Association and Key Management Protocol*, RFC 2408)
- IKE (*Internet Key Exchange*, RFC 2409)
- El protocolo de intercambio de claves OAKLEY (RFC 2412)

Los agentes que intervienen en la arquitectura IPSec son:

- Los nodos extremos de la comunicación: el origen y el destino final de los datagramas.
- Los nodos intermedios que suporten IPsec, llamados **pasarelas seguras**, como por ejemplo los encaminadores o cortafuegos con IPsec.

Pasarelas seguras

Un encaminador IPsec normalmente actúa como pasarela, pero puede pasar que en alguna comunicación actue como nodo extremo, como por ejemplo cuando se le envían comandos de configuración con el protocolo de gestión SNMP.

El tráfico IPsec también puede pasar por nodos intermedios que no soporten IPsec. Estos nodos son transparentes al protocolo porque para ellos los datagramas IPsec son como cualquier otro datagrama IP.

La relación que se establece entre dos nodos que se envían datagramas IPsec el uno al otro se llama **asociación de seguridad** (SA). Estos dos nodos pueden

ser o no los extremos de la comunicación, es decir, el origen de los datagramas o su destino final. Por tanto, podemos distinguir dos tipos de SA:

SA

SA es la sigla de *Security Association*.

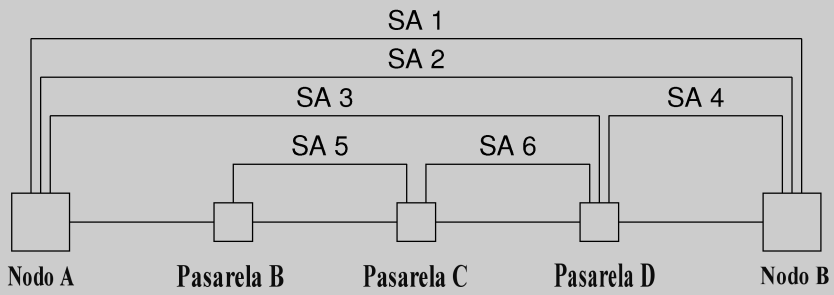
- Las SA extremo a extremo: se establecen entre el nodo que origina los datagramas y el nodo al cual van destinados.
- Las SA con una pasarela segura: al menos uno de los nodos es una pasarela segura (también pueden serlo ambos). Por tanto, los datagramas vienen de otro nodo y/o van hacia otro nodo.

Además, se considera que las SA son unidireccionales, es decir, si *A* envía datagramas IPsec a *B*, y *B* envía datagramas IPsec a *A*, tenemos dos SA, una en cada sentido.

por otro lado, cuando se establece una SA entre dos nodo, se utiliza uno de los dos protocolos básicos IPsec: o AH o ESP. Si se quiere utilizar ambos a la vez, se deberá establecer dos SA, una para cada protocolo.

Por tanto, puede pasar que en una comunicación entre dos nodos extremos intervengan diversas SA, cada una con sus nodos de inicio y de final y con su protocolo.

Ejemplo de combinación de asociaciones de seguridad



Cada nodo debe guardar información sobre sus SA, como por ejemplo los algoritmos criptográficos que utiliza cada una, las claves, etc. En la terminología IPsec, el lugar donde se guarda esta información se llama **base de datos de asociaciones de seguridad** o SAD. A cada SA le corresponden un número llamado **índice de parámetros de seguridad** o SPI. Todas las SA que un nodo tenga establecidas con otro nodo han de tener SPI diferentes. Por tanto, cada SA en que participa un nodo queda identificada por la dirección IP de destino y su SPI.

SAD

SAD es la sigla de *Security Association Database*.

SPI

SPI es la sigla de *Security Parameters Index*.

Para cada datagrama que llega a un nodo IPsec, se consulta una **base de datos**

de políticas de seguridad (SPD) donde se especifican criterios para determinar cual de las siguientes 3 acciones se debe realizar:

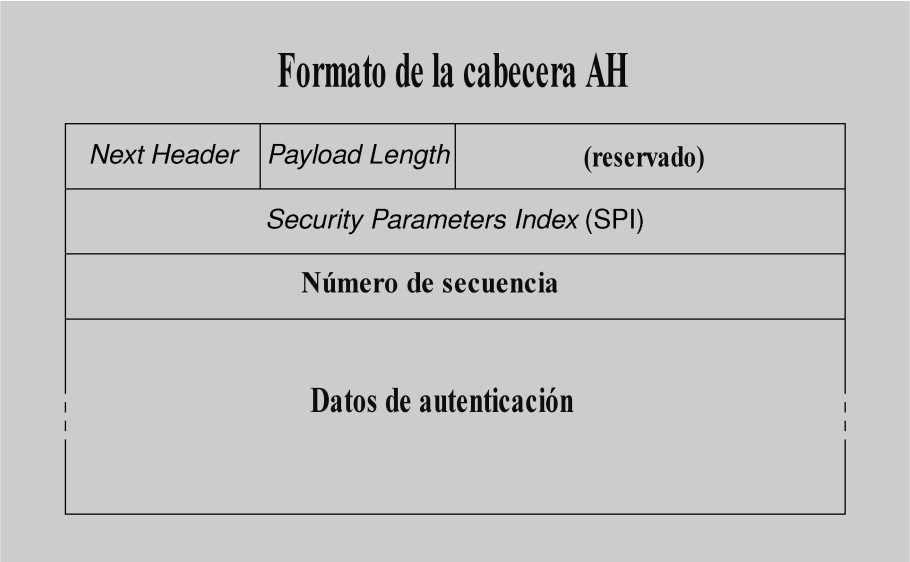
- Aplicar servicios de seguridad IPsec al datagrama, es decir, procesarlo según AH y/o ESP.
- Procesarlo como un datagrama IP normal, es decir, de forma transparente a IPsec.
- Descartar el datagrama.

SPD

SPD es la sigla de *Security Policy Database*.

3.2. El protocolo AH

El protocolo AH define una cabecera que contiene la información necesaria para a la autenticación de origen de un datagrama.



El campo *Next Header* sirve para indicar a que protocolo corresponden los datos que vienen a continuación de la cabecera AH. El campo *Payload Length* indica la longitud de la cabecera (esta información se necesita porque el último campo es de longitud variable, ya que depende del algoritmo de autentificación).

Campo *Next Header* en AH

Posibles valores del campo *Next Header* son, por ejemplo, 6 (TCP), 17 (UDP), 50 (si a continuación viene un cabecera ESP), etc.

El campo SPI sirve para identificar a que SA corresponde esta cabecera AH, y el número de secuencia que se utiliza si se quiere proporcionar el servicio de protección contra repetición de datagramas.

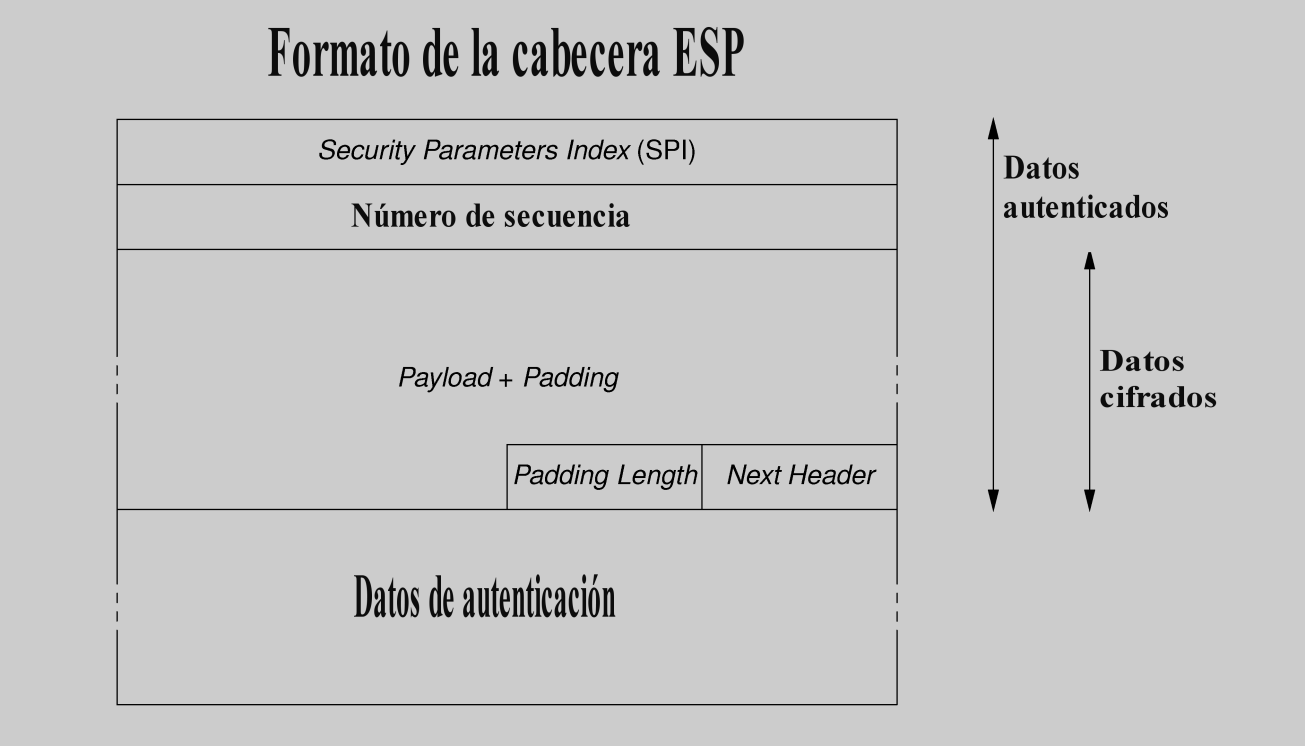
Finalmente, el último campo contiene un código de autentificación, por ejemplo un código MAC, calculado según el algoritmo que corresponda a esta SA. El código se obtiene a partir del datagrama entero, excepto los campos

que puedan variar de nodo a nodo (como los campos TTL y *Checksum* de la cabecera IP), o los que tienen un valor desconocido *a priori* (como el propio campo de datos de autentificación de la cabecera AH).

El nodo que reciba un datagrama con encabezamiento AH verificará el código de autentificación, y si es correcto dará el datagrama por bueno y lo procesará normalmente. Si a parte se utiliza el servicio de protección contra repeticiones, se necesita comprobar que el número de secuencia no sea repetido: si lo es, se descarta el datagrama.

3.3. El protocolo ESP

El protocolo ESP define otra cabecera, que de hecho incluye dentro todos los datos que vengan a continuación en el datagrama (lo que en inglés se llama *"payload"*).



Los campos SPI y número de secuencia son análogos a los de la cabecera AH.

A continuación vienen los datos del datagrama (*payload*), a los cuales puede ser necesario añadir bytes adicionales (*padding*) si se utiliza un algoritmo de cifrado en bloque, para conseguir que el número de bytes a cifrar sea múltiple de la longitud de bloque. El campo *Padding Length* indica exactamente el número de bytes que se han añadido (puede ser 0). El campo *Next Header* indica de que protocolo son los datos del datagrama.

Datos cifrados en ESP

Dependiendo del algoritmo de cifrado utilizado, puede ser necesario incluir antes de los datos cifrados parámetros como el vector de inicialización, etc.

Camp *Next Header* en ESP

Dependiendo del servicio o servicios que proporcione esta cabecera ESP, puede ser que los datos estén cifrados (incluyendo el *padding* y los campos *Padding Length* y *Next Header*), que se le añade un código de autenticación calculado a partir de la cabecera ESP (pero no de las cabeceras que pueda haber antes), o ambas cosas a la vez.

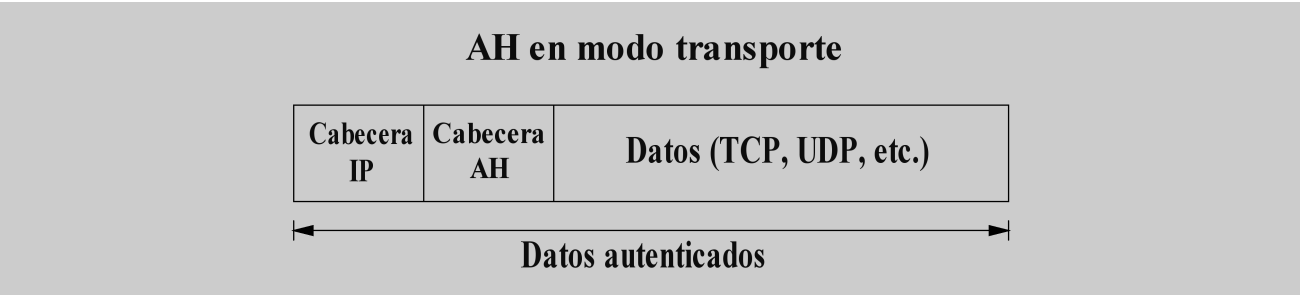
El nodo que reciba un datagrama con cabecera ESP deberá verificar el código de autenticación, o descifrar los datos, o ambas cosas (por este orden, porque si se aplican los dos servicios primero se cifra y después se autentican los datos cifrados). Si a más se utiliza el servicio de protección contra repeticiones, también se debe comprobar el número de secuencia. Este último servicio, pero, sólo se puede utilizar cuando la cabecera ESP está autenticada.

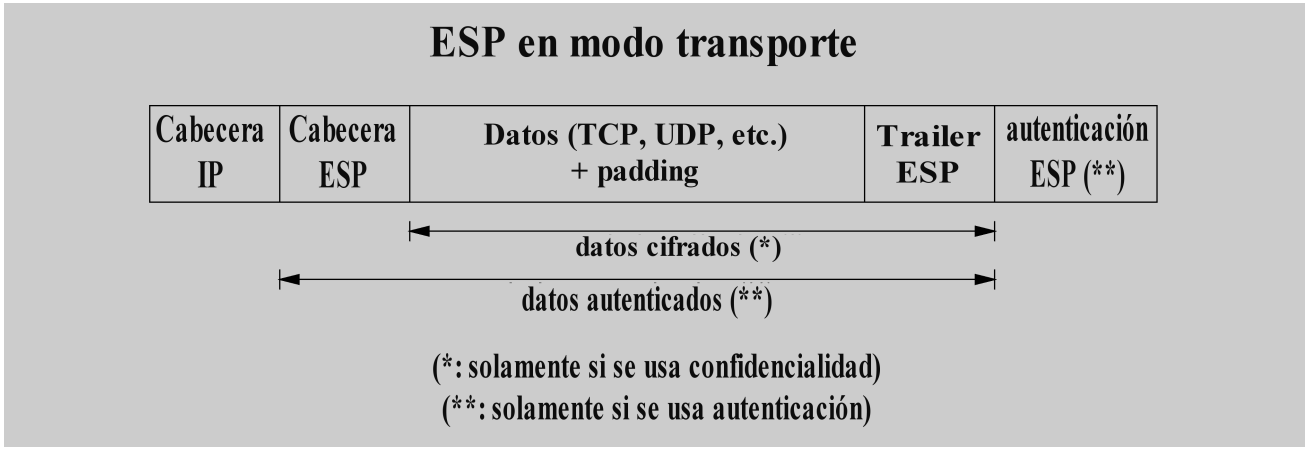
3.4. Modos de uso de los protocolos IPsec

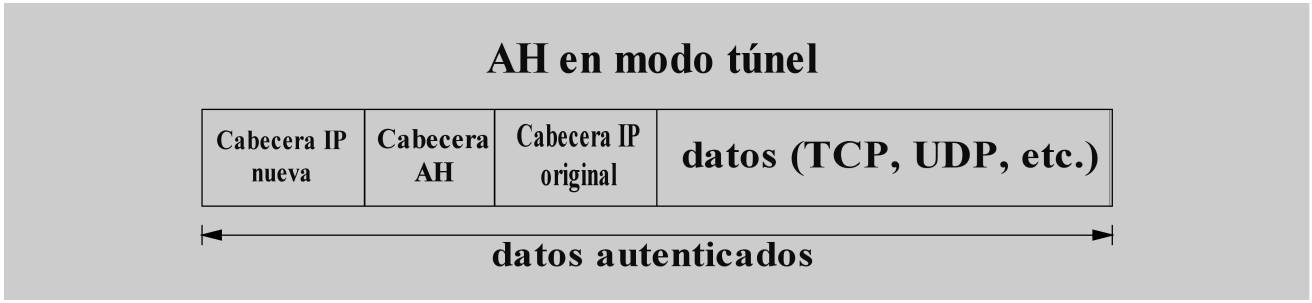
La arquitectura IPsec define dos modos de uso de los protocolos AH y ESP, dependiendo de como se incluyan las cabeceras correspondientes en un datagrama IP.

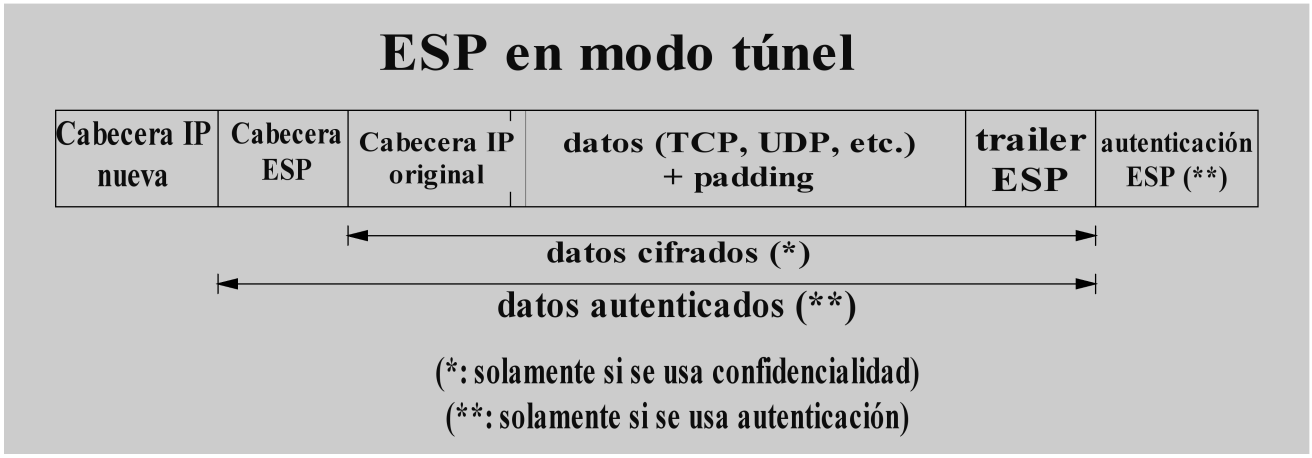
- En el **modo transporte**, la cabecera AH o ESP se incluye después de la cabecera IP convencional, como si fuera una cabecera de un protocolo de nivel superior, y a continuación van los datos del datagrama (por ejemplo, un segmento TCP con su cabecera correspondiente, etc.).
- En el **modo túnel**, el datagrama original se encapsula entero, con su cabecera y sus datos, dentro de otro datagrama. Este otro datagrama tendrá una cabecera IP en la cual las direcciones de origen y de destino serán las de los nodos inicio y final de la SA. Por tanto, se dice que entre estos dos nodos hay un "túnel" dentro del cual viajan intactos los datagramas originales. A continuación de la cabecera IP del datagrama "externo" hay la cabecera AH o ESP.

Las siguientes figuras muestran la disposición de las cabeceras IPsec en cada modo. En estas figuras, "*trailer* ESP" se refiere a los campos *Padding Length* y *Next Header* de la cabecera ESP.









El protocolo IP prevé que un datagrama se pueda fragmentar, y se puede dar el caso que los fragmentos de un mismo datagrama vayan por caminos diferentes hasta llegar a su destino final. Esto representaría un problema en una SA entre pasarelas seguras (o entre un nodo extremo y una pasarela segura) si se utilizara el modo transporte: por ejemplo, algunos fragmentos podrían quedar sin proteger, otros podrían resultar indescifrables porque no han pasado por la pasarela que los había de descifrar, etc. Para evitar estas situaciones, en IPsec sólo se permite el modo transporte en las SA extremo a extremo.

El modo túnel no tiene este problema porque, aunque la SA sea entre pasarelas, cada datagrama tiene como dirección de destino la del nodo que hay al final del túnel, y todos los fragmentos finalmente tienen que llegar a este nodo. Por tanto, el modo túnel se puede utilizar en cualquier SA, tanto si es extremo a extremo como si interviene una pasarela segura.

Como hemos visto antes, puede haber diversas SA en el camino entre el originador de los datagramas y el destino final. Esto quiere decir que las disposiciones de cabecera AH y ESP que muestran las figuras anteriores se pueden combinar entre ellas. Por ejemplo, puede haber un túnel dentro de otro túnel, o un túnel dentro de una SA en modo transporte, etc.

Otro caso que se puede dar es el de dos SA entre los mismos nodos de origen y de destino, una con el protocolo AH y la otra con el protocolo ESP. En este caso, el orden más lógico es aplicar primero ESP con servicio de confidencialidad y después AH, ya que de así la protección que ofrece AH, es decir, la autenticación, se extiende a todo el datagrama resultante.

4. Protección del nivel de transporte: SSL/TLS/WTLS

Tal y como hemos visto en el apartado anterior, el uso de un protocolo seguro a nivel de red puede requerir la adaptación de la infraestructura de comunicaciones, por ejemplo cambiar los encaminadores IP por otros que entiendan IPsec.

Un método alternativo que no necesita modificaciones en los equipos de interconexión es introducir la seguridad en los protocolos de transporte. La solución más usada actualmente es el uso del protocolo SSL o de otros basados en SSL.

Este grupo de protocolos comprende:

- El protocolo de transporte *Secure Sockets Layer* (SSL), desarrollado por Netscape Communications a principios de los años 90. La primera versión de este protocolo ampliamente difundida y implementada fue la 2.0. Poco después Netscape publicó la versión 3.0, con muchos cambios respecto a la anterior, que hoy ya casi no e utiliza.
- La especificación *Transport Layer Security* (TLS), elaborada por la IETF (*Internet Engineering Task Force*). La versión 1.0 del protocolo TLS está publicada en el documento RFC 2246. Es prácticamente equivalente a SSL 3.0 con algunas pequeñas diferencias, por lo que en ciertos contextos se considera el TLS 1.0 como si fuera el protocolo "SSL 3.1".
- El protocolo *Wireless Transport Layer Security* (WTLS), perteneciente a la familia de protocolos WAP (*Wireless Application Protocol*) para el acceso a la red des de dispositivos móviles. La mayoría de los protocolos WAP son adaptaciones de los ya existentes a las características de las comunicaciones inalámbricas, y en particular el WTLS está basado en el TLS 1.0. Las diferencias se centran principalmente en aspectos relativos a el uso eficiente del ancho de banda y de la capacidad de cálculo de los dispositivos, que puede ser limitada.

En este apartado hablaremos de les características comunes a SSL 3.0 y TLS 1.0, con algún detalle particular a las diferencias entre ellos. La mayoría de referencias a los "protocolos SSL/TLS" se deben entender aplicables también a WTLS.

4.1. Características del protocolo SSL/TLS

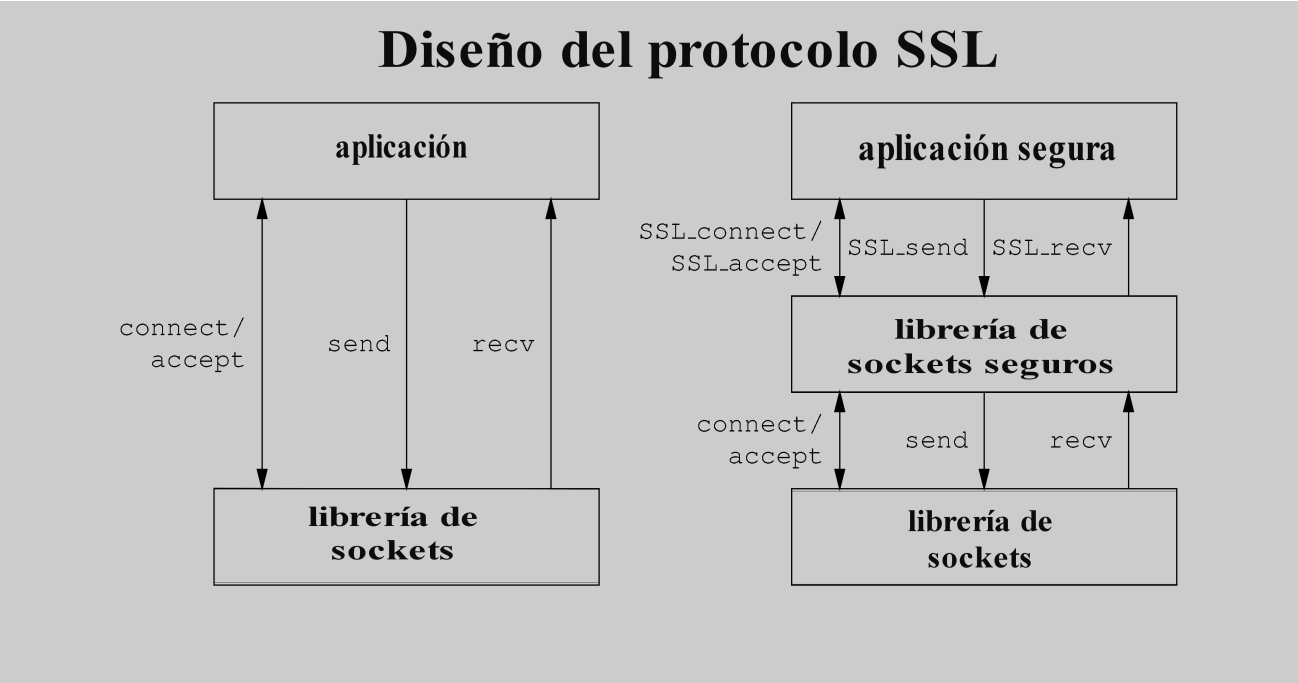
El objetivo inicial del diseño del protocolo SSL fue proteger las conexiones entre clientes y servidores web con el protocolo HTTP. Esta protección debía permitir al cliente asegurarse que se había conectado al servidor auténtico, y enviarle datos confidenciales, como por ejemplo un número de tarjeta de crédito, con la confianza que nadie más que el servidor sería capaz de ver estos datos.

Las funciones de seguridad, pero, no se implementaron directamente en el protocolo de aplicación HTTP, si no que se optó por introducirlas a nivel de transporte. De este modo podría haber muchas más aplicaciones que hicieran uso de esta funcionalidad.

Con este fin se desarrolló una interfaz de acceso a los servicios del nivel de transporte basada en la interfaz estándar de los *sockets*. En esta nueva interfaz, funciones como `connect`, `accept`, `send` o `recv` fueron sustituidas por otras equivalentes pero que utilizaban un protocolo de transporte seguro: `SSL_connect`, `SSL_accept`, `SSL_send`, `SSL_recv`, etc. El diseño se realizó de tal modo que cualquier aplicación que utilizara TCP a través de las llamadas de los *sockets* podía hacer uso del protocolo SSL solamente cambiando estas llamadas. De aquí proviene el nombre del protocolo.

Datagramas en WTLS

Una característica distintiva del WTLS es que no solamente permite proteger conexiones TCP, como hacen SSL y TLS, si no que también define un mecanismo de protección para las comunicaciones en modo datagrama, usadas en diversas aplicaciones móviles.



Los servicios de seguridad que proporcionan los protocolos SSL/TLS son:

Confidencialidad. El flujo normal de información en una conexión SSL/TLS

consiste en intercambiar paquetes con datos cifrados mediante claves simétricas (por motivos de eficiencia y rapidez). Al inicio de cada sesión, cliente y servidor se ponen de acuerdo en que claves utilizarán para cifrar los datos. Siempre se utilizan dos claves distintas: una para los paquetes enviados del cliente al servidor, y la otra para los paquetes enviados en sentido contrario.

Para evitar que un intruso que esté escuchando el diálogo inicial pueda saber cuales son las claves acordadas, se sigue un mecanismo seguro de intercambio de claves, basado en criptografía de clave pública. El algoritmo concreto para este intercambio también se negocia durante el establecimiento de la conexión.

Autenticación de entidad. Con un protocolo de reto-respuesta basado en firmas digitales el cliente pude confirmar la identidad del servidor al cual se ha conectado. Para validar las firmas el cliente necesita conocer la clave pública del servidor, y esto normalmente se realiza a través de certificados digitales.

SSL/TLS también prevé la autenticación del cliente frente al servidor. Esta posibilidad, pero, no se usa tan a menudo porque muchas veces, en lugar de autenticar automáticamente el cliente a nivel de transporte, las mismas aplicaciones utilizan su propio método de autenticación.

Autenticación de cliente

Un ejemplo de autenticación de cliente a nivel de aplicación son las contraseñas que pueden introducir los usuarios en formularios HTML. Si la aplicación utiliza este método, al servidor ya no le hace falta autenticar al cliente a nivel de transporte.

Autenticación de mensaje. Cada paquete enviado en una conexión SSL/TLS, a más de ir cifrado, puede incorporar un código MAC para que el destinatario compruebe que nadie ha modificado el paquete. Las claves secretas par el cálculo de los códigos MAC (una para cada sentido) también se acuerdan de forma segura en el diálogo inicial.

A más, los protocolos SSL/TLS están diseñados con estos criterios adicionales:

Eficiencia. Dos de las características de SSL/TLS, la definición de sesiones y la compresión de los datos, permiten mejorar la eficiencia de la comunicación.

- Si el cliente pide dos o más conexiones simultáneas o muy seguidas, en lugar de repetir la autenticación y el intercambio de claves (operaciones computacionalmente costosas porque intervienen algoritmos de clave pública), hay la opción de reutilizar los parámetros previamente acordados. Si se hace uso de esta opción, se considera que la nueva conexión pertenece a la misma **sesión** que la anterior. En el establecimiento de cada conexión se especifica un **identificador de sesión**, que permite saber si la conexión empieza una sesión nueva o es continuación de otra.
- SSL/TLS prevé la negociación de algoritmos de **compresión** para los datos intercambiados, para compensar el tráfico adicional que introduce la seguridad. Pero ni SSL 3.0 ni TLS 1.0 especifican ningún algoritmo concreto de compresión.

Conexiones consecutivas o simultáneas

Una situación típica en que se utiliza SSL/TLS es la de un navegador web que accede a una página HTML que contiene imágenes: con HTTP "no persistente" (el único modo definido en HTTP 1.0), esto requiere una primera conexión para la página y a continuación tantas conexiones como imágenes haya. Si las conexiones pertenecen a la misma sesión SSL/TLS, sólo hace falta realizar la negociación una vez.

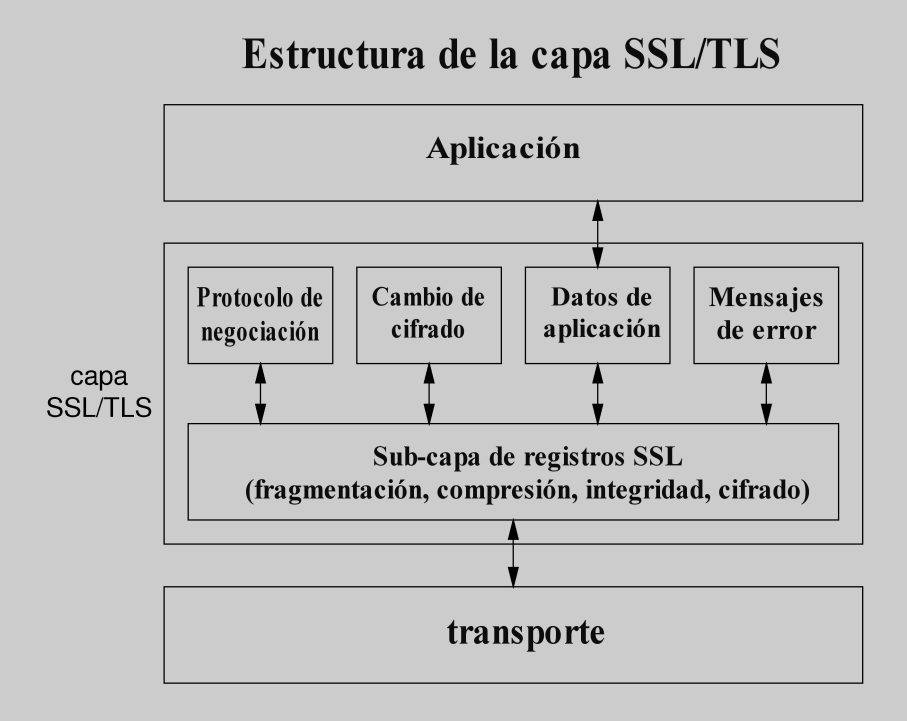
Extensibilidad. Al inicio de cada sesión, cliente y servidor negocian los algoritmos que utilizarán para el intercambio de claves, la autenticación y el cifrado (a más del algoritmo de compresión). Las especificaciones de los protocolos incluyen unas combinaciones predefinidas de algoritmos criptográficos,

pero dejan abierta la posibilidad de añadir nuevos algoritmos si se descubren otros que sean más eficientes o más seguros.

4.2. El transporte seguro SSL/TLS

La capa de transporte seguro que proporciona SSL/TLS se puede considerar dividida en dos subcapas.

- La subcapa superior se encarga básicamente de negociar los parámetros de seguridad y de transferir los datos de la aplicación. Tanto los datos de negociación como los de aplicación se intercambian en **mensajes**.
- En la subcapa inferior, estos mensajes son estructurados en **registros** a los cuales se les aplica, según corresponda, la compresión, la autenticación y el cifrado.



El **protocolo de registros SSL/TLS** es el que permite que los datos protegidos sean convenientemente codificados por el emisor y interpretados por el receptor. Los parámetros necesarios para la protección, como pueden ser los algoritmos y las claves, se establecen de forma segura al inicio de la conexión mediante el **protocolo de negociación SSL/TLS**. A continuación veremos las características de cada uno de estos dos protocolos.

4.2.1. El protocolo de registros SSL/TLS

La información que se intercambian cliente y servidor en una conexión SSL/TLS se empaqueta en registros, que tienen este formato:

El significado de cada campo es el siguiente:

- El primer campo indica cual es el tipo de contenido de los datos, que puede ser:
  - un mensaje del protocolo de negociación,
  - una notificación de cambio de cifrado,
  - un mensaje de error, o
  - datos de aplicación.
- El segundo campo son dos bytes que indican la versión del protocolo: si son iguales a 3 y 0 el protocolo es SSL 3.0, y si son iguales a 3 y 1 el protocolo es TLS 1.0.
- El tercer campo indica la longitud del resto del registro. Por tanto, es igual a la suma de L_d y L_{MAC} y, si los datos están cifrados con un algoritmo en bloque, $L_p + 1$.
- El cuarto campo son los datos, comprimidos si se ha acordado algún algoritmo de compresión.
- El quinto campo es el código de autenticación (MAC). En el cálculo de este MAC intervienen la clave MAC, un número de secuencia implícito de 64 bits (que se incrementa en cada registro pero no se incluye en ningún campo) y, naturalmente, el contenido del registro.

  La longitud de este campo depende del algoritmo de MAC que se haya acordado utilizar. Puede ser igual a 0 si se utiliza el algoritmo nulo, que es el que se utiliza al inicio de la negociación mientras no se ha acordado ningún otro.

Datos de un registro SSL/TLS

Normalmente los datos de un registro corresponden a un mensaje de la subcapa superior, pero también es posible juntar en un mismo registro dos o más mensajes, siempre que todos pertenecen al tipo indicado por el primer campo. También puede pasar que un mensaje se fragmente en diversos registros, si su longitud es superior a un cierto máximo (16384 bytes antes de comprimir).

- Si se ha acordado utilizar un algoritmo en bloque para cifrar los datos, es preciso añadir bytes adicionales (*padding*) a cada registro para tener un número total que sea múltiple de la longitud del bloque.

  La técnica que se usa para saber cuantos bytes adicionales hay es poner al menos uno, y el valor del último byte siempre indica cuantos otros bytes de *padding* hay antes (este valor puede ser 0 si sólo faltaba un byte para tener un bloque entero).

***Padding* en SSL y TLS**

Otra diferencia entre SSL y TLS está en los bytes de *padding*. En SSL debe haber el mínimo necesario, y su valor (excepto el último byte) es irrelevante. En TLS todos los bytes de *padding* deben tener el mismo valor que el último.

El protocolo de registros SSL/TLS se encarga de formar cada registro con sus campos correspondientes, calcular el MAC, y cifrar los datos, el MAC y el *padding* con los algoritmos y las claves que pertocan.

En la fase de negociación, mientras no se hayan acordado los algoritmos, los registros no se cifran ni se autentican, es decir, se aplican algoritmos nulos. Como veremos después, pero, todo el proceso de negociación queda autenticado *a posteriori*.

4.2.2. El protocolo de negociación SSL/TLS

El protocolo de negociación SSL/TLS, también llamado **protocolo de encajada de manos** (*"Handshake Protocol"*), tiene por finalidad autenticar el cliente y/o el servidor, y acordar los algoritmos y claves que se utilizaran de forma segura, es decir, garantizando la confidencialidad y la integridad de la negociación.

Como todos los mensajes SSL/TLS, los mensajes del protocolo de negociación se incluyen dentro del campo de datos de los registros SSL/TLS para ser transmitidos al destinatario. La estructura de un mensaje de negociación es la siguiente:

Formato de los mensajes de negociación SSL/TLS

1	3	L_m
Tipo de mensaje	longitud (L_m)	**Contenido del mensaje**

El contenido del mensaje tendrá unos determinados campos dependiendo del tipo de mensaje de negociación del que se trate. En total hay 10 tipos distintos,

que veremos a continuación en el orden en que se tienen que enviar.

1) Petición de saludo (*Hello Request*)

Cuando se establece una conexión, el servidor normalmente espera que el cliente inicie la negociación. Alternativamente, puede optar por enviar un mensaje *Hello Request* para indicar al cliente que está preparado para empezar. Si durante la sesión el servidor quiere iniciar una renegociación, también lo puede indicar al cliente enviando-le un mensaje de este tipo.

2) Saludo de cliente (*Client Hello*)

El cliente envía un mensaje *Client Hello* al inicio de la conexión o como respuesta a un *Hello Request*. Este mensaje contiene la siguiente información:

- La versión del protocolo que el cliente quiere utilizar.
- Una cadena de 32 bytes aleatorios.
- Opcionalmente, el identificador de una sesión anterior, si el cliente desea volver a utilizar los parámetros que se han acordado.
- La lista de las combinaciones de algoritmos criptográficos que el cliente ofrece utilizar, por orden de preferencia. Cada combinación incluye el algoritmo de cifrado, el algoritmo de MAC y el método de intercambio de claves.

Bytes aleatorios

De los 32 bytes aleatorios que se envían en los mensajes de saludo, los 4 primeros deben ser una marca de tiempo, con precisión de segundos.

Algoritmos criptográficos previstos en SSL/TLS

SSL/TLS contempla los algoritmos criptográficos siguientes:

– Cifrado: RC4, DES, Triple DES, RC2, IDEA y FORTEZZA (este último sólo en SSL 3.0).
– MAC: MD5 y SHA-1.
– Intercambio de claves: RSA, Diffie-Hellman y FORTEZZA KEA (este último sólo en SSL 3.0).

Si solamente interesa autenticar la conexión, sin confidencialidad, también se puede usar el algoritmo de cifrado nulo.

- La lista de los algoritmos de compresión ofrecidos, por orden de preferencia (como mínimo debe haber uno, aunque sea el algoritmo nulo).

Algoritmos de compresión

El único algoritmo de compresión previsto en SSL/TLS es el algoritmo nulo, es decir, sin compresión.

3) Saludo de servidor (*Server Hello*)

Como respuesta, el servidor envía un mensaje *Server Hello*, que contiene esta información:

- La versión del protocolo que se usará en la conexión. La versión será igual a la que envió el cliente, o inferior si esta no es soportada por el servidor.
- Otra cadena de 32 bytes aleatorios.

- El identificador de la sesión actual. Si el cliente envió uno y el servidor quiere reprender la sesión correspondiente, debe responder con el mismo identificador. Si el servidor no quiere reemprender la sesión (o no puede porque ya no guarda la información necesaria), el identificador enviado será diferente. Opcionalmente, el servidor puede no enviar ningún identificador para indicar que la sesión actual nunca no podrá ser reemprendida.
- La combinación de algoritmos criptográficos escogida por el servidor de entre la lista de las enviadas por el cliente. Si se reemprende una sesión anterior, esta combinación debe ser la misma que se utilizó entonces.
- El algoritmo de compresión escogido por el servidor, o el que se utilizó en la sesión que se reemprende.

Si se ha decidido continuar una sesión anterior, cliente y servidor ya pueden empezar a utilizar los algoritmos y claves previamente acordados y se saltan los mensajes que vienen a continuación pasando directamente a los de finalización de la negociación (mensajes *Finished*).

4) Certificado de servidor (*Certificate*) o **intercambio de claves de servidor** (*Server Key Exchange*)

Si el servidor puede autenticarse frente al cliente, que es el caso más habitual, envía el mensaje *Certificate*. Este mensaje normalmente contendrá el certificado X.509 del servidor, o una cadena de certificados.

Si el servidor no tiene certificado, o se ha acordado un método de intercambio de claves que no precisa de él, debe mandar un mensaje *Server Key Exchange*, que contiene los parámetros necesarios para el método a seguir.

5) Petición de certificado (*Certificate Request*)

En caso que se deba realizar también la autenticación del cliente, el servidor le envía un mensaje *Certificate Request*. Este mensaje contiene una lista de los posibles tipos de certificado que el servidor puede admitir, por orden de preferencia, y una lista de los DN de las autoridades de certificación que el servidor reconoce.

Tipo de certificados

En SSL/TLS están contemplados los certificados de clave pública RSA, DSA o FORTEZZA KEA (este último tipo solamente en SSL 3.0).

6) Fi de saludo de servidor (*Server Hello Done*)

Para terminar esta primera fase del diálogo, el servidor envía un mensaje *Server Hello Done*.

7) Certificado de cliente (*Certificate*)

Una vez el servidor ha mandado sus mensajes iniciales, el cliente ya sabe como continuar el protocolo de negociación. En primer lugar, si el servidor le ha pedido un certificado y el cliente tiene alguno de las características solicitadas, lo envía en un mensaje *Certificate*.

Cliente sin certificado

Si el cliente recibe una petición de certificado pero no tiene ninguno apropiado, en SSL 3.0 debe enviar un mensaje de aviso, pero en TLS 1.0 debe enviar un mensaje *Certificate* vacío. En cualquier caso, el servidor puede responder con un error fatal, o bien continuar sin autenticar el cliente.

8) Intercambio de claves de cliente (*Client Key Exchange*)

El cliente envía un mensaje *Client Key Exchange*, el contenido del cual

depende del método de intercambio de claves acordado. En caso de seguir el método RSA, en este mensaje hay una cadena de 48 bytes que se usará como **secreto pre-maestro**, cifrada con la clave pública del servidor.

Entonces, cliente y servidor calculan el **secreto maestro**, que es otra cadena de 48 bytes. Para realizar esta cálculo, se aplican funciones *hash* al secreto pre-maestro y a las cadenas aleatorias que se enviaron en los mensajes de saludo.

A partir del secreto maestro y las cadenas aleatorias, se obtienen:

- Las dos claves para el cifrado simétrico de los datos (una para cada sentido: de cliente a servidor y de servidor a cliente).
- Las dos claves MAC (también una para cada sentido).
- Los dos vectores de inicialización para el cifrado, si se utiliza un algoritmo en bloque.

Ataques de versión del protocolo

Un posible ataque contra la negociación es modificar los mensajes para que las dos partes acuerden utilizar el protocolo SSL 2.0, que es más vulnerable. Para evitar este ataque, a los dos primeros bytes del secreto pre-maestro debe haber el número de versión que se envió en el mensaje *Client Hello*.

9) Verificación de certificado (*Certificate Verify*)

Si el cliente ha mandado un certificado en respuesta a un mensaje *Certificate Request*, ya puede autenticarse demostrando que posee la clave privada correspondiente mediante un mensaje *Certificate Verify*. Este mensaje contiene una firma, generada con la clave privada del cliente, de una cadena de bytes obtenida a partir de la concatenación de todos los mensajes de negociación intercambiados hasta el momento, desde el *Client Hello* hasta el *Client Key Exchange*.

10) Finalización (*Finished*)

A partir de este punto ya se pueden utilizar los algoritmos criptográficos negociados. Cada parte manda a la otra una notificación de cambio de cifrado seguida de un mensaje *Finished*. La notificación de cambio de cifrado sirve para indicar que el siguiente mensaje será el primer enviado con los nuevos algoritmos y claves.

El mensaje *Finished* sigue inmediatamente la notificación de cambio de cifrado. Su contenido se obtiene aplicando funciones *hash* al secreto maestro y a la concatenación de todos los mensajes de negociación intercambiados, des de el *Client Hello* hasta el anterior a este (incluyendo el mensaje *Finished* de la otra parte, si ya lo ha enviado). Normalmente será el cliente el primer en enviar el mensaje *Finished*, pero en el caso de reemprender una sesión anterior, será el servidor quien lo enviará primero, justo después del *Server Hello*.

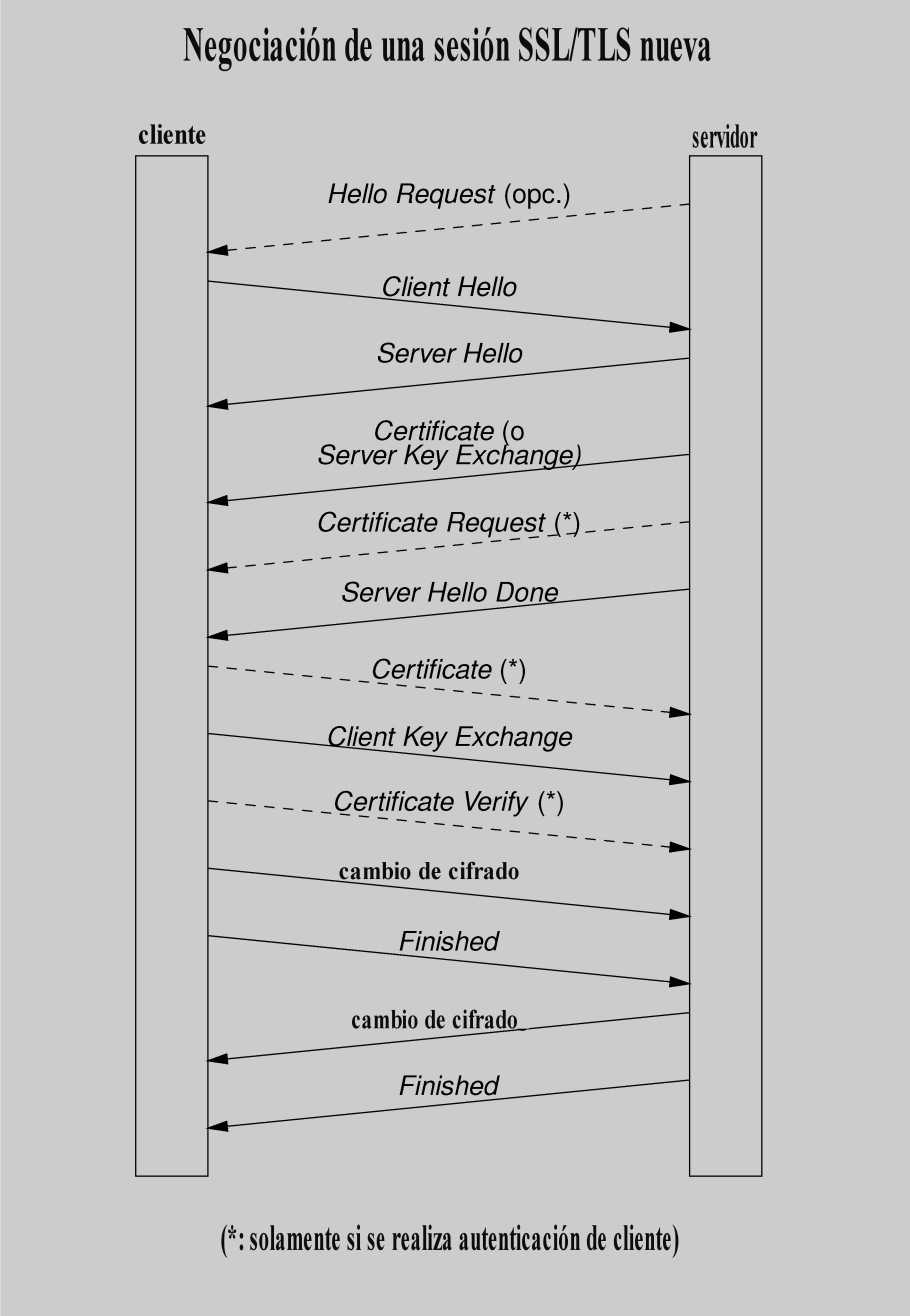
El contenido del mensaje *Finished* sirve para verificar que la negociación se ha llevado a cabo correctamente. Este mensaje también permite autenticar el servidor frente al cliente, ya que el primer necesita su clave privada para descifrar el mensaje *Client Key Exchange* y obtener las claves que se usarán en la comunicación.

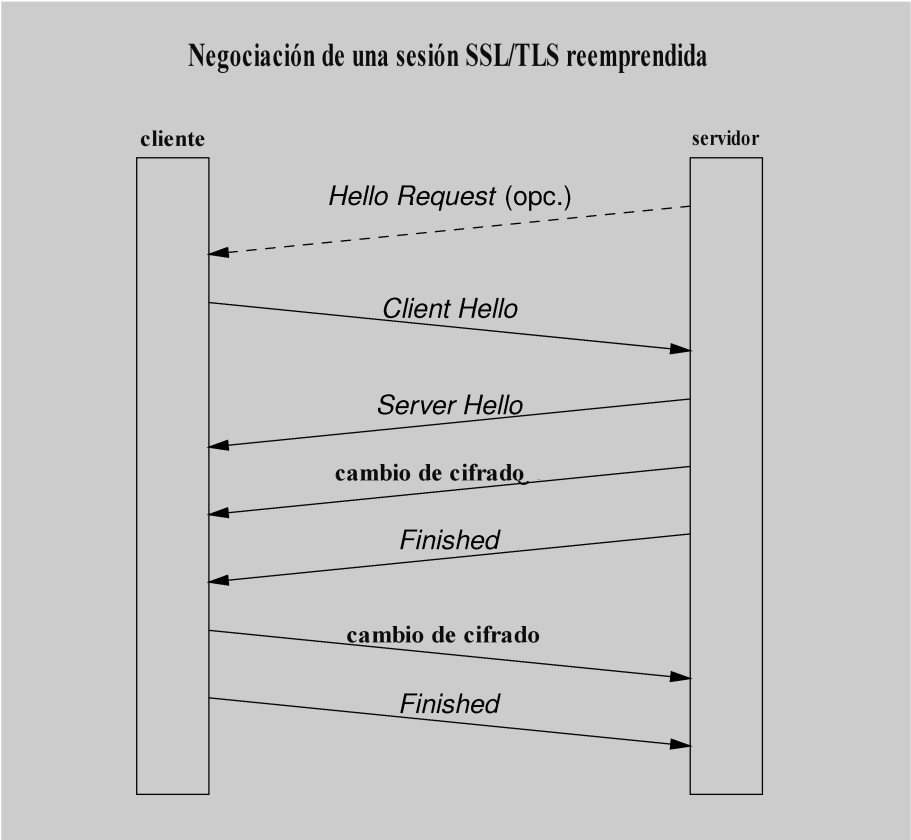
Verificación de autenticidad en SSL y TLS

Una de las principales diferencias entre SSL 3.0 y TLS 1.0 está en la técnica usada para obtener los códigos de verificación de los mensajes *Finished*, y también para calcular el secreto maestro y para obtener las claves a partir de este secreto (en SSL se utilizan funciones *hash* directamente, y en TLS se utilizan códigos HMAC).

Una vez enviado el mensaje *Finished*, se da por acabada la negociación, y cliente y servidor pueden empezar a enviar los datos de aplicación utilizando los algoritmos y claves acordados.

Los siguientes diagramas resumen los mensajes intercambiados durante la fase de negociación SSL/TLS:





A más de los mensajes de negociación, notificaciones de cambio de cifrado y datos de aplicación, también se pueden enviar mensajes de error. Estos mensajes contienen un código de nivel de gravedad, que puede ser "mensaje de aviso" o "error fatal", y un código de descripción del error. Un error fatal provoca el fin de la conexión y la invalidación del identificador de sesión correspondiente, es decir, la sesión no podrá ser reemprendida. Son ejemplos de errores fatales: MAC incorrecto, tipo de mensaje inesperado, error de negociación, etc. (TLS 1.0 prevé más códigos de error que SSL 3.0).

También se puede enviar un mensaje de aviso para indicar el fin normal de la conexión. Para evitar ataques de truncamiento, si una conexión acaba sin haber enviado este aviso se invalidará su identificador de sesión.

4.3. Ataques contra el protocolo SSL/TLS

Los protocolos SSL/TLS están diseñados para resistir los siguientes ataques:

Lectura de los paquetes enviados por el cliente y servidor. Cuando los datos se envían cifrados, un atacante que pueda leer los paquetes, por ejemplo utilizando técnicas de *sniffing*, se enfrenta al problema de romper el cifrado si quiere interpretar su contenido. Las claves que se utilizan para el cifrado se intercambian con métodos de clave pública, que el atacante tendría que romper si quiere saber cuales son los valores acordados.

Es preciso advertir, pero, que dependiendo de la aplicación que lo utilice, el protocolo SSL/TLS puede ser objeto de ataques con texto en claro conocido. Por ejemplo, cuando se utiliza juntamente con HTTP para acceder a servidores web con contenidos conocidos.

Si la comunicación es totalmente anónima, es decir sin autenticación de servidor ni cliente, sí que existe la posibilidad de capturar las claves secretas con un ataque conocido como "hombre a medio camino" (en inglés, "*man-in-the-middle attack*"). En este ataque el espía genera sus propias claves públicas y privadas, y cuando una parte envía a la otra información sobre su clave pública, tanto en un sentido como en el otro, el atacante la intercepta y la sustituye por la equivalente con la clave pública fraudulenta. Dado que el intercambio es anónimo, el receptor no tiene manera de saber si la clave pública que recibe es la del emisor auténtico o no.

En cambio, si se realiza la autenticación de servidor y/o cliente, es necesario enviar un certificado donde tiene que haber la clave pública del emisor firmada por una autoridad de certificación que el receptor reconozca, y por tanto no puede ser sustituida por otra.

Suplantación de servidor o cliente. Cuando se realiza la autenticación de servidor o cliente, el certificado digital debidamente firmado por la CA sirve para verificar la identidad de su propietario. Un atacante que quiera hacerse pasar por el servidor (o cliente) auténtico debería obtener su clave privada, o bien la de la autoridad de certificación que ha emitido el certificado para poder generar otro con una clave pública diferente y que parezca auténtico.

Alteración de los paquetes. Un atacante puede modificar los paquetes para que lleguen al destinatario con un contenido distinto del original (si están cifrados no podrá controlar cual será el contenido final descifrado, solamente sabrá que será distinto al original). Si pasa esto, el receptor detectará que el paquete ha sido alterado porque el código de autenticación (MAC) casi con total seguridad será incorrecto.

Si la alteración se realiza en los mensajes de negociación cuando aun no se aplica ningún código MAC, con la finalidad por ejemplo de forzar la adopción de algoritmos criptográficos más débiles y vulnerables, esta manipulación será detectada en la verificación de los mensajes *Finished*.

Repetición, eliminación o reordenación de paquetes. Si el atacante vuelve a enviar un paquete correcto que ya había sido enviado antes, o suprime algún paquete haciendo que no llegue a su destino, o los cambia de orden, el receptor lo detectará porque los códigos MAC no coincidirán con el valor esperado. Esto es así porque en el cálculo del MAC se utiliza un número de secuencia que se va incrementando en cada paquete.

Tampoco se pueden copiar los mensajes enviados en un sentido (de cliente a servidor o de servidor a cliente) al sentido contrario, porque en los dos flujos de la comunicación se utilizan claves de cifrado y de MAC diferentes.

Como consideración final, cabe destacar que la fortaleza de los protocolos seguros recae no solamente en su diseño si no en el de las implementaciones. Si una implementación solamente soporta algoritmos criptográficos débiles (con pocos bits de clave), o genera números pseudoaleatórios fácilmente predecibles, o guarda los valores secretos en almacenamiento (memoria o disco) accesible por atacantes, etc., no estará garantizando la seguridad del protocolo.

4.4. Aplicaciones que utilizan SSL/TLS

Como hemos visto al inicio de este apartado, los protocolos SSL/TLS fueron diseñados para permitir la protección de cualquier aplicación basada en un protocolo de transporte como TCP. Algunas aplicaciones que utilizan esta característica son:

- HTTPS (HTTP sobre SSL/TLS): el protocolo más utilizado actualmente para la navegación web segura.
- NNTPS (NNTP sobre SSL): para el acceso seguro al servicio de News.

Estas aplicaciones con SSL/TLS funcionan exactamente igual que las originales. Las únicas diferencias son el uso de la capa de transporte seguro que proporciona SSL/TLS y la asignación de números de puerto TCP propios: 443 para HTTPS y 563 para NNTPS.

En muchos otros casos, pero, es preferible aprovechar los mecanismos de extensión previstos en el propio protocolo de aplicación, si hay, para negociar el uso de SSL/TLS, a fin de evitar la utilización innecesaria de nuevos puertos TCP. Así lo hacen aplicaciones como:

- TELNET, usando la opción de autenticación (RFC 1416).
- FTP, usando las extensiones de seguridad (RFC 2228).
- SMTP, usando sus extensiones para SSL/TLS (RFC 2487).
- POP3 y IMAP, también usando comandos específicos para SSL/TLS (RFC 2595).

También hay definido un mecanismo para negociar el uso de SSL/TLS en HTTP (RFC 2817), como alternativa a HTTPS.

5. Redes privadas virtuales (VPN)

Los protocolos seguros que hemos visto hasta este punto permiten proteger las comunicaciones, por ejemplo, de una aplicación implementada como un proceso cliente que se ejecuta en un ordenador y un proceso servidor que se ejecuta en otro ordenador. Si hay otras aplicaciones que también necesiten una comunicación segura entre estos dos ordenadores, o entre ordenadores situados en las mismas redes locales, pueden hacer uso de otras instancias de los protocolos seguros: nuevas asociaciones de seguridad IPsec, nuevas conexiones SSL/TLS, etc.

Una posibilidad alternativa es establecer una **red privada virtual** o VPN entre estos ordenadores o las redes locales donde están situados. En este apartado veremos las características principales de las redes privadas virtuales.

VPN

VPN es la sigla de *Virtual Private Network*.

5.1. Definición y tipos de VPN

Una **red privada virtual** (VPN) es una configuración que combina el uso de dos tipos de tecnologías:

- Las tecnologías de seguridad que permiten la definición de una **red privada**, es decir, un medio de comunicación confidencial que no puede ser interceptado por usuarios ajenos a la red.
- Las tecnologías de encapsulamiento de protocolos que permiten que, en lugar de una conexión física dedicada para la red privada, se pueda utilizar una infraestructura de red pública, como Internet, para definir por encima de ella una **red virtual**.

Por tanto, una VPN es una red lógica o virtual creada sobre una infraestructura compartida, pero que proporciona los servicios de protección necesarios para una comunicación segura.

Dependiendo de la situación de los nodos que utilizan esta red, podemos considerar tres tipos de VPN:

VPN entre redes locales o intranets. Este es el caso habitual en que una empresa dispone de redes locales en diferentes sedes, geográficamente separadas, en cada una de las cuales hay una red privada o **intranet**, de acceso restringido

a sus empleados. Si interesa que desde una de sus sedes se pueda acceder a las intranets de otras sedes, se puede usar una VPN para interconectar estas redes privadas y formar una intranet única.

VPN de acceso remoto. Cuando un empleado de la empresa quiere acceder a la intranet des de un ordenador remoto, puede establecer una VPN de este tipo entre este ordenador y la intranet de la empresa. El ordenador remoto puede ser, por ejemplo, un PC que el empleado tiene en su casa, o un ordenador portátil des del cual se conecta a la red de la empresa cuando está de viaje.

VPN extranet. A veces, a una empresa le interesa compartir una parte de los recursos de su intranet con determinados usuarios externos, como por ejemplo proveedores o clientes de la empresa. La red que permite estos accesos externos a una intranet se llama **extranet**, y su protección se consigue mediante una VPN extranet.

5.2. Configuraciones y protocolos utilizados en VPN

A cada uno de los tipos de VPN que acabamos de ver le suele corresponder una configuración específica.

- En las VPN entre intranets, la situación más habitual es que en cada intranet hay una **pasarela VPN**, que conecte la red local con Internet. Esta pasarela se comunica con la de las otras intranets, aplicando el cifrado y las protecciones que sean necesarias a las comunicaciones de pasarela a pasarela a través de Internet. Cuando los paquetes llegan a la intranet de destino, la pasarela correspondiente los descifra y los reenvía por la red local hasta el ordenador que los tenga que recibir.

  De esta manera es utiliza la infraestructura pública de Internet, en lugar de establecer líneas privadas dedicadas, que supondrían un coste más elevado. También se aprovecha la fiabilidad y redundancia que proporciona Internet, ya que si una ruta no está disponible siempre se pueden encaminar los paquetes por otro camino, mientras que con una línea dedicada la redundancia supondría un coste aún más elevado.

- En las VPN de acceso remoto, a veces llamadas VPDN, un usuario se puede comunicar con una intranet a través de un proveedor de acceso a Internet, utilizando tecnología convencional como por ejemplo a través de un módem ADSL. El ordenador del usuario ha de disponer de software **cliente VPN** para comunicarse con la pasarela VPN de la intranet y llevar a cabo la autenticación necesaria, el cifrado, etc.

  De este modo también se aprovecha la infraestructura de los proveedores de Internet para el acceso a la intranet, sin necesidad de llamadas a un módem de la empresa, que pueden llegar a tener un coste considerable.

- El caso de las VPN extranet puede ser como el de las VPN entre intranets, en que la comunicación segura se establece entre pasarelas VPN, o bien

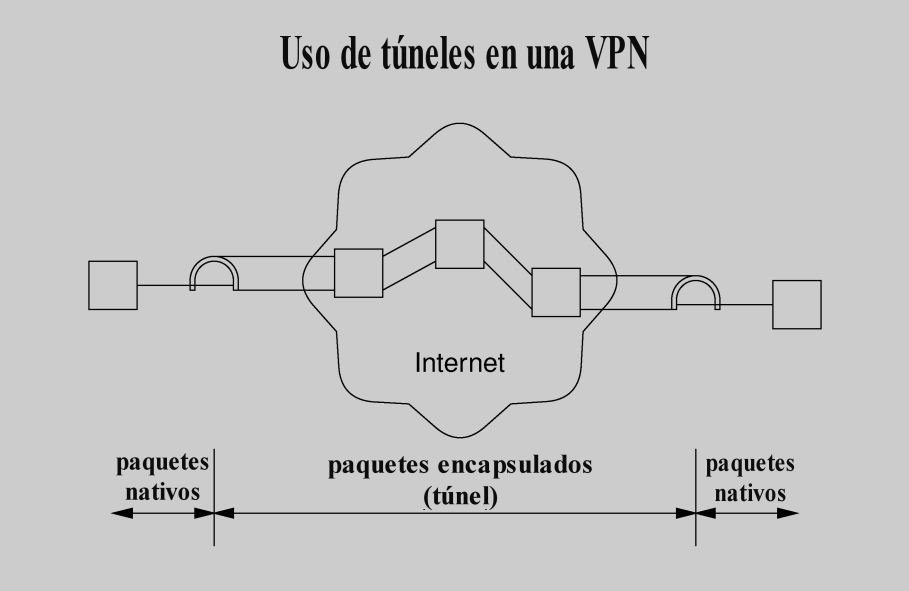
VPDN

VPDN es la sigla de *Virtual Private Dial Network*.

como el de las VPN de acceso remoto, en que un cliente VPN se comunica con la pasarela de la intranet. La diferencia, pero, es que en este caso normalmente el control de acceso es más restrictivo para permitir solamente el acceso a los recursos autorizados.

La definición de una red virtual lleva a cabo mediante el establecimiento de **túneles**, que permiten encapsular paquetes de la red virtual, con sus protocolos, dentro de paquetes de otra red, que normalmente es Internet, con su protocolo, es decir IP.

Para la comunicación entre las distintas intranets, o entre el ordenador que accede remotamente y la intranet, se pueden utilizar los protocolos que sean más convenientes. Los paquetes de estos protocolos, para poderlos hacer llegar a su destino a través de Internet, se pueden encapsular en datagramas IP, que dentro suyo contendrán los paquetes originales. Cuando lleguen a su destino, se desencapsulan estos datagramas para recuperar los paquetes con el formato "nativo" del protocolo correspondiente.



Hay protocolos que pueden ser utilizados para establecer los túneles, dependiendo del nivel de la comunicación al cual se quiera realizar la protección.

Túneles a nivel de red. El protocolo utilizado en la gran mayoría de configuraciones VPN es IPsec en modo túnel, generalmente con ESP par cifrar los datos, y opcionalmente con AH para autenticar los paquetes encapsulados. Las pasarelas VPN son, en este caso, pasarelas seguras IPsec.

Túneles a nivel de enlace. En el caso de las VPN de acceso remoto o VPDN, existe la posibilidad de encapsular tramas PPP, que son las que transmite normalmente un cliente VPN de este tipo, sobre datagramas IP. Hay diversas

opciones para encapsular PPP (que a su vez puede encapsular otros protocolos de red, como IPX, etc. o posiblemente IP) sobre IP:

- El protocolo PPTP (*Point-to-Point Tunneling Protocol*, RFC 2637) especifica una técnica para el encapsulamiento de tramas PPP pero no añade servicios de autenticación. Estos servicios se pueden realizar con los mismos protocolos que utiliza PPP, como PAP (*Password Authentication Protocol*) o CHAP (*Challenge Handshake Authentication Protocol*).
- El protocolo L2F (*Layer Two Forwarding*, RFC 2637) es parecido al PPTP pero también puede trabajar con SLIP a más de PPP. Para la autenticación puede utilizar protocolos auxiliares como RADIUS (*Remote Authentication Dial-In User Service*).
- El protocolo L2TP (*Layer Two Tunneling Protocol*, RFC 2661) combina las funcionalidades que ofrecen PPTP y L2F.

Túneles a nivel de transporte. El protocolo SSH (*Secure Shell*), como veremos en el módulo de aplicaciones seguras, ofrece la posibilidad de redirigir puertos TCP sobre un canal seguro, que podemos considerar como un túnel a nivel de transporte. Des de este punto de vista, también se podría considerar una conexión SSL/TLS como un túnel a nivel de transporte que proporciona confidencialidad y autenticación. Habitualmente, este último tipo de túnel no sirve para cualquier tipo de tráfico si no solamente para datos TCP, y por tanto no se considera parte integrante de una VPN.

Resumen

En este módulo hemos visto que las **técnicas criptográficas** permiten cifrar un texto mediante una **clave de cifrado**, y solamente quien conozca la **clave de descifrado** correspondiente será capaz de obtener el texto original.

Según la relación que haya entre las dos claves, los algoritmos criptográficos se clasifican en **algoritmos simétricos** si la clave de cifrado y la de descifrado son la misma, o **algoritmos de clave pública** si las claves son distintas. Los algoritmos simétricos, a su vez, se pueden clasificar en **algoritmos de cifrado en flujo**, si el cifrado consiste en añadir al texto datos pseudoaleatórios calculados a partir de la clave, o **algoritmos de cifrado en bloque**, si el cifrado se realiza sobre bloques de medida fija del texto original.

La particularidad de la criptografía de clave pública es que a partir de la **clave pública** es prácticamente imposible deducir la **clave privada**. Esto permite que cualquiera que conozca la clave pública de un usuario pueda usarla para cifrar datos confidenciales, con la seguridad que solamente quien tenga la clave privada correspondiente podrá descifrarlos, y sin necesidad de acordar ninguna clave secreta a través de un canal seguro. El uso de las claves al revés (la privada para cifrar y la pública para descifrar) es la base de las **firmas digitales**.

Dado que la criptografía de clave pública es computacionalmente más costosa que la simétrica, no se utiliza nunca directamente para obtener confidencialidad, si no siempre a través de una **clave de sesión** simétrica. Del mismo modo, la firma de un texto no se calcula directamente a partir del texto, si no aplicándole una **función *hash* segura**. La propiedad de este tipo de función es que es muy difícil encontrar un mensaje que de el mismo *hash* que otro.

Para garantizar que las claves públicas son auténticas, y pertenecen a quien se supone que han de pertenecer, se pueden utilizar **certificados digitales** o de clave pública, como por ejemplo los certificados X.509. Cuando una **autoridad de certificación** (CA) firma un certificado, está dando fe de la autenticidad entre el vínculo entre de la clave pública correspondiente y la identidad del usuario. Los certificados son un componente básico de la **infraestructura de clave pública** (PKI), como también lo son las **listas de revocación de certificados** (CRL).

Las firmas digitales proporcionan el servicio de **autenticación de mensaje**. Los llamados **códigos MAC** también proporcionan este servicio, pero utilizando claves secretas compartidas en lugar de claves públicas.

Otro servicio de autenticación es el de **autenticación de entidad**. Este mecanismo permite comprobar que la otra parte de la comunicación es quien dice ser, y no un impostor. Esto se puede conseguir con técnicas de **autenticación dé-**

bil basadas en contraseñas o, si es necesario, con técnicas de **autenticación fuerte** basadas en **protocolos de reto-respuesta**, que a diferencia de las anteriores son resistentes a muchos más ataques que las primeras.

Cuando se aplican los mecanismos de confidencialidad y autenticación a los protocolos de comunicación, es posible realizarlo a distintos niveles. Para proteger las comunicaciones a nivel de red se puede utilizar la **arquitectura IPsec**, que incluye los protocolos **AH**, para autenticar datagramas IP, y **ESP** para cifrar y/o autenticar los datos de los datagramas IP. También hay protocolos para el intercambio seguro de las claves necesarias.

Toda comunicación entre dos nodos de la red mediante protocolos IPsec pertenece a una **asociación de seguridad** (SA). Cada SA establece el protocolo a utilizar, y en cual de los dos modos posibles trabaja: el **modo transporte**, en el cual la cabecera AH o ESP actúa como si fuera la cabecera de los datos de nivel superior (transporte), o el **modo túnel**, en el cual se construye un nuevo datagrama IP que tiene como datos el datagrama original convenientemente protegido. El modo transporte solamente se puede usar en las SA que vayan de extremo a extremo, es decir, des del nodo que origina los datagramas hasta el que los recibe.

También hay la posibilidad de proteger las comunicaciones a nivel de transporte. En este caso se pueden usar los protocolos **SSL/TLS**, que utilizan el servicio de transporte TCP estándar. En estos protocolos existe una **negociación** inicial que permite autenticar el servidor y, si es el caso, el cliente, mediante sus certificados. El mismo protocolo de negociación sirve para establecer las claves de sesión que se usarán en la comunicación posterior, como les claves para el cifrado simétrico de los datos o las claves para los códigos MAC.

El usi típico de los protocolos SSL/TLS es para proteger de forma transparente un protocolo de aplicación como es el caso del HTTP. El protocolo **HTTPS** es simplemente la combinación de HTTP con el transporte seguro SSL/TLS.

Finalmente, las **redes privadas virtuales** (VPN) permiten utilizar la red pública Internet como si fuera una red privada dedicada, por ejemplo, entre diversas intranets de una misma organización. La técnica básica que utilizan las VPN son los **túneles**, en los cuales los paquetes protegidos se encapsulan des de datagramas IP que circulan de manera normal por la red Internet.

Actividades

3-1 Visitad páginas que contengan listas de algoritmos criptográficos y sus ataques conocidos (por ejemplo `www.ramkilde.com/bc.html` para el cifrado en bloque, `planeta.terra.com.br/informatica/paulobarreto/hflounge.html` para las funciones *hash*, etc.), y comprobad que algoritmos se pueden considerar actualmente seguros y cuales no.

3-2 Visitad la página `www.rsasecurity.com/rsalabs/challenges/`, el apartado *"RSA factoring challenge"*, y comprobad cuantos bits tiene el último número que se ha conseguido factorizar.

3-3 El proyecto EuroPKI pretende crear una infraestructura de clave pública a nivel europeo. Visitad su página web (`www.europki.org`) y examinad el certificado de su CA. Es una CA raíz? De cuantos bits es su clave pública? Examinad también la lista de certificados emitidos por esta CA y su CRL. Hay algún certificado revocado? Cuando se emitirá la próxima CRL? Si hay certificados revocados, podeis averiguar si volverán a aparecer en la próxima CRL?

Navegad también por las páginas web de otras CA de la jerarquía, como por ejemplo la de RedIRIS (`www.rediris.es/cert/iris-pca/`) o la de la Anella Científica del CESCA (`www.cesca.es/comunicacions/scd/`). Alguna de estas CA incluye el subcampo `pathLenConstraint` en su certificado?

3-4 Una implementación *"open source"* de los protocolos SSL/TLS bastante conocida es la del proyecto OpenSSL. Visitad su página web (`www.openssl.org`) y comprobad que algoritmos criptográficos soporta la última versión.

Ejercicios de autoevaluación

3-1 En el cifrado en bloque en modo ECB, si hay un error de transmisión en un bloque de texto cifrado, solamente se ve afectado el bloque correspondiente del texto descifrado. En modo CBC, el error se propaga: un error en la transmisión de C_y afecta al descifrado de M_y y M_{y+1}.

a) Afectaría el error a algún otro bloque más allá de M_{y+1}?

b) Suponed que hay un error en un bit de la versión original (antes de cifrar) de M_y. A cuantos bloques del texto cifrado se propagará este error? Cual será el efecto en la recepción?

3-2 Si se produce un error de transmisión en un bit de texto cifrado en modo CFB de 8 bits (longitud de cada unidad de texto cifrado C_y igual a 8 bits), hasta donde se propagara el error?

3-3 Considerad la siguiente propuesta de algoritmo para verificar si, después de un intercambio de clave secreta, las dos partes A y B han obtenido el mismo valor de la clave k. Primero, A crea una cadena de bits aleatorios r de la misma longitud que la clave, calcula $a = k \oplus r$, y envía este valor a B. Entonces B deduce r calculando $b = a \oplus k$ $(= k \oplus r \oplus k = r)$ y envía este valor b a A. Si A ve que el valor recibido b coincide con r, sabrá que B tiene el mismo valor de k, sin que ninguno de los dos haya mandado este valor en claro. Tiene algún problema esta propuesta?

3-4 El PKCS #1 especifica como formatear los datos que se quieren cifrar con una clave pública RSA antes de aplicarlos al algoritmo de cifrado. Según la versión 1.5, es preciso crear una secuencia de L bytes (donde L es la longitud en bytes del módulo n):

- El primer byte es igual a 0.
- El segundo byte indica el tipos de formato, y en este caso es igual a 2.
- Los siguientes bytes (como mínimo, 8) han de tener valores aleatorios distintos de 0.
- El siguiente byte es igual a 0.
- El resto de bytes (como máximo, $L - 11$) son el mensaje que se quiere cifrar.

a) Que seguridad proporcionan el segundo byte y los bytes aleatorios?

b) Que utilidad tiene el byte igual a 0 antes del mensaje?

3-5 Mientras que una clave de cifrado AES de 128 bits actualmente se considera bastante segura, una clave RSA de 512 bits se considera poco segura. Por qué?

3-6 Si un certificado X.509 ha dejado de ser válido antes de su caducidad, la CA que lo emitió lo puede incluir en su lista de certificados revocados (CRL). Sabiendo que en la CRL no hay el nombre (DN) del usuario a quien se le revoca el certificado, si no solamente el número de serie del certificado, hay algún modo que un atacante pueda manipular la CRL para hacer creer que el certificado que se está revocando es el de otro usuario?

3-7 La Recomendación X.509 describe diversos protocolos de autenticación, uno de los cuales se la llamada "autenticación en dos pasos", que se puede resumir de la siguiente forma:

$$A \rightarrow B: \quad S_A(\{r_A, t_A, B\})$$
$$A \leftarrow B: \quad S_B(\{r_B, t_B, A, r_A\})$$

La misma Recomendación X.509 define otro protocolo, llamado "autenticación en tres pasos", donde las marcas de tiempo son opcionales y por tanto no requiere sincronía entre A

y B. Este otro protocolo en su versión original era equivalente al siguiente intercambio:

$$\begin{aligned} A \rightarrow B: &\quad S_A(\{r_A, B\}) \\ A \leftarrow B: &\quad S_B(\{r_B, A, r_A\}) \\ A \rightarrow B: &\quad S_A(\{r_B\}) \end{aligned}$$

Pero este protocolo tiene un problema potencial que puede ser explotado por un impostor C que se quiera hacer pasar por A delante de B. El impostor, por un lado, puede repetir un mensaje inicial previamente capturado:

$$\begin{aligned} C \rightarrow B: &\quad S_A(\{r_A, B\}) \\ C \leftarrow B: &\quad S_B(\{r'_B, A, r_A\}) \end{aligned}$$

y por otro lado puede hacer que A inicie una autenticación con C:

$$\begin{aligned} A \rightarrow C: &\quad S_A(\{r'_A, C\}) \\ A \leftarrow C: &\quad S_C(\{r'_B, A, r'_A\}) \\ A \rightarrow C: &\quad S_A(\{r'_B\}) \end{aligned}$$

Entonces, C solamente tiene que mandar a B este último mensaje, que es el que necesita para que se convenza que está hablando con A, cuando en realidad está hablando con C:

$$C \rightarrow B: \quad S_A(\{r'_B\})$$

Cual sería una posible solución simple para evitar este ataque? (En la versión actual de la Recomendación X.509 este problema ya está solucionado.)

3-8 La firma digital de un mensaje se calcula cifrando con la clave privada del firmante el *hash* del mensaje. Porqué no se cifra directamente el mensaje a firmar?

3-9 Un posible ataque contra las firmas digitales consiste en hacer creer que el firmante ha calculado el *hash* con otro algoritme (p. ex. MD4 en lugar de MD5), y si este algoritmo es menos seguro que el original, puede ser que el atacante sea capaz de obtener un mensaje diferente que de el mismo *hash* con este otro algoritmo, de modo que la firma continuaría siendo válida. Com se puede evitar este ataque? (Uno de los PKCSs, el número #7, incluye una medida contra este ataque.)

3-10 La especificación IPsec indica que cuando dos asociaciones de seguridad (SA) en modo transporte se combinan para usar AH y ESP en una misma comunicación extremo a extremo, solamente uno de los dos posibles ordenes es apropiado: aplicar primero el protocolo ESP y después el protocolo AH. Por qué?

3-11 Una organización tiene instalada una red con direcciones IP privadas, y conectada a Internet mediante un *router* NAT (*Network Address Translator*), que traduce las direcciones privadas en una o más direcciones públicas (asignadas por un registrador oficial). Si se quiere utilizar IPsec para conectarse a servidores externos, sin realizar ningún cambio en la red, que combinaciones de protocolos (AH, ESP) y modos de operación (transporte, túnel) serán apropiadas, cuales no, y por qué?

3-12 Como puede el protocolo HTTPS (HTTP sobre SSL/TLS) contrarrestar las siguientes amenazas a la seguridad del servicio WWW?

a) Ataque criptográfico de fuerza bruta: búsqueda exhaustiva en el espacio de claves para descifrar los paquetes cifrados simétricamente.

b) Ataque de texto claro conocido, teniendo en cuenta que muchos mensajes HTTP, como por ejemplo las peticiones "`GET`", contienen texto predecible.

c) Ataque de repetición: reenviar mensajes de negociación SSL/TLS capturados previamente.

d) Ataque "de hombre a medio camino" ("*man-in-the-middle*"): interceptar una negociación SSL/TLS, reenviando paquetes modificados al cliente como si fueran del servidor auténtico, y viceversa.

e) Obtención de contraseñas: captura de *passwords* HTTP o de otras aplicaciones.

f) Falsificación IP ("*IP spoofing*"): generar paquetes con direcciones IP falsas.

g) "Secuestro" IP ("*IP hijacking*"): interrumpir una conexión autenticada entre dos nodos y continuarla haciéndose pasar por uno de ellos.

h) "Inundación" de paquetes SYN ("*SYN flooding*"): enviar paquetes TCP con el *flag* SYN para iniciar una conexión pero no responder al mensaje final para terminar el establecimiento, con la intención de saturar el servidor con conexiones TCP medio abiertas.

3-13 Un cliente C quiere establecer una conexión SSL/TLS con un servidor S que tiene una clave pública K. Durante la fase inicial de negociación, un atacante A intercepta los mensajes SSL/TLS y responde a C en nombre de S simulando que la clave pública del servidor es K' en lugar de K, y siguiendo todos los pasos de la negociación utilizando esta clave K'. Como puede C detectar este fraude?

3-14 En el protocolo SSL/TLS, le es posible al receptor reordenar registros SSL/TLS que le lleguen desordenados? Por qué?

Soluciones

3-1

a)No.

b)El error en M_y hará que todos los bloques a partir de C_y sean distintos de los que se tendrían que transmitir. En recepción, pero, todos los bloques a partir de M_{y+1} se recuperarán correctamente (el bloque M_y se recuperará con el mismo error de origen).

3-2 Se recuperarán incorrectamente los $N+1$ bytes de texto en claro a partir del error, donde N es $L/8$ (L = longitud de bloque del algoritmo de cifrado). Por ejemplo, en DES ($L = 64$), $N+1$ = 9 bytes.

3-3 Un atacante que tenga acceso a la comunicación verá los valores $a = k \oplus r$ y $b = r$. Entonces solamente se tiene que calcular $a \oplus b$ para obtener k.

3-4

a)El segundo byte indica como interpretar los datos una vez descifrados, y los bytes aleatorios aseguran que el número M a cifrar será grande ($M > 2^{8 \cdot L-24}$, y con el segundo byte igual a 2, $M > 2^{8 \cdot L-17}$). Si M fuera demasiado pequeño, el descifrado podría ser trivial (especialmente si $M^e < n$). A más, los bytes aleatorios dificultan los ataques por fuerza bruta cifrando con la misma clave pública: hace falta probar al menos 2^{64} combinaciones para cada posible valor del mensaje.

b)El byte igual a 0 sirve para saber donde acaban los bytes aleatorios (que han de ser diferentes de 0) y donde empieza el mensaje.

3-5 Porqué en los algoritmos simétricos cualquier combinación de bits es una clave válida, y por tanto el esfuerzo para romper una clave de 128 bits debe de ser del orden de 2^{128} operaciones. En cambio, las claves públicas deben cumplir unas propiedades (y por tanto no cualquier combinación de 512 bits es una clave RSA válida), y los métodos para romper claves públicas aprovechan estas propiedades.

3-6 Los números de serie identifican de manera única los certificados que emite una CA, y la manera de hacer creer que se está revocando otro certificado es modificando la CRL. Dado que la CRL está firmada por la CA que la emite, el atacante debería de ser capaz de falsificar la firma de la CA.

3-7 Una solución simple es que el mensaje final de la autenticación en tres pasos incluya el identificador del destinatario:

$$A \rightarrow B: \quad S_A(\{r_B, B\})$$

(Esta es la solución que incorpora la versión actual de la Recomendación X.509.)

3-8 Porqué cifrar con la clave privada un mensaje de longitud arbitraria puede ser muy costoso, ya que la criptografía de clave pública requiere muchos más cálculos que la criptografía simétrica. Por esto se cifra solamente el *hash*, que es de longitud corta.

3-9 Una posible solución (la que utiliza el PKCS #7) es que los datos que se cifran con la clave privada no sean únicamente el *hash* del mensaje, si no una concatenación de este *hash* con un identificador del algoritmo de *hash* utilizado.

3-10 Porqué con ESP se pueden cifrar los datos de los paquetes IP, y después con AH se pueden autenticar los paquetes enteros, incluido la cabecera. Si se hiciera a la inversa, se estaría autenticando el paquete interno con AH, pero en el paquete ESP externo no se estarían protegiendo las cabeceras (con confidencialidad y/o autenticación).

3-11 Dado que el NAT modifica las cabeceras IP (concretamente las direcciones de origen o destino), la combinación más apropiada es utilizar ESP en modo túnel, de modo que el *router* no modifique el paquete encapsulado. No se puede utilizar AH porqué autentica todo el paquete, incluidas las cabeceras. El modo transporte tiene el problema que los *routers* NAT también deben modificar el *checksum* de las cabeceras TCP y UDP, ya que en su cálculo intervienen las direcciones de la cabecera IP. (Alternativamente, se puede utilizar IPsec en la parte externa de la red, después del NAT, o, si las implementaciones lo soportan, deshabilitar los *checksums* TCP y UDP).

3-12

a) La protección es la que da el algoritmo de cifrado simétrico escogido, y dependerá de la longitud de la clave de sesión.

b) Los ataques de texto en claro conocido son posibles, y pueden reducir en parte el esfuerzo necesario para el descifrado por fuerza bruta.

c) Dentro de una misma sesión se detectaría la repetición de los datos, porqué el MAC sería incorrecto. Aunque su usara un cifrado en bloque en modo ECB, el MAC continuaría siendo inválido porqué se calcula a partir de un número de secuencia implícito. Tampoco se pueden copiar datos en sentido contrario porqué se utilizan claves de cifrado y de MAC distintas en cada sentido.

d) Si el intercambio de claves es anónimo, el ataque tendría éxito. Si se utiliza autenticación (del servidor y/o del cliente), el atacante tendría que romper el algoritmo de autenticación.

e) Este problema en general se reduce a un ataque al cifrado simétrico de la comunicación.

f) Si no hay autenticación, este ataque tendría éxito. Si hay autenticación, los certificados (que pueden incluir la dirección IP o nombre DNS del servidor y/o del cliente) sirven para evitar este ataque.

g) Sin conocer las claves de sesión, el atacante no puede continuar la comunicación.

h) El protocolo SSL/TLS trabaja sobre TCP, y no tiene acceso a los mecanismos de establecimiento de la conexión TCP. Por tanto, SSL/TLS no protege contra este ataque.

3-13 Si la clave K está autenticada mediante un certificado, C descubrirá que K' es una clave falsa porqué no habrá un certificado válido para esta clave.

3-14 Los registros SSL/TLS no tendrían que llegar desordenados porqué el protocolo SSL/TLS se usa sobre TCP, que garantiza la secuencia correcta de los datos. Si un atacante intencionadamente desordenara los registros, el receptor no sabría en principio como reordenarlos porqué en el registro no hay ningún número de secuencia explícito (pero el MAC permitiría detectar el cambio de secuencia).

Glosario

AH: Ver *Authentication Header*.

Asociación de seguridad (SA): Relación entre un nodo origen y un nodo destino que utilizan uno de los protocolos IPsec (AH o ESP) para enviar datagramas IP protegidos.

Ataque: Acción realizada por una tercera parte, distinta del emisor y del receptor de la información protegida, para intentar contrarrestar esta protección.

Ataque de cumpleaños: Ataque contra las funciones *hash*, consistente en encontrar dos mensajes que den el mismo resumen, en lugar de encontrar un mensaje que de el mismo resumen que otro determinado, cosa, esta última, que requiere mouchas más operaciones.

Ataque de diccionario: Ataque contra los métodos de autenticación de entidad basados en contraseñas, consistente en probar las palabras de un diccionario hasta encontrar la correcta.

Ataque de fuerza bruta: Ataque contra las funciones criptográficas, consistente en probar todos los posibles valores de la clave hasta encontrar el correcto.

Ataque del hombre a medio camino: Ataque contra la autenticación en los protocolos de comunicación seguros, en qué el atacante intercepta los mensajes de autenticación y los sustituye por otros con las claves públicas cambiadas, de modo que se puede producir una suplantación si no se comprueba la autenticidad de estas claves.

Autenticación: Protección de la información contra falsificaciones.

Autenticación de entidad: Servicio de seguridad que permite confirmar que un participante en una comunicación es auténtico, y no se trata de un impostor que está intentando suplantarlo.

Autenticación de mensaje: Servicio de seguridad que permite confirmar que el originador de un mensaje es auténtico, y que el mensaje no ha sido creado o modificado por un falsificador.

Autenticación de origen de datos: Nombre con el que se conoce a veces la autenticación de mensaje.

***Authentication Header* (AH):** Protocolo de la arquitectura IPsec que proporciona autenticación de los datagramas IP.

Autoridad de certificación (CA): Entidad que emite certificados de clave pública, que sirven para que los usuarios que confíen en esta autoridad se convenzan de la autenticidad de les claves públicas.

Autoridad de certificación (CA) raíz: CA que no tiene ninguna otra superior que certifique la autenticidad de su clave pública, y que por tanto tiene un certificado firmado por ella misma.

CA: Ver *Autoridad de certificación*.

Cadena de certificados: Lista de certificados, cada uno de los cuales permite verificar la autenticidad de la clave pública de la CA que ha emitido el anterior, hasta llegar al certificado de una CA raíz.

Certificado de clave pública: También conocido como certificado digital, es una estructura de datos que contiene un nombre de usuario y su clave pública, y que está firmado digitalmente por una autoridad de certificación dando fe de esta asociación usuario-clave pública.

Certificado digital: Certificado de clave pública.

Cifrado: Transformación de un texto en claro, mediante un algoritmo que tiene como parámetro una clave, en un texto cifrado ininteligible para quien no conozca la clave de descifrado.

Cifrado en bloque: Transformación criptográfica en que el texto en claro se divide en bloques y se aplica un algoritmo de cifrado a cada uno de estos bloques.

Cifrado en flujo: Transformación criptográfica en que el texto en claro se combina con una secuencia pseudoaleatoria obtenida a partir de la clave.

Clave: Parámetro de un algoritmo de cifrado o de descifrado, que permite definir transformaciones criptográficas distintas sin necesidad de cambiar el algoritmo.

Clave de sesión.: Clave simétrica generada *ad hoc* para proteger un determinado intercambio de información, y que es conocida por las dos partes utilizando criptografía de clave pública, para que no pueda ser descubierta por un atacante.

Clave privada.: Clave que permite realizar la transformación criptográfica inversa a la que se obtiene con una clave pública, y que es computacionalmente inviable obtener a partir de esta última.

Clave pública.: Clave que permite realizar la transformación criptográfica inversa a la que se obtiene con una clave privada.

Clave simétrica.: Clave que permite realizar tanto una transformación criptográfica como la transformación inversa, es decir, cifrado y descifrado.

Código de autenticación de mensaje (MAC): Valor calculado a partir de un texto con una clave secreta, y que puede ser utilizado por quien conozca la clave para comprobar la autenticidad del mensaje.

Confidencialidad: Protección de la información contra lectura por parte de terceros no autorizados.

Contraseña: Palabra (*"password'*) o cadena de caracteres secreta, de longitud relativamente corta, usada por una entidad para autenticarse.

Criptoanálisis: Estudio de las técnicas matemáticas para anular la protección que proporciona la criptografía.

Criptografía: Estudio de las técnicas matemáticas para proteger la información, de modo que no pueda ser interpretada por partes no autorizadas.

Criptología: Disciplina que engloba la criptografía y el criptoanálisis.

CRL: Ver *Lista de revocación de certificados*.

Descifrado: Transformación inversa al cifrado, para obtener el texto en claro a partir del texto cifrado y la clave de descifrado.

***Digest*:** Nombre que se da a veces a un resumen o *hash*.

***Encapsulating Security Payload* (ESP):** Protocolo de la arquitectura IPsec que proporciona autenticación y/o confidencialidad de los datos de los datagramas IP.

ESP: Ver *Encapsulating Security Payload*.

Extranet: Red privada de una organización en que una parte de sus recursos son accesibles a determinados usuarios externos a esta organización.

Firma digital: Valor calculado a partir de un texto con una clave privada, y que puede ser comprobado con la correspondiente clave pública, la cual cosa permite confirmar que solamente lo puede haber generado el poseedor de la clave privada.

***Hash*:** Cadena de bits, de longitud predeterminada, que se obtiene a partir de una secuencia de bits de longitud arbitraria, como "resumen" de esta secuencia.

Índice de parámetros de seguridad (SPI): Número que, juntamente con la dirección IP del nodo de destino, permite a un nodo origen identificar una asociación de seguridad IPsec.

Infraestructura de clave pública (PKI): Conjunto de estructuras de datos, procedimientos y agentes que permiten el uso de la criptografía de clave pública.

Intranet: Red privada corporativa de una organización, con acceso restringido a los usuarios que pertenecen a esta organización.

IPsec: Conjunto de protocolos a nivel de red (AH, ESP, etc.) que añaden seguridad al protocolo IP.

Lista de revocación de certificados (CRL): Lista de certificados que han dejado de ser válidos antes de la su fecha de caducidad, emitida y firmada por la misma CA que emitió estos certificados.

MAC: Ver *Código de autenticación de mensaje*.

No repudio: Protección de la información contra denegación de autoría por parte de su originador.

***Padding*:** Datos adicionales que puede ser necesario añadir a un texto en claro antes de aplicarle un algoritmo de cifrado en bloque, para que su longitud sea múltiple de la longitud del bloque.

***Passphrase*:** Cadena de caracteres secreta, de longitud generalmente mayor que una contraseña, usada por una entidad para autenticarse.

***Password*:** Ver *Contraseña*.

PKI: Ver *Infraestructura de clave pública*.

Resumen: Ver *Hash*.

Reto-respuesta: Método de autenticación de entidad basado en un valor secreto, que la entidad a autenticar debe utilizar para calcular una respuesta válida a un reto que le envía el verificador.

SA: Ver *Asociación de seguridad*.

Sal: Conjunto de bits aleatorios que se generan *ad hoc* para modificar una clave de cifrado, y que permiten que un mismo texto resulte en textos cifrados distintos aunque se cifre con la misma clave.

***Secure Sockets Layer* (SSL):** Protocolo para proteger las comunicaciones a nivel de transporte, que ofrece unos servicios de comunicación segura análogos a los que ofrece la interfaz de los *sockets*.

Seguridad computacional: Seguridad que proporciona una técnica criptográfica el criptoanálisis de la cual requeriría una cantidad de recursos computacionales mucho más grande de lo que está al alcance de nadie.

Seguridad incondicional: Seguridad que proporciona una técnica criptográfica que no permite obtener ninguna información sobre el texto en claro, independientemente de la cantidad de recursos disponibles para el criptoanálisis.

SPI: Ver *Índice de parámetros de seguridad*.

SSL: Ver *Secure Sockets Layer*.

Texto en claro: Información directamente inteligible.

Texto cifrado: Resultado de aplicar un cifrado a un texto en claro.

TLS: Ver *Transport Layer Security*.

***Transport Layer Security* (TLS):** Versión del protocolo SSL estandardizada por el IETF (*Internet Engineering Task Force*).