PRIVACIDAD Y SEGURIDAD DE LA INFORMACIÓN

Rafael Álvarez
DCCIA, Universidad de Alicante

Estas notas de apoyo tienen como objetivo suplementar las transparencias y facilitar la comprensión de los contenidos vistos en clase. Son, por tanto, un complemento y no un sustituto de estas.

1 INTRODUCCIÓN

En este tema, analizamos diferentes primitivas (o herramientas) criptográficas que nos pueden ser de utilidad para diseñar, implementar, analizar o auditar sistemas seguros.

Definamos brevemente, a continuación, la terminología de uso común en relación con estos temas:

- *Cifrar (o encriptar)* un mensaje consiste en aplicarle una transformación reversible de forma que sea irreconocible para un atacante. *Descifrar* supone obtener el mensaje original. El proceso de cifrado y descifrado requiere de una clave secreta, en caso contrario estaríamos hablando de *codificación*.
- El mensaje a cifrar se denomina *texto en claro (o texto plano)*, mientras que el mensaje ya cifrado se llama texto cifrado *(o criptograma)*.
- Esta transformación reversible que aplicamos para cifrar y descifrar se denomina *algoritmo de cifrado, criptosistema o cifrador*.
- Los criptosistemas requieren de un valor secreto, llamado *clave*, que determina qué transformación se aplica. Para descifrar correctamente, se debe conocer la clave con la que se ha cifrado el mensaje. Por el contrario, una *contraseña* es una cadena de texto introducida por el usuario de la que se deriva una clave de cifrado.
- El número total de claves posibles se conoce como *espacio de claves*. Es esencial que sea lo suficientemente grande para que un atacante no pueda romper el cifrado simplemente probando todas las claves, lo que se conoce como *ataque por fuerza bruta o búsqueda exhaustiva*.
- La *criptografía* que estudia el diseño de cifradores y el cifrado de mensajes, el *criptoanálisis* intenta romper la criptografía y buscar ataques más eficientes que la mera fuerza bruta, mientras que la *criptología* es la ciencia que combina ambas disciplinas en conjunto.
- Por otra parte, la *esteganografía* consiste en ocultar información dentro de otra información, pero no utiliza clave por lo que no es criptografía, es simplemente codificación. Se puede combinar criptografía con esteganografía, introduciendo datos cifrados, y ocultos, en imágenes, vídeos, sonidos u otros ficheros.

Recordemos, también, los principales tipos de primitivas disponibles en la actualidad:

- Criptografía
  - Simétrica (o de clave privada)
    - Cifrado en flujo
    - Cifrado en bloque
  - Asimétrica (o de clave pública)
- Auxiliares
  - PRNG (generadores pseudoaleatorios)
  - Funciones resumen (o hash) y MAC (autentificación)
  - PBKDF (gestión de contraseñas)

Dentro de la criptografía tenemos dos grandes tipos con aplicaciones muy distintas: simétrica y asimétrica.

La criptografía simétrica, o de clave secreta, se llama así porque utiliza la misma clave para cifrar que para descifrar y porque dicha clave debe permanecer en secreto para garantizar la seguridad. Se utiliza para el cifrado en general ya que suele ser la opción más eficiente. Dentro de este tipo de criptografía, tenemos dos clases de algoritmos distintos: cifradores en flujo y cifradores en bloque.

La criptografía asimétrica, o de clave pública, toma su nombre al utilizar dos claves distintas: una para cifrar y otra para descifrar. En general, una se ha de mantener en secreto (la clave privada) y otra se deja a disposición de otros (la clave pública). Este tipo de criptografía suele hacer uso de funciones matemáticas muy complejas y es mucho más lenta que la criptografía simétrica; por eso, no se utiliza para el cifrado en general. No obstante, permite una serie de aplicaciones que no son posibles o prácticas con criptografía simétrica, como el intercambio seguro de clave o la firma digital.

Además de criptografía, en los sistemas seguros se utilizan otra serie de algoritmos o funciones para otros cometidos distintos al cifrado:

Los generadores de números pseudoaleatorios, o PRNG de sus siglas en inglés, permiten obtener secuencias de números (o bits) impredecibles y tienen muchas aplicaciones entre las que se encuentran la generación de claves u otros valores que tengan que ser impredecibles para garantizar la seguridad del sistema, etc. Además, tienen una relación directa con los cifradores en flujo (un tipo de criptografía simétrica) que son, básicamente, PRNGs rápidos y seguros.

Las funciones hash, o resumen, permiten procesar un bloque de datos (o mensaje) de longitud variable obteniendo un identificador (llamado resumen) de un tamaño fijo (generalmente entre 256 y 512 bits). Cualquier cambio en el mensaje produce (con mucha probabilidad) un resumen distinto y es muy difícil (supone un gran coste computacional) encontrar un mensaje que produzca un resumen determinado o dos mensajes con el mismo resumen. Se utilizan para garantizar la integridad de los datos, para la gestión de contraseñas, para la firma digital, etc. Las *funciones MAC* (código de autentificación de mensaje) son similares a las funciones hash, aceptan un valor secreto (como una clave) además de los datos a procesar y permiten garantizar la autenticidad además de la integridad de los datos.

Las PBKDF, Password Based Key Derivation Function (función de derivación de clave basada en contraseña), son similares una función hash pero que se utiliza exclusivamente en la gestión de contraseñas. Sirven para obtener una clave (lo que necesita un cifrador simétrico) a partir de una contraseña (lo que introduce un usuario) y suelen ser más lentas y/o utilizar más memoria RAM que una función hash normal con el objetivo de ralentizar a un atacante.

2 NÚMEROS ALEATORIOS Y PRNG

2.1 CONCEPTOS BÁSICOS

Las secuencias de números aleatorios tienen muchas aplicaciones en criptografía (entre otras):

- *Generación y distribución de claves.* En muchos casos, las claves no provienen del usuario, se generan de forma automática por el sistema a partir de números aleatorios.
- *Generación de números primos para clave pública.* Estos criptosistemas se suelen basar en números primos muy grandes (de miles de bits) que, por eficiencia, se obtienen mediante números aleatorios a los que luego se les aplican test de primalidad.
- *Valores impredecibles.* En muchos sistemas o protocolos seguros son necesarios valores impredecibles para garantizar la integridad, autenticidad y/o privacidad. Serían buenos ejemplos los números de secuencia de paquetes en TCP, los esquemas de autentificación basados en desafío y respuesta utilizados en esquemas antipiratería o las cookies de sesión para aplicaciones web.
- *Secuencias cifrantes para el cifrado en flujo*. Los cifradores en flujo (como veremos más adelante) son, en esencia, un generador aleatorio.

Las características que debe cumplir una secuencia aleatoria para su uso en criptografía y seguridad son que sea aleatoria e impredecible.

Aleatoriedad. A una secuencia se le pueden aplicar muchos test para detectar redundancias o problemas y determinar que **no** es aleatoria. Una secuencia aleatoria pasa dichos test y tiene una distribución estadística uniforme, y sus valores son independientes unos de otros (un valor de la secuencia no depende de los anteriores ni influye en los posteriores).

Impredecibilidad. Aunque un atacante pudiera obtener los valores de la secuencia no debe ser capaz de predecir el siguiente bit (o valor) de la secuencia.

No todas las secuencias que aparentan aleatoriedad son impredecibles.

Tenemos dos clases de generadores de secuencias aleatorias: realmente aleatorios y pseudoaleatorios.

Los generadores realmente aleatorios se basan, usualmente, en procesos físicos o externos al computador (como el ruido en un circuito eléctrico, la frecuencia de emisión de partículas en radioactividad, la frecuencia del tecleo o de paquetes de red, etc.) de los que se extrae aleatoriedad. Son costosos de implementar y mantener en algunos casos y requieren ser procesados para "destilar" la aleatoriedad que ofrecen, lo que los hace muy lentos. Además, no son deterministas por lo que la secuencia que generan no se puede reproducir puesto que provienen de un origen que no controlamos.

Por el contrario, los generadores pseudoaleatorios (o PRNG) son algoritmos (programas de ordenador), deterministas (la secuencia se puede reproducir si utilizamos el mismo valor inicial) y muy eficientes. Además, las secuencias producidas por un PRNG son indistinguibles en la práctica de las secuencias producidas por un generador realmente aleatorio.

Por las ventajas que ofrecen, utilizamos generadores pseudoaleatorios en la gran mayoría de los casos.

2.2 GENERADORES PSEUDOALEATORIOS

Los generadores pseudoaleatorios (o PRNG) son algoritmos que se basan en un esquema muy simple:

- Parten de un valor inicial llamado semilla
- Mediante una función de evolución toman el valor inicial y obtienen el valor siguiente.
- Progresan de la misma forma para obtener todos los valores de la secuencia, tomando el valor anterior, aplicando la función de evolución y obteniendo el valor siguiente.

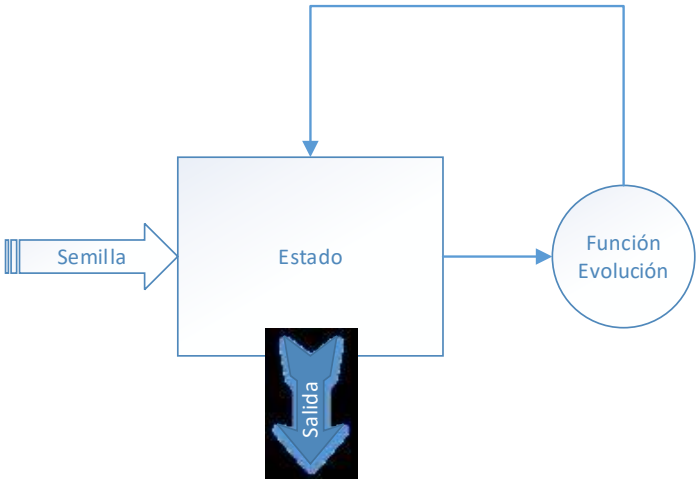

Ilustración 1: diagrama de un PRNG

De esta forma, la semilla determina la secuencia producida. Para una misma semilla, el PRNG generará siempre la misma secuencia mientras que para semillas distintas generará secuencias distintas.

Todos los PRNG producen secuencias que acaban por repetirse tras una longitud determinada. Esta longitud se denomina **período de la secuencia**. Es importante **evitar que la secuencia se repita** (cambiando la semilla antes de que esto ocurra) puesto que dejaría de ser aleatoria e impredecible y, por lo tanto, sería insegura.

Además, el **tamaño de la semilla** es un factor importante ya que determina el número de semillas posibles y, en definitiva, el número de semillas que un atacante tendría que probar para encontrar la semilla que produce una secuencia determinada. La semilla es un concepto similar al de clave en un cifrador por lo que su tamaño debe ser similar (128 o 256 bits, por ejemplo) para garantizar la seguridad.

La semilla deberá ser un valor aleatorio y por lo tanto **impredecible** si queremos que la secuencia generada a partir de la misma también lo sea: de poco valdría tener un PRNG excelente si siempre utilizamos la misma semilla y el atacante la conoce.

2.3 ALGORITMOS PRNG

A la hora de elegir un generador pseudoaleatorio podemos tomar diferentes alternativas: emplear algoritmos que han sido diseñados a propósito para ser PRNG, tomar un cifrador en flujo (puesto que son prácticamente lo mismo) o adaptar otras primitivas existentes para que funciones a modo de PRNG.

Analizamos, a continuación, algunos de los PRNG más populares.

2.3.1 CONGRUENCIAL LINEAL

Este algoritmo es también conocido como *"random de C"* (C es un lenguaje de programación muy utilizado) y es inmensamente popular ya que se utiliza por defecto en casi todos los lenguajes de programación. Se basa en la siguiente expresión:

$$X_{n+1} = (a \cdot X_n + c) \bmod m$$

Donde el valor anterior (X_n) se multiplica por un coeficiente (a), se le suma un desplazamiento (c) y todo ello módulo m constituye el siguiente valor de la secuencia (X_{n+1}). El nombre de congruencial lineal hace referencia a que es la expresión de una recta (lineal) en aritmética modular (congruencial).

A pesar de generar secuencias que, desde el punto de vista de la aleatoriedad estadística, son bastante buenas y que pueden tener un período bastante largo si se eligen los parámetros de forma adecuada, **resulta inadecuado para su uso en criptografía o seguridad ya que es predecible.**

Bastan cuatro valores de la secuencia para poder determinar la semilla y los parámetros necesarios (a, c y m) con los que reproducir toda la secuencia. Lamentablemente, al ser tan popular, se utiliza con mucha frecuencia por programadores inexpertos en tareas de seguridad, permitiendo ataques relativamente sencillos.

2.3.2 BLUM BLUM SHUB

Este PRNG debe su nombre a sus autores y, al contrario que con el congruencial lineal, es demostrablemente seguro al basarse en problemas matemáticos bien conocidos. Sigue el siguiente algoritmo:

$$\mathrm{X}_{n+1} = (X_n)^2 \bmod m$$

En el que el valor anterior se eleva al cuadrado y se aplica el módulo m para obtener el valor siguiente y donde m es el producto de dos números primos muy grandes (miles de bits):

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

$$m = pq$$

A pesar de su gran seguridad, es **extremadamente lento** (al utilizar números tan grandes) por lo que no se utiliza en la práctica y solo tendría sentido su aplicación para la generación de secuencias muy cortas.

2.3.3 PRNG BASADOS EN CIFRADORES EN BLOQUE Y OTRAS PRIMITIVAS

Partiendo de un *cifrador en bloque seguro*, podemos obtener un PRNG seguro (y bastante eficiente) siguiendo esquemas o métodos bien conocidos como los modos de operación CTR y OFB o esquemas como el estándar ANSI X9.17. Tratamos los cifradores en bloque más adelante.

También se pueden emplear las *funciones hash, MAC o PBKDF a modo de PRNG* para obtener secuencias pseudoaleatorias.

Hay que tener en cuenta que los cifradores en flujo son, básicamente, PRNG rápidos y seguros por lo que podemos utilizar cualquier *cifrador en flujo* como PRNG también. Al igual que los cifradores en bloque, vemos los cifradores en flujo a continuación.

3 CRIPTOGRAFÍA SIMÉTRICA (O DE CLAVE PRIVADA)

La criptografía simétrica es la **herramienta básica de cifrado** que utilizamos en la gran mayoría de los casos, ya que es rápida, práctica y segura. Como contrapartida, la criptografía asimétrica es mucho más lenta, pero permite funcionalidades adicionales como la firma digital o el intercambio de claves seguro.

Dentro de la criptografía simétrica tenemos dos grandes clases de criptosistemas: los cifradores en bloque y los cifradores en flujo. Si bien son dos enfoques o diseños distintos, ambos tienen una funcionalidad, rendimiento y seguridad similar. De hecho, en la práctica, es muy común convertir los cifradores en bloque en cifradores en flujo empleando modos de operación específicos (que vemos más adelante).

Las principales diferencias entre ambos son las siguientes:

- Los *cifradores en bloque*
  - Dividen el mensaje en **bloques de varios elementos** (128 bits, por ejemplo) y lo cifran bloque a bloque.
  - Aplican una **transformación fija** que depende únicamente de la clave utilizada. Esto significa que, para la misma clave y cifrador, el mismo bloque de texto en claro produce siempre el mismo bloque de texto cifrado.
  - La gran mayoría (aunque no todos) se basan en una estructura común conocida como red **Feistel**.

- Los *cifradores en flujo*
  - Toman el mensaje **elemento a elemento** (bit a bit o byte a byte) y lo procesan a modo de cadena (como si fuera letra a letra).
  - Aplican una **transformación variable** que depende de la clave y de la posición del elemento en el mensaje. De esta forma, el mismo byte (o bit) se cifra de forma distinta en función de su posición.
  - Todos los cifradores en flujo actuales se basan en una estructura común conocida como esquema **Vernam**.

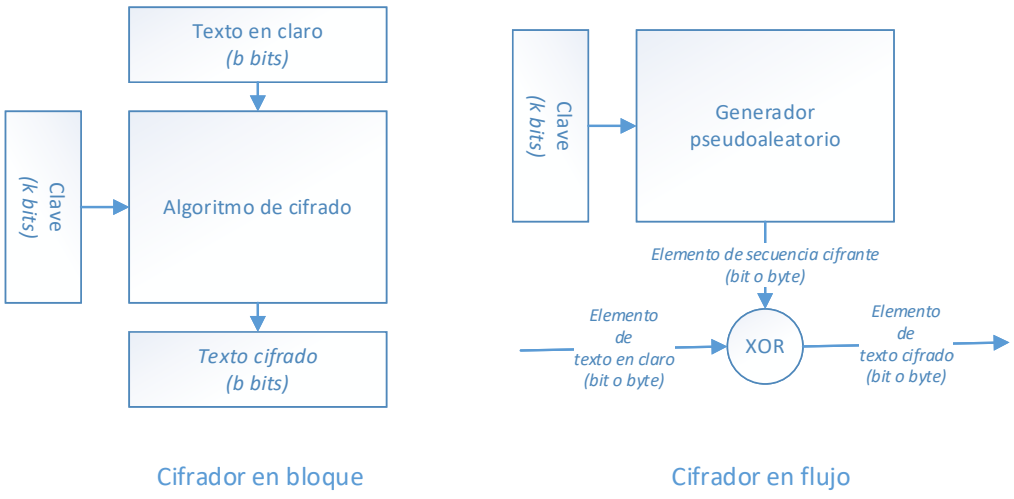

Ilustración 2: tipos de cifrado simétrico

3.1 CIFRADO EN FLUJO

3.1.1 VERNAM

El esquema de *Vernam* consiste en utilizar una *secuencia aleatoria* (llamada *secuencia cifrante*) de la misma longitud que el mensaje y cifrar mediante la operación XOR (or exclusiva bit a bit) entre el mensaje y la secuencia cifrante, obteniendo el texto cifrado.

El XOR posee la propiedad de anularse al ser aplicado dos veces, de forma que **descifrar es simplemente cifrar de nuevo el texto cifrado** (hacer XOR dos veces con la secuencia cifrante), obteniendo el mensaje original.

Utilizar una secuencia aleatoria tan larga como el mensaje no es viable en la práctica, ya que sería necesario que el emisor y el receptor compartieran dicha secuencia de forma segura; lo que no tiene sentido al ser equivalente a compartir directamente el mensaje de forma segura sin necesidad de criptografía. Por ello, en la práctica, se utiliza un generador pseudoaleatorio para generar la secuencia cifrante, de forma que solamente hace falta compartir la semilla (que actúa como clave en este esquema) y que es, en general, mucho más corta que el mensaje completo.

Para que el esquema Vernam sea seguro, debemos utilizar un PRNG **impredecible** y que tenga un **período suficientemente largo** (debemos cambiar de semilla o clave antes de que se repita). Además, este PRNG ha de aceptar **semillas** suficientemente **largas** para que el espacio de claves sea seguro ante un ataque por fuerza bruta. En la actualidad, se considera seguro una semilla (o clave) de 128 bits, si bien es ideal utilizar **256 bits** para tener un cierto margen de seguridad adicional.

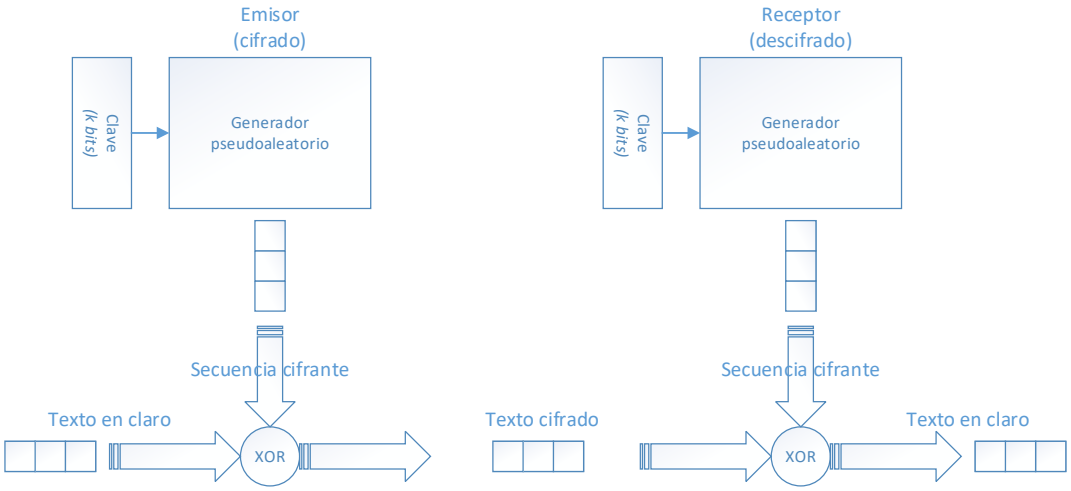

Ilustración 3: esquema de Vernam con PRNG

Este esquema requiere sincronización entre emisor y receptor de forma que no se pierdan datos ni se modifique el texto cifrado (algo que garantiza internet), aunque existen variantes asíncronas (o autosincronizantes) para canales de comunicación sin dichas garantías (como transmisiones espaciales o ruidosas), pero su uso es poco frecuente.

3.1.2 LFSR

Los registros de desplazamiento con retroalimentación lineal (o LFSR de sus siglas en inglés) son muy populares para la generación de secuencias pseudoaleatorias y, por tanto, para el diseño de cifradores en flujo (ver el diagrama en las transparencias de clase).

Estos registros poseen ciertas características deseables:

- Se pueden analizar de forma sencilla dado que tienen propiedades matemáticas conocidas.
- Su implementación en hardware es eficiente y directa (aunque no tanto en software).
- Con ciertas funciones de retroalimentación determinadas, se puede garantizar un cierto período (longitud) de la secuencia antes de que se repita.
- Las secuencias generadas tienen muy buenas propiedades de aleatoriedad estadística.

Si bien estas propiedades son muy positivas, los LFSR por sí solos no son seguros puesto que **son predecibles** y **su uso en criptografía no estaría muy recomendado**. No obstante, puesto que poseen estas propiedades tan atractivas desde el punto de vista matemático, se han utilizado (y se utilizan) combinados con otras técnicas para intentar explotar sus características de aleatoriedad de forma segura.

3.1.3 RC4

RC4 (o Rivest Cipher 4) es un algoritmo de cifrado en flujo tremendamente popular. Diseñado por Ron Rivest (la R de la empresa de seguridad RSA) en 1987, se mantuvo en secreto hasta que en 1994 se publicó de forma anónima en internet.

Dado que es tremendamente eficiente y sencillo (ver detalles en transparencias y materiales de ampliación), se ha utilizado para el cifrado en multitud de protocolos de seguridad como SSL/TLS (el protocolo utilizado para conectar a web seguras) o WEP/WPA (protocolos de seguridad para WiFi).

Lamentablemente, recientemente se han publicado ataques cada vez más efectivos contra RC4 generando mucha desconfianza en su diseño, llegando a ser prohibido en las versiones actuales de SSL/TLS.

A pesar de que no hay un ataque definitivo contra RC4, su uso **no sería recomendable**.

3.1.4 SALSA20

Salsa20 es un cifrador en flujo diseñado por Dan Bernstein para el concurso europeo eStream. Está considerado seguro y es muy eficiente en software al basarse únicamente en 3 operaciones: suma de valores de 32 bits, XOR bit a bit y rotaciones de bits.

Tiene como característica interesante que su PRNG es capaz de generar cualquier punto de la secuencia cifrante sin tener que generar los valores anteriores (esto es algo bastante inusual). Existen variantes de menos rondas (más rápidas pero menos seguras) como Salsa20/12 o Salsa20/8, pero no se recomienda su uso. También existe una versión mejorada que se llama ChaCha[1] aunque no goza de la misma popularidad que Salsa20.

Salsa20 sería el cifrador en flujo recomendado en la actualidad.

[1] Bernstein parece tener cierta predilección por los bailes para nombrar a sus cifradores.

3.1.5 OTROS

Si bien nuestra recomendación es utilizar Salsa20 (o ChaCha), hay muchos cifradores en flujo interesantes (se recomienda consultar Wikipedia para ampliar) desde el punto de vista histórico o del diseño, entre otros:

- Los cifradores *A5* (tanto A5/1 como A5/2) basados en LFSRs y utilizados en el estándar de *telefonía GSM*.
- El cifrador en flujo *SEAL*, diseñado en 1997 por IBM y que es bastante rápido en software al basar su diseño en la generación de tablas que dependen de la clave. El hecho de haber sido patentado a afectado negativamente a su popularidad.
- *Phelix* (y su versión original *Helix*) es un cifrador en flujo diseñado por Bruce Schneier y otros para el concurso *eStream*. Muy eficiente en plataformas de 32 bits, ha sufrido ciertos ataques que han impedido su uso generalizado.
- *HC-128* (y su versión mejorada *HC-256*) es un diseño de Hongjun Wu publicado en 2004 y no patentado que también participó en *eStream*. No se conocen ataques y es bastante eficiente, aunque requiere de una inicialización (para cada cambio de clave) muy lenta. No es tan popular como Salsa20.
- *SNOW 3G* es un algoritmo propuesto por Johansson y Ekdahl en 2006 y está basado en un LFSR combinado con una máquina de estados finitos (FSM en inglés). Es el cifrador elegido para el protocolo 3GPP *(telefonía 3G y 4G)*.
- *Spritz* fue propuesto por Rivest (el autor de RC4) en 2014 como sustituto de RC4. Es una modificación extensa de RC4 basada en un diseño de función esponja que, si bien es mucho más seguro, también es bastante más lento por lo que no está gozando de la misma popularidad que RC4 o Salsa20.

3.2 CIFRADO EN BLOQUE

3.2.1 FEISTEL

La mayoría de los cifradores en bloque sigue un esquema común que se llama *red Feistel*, por su autor. *Horst Feistel* propuso un diseño de cifrador en bloque basado en unos conceptos sencillos:

- Propone una aproximación al cifrador en bloque ideal (que no es realizable en la práctica) mediante un cifrador producto. Esto consiste en ejecutar operaciones más o menos sencillas en secuencia, o por rondas, de forma que se incremente la complejidad al aplicar cada ronda sucesiva.
- También indica la necesidad de que el cifrador cuente con un tamaño de clave independiente (k bits) del tamaño del bloque (n bits), de forma que el número de transformaciones posibles (o espacio de claves) sea 2^k.
- Por último, basa su diseño en un cifrador que alterna sustitución (cambiar un elemento por otro) y permutación (reordenar elementos).

En general, todos los cifradores en bloque basados en una red Feistel son parecidos, pero se diferencian en lo siguiente:

- El *tamaño del bloque*, un valor común en la actualidad sería 128 bits.
- El *tamaño de la clave*, se considera que 128 bits es seguro, pero 256 bits es lo ideal para tener cierto margen de seguridad.
- El *algoritmo de generación de subclaves*, que se encarga de derivar valores para cada ronda a partir de la clave de cifrado/descifrado. Su complejidad es uno de los factores que determinan la seguridad general del cifrador.
- La *función de ronda*, que se encarga transformar una de las dos mitades en cada ronda y cuyo diseño es el otro factor principal de la seguridad del cifrador.
- El *número de rondas*, un número común es 16 pero depende completamente del diseño del resto del cifrador. A más rondas, más complejidad y seguridad, pero también peor rendimiento. Esto lleva a buscar un equilibrio entre seguridad y velocidad de cifrado.

En definitiva, muchos autores basan sus diseños en una red Feistel puesto que es sencilla de comprender y aporta ciertas garantías como que el cifrador se pueda descifrar, que no es tan evidente a priori.

3.2.2 DES

Horst Feistel dirigió un proyecto en IBM a finales de los años 60 que dio como resultado un cifrador llamado *Lucifer*. Este algoritmo fue uno de los candidatos del concurso *DES* (Data Encryption Standard o estándar de cifrado de datos) y cuando ganó se convirtió en DES, el primer estándar de cifrado de datos moderno.

Las diferencias entre DES y Lucifer no son simplemente un cambio de nombre, la NSA modificó el diseño para mejorar la seguridad y redujo la clave de 128 a 56 bits para que cupiera en un único chip (el hardware de la época era mucho más limitado que el actual).

Estas modificaciones provocaron el recelo de la comunidad científica, ya que se había reducido la clave de forma significativa y los criterios que habían motivado los cambios sobre el diseño de Lucifer no se hicieron públicos. Más tarde, se descubrió que la NSA conocía entonces un ataque (criptoanálisis diferencial) que la comunidad científica no descubrió hasta más tarde y modificó Lucifer para hacerlo más resistente.

No obstante, la clave de 56 bits que utiliza DES no se debería considerar segura en la actualidad ya que es demasiado fácil (o rápido) de romper por fuerza bruta con la capacidad computacional que tenemos hoy en día (ordenadores mucho más rápidos y baratos, computación en la nube, etc.).

Si bien DES ha gozado de gran popularidad, no tiene sentido utilizarlo en la actualidad: es un diseño arcaico, ineficiente en software y con una longitud de clave insuficiente. Es posible que en algunos sectores se utilicen aplicaciones o sistemas antiguos heredados que todavía hagan uso de chips DES para cifrar datos. En estos casos y para que sea seguro, se utiliza de una forma especial que se llama triple DES y que supone hacer tres cifrados DES (en realidad dos cifrados y un descifrado) y duplicar la clave de 56 a 112 bits (que se podría considerar seguro hoy en día). A pesar de triple DES, **su uso no sería recomendable para ninguna aplicación actual.**

3.2.3 AES

Una vez quedó claro que DES estaba quedando desfasado muy rápidamente, el *NIST* (instituto de estandarización y tecnología de E.E.U.U.) convocó otro concurso para establecer un nuevo estándar de cifrado más avanzado. Esto dio lugar al concurso *AES* (Advanced Encryption Standard o estándar de cifrado avanzado) en el año *2001* y que ganó un algoritmo originalmente llamado *Rijndael* (por sus autores, Joan Daemen y Vincent Rijmen de Bélgica).

Este nuevo estándar, AES, tiene ciertas peculiaridades interesantes: *no utiliza una red Feistel*, se basa en *operaciones matemáticas relativamente complejas* (suma, producto y división sobre un cuerpo de Galois, que es un concepto de álgebra) y está diseñado para ser *implementado de forma eficiente en software* (tanto en máquinas de 8 bits como en procesadores más complejos de 32 bits). Además, la gran mayoría de procesadores actuales soportan *aceleración por hardware* para AES, mejorando el rendimiento de forma muy significativa y poniéndolo en el mismo nivel que cifradores en flujo muy rápidos como RC4 o Salsa20.

A pesar de no ser estrictamente una red Feistel, sí que se basa en conceptos similares: varias rondas idénticas que se parametrizan con una subclave distinta para cada ronda, etc. En AES el bloque se considera como una matriz de 4x4 bytes (16 bytes o 128 bits) y las rondas comprenden 4 operaciones básicas todas utilizando esta aritmética especial sobre un cuerpo de Galois: sustituir elementos mediante una tabla (*SubBytes*), reordenar filas (*ShiftRows*), producto matriz vector (*MixColumns*) y combinar la subclave de ronda con el bloque (*AddRoundKey*). Se puede consultar las transparencias de clase para más detalle.

AES permite claves de 128, 192 o 256 bits simplemente incrementando el número de rondas para los tamaños de clave mayores y es considerado seguro en la actualidad. Dado que está acelerado por hardware en muchos sistemas, es una opción idónea para el cifrado de propósito general y **es el cifrador en bloque recomendado en la actualidad**.

3.2.4 OTROS

Si bien nuestra recomendación es utilizar AES (idealmente con clave de 256 bits), hay muchos cifradores en bloque interesantes (se recomienda consultar Wikipedia para ampliar) desde el punto de vista histórico o del diseño, entre otros:

- El cifrador IDEA, un estándar internacional de cifrado en bloque publicado en 1991 y que no ha gozado de la popularidad de DES o AES.
- Los algoritmos de Ron Rivest RC5 y RC6 que, al contrario que RC4, son cifradores en bloque y no han tenido éxito en la práctica.
- El algoritmo Blowfish (Schneier y otros en 1993) y su versión mejorada Twofish (1998) que tienen un diseño interesante y han gozado de cierta popularidad en el mundo del software libre y la gestión de contraseñas.

3.2.5 MODOS DE OPERACIÓN

Los cifradores en bloque no se suelen utilizar directamente siguiendo su diseño básico, se aplican en distintos modos de operación que, partiendo del cifrador en bloque básico, obtienen mejoras significativas en ciertos aspectos. Se recomienda consultar los diagramas de las transparencias de clase. Los modos más comunes son:

- **Cifrado triple** (por ejemplo, triple DES). Este modo se utiliza en DES para aumentar la seguridad y duplicar el tamaño de la clave. Se realiza un cifrado con una clave K1, un descifrado con una clave K2 y otra vez un cifrado con la clave K1; de esta forma tenemos un rendimiento 3 veces peor, pero utilizamos dos claves distintas y, por lo tanto, duplicamos el espacio de claves y la seguridad frente a un ataque de fuerza bruta. Se podría aplicar a cualquier cifrador, pero sólo tiene sentido en el caso de DES ya que su clave es de 56 bits y no es seguro sin aplicar el cifrado triple. **No se recomienda su utilización.**
- **Modo ECB** (libro de código electrónico). Este es el modo directo de utilizar un cifrador en bloque, sin aplicar ningún modo de operación avanzado. Presenta el gran problema de que, para la misma clave, la misma entrada siempre produce la misma salida; lo que implica que, muchas veces, no esconde los patrones estadísticos o de frecuencia de la información original (ver la imagen del pingüino Tux en la transparencia 2b-29). Además, es necesario conformar el texto en claro al tamaño del bloque, esperando a tener información suficiente para rellenar un bloque completo o rellenar el último bloque si el mensaje no tiene una longitud múltiplo del tamaño del bloque. Por estas razones, **no se recomienda utilizar el modo ECB**.
- **Modo CBC** (encadenamiento de bloque cifrado). En este modo se utiliza el bloque cifrado anterior para hacer el XOR con la entrada del bloque siguiente; como en el primer bloque no tenemos bloque anterior, se utiliza un valor inicial aleatorio (IV) que no tiene porqué ser secreto, pero debe estar disponible para el cifrado. De esta forma se evita que la misma entrada, con la misma clave, produzca siempre la misma salida. No obstante, sigue presentando el inconveniente de tener que conformar el mensaje al tamaño del bloque y, además, este encadenamiento fuerza a una ejecución secuencial (no permite paralelismo o ejecución sobre múltiples núcleos).
- **Modo CFB** (realimentación de cifrado). Este modo convierte el cifrador en bloque en un generador pseudoaleatorio que se utiliza a modo de cifrador en flujo. Para ello, encadena la salida cifrada como entrada del siguiente bloque y genera un bloque cifrado que utiliza como secuencia para cifrar mediante XOR (al estilo Vernam) con el texto en claro. Esto mejora el modo CBC ya que no requiere conformar el mensaje al tamaño del bloque, pero tiene penalizaciones en el rendimiento.
- **Modo OFB** (realimentación de salida). Es similar al anterior, pero realimenta la salida del cifrador en bloque en lugar de la salida del cifrado final. Es muy parecido a un cifrador en flujo, pero con elementos de tamaño bloque en lugar de bit o byte. Como el encadenamiento no utiliza el texto en claro, permite precalcular varios cifrados mientras se espera la recepción de todo el mensaje.
- **Modo CTR** (contador). Este modo no realiza encadenamiento, emplea un contador cuyo valor se va incrementando de uno en uno y utiliza la salida del cifrado en bloque como secuencia cifrante. El valor inicial del contador no debe repetirse, pero puede ser público (lo secreto es la clave). Al no realizar encadenamiento, permite su ejecución en paralelo. **El modo CTR sería la recomendación tanto por seguridad y eficiencia en la mayoría de los casos** (salvo que necesitemos alguna característica concreta de los modos anteriores).
- **Modo XTS** (cifrado de disco). Este modo está *indicado únicamente para el cifrado de discos* ya que tienen características especiales. Utiliza dos claves y varios cifrados y tiene en cuenta la posición de los datos en el disco para el cifrado. La mayoría de los sistemas de cifrado de disco (Bitlocker de Microsoft o FileVault de Apple, etc.) utilizan el modo XTS con AES; aunque no son necesariamente compatibles entre sí.

4 FUNCIONES HASH

4.1 CONCEPTOS BÁSICOS

Una función hash (o función resumen) criptográfica toma una entrada de tamaño variable y genera una salida de tamaño fijo (resumen). En realidad, las funciones hash tienen una entrada de tamaño fijo también, pero dividen el mensaje en bloques de ese tamaño y los van procesando iterativamente (uno tras otro), generando un único resumen para todo el mensaje. Como el mensaje no suele ser de una longitud múltiplo del tamaño de entrada de la función resumen, se hace necesario rellenar el último bloque y la mayoría de funciones hash usan un esquema conocido como Merkle-Damgärd para esto.

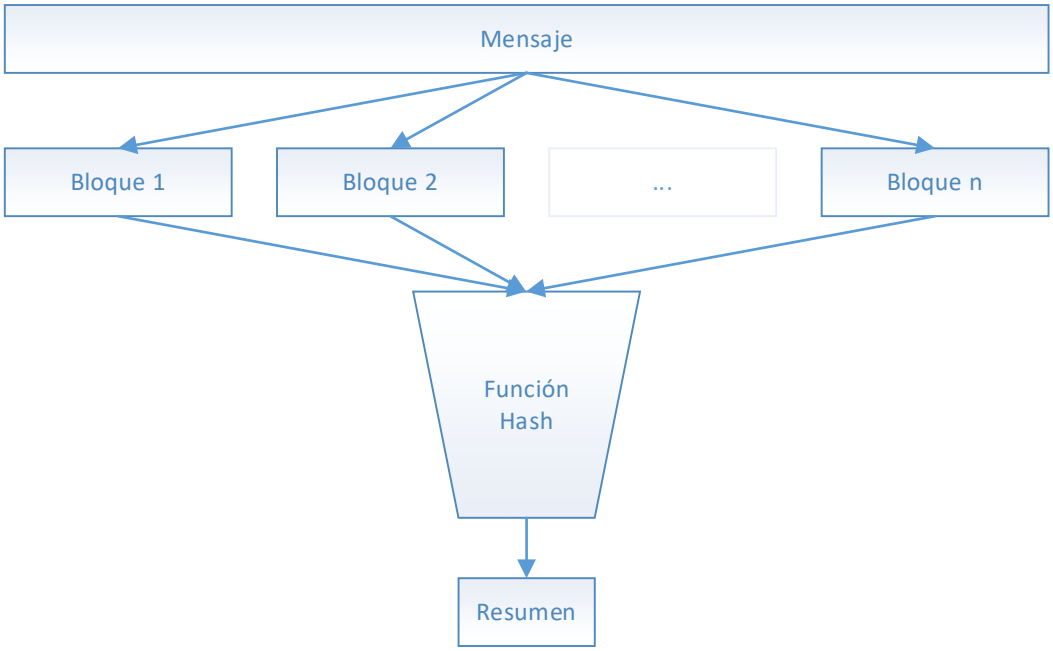

Ilustración 4: esquema básico de una función hash

La propiedad principal de una función hash es que se puede utilizar el resumen como identificador del mensaje: es fácil calcular el resumen a partir de un mensaje y el resumen varía mucho ante cualquier modificación del mensaje. Desde el punto de vista de la seguridad, es muy costoso para un atacante:

- Encontrar dos mensajes cualesquiera que tengan el mismo resumen (lo que se conoce como *colisión débil*).
- Partiendo de un mensaje determinado, encontrar otro mensaje con el mismo resumen (*colisión fuerte*).
- Teniendo un resumen, generar un mensaje que produzca dicho resumen (a esto se le llama *preimagen*).

En la actualidad, consideramos que una función hash es **segura a partir de 256 bits de resumen**, lo que implica una resistencia a colisiones de 2^{128} (por la paradoja del cumpleaños se reduce a la mitad de bits y no 2^{256}).

Debe quedar claro que **una función hash no es una función de cifrado,** no se puede descifrar y además no hay clave, pero permite una serie de aplicaciones muy útiles, entre las que se encuentran:

- **La autentificación de mensajes.** Dado que al variar el mensaje también varía el resumen, podemos utilizar las funciones hash para garantizar la integridad de los datos.
- **Mayor eficiencia en la firma digital.** Puesto que firmar el mensaje completo es muy lento (hace uso de criptografía de clave pública o asimétrica) podemos, en su lugar, firmar el resumen del mensaje que suele ser de tamaño más reducido y, por lo tanto, más rápido de firmar.
- **La gestión de contraseñas.** No resulta seguro guardar las contraseñas de los usuarios en claro en una base de datos que podría ser robada, etc.; para evitar esto podemos hacer uso de funciones hash para guardar el resumen de la contraseña en lugar de la contraseña en sí. De esta forma, si la base de datos con las contraseñas fuera robada, sólo tendrían acceso a los resúmenes que carecerían de utilidad a priori (ver sección de gestión de contraseñas).
- **Miscelánea.** Las propiedades de las funciones hash permiten su uso en múltiples aplicaciones: detección de intrusos en la red, antivirus, detección de modificación de ficheros del sistema operativo, construcción de generadores pseudoaleatorios, etc.

4.2 ALGORITMOS HASH COMUNES

Muchas de las funciones hash actuales provienen de los diseños originales de Ron Rivest para su línea de funciones hash: *MD2* (Message Digest 2 o digestión de mensaje, 1989), *MD4* (1990) o *MD5* (1992). Todas ellas están rotas, por lo que **no se recomienda su uso** en ninguna situación que requiera seguridad.

En 1995, el *NIST* estadounidense decide crear el estándar *SHA-1* (Secure Hash Algorithm o algoritmo hash seguro) que es un diseño de la *NSA*. Este hash de 160 bits de resumen **ha sido roto recientemente** por Google, demostrando la posibilidad de crear 2 mensajes distintos con el mismo resumen.

SHA-2. Como mejora a SHA-1, el NIST estandarizó SHA-2 en 2001. Es una evolución de este, también diseñado por la NSA, que tiene dos variantes distintas: una con un resumen de 256 bits (diseñada especialmente para procesadores de 32 bits) y otra con un resumen de 512 (diseñada para procesadores de 64 bits). Ambas versiones son consideradas seguras en la actualidad y gozan de gran popularidad.

SHA-3. A pesar de que SHA-2 es considerado seguro, el NIST creó el concurso SHA-3 (de 2008 a 2012) para tener un diseño completamente nuevo que no tuviera la herencia de MD4 ni del esquema Merkle-Damgärd. De esta forma, si se encontrara un ataque satisfactorio para SHA-2, SHA-3 sería inmune al ser radicalmente diferente. Este concurso lo ganó el algoritmo Keccak, propuesto por un equipo que incluye a Joan Daemen (uno de los creadores de AES).

A pesar de que SHA-3 es el estándar actual, se sigue empleando SHA-2 con asiduidad, ya que es bastante más rápido que SHA-3. **Se recomienda utilizar tanto SHA-2 como SHA-3 indistintamente, idealmente con un resumen de 512 bits.**

5 GESTIÓN DE CONTRASEÑAS

5.1 CONCEPTOS BÁSICOS

Hemos comentado con anterioridad que las funciones hash pueden ayudar en la gestión de contraseñas. Entendemos por **gestión de contraseñas** al modo en el que se manejan y autentifican los usuarios en un servidor, aplicación o servicio a partir de cadenas o frases de texto que sólo conoce el usuario y que denominamos **contraseña**.

El proceso de autentificación (o login) suele consistir en tener previamente almacenada una copia de la contraseña en una base de datos o fichero en el servidor para comprobar después que la contraseña introducida por el usuario coincide con la almacenad; esta copia de la contraseña se almacena generalmente durante el registro. Existen múltiples formas de gestionar la autentificación y el almacenamiento de contraseñas en el servidor, podemos destacar las siguientes:

En claro. Las contraseñas se almacenan directamente en la base de datos por lo que, si ésta fuera robada, se tendría acceso a todas las contraseñas, con los problemas de seguridad que esto conllevaría: suplantación de identidad, más aún en caso de que el usuario utilice la misma contraseña para múltiples servicios, etc.

Cifrado. Una solución inicial consiste en almacenar las contraseñas cifradas con una clave que, generalmente, gestiona el administrador del servidor. Si bien es más seguro que el almacenamiento en claro, el hecho de usar un cifrador implica que se podrían descifrar todas las contraseñas en caso de encontrar dicha clave; además, se obtiene el mismo resultado cifrado para todas las contraseñas idénticas, facilitando la labor a un atacante (las que más veces se repitan serán "1234", etc.).

Hashing. Una mejora significativa consiste en utilizar una función hash en lugar de un cifrador, guardando todos los resúmenes de las contraseñas en lugar de las contraseñas en sí. Esto impide que se descifren puesto que no hay clave que custodiar.

Si bien el hashing parece ser una opción idónea, tiene tres inconvenientes significativos. Un atacante puede crear una lista de contraseñas habituales y calcular sus resúmenes a priori, por lo que luego sólo tendría que comparar su **tabla precalculada** con la base de datos robada sin tener que calcular los resúmenes de nuevo. Otro problema es que las **GPUs** (tarjetas gráficas específicas) permiten calcular hashes a mucha velocidad, posibilitando a un atacante probar muchas contraseñas muy rápido e, incluso, probar todas las contraseñas posibles de una determinada longitud (10 caracteres, por ejemplo). Por último, **la misma contraseña va a producir el mismo resumen** en la base de datos, por lo que un atacante podrá comprobar cuáles se repiten más, etc.

Hashing + sal. Para evitar el problema del precálculo de contraseñas habituales, se puede concatenar una cadena de bits aleatoria, que llamamos sal, a la contraseña y calcular el resumen de ambas. De esta forma, el atacante no puede hacer una tabla precalculada puesto que el resumen depende de la sal y no sólo de la contraseña. Además, la misma contraseña para distintos usuarios producirá resúmenes distintos, evitando que el atacante descubra las más frecuentes, etc. No obstante, no evitamos el ataque por fuerza bruta mediante GPUs.

Para evitar el ataque por GPUs, han surgido diseños de funciones hash que tienen un coste computacional configurable, es decir, se puede elegir cuanto tiempo tardan. Este tipo de funciones se llaman **PBKDF** (Password Based Key-Derivation Function o función de derivación de clave basada en contraseña) y son, básicamente, funciones hash más lentas y que pretenden ralentizar a un atacante. **Esta sería la opción más recomendable en la actualidad, utilizar una función PBKDF junto con la sal.**

5.2 ALGORITMOS PBKDF COMUNES

Entre los algoritmos PBKDF más comunes, tenemos tres:

PBKDF2[2]. Esta función es un estándar de internet del año 2000 y se basa en repetir mediante un bucle una función hash convencional. El número de repeticiones, y por tanto el tiempo de cómputo, es configurable. Si bien es un estándar empleado por Microsoft o Apple entre otros, es de los peores PBKDF frente a GPUs, ya que utiliza internamente funciones hash convencionales que se pueden calcular muy eficientemente en dichos dispositivos. **No se recomienda su uso.**

BCRYPT. Este PBKDF de 1999 utiliza el cifrador en bloque blowfish como parte de su diseño. Además, emplea una modesta cantidad de memoria RAM lo que ralentiza a las GPUs al limitar su paralelismo. Es mejor que PBKDF2, pero **no es la mejor opción posible.**

SCRYPT. Se basa en el cifrador en flujo Salsa20 y hace uso de grandes cantidades de memoria RAM; esto lo hace mucho más efectivo frente ataques de GPUs y ha gozado de gran popularidad tanto para la gestión de contraseñas como en el mundo de las *criptomonedas* alternativas a *bitcoin* (que utiliza SHA-2) como *litecoin* o *dogecoin*. Esta es la **opción recomendada para gestionar contraseñas en la actualidad.**

[2] Es importante destacar que dentro de las funciones PBKDF, hay una función concreta que se llama PBKDF2. Es como si hubiera un cifrador que se llamara CIFRADOR2.

6 SEGURIDAD A NIVEL DE TRANSPORTE

6.1 SSL Y TLS

Ambos, SSL (Secure Sockets Layer) y TLS (Transport Layer Security) son en realidad el mismo protocolo; siendo TLS la versión moderna de SSL. SSL surge como iniciativa de Netscape (actualmente Mozilla) y pasa a ser adoptado de forma estándar por toda la comunidad a partir de la versión 3.

TLS es un protocolo orientado a proporcionar seguridad sobre conexiones TCP/IP por lo que, a pesar de ser asociado a las webs seguras, puede servir para muchas otras aplicaciones. En concreto, TLS nos ofrece las siguientes ventajas de seguridad:

- *Autenticidad*. El protocolo hace uso de certificados lo que permite comprobar la identidad del servidor.
- *Confidencialidad*. Se intercambia una clave mediante criptografía de clave pública y luego se crea un túnel cifrado mediante criptografía simétrica ofreciendo, así, seguridad y eficiencia.
- *Integridad*. Se incorpora un resumen o hash criptográfico de cada paquete de datos, garantizando que no ha sido modificado en tránsito.

Además, TLS proporciona compresión de datos. El protocolo TLS es la base de gran parte de la seguridad de la que gozamos hoy en día en Internet y es **altamente recomendable**.

6.2 HTTPS

El protocolo HTTPS significa HTTP sobre SSL (o TLS en la actualidad). El protocolo HTTP es el protocolo de la web estándar, por lo que HTTPS es el estándar sobre el que se fundamentan las webs seguras que tenemos hoy en día. Básicamente, consiste en utilizar TLS para enviar los datos de HTTP.

HTTPS puede ser útil no sólo en aquellas situaciones en las que las webs tengan información privada o sensible, *también permite proporcionar privacidad* al ser una conexión cifrada entre el navegador y el servidor web y evitando que nuestro operador de red (u otras terceras partes) puedan descubrir que páginas visitamos.

Resulta interesante destacar que el nuevo protocolo HTTP2 incorpora HTTPS de forma obligatoria en muchos casos y que esto va a redundar en una mejora significativa de la seguridad de la web en general.

6.3 SSH

El protocolo Secure Shell (SSH) se diseñó originalmente para permitir la conexión a consolas y terminales remotos de forma segura, pero tiene funcionalidades avanzadas similares a TLS que lo hacen muy versátil.

Se puede utilizar SSH para conexiones de otros protocolos como transferencia de ficheros (SFTP), email, etc. Además, SSH permite la redirección de puertos, posibilitando añadir seguridad a servicios que, en principio, no disponen de soporte cifrado.

Al igual que TLS, SSH garantiza la *autenticidad*, la *confidencialidad* y la *integridad* de los datos y es muy común su uso en la gestión de máquinas virtuales remotas y la computación en la nube.

7 CRIPTOGRAFÍA ASIMÉTRICA (O DE CLAVE PÚBLICA)

7.1 CONCEPTOS BÁSICOS

La criptografía de clave pública o asimétrica se diferencia de la tradicional criptografía simétrica (cifrado en bloque o en flujo) en que *utiliza dos claves asociadas*, una para cifrar y otra distinta para descifrar. Esta característica permite funcionalidades adicionales que son imposibles, o muy difíciles, en la criptografía simétrica: la *distribución de claves* mediante un canal inseguro y la *firma digital*.

En la criptografía asimétrica, cada usuario posee un par de claves: una clave pública y otra privada. Como su nombre indica, la clave pública se puede transmitir e incorporar en repositorios públicos, pero la privada ha de permanecer en secreto. Si ciframos con la clave pública, podremos descifrar con la privada y al revés, si ciframos con la clave privada, podremos descifrar con la pública.

Las tres operaciones básicas que permite un criptosistema de clave pública son:

Cifrado. Para enviar un mensaje cifrado de Alicia a Bernardo (de A a B), Alicia cifra el mensaje con la clave pública de Bernardo; luego, Bernardo puede descifrar el mensaje haciendo uso de su clave privada. Como sólo Bernardo tiene su clave privada, sólo Bernardo puede descifrar el mensaje. Manolo, que es un espía, sólo puede ver el mensaje cifrado y la clave pública de Bernardo, pero no tiene la clave privada para descifrar el mensaje. **Para cifrar un mensaje, se cifra con la clave pública del destino y se descifra con la clave privada del destino.**

Firma. Para firmar un mensaje, Alicia cifra el mensaje con su clave privada y lo envía a Bernardo; luego, Bernardo puede comprobar que el mensaje proviene de Alicia descifrando el mensaje con la clave pública de Alicia y, comprobando que coincide con el mensaje original. Como sólo Alicia tiene su clave privada, ella es la única capaz de firmar un mensaje que se descifre correctamente con su clave pública y, por lo tanto, queda garantizada su autoría. El espía Manolo también puede comprobar la firma de Alicia, puesto que tiene su clave pública, pero no puede firmar ningún mensaje en nombre de Alicia. Es importante destacar que la firma digital garantiza la autenticidad o identidad de los datos, pero no la confidencialidad; si queremos que el mensaje sea privado, debemos cifrar tras firmar el mensaje. **Para firmar un mensaje, se cifra con la clave privada del firmante y se comprueba dicha firma descifrando con la clave pública del firmante.**

Intercambio de clave. En muchos casos, nos basta con intercambiar una clave de forma segura entre Alicia y Bernardo para después utilizar un cifrador convencional (AES o Salsa20, por ejemplo) para cifrar los datos; esto es lo que realizan protocolos como TLS o SSH. En clave pública, Alicia puede intercambiar una clave con Bernardo simplemente generándola de forma aleatoria y cifrándola con la clave pública de Bernardo. No obstante, hay algoritmos de clave pública que no permiten ni cifrar ni firmar, pero sí **compartir de forma segura un mismo número en ambos extremos a partir del cual se puede derivar una clave a utilizar para criptografía simétrica**.

Los conceptos importantes a tener en cuenta son:

Rendimiento. La criptografía asimétrica es demasiado lenta para el cifrado de datos en general, por lo que recurrimos a un intercambio de clave y al cifrado mediante criptografía simétrica del resto de los datos en la mayoría de los casos. Lo mismo ocurre con la firma digital, que es demasiado lenta para firmar mensajes muy largos; aquí recurrimos a calcular el resumen del mensaje mediante una función hash y firmar el resumen en lugar del mensaje completo por motivos de eficiencia.

Seguridad. La criptografía asimétrica no es más segura que la criptografía simétrica, depende del tamaño de la clave y del coste computacional asociado a probar todas las claves posibles. Hay que utilizar tamaños de clave tanto para la criptografía asimétrica como la simétrica que sean equivalentes en cuanto a seguridad.

Distribución de claves. La criptografía de clave pública requiere de ciertas infraestructuras y protocolos para la gestión de las claves: repositorios de claves públicas, terceras partes de confianza que garanticen la autenticidad de las claves, etc. Sin estas medidas, el espía Manolo puede realizar lo que se conoce como ataque del hombre en el medio (man-in-the-middle), haciéndose pasar por Bernardo frente a Alicia y como a Alicia frente a Bernardo y filtrando toda la información que, en teoría, debería ser privada entre ambos.

Se recomienda consultar las transparencias de clase acerca de este tema, así como los materiales de ampliación que en ellas se indican para profundizar más, ver esquemas, etc.

A continuación, describimos los algoritmos principales que existen para realizar criptografía asimétrica o de clave pública.

7.2 RSA

Se propone en 1978 y es el primer algoritmo de clave pública completo en salir al mercado. Proviene de Rivest (creador de RC4, MD5, etc.), Shamir y Adleman. Su seguridad reside en el problema de la factorización de un número en factores primos cuando este número es muy grande. El algoritmo RSA ha estado patentado hasta el año 2000, por lo que era necesario pagar para su utilización.

Permite tanto el cifrado (y por tanto el intercambio de clave) como la firma digital y está considerado como seguro en la actualidad. Si bien, dejaría de ser seguro (al igual que los otros criptosistemas de clave pública) si tuviéramos computadores cuánticos[3] viables.

RSA es extremadamente popular y se utiliza en multitud de aplicaciones. S**e recomienda su uso con una clave de 3072 bits** (equivalente a 128 bits en criptografía simétrica) como mínimo, si bien pueden existir alternativas más eficientes para ciertas aplicaciones.

[3] Los computadores cuánticos podrían realizar los cálculos asociados a romper los algoritmos de clave pública de forma mucho más eficiente que los computadores convencionales. Esto ha dado lugar al concepto de *criptografía postcuántica*, en donde se intenta diseñar algoritmos capaces de resistir un ataque mediante un computador cuántico.

7.3 DIFFIE-HELLMAN

Diffie-Hellman es un algoritmo que sólo permite el intercambio de claves, no permite ni el cifrado ni la firma digital. Se basa en la dificultad de calcular el logaritmo discreto (problema matemático similar al de RSA) y permite llegar al mismo número en ambos extremos.

Este número que se comparte de forma segura, puede ser utilizado para derivar una clave que se utilizará en criptografía simétrica convencional y depende de las claves públicas y privadas de cada extremo. De esta forma, si queremos compartir una nueva clave, tendremos que generar nuevas claves públicas y privadas.

A pesar de ser exclusivamente un intercambio de clave, ha gozado de mucha popularidad ya que es más rápido que RSA en este cometido y no ha estado patentado, por lo que no era necesario pagar por su uso. Se utiliza en muchos protocolos actuales como TLS o el de cifrado de Whatsapp.

Requiere un tamaño de clave similar a RSA para ser seguro (3072 bits).

7.4 CURVAS ELÍPTICAS

La criptografía sobre curvas elípticas es, esencialmente, una mejora sobre Diffie-Hellman utilizando una aritmética distinta que permite claves mucho más pequeñas y rendimientos mucho mejores.

A esta técnica se le denomina ECDH (Elliptic Curve Diffie-Hellman) y es la forma que se utiliza tanto en TLS como Whatsapp. **Se recomienda el uso de ECDH con un tamaño de clave de 256 bits como mínimo dado su mayor rendimiento** respecto a RSA o Diffie-Hellman convencional.

7.5 ELGAMAL

El algoritmo de Taher Elgamal, es una variante sobre Diffie Hellman que permite el cifrado y la firma digital. Se ha utilizado en muchos protocolos (como el estándar DSS de firma digital del NIST o el DSA) para implementar firma digital sin tener que licenciar la patente de RSA. Elgamal fue también uno de los creadores de SSL en Netscape.