Instituto Politécnico Nacional

Escuela Superior de Cómputo

Trabajo terminal

Dra. Sandra Díaz Santiago

Generación de *tokens* para proteger los datos de tarjetas bancarias

Trabajo terminal No. 2017-B008

Daniel Ayala Zamorano
Laura Natalia Borbolla Palacios
Ricardo Quezada Figueroa

Enero de 2018

Contenido

Simbología

A continuación se describe la simbología que se utilizará a lo largo de este documento.

Tabla 1: Simbología

Símbolo	Descripción
K	Llave
pk	Llave pública
sk	Llave privada o secreta
k_i	I*é*-sima subllave
E	Operación de cifrado
E_K	Operación de cifrado utilizando la llave K
D	Operación de descifrado
D_K	Operación de descifrado utilizando la llave K
h	Función hash
h_k	Función hash que utiliza una llave k para calcular el valor
M	Mensaje en claro
C	Mensaje cifrado
$\mathbb{Z}_n$	Conjunto de los números enteros módulo n
$\{0,1\}^r$	Cadena de bits de longitud r
$\{0,1\}^*$	Cadena de bits de longitud arbitraria
mód	Operación módulo
mcd	Máximo común divisor
$\oplus$	Operación XOR
φ	Función ϕ de Euler

Es menester aclarar que un mensaje no consiste solo en letras y números; el *mensaje* se refiere al conjunto de datos que van a ser cifrados o descifrados.

Capítulo 1

Introducción

1.1. Justificación

1.2. Objetivos

Capítulo 2

Antecedentes

2.1. Introducción a la criptografía

La información presentada a continuación puede consultarse con más profundidad en las siguientes referencias [1], [2].

La palabra criptografía proviene de las etimologías griegas *Kriptos* (ocultar) y *Graphos* (escritura), y es definida por la Real Academia Española (RAE) como «el arte de escribir con clave secreta o de un modo enigmático». De manera más formal se puede definir a la criptografía como la ciencia encargada de estudiar y diseñar por medio de técnicas matemáticas, métodos y modelos capaces de resolver problemas en la seguridad de la información, como la confidencialidad de esta, su integridad y la autenticación de su origen.

2.1.1. Objetivos de la criptografía

La criptografía tiene como finalidad proveer los siguientes cuatro servicios:

1. **Confidencialidad**

   Es el servicio encargado de mantener legible la información solo a aquellos que estén autorizados a visualizarla.

2. **Integridad**

   Este servicio se encarga de evitar la alteración de la información de forma no autorizada, esto incluye la inserción, sustitución y eliminación de los datos.

3. **Autenticación**

   Este servicio se refiere a la identificación tanto de las personas que establecen una comunicación, garantizando que cada una es quien dice ser; como del origen de la información que se maneja, garantizando la veracidad de la hora y fecha de origen, el contenido, tiempos de envío, entre otros.

4. **No repudio**

   Es el servicio que evita que el autor de la información o de alguna acción determinada, pueda negar su validez, ayudando así a prevenir situaciones de disputa.

2.1.2. Criptoanálisis y ataques

La criptografía forma parte de una ciencia más general llamada criptología, la cual tiene otras ramas de estudio, como es el criptoanálisis que es la ciencia encargada de estudiar los posibles ataques a sistemas criptográficos, que son capaces de contrariar sus servicios ofrecidos. Entre los principales objetivos del criptoanálisis están interceptar la información que circula en un canal de comunicación, alterar dicha infor-

mación y suplantar la identidad, rompiendo con los servicios de confidencialidad, integridad y autenticación respectivamente.

Los ataques que se realizan a sistemas criptográficos dependen de la cantidad de recursos o conocimientos con los que cuenta el adversario que realiza dicho ataque, dando como resultado la siguiente clasificación.

1. **Ataque con sólo texto cifrado**

   En este ataque el adversario solamente es capaz de obtener la información cifrada, y tratará de conocer su contenido en claro a partir de ella. Esta forma de atacar es la más básica, y todos los métodos criptográficos deben poder soportarla.

2. **Ataque con texto en claro conocido**

   Esta clase de ataques ocurren cuando el adversario puede obtener pares de información cifrada y su correspondiente información en claro, y por medio de su estudio, trata de descifrar otra información cifrada para la cual no conoce su contenido.

3. **Ataque con texto en claro elegido**

   Este ataque es muy parecido al anterior, con la diferencia de que en este el adversario es capaz de obtener los pares de información cifrada y en claro con el contenido que desee.

4. **Ataque con texto en claro conocido adaptativo**

   En este ataque el adversario es capaz de obtener los pares de información cifrada y en claro con el contenido que desee, además tiene amplio acceso o puede usar de forma repetitiva el mecanismo de cifrado.

5. **Ataque con texto en claro elegido adaptativo**

   En este caso el adversario puede elegir información cifrada y conocer su contenido, dado que tiene acceso a los mecanismos de descifrado.

2.1.3. Clasificación de la criptografía

La criptografía puede clasificarse de forma histórica en dos categorías, la criptografía clásica y la criptografía moderna. La criptografía clásica es aquella que se utilizó desde la antigüedad, teniéndose registro de su uso desde hace más 4000 años por los egipcios, hasta la mitad del siglo XX. En esta los métodos utilizados para cifrar eran variados, pero en su mayoría usaban la transposición y la sustitución, además de que la mayoría se mantenían en secreto. Mientras que la criptografía moderna es la que se inició después la publicación de la *Teoría de la información* por Claude Elwood Shannon[3], dado que esta sentó las bases matemáticas para la criptología en general.

Una manera de clasificar es de acuerdo a las técnicas y métodos empleados para cifrar la información, esta clasificación se puede observar en la figura 2.1.

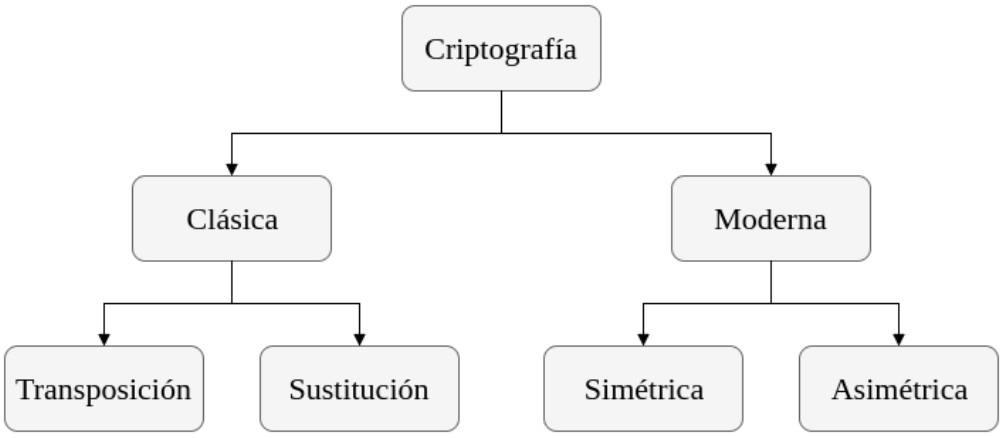


Figura 2.1: Clasificación de la criptografía.

Adentrándose en la clasificación de la criptografía clásica, se tienen los cifrados por transposición, los cuales se basan en técnicas de permutación de forma que los caracteres de la información en claro se reordenen mediante algoritmos específicos, y los cifrados por sustitución, que utilizan técnicas de modificación de los caracteres por otros correspondientes a un alfabeto específico para el cifrado.

En cuanto a la criptografía moderna, esta tiene dos vertientes: la criptografía simétrica o de llave secreta, y la asimétrica o de llave pública. Hablando de la primer vertiente, se puede decir que es aquella que utiliza un modelo matemático para cifrar y descifrar un mensaje utilizando únicamente una llave que permanece secreta.

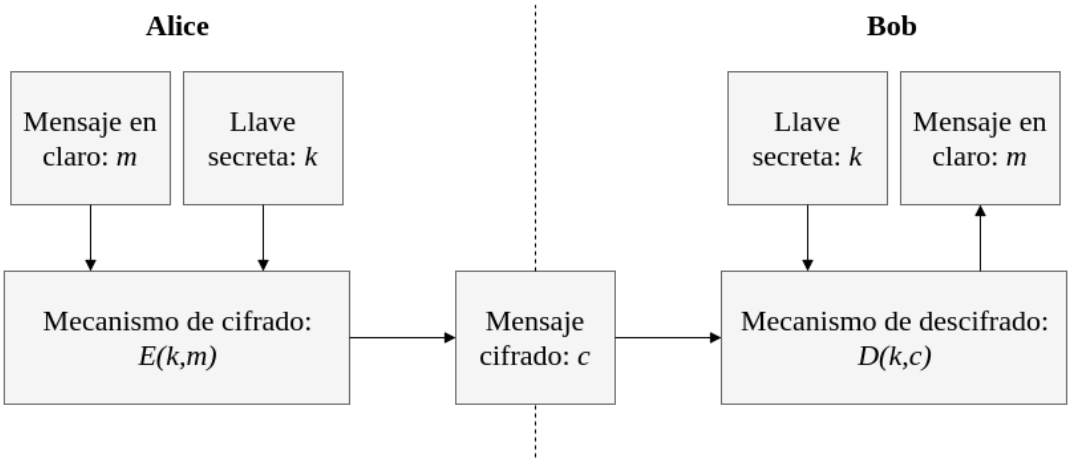


Figura 2.2: Canal de comunicación con criptografía simétrica.

En la figura 2.2 se puede observar el proceso para establecer una comunicación segura por medio de la criptografía simétrica. Primero, tanto Alice como Bob deben de establecer una llave única y compartida k, para que después, Alice, actuando como el emisor, cifre un mensaje m usando la llave k por medio del algoritmo de cifrado $E(k, m)$ para obtener el mensaje cifrado c y enviárselo a Bob. Posteriormente Bob, como receptor, se encarga de descifrar c con ayuda de la llave k por medio del algoritmo de descifrado $D(k, c)$ para obtener el mensaje original m.

Gran parte de los algoritmos de cifrado que caen en este tipo de criptografía están basados en las redes Feistel, que son un método de cifrado propuesto por el criptógrafo Horst Feistel, mismo que desarrolló el *Data Encryption Standard* (DES) a principios de la década de los 70, que fue el cifrado usado por el gobierno estadounidense hasta 2002, año que el *Advanced Encryption Standard* (AES) lo sustituyó.

Ahora, adentrándose en la criptografía asimétrica, se tiene que su idea principal es el uso de 2 llaves distintas para cada persona, una llave pública para cifrar, que esté disponible para cualquier otra persona, y una llave privada para descifrar, que se mantiene disponible solo para su propietario.

El proceso para establecer una comunicación segura por medio de este tipo de criptografía es el siguiente: primero, Alice nuevamente como el emisor, cifra un mensaje m con la llave pública de Bob pk, y usa el algoritmo de cifrado $E(pk, m)$ para obtener c y enviarlo. Después Bob como receptor, se encarga de descifrar c por medio del algoritmo de descifrado $D(sk, c)$ haciendo uso de su llave privada sk. Este proceso se refleja gráficamente el la figura 2.3.

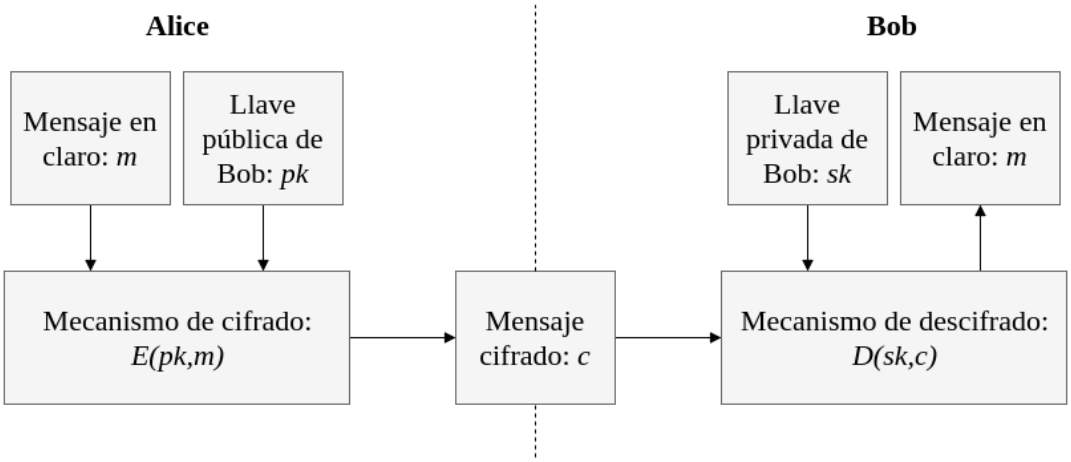


Figura 2.3: Canal de comunicación con criptografía asimétrica.

Entre los uso que se le da a esta criptografía está el mantener la distribución de llaves privadas segura y establecer métodos que garanticen la autenticación y el no repudio; por ejemplo, en las firmas y certificados digitales.

El principal precursor de la criptografía asimétrica fue el método de intercambio de llaves de Diffie-Hellman, desarrollado y publicado por Whitfield Diffie y Martin Hellman, en 1976 en el artículo *New Directions in Cryptography*[4], siendo la primera forma práctica para poder establecer una llave secreta compartida entre dos partes sin contacto previo por medio de un canal público para intercambiar mensajes.

Otro precursor fue el sistema criptográfico *Ron Rivest, Adi Shamir, Leonard Adleman* (RSA) (nombre obtenido por las siglas de los apellidos de sus desarrolladores: Ron Rivest, Adi Shamir y Leonard Adleman), publicado en 1978[5] y que fue el primer sistema criptográfico capaz de servir tanto para cifrar mensajes, como para la implementación de firmas digitales. A pesar de que sus orígenes son de ya hace casi cuatro décadas, este sistema aún es uno de los más ampliamente usados.

Entre los motivos del éxito de RSA está que su funcionamiento se basa en la teoría elemental de números, ya que usa propiedades descritas en esta teoría; y en su seguridad, ya que se basa en la incapacidad de poder factorizar números grandes de forma eficiente.

El algoritmo de RSA consta de 3 partes, la generación de llaves, el cifrado y el descifrado. El proceso para poder generar un par de llaves (pública y privada) con RSA se muestra en el pseudocódigo 2.1.

```
entrada: ninguna.
salida: llave pública (n,e) y privada (n,d).
inicio
  Elegir de forma aleatoria 2 números primos p y q, que sean de gran
  magnitud y de una longitud parecida.
  Calcular n = p · q
  Calcular φ(n) = (p − 1)(q − 1).
  Elegir un exponente de cifrado e tal que e < φ(n) y mcd(e, φ(n)) = 1.
  Encontrar el exponente de descifrado d tal que e·d mód φ(n) = 1.
fin
```

Pseudocódigo 2.1: Proceso de generación de llaves de RSA.

Las funciones de cifrado y descifrado, están definidas en las ecuaciones 2.1 y 2.2 respectivamente, y se aplican números enteros o bloques de bits, siendo funciones biyectivas e inversas entre sí.

$$E : \mathbb{Z}_n \longrightarrow \mathbb{Z}_n, x \longmapsto x^e \tag{2.1}$$

$$D : \mathbb{Z}_n \longrightarrow \mathbb{Z}_n, x \longmapsto x^d \tag{2.2}$$

2.2. Cifrados por bloques

La información presentada a continuación puede consultarse con más profundidad en las siguientes referencias [1], [2].

Los cifrados por bloque son esquemas de cifrado que, como bien lo explica su nombre, operan mediante bloques de datos. Normalmente los bloques tienen una longitud de 64 o de 128 bits, mientras que las llaves pueden ser de 56, 128, 192 o 256 bits.

En muchos sistemas criptográficos, los cifrados por bloque simétricos son elementos importantes, pues su versatilidad permite construir con ellos generadores de números pseudoaleatorios, cifrados de flujo MACs y funciones hash. Sirven también como componentes centrales en técnicas de autenticación de mensajes, mecanismos de integridad de datos, protocolos de autenticación de entidad y esquemas de firma electrónica que usan llaves simétricas.

Los cifrados por bloque están limitados en la práctica por varios factores, tales como el límite de memoria, la velocidad requerida o restricciones impuestas por el hardware o el software en el que se implementan. Normalmente, se debe escoger entre eficiencia y seguridad

Idealmente, al cifrar por bloques, cada bit del bloque cifrado depende de todos los bits de la llave y del texto en claro; no debería existir una relación estadística evidente entre el texto en claro y el texto cifrado; el alterar tan solo un bit en el texto en claro o en la llave debería alterar cada uno de los bits del texto cifrado con una probabilidad de $\frac{1}{2}$; y alterar un bit del texto cifrado debería provocar resultados impredecibles al recuperar el texto en claro.

2.2.1. Definición

$$\begin{aligned} E : \{0,1\}^r \times \{0,1\}^n &\longrightarrow \{0,1\}^n \\ (k, m) &\longmapsto E(k, m) \end{aligned} \tag{2.3}$$

Utilizando una llave secreta k de longitud binaria r el algoritmo de cifrado E cifra bloques en claro m de una longitud binaria fija n y da como resultado bloques cifrados $c = E(k, m)$ cuya longitud también es n. n es el tamaño de bloque del cifrado. El espacio de llave está dado por $K = \{0,1\}^r$, para cada llave existe una función $D_k(c)$ que permite tomar un bloque cifrado c y regresarlo a su forma original m.

Generalmente, los cifrados por bloque procesan el texto claro en bloques relativamente grandes ($n \geq 64$), contrastando con los cifradores de flujo, que toman bit por bit. Cuando la longitud del mensaje en claro excede el tamaño de bloque, se utilizan los modos de operación.

Los parámetros más importantes de los cifrados por bloque son los siguientes:

- Tamaño de bloque
- Tamaño de llave

2.2.2. Criterios para evaluar los cifrados por bloque

A continuación se listan algunos de los criterios que pueden ser tomados en cuenta para evaluar estos cifrados:

1. **Nivel de seguridad.** La confianza que se le tiene a un cifrado va creciendo con el tiempo, pues va siendo analizado y sometido a pruebas.
2. **Tamaño de llave.** La entropía del espacio de la llave define un límite superior en la seguridad del cifrado al tomar en cuenta la búsqueda exhaustiva. Sin embargo, hay que tener cuidado con su tamaño, pues también aumentan los costos de generación, transmisión, almacenamiento, etcétera.
3. **Tamaño de bloque.** Impacta la seguridad, pues entre más grandes, mejor; sin embargo, tiene repercusiones en el costo de la implementación, además de que puede afectar el rendimiento del cifrado.
4. **Expansión de datos.** Es extremadamente deseable que los datos cifrados no aumenten su tamaño respecto a los datos en claro.
5. **Propagación de error.** Descifrar datos que contienen errores de bit puede llevar a recuperar incorrectamente el texto en claro, además de propagar errores en los bloques pendientes por descifrar. Normalmente, el tamaño de bloque afecta el error de propagación.

   A continuación se listan algunos algoritmos de cifrado por bloques.

2.2.3. Cifrado Feistel

Consiste en un cifrado iterativo que mapea bloques de texto en claro de tamaño $2t$bits (separados en bloques L_0, R_0 de tamaño t) a un texto cifrado R_r, L_r mediante un proceso de r rondas.

```
inicio
para_todo i desde 1 hasta r:
  L_i = R_{i-1}
  R_i = L_{i-1} ⊕ f(R_{i-1}, K_i)
fin
fin
```

Pseudocódigo 2.2: Feistel, cifrado.

Donde cada subllave K_i se obtiene de la llave K.

Normalmente el número de rondas r es mayor o igual a tres y par. Además, casi siempre intercambia el orden de los bloques de salida al revés en la última ronda: (R_r, L_r) en vez de (L_r, R_r).

El descifrado se realiza utilizando el mismo proceso de cifrado pero con las llaves en el orden inverso (comenzando con K_r hasta K_1).

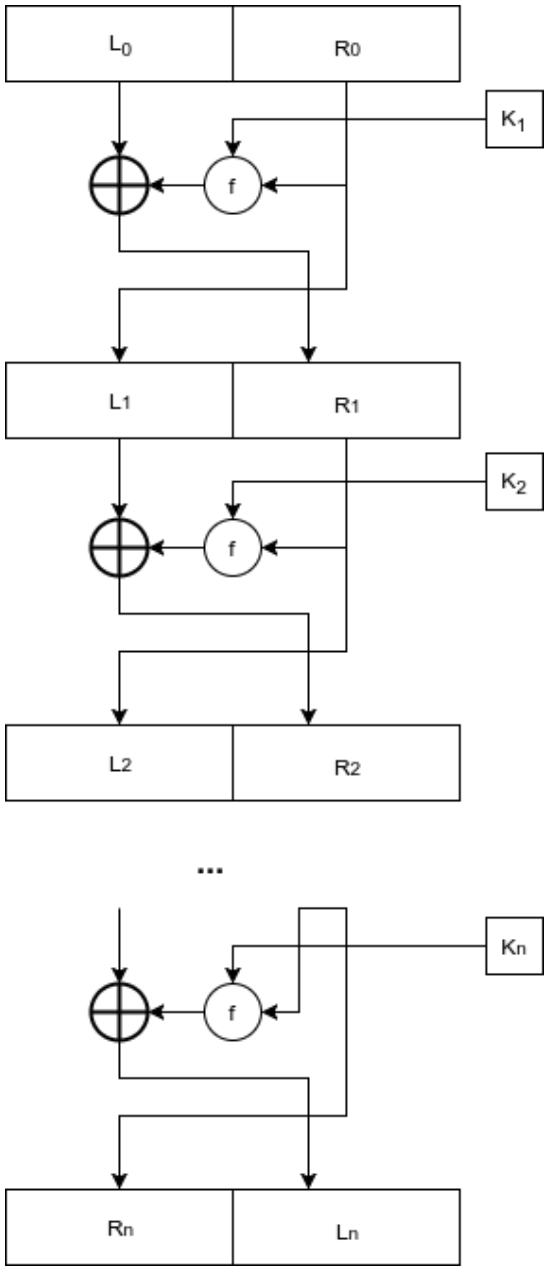


Figura 2.4: Diagrama genérico de una red Feistel.

2.2.4. Data Encryption Standard (DES)

Este es, probablemente, el cifrado simétrico por bloques más conocido; ya que en la década de los 70 estableció un precedente al ser el primer algoritmo a nivel comercial que publicó abiertamente sus especificaciones y detalles de implementación. Se encuentra definido en el estándar americano *Federal Information Processing Standard* (FIPS) 46-2.

DES es un cifrado Feistel que procesa bloques de $n = 64$ bits y produce bloques cifrados de la misma longitud. Aunque la llave es de 64 bits, 8 son de paridad, por lo que el tamaño *efectivo* de la llave es de 56 bits. Las 2^{56} llaves implementan, máximo, 2^{56} de las $2^{64}!$ posibles biyecciones en bloques de 64 bits.

Con la llave K se generan 16 subllaves K_i de 48 bits; una para cada ronda. En cada ronda se utilizan 8 *cajas-s* (mapeos de sustitución de 6 a 4 bits). La entrada de 64 bits es dividida por la mitad en L_0 y R_0. Cada ronda i va tomando las entradas L_{i-1} y R_{i-1} de la ronda anterior y produce salidas de 32 bits L_i y R_i mientras $1 \leq i \leq 16$ de la siguiente manera:

$$\begin{aligned} L_i &= R_{i-1} \\ R_i &= L_{i-1} \oplus f(R_{i-1}, K_i) \\ &\quad donde \quad f(R_{i-1}, K_i) = P(S(E(R_{i-1}) \oplus K_i)) \end{aligned} \tag{2.4}$$

E se encarga de expandir R_{i-1} de 32 bits a 48, P es una permutación de 32 bits y S son las cajas-s.

entrada: 64 bits de texto en claro $M = m_1 \dots m_{64}$;
llave de 64 bits $K = k_1 \dots k_{64}$.
salida: bloque de texto cifrado de 64 bits $C = c_1 \dots c_{64}$.
inicio
Calcular 16 subllaves K_i de 48 bits partiendo de K.
Obtener (L_0, R_0) de la tabla de permutaciones iniciales $IP(m_1 m_2 \dots m_{64})$
para_todo i desde 1 hasta 16:
L_i = R_{i-1}
Obtener $f(R_{i-1}, K_i)$:
a) Expandir $R_{i-1} = r_1 r_2 \dots r_{32}$ de 32 a 48 bits
usando E: $T \leftarrow E(R_{i-1})$.
b) $T' \leftarrow T \oplus K_i$. Donde T' es representado
como ocho cadenas de 6 bits cada una $(B_1, \dots, B_8)$.
c) $T'' \leftarrow (S_1(B_1), S_2(B_2), \dots S_8(B_8))$
d) $T''' \leftarrow P(T'')$
R_i = L_{i-1} \oplus f(R_{i-1}, K_i)
fin
$b_1 b_2 \dots b_{64} \leftarrow (R_{16}, L_{16})$.
$C \leftarrow IP^{-1}(b_1 b_2 \dots b_{64})$
fin

Pseudocódigo 2.3: DES, cifrado.

El descifrado DES consiste en el mismo algoritmo de cifrado, con la misma llave K, pero utilizando las subllaves en orden inverso: $K_{16}, K_{15}, \ldots, K_1$.

Llaves débiles

Tomando en cuenta las siguientes definiciones

- Llave débil: una llave K tal que $E_K(E_K(M)) = M$ para toda x; en otras palabras, una llave débil permite que, al cifrar dos veces con la misma llave, se obtenga de nuevo el mensaje en claro.
- Llaves semidébiles: se tiene un par de llaves K_1, K_2 tal que $E_{K_1}(E_{K_2}(x)) = x$.

DES tiene cuatro llaves débiles y seis pares de llaves semidébiles. Las cuatro llaves débiles generan subllaves K_i iguales y, debido a que DES es un cifrado Feistel, el cifrado es autorreversible. O sea que al final se obtiene de nuevo el texto en claro, pues cifrar dos veces con la misma llave regresa la entrada original. Respecto a los pares semidébiles, el cifrado con una de las llaves del par es equivalente al descifrado con la otra (o viceversa).

2.2.5. Advanced Encryption Standard (AES)

Dado que el tamaño de bloque y la longitud de la llave de DES se volvieron muy pequeños para resistir los embates del progreso de la tecnología, el *National Institute of Standards and Technology* (NIST) comenzó la búsqueda de un nuevo cifrado estándar en 1997; este cifrado debía tener un tamaño de bloque de, al menos, 128 bits y soportar tres tamaños de llave: 128, 192 y 256 bits.

Después de pasar por un proceso de selección, la propuesta Rijndael fue seleccionada. Se le hicieron algunas modificaciones, pues Rijndael soporta combinaciones de llaves y bloques de longitud 128, 169, 192, 224 y 256; mientras que AES tiene fijo el tamaño de bloque y solo utiliza los tres tamaños de llave mencionados anteriormente. Dependiendo del tamaño de la llave, se tiene el número de rondas: 10 para las de 128 bits, 12 para las de 192 y 14 para las de 256.

El cifrado requiere de una matriz de 4×4 denominada matriz de estado.

```
entrada:    128 bits de texto en claro M; llave de n bits K.
salida:     bloque de texto cifrado de 64 bits C = c1...c64.
inicio
  Obtener las subllaves de 128 bits necesarias: una para cada ronda y una extra.
  Iniciar matriz de estado con el bloque en claro.
  Realizar AddRoundKey(matriz_estado, k0)
  para_todo i desde 1 hasta num_rondas - 1:
    SubBytes(matriz_estado)
```

```
    ShiftRows(matriz_estado)
    MixColumns(matriz_estado)
    AddRoundKey(matriz_estado, k_i)
  fin
  SubBytes(matriz_estado)
  ShiftRows(matriz_estado)
  MixColumns(matriz_estado)
  AddRoundKey(matriz_estado, k_num_rondas)
  regresa matriz_estado
fin
```

Pseudocódigo 2.4: AES, cifrado.

Como todos los pasos realizados en las rondas son invertibles, el proceso de descifrado consiste en aplicar las funciones inversas a $SubBytes$, $ShiftRows$, $MixColumns$ y $AddRoundKey$ en el orden opuesto. Tanto el algoritmo como sus pasos están pensados con bytes. En el algoritmo Rijndael los bytes son considerados como elementos del campo finito $\mathbb{F}_{2^8}$ con 2^8 elementos; $\mathbb{F}_{2^8}$ es construido como una extensión del campo $\mathbb{F}_2$ con 2 elementos mediante el uso del polinomio irreducible $X^8 + X^4 + X^3 + X + 1$. Por lo tanto, las operaciones que se hagan a continuación de adición y el producto entre bytes significa sumarlos y multiplicarlos como elementos del campo $\mathbb{F}_{2^8}$.

SubBytes

Esta es la única transformación no lineal de Rijndael. Sustituye los bytes de la matriz de estado byte a byte al aplicar la función S_{RD} a cada elemento de la matriz. La función S_{RD} es también conocida como Caja-S y no depende de la llave. La misma caja es utilizada para los bytes en todas las posiciones.

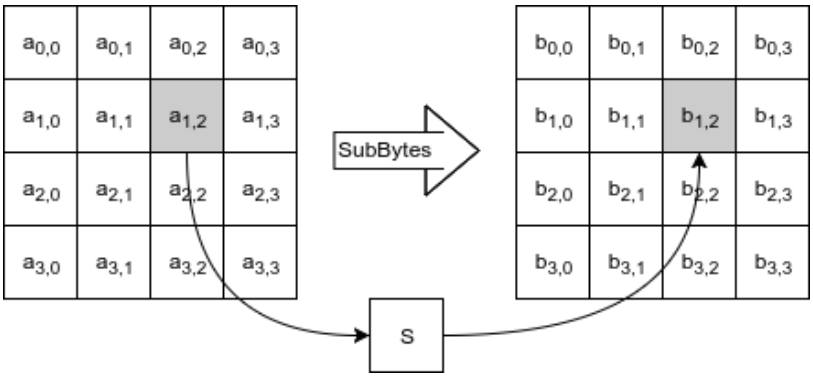


Figura 2.5: Diagrama de la operación $SubBytes$.

ShiftRows

Esta transformación hace un corrimiento cíclico hacia la izquierda de las filas de la matriz de estado. Los desplazamientos son distintos para cada fila y dependen de la longitud del bloque (N_b).

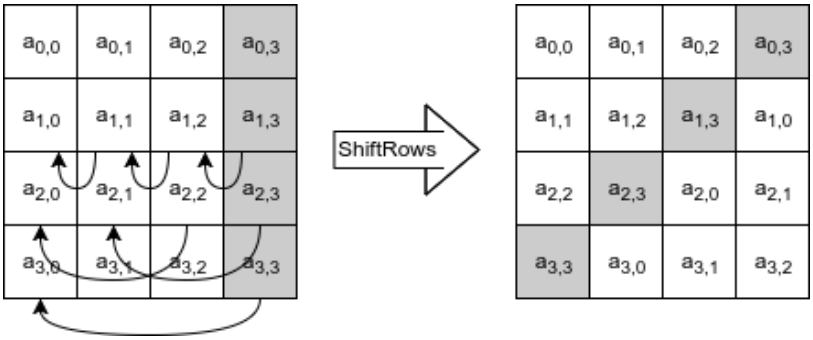


Figura 2.6: Diagrama de la operación $ShiftRows$.

MixColumns

Esta transformación opera en cada columna de la matriz de estado independientemente. Se considera una columna $a = (a_0, a_1, a_2, a_3)$ como el polinomio $a(X) = a_3X^3 + a_2X^2 + a_1X + a_0$. Entonces este paso transforma una columna a al multiplicarla con el siguiente polinomio fijo:

$$c(X) = 03X^3 + 01X^2 + 01X + 02 \tag{2.5}$$

y se toma el residuo del producto módulo $X^4 + 1$:

$$a(X) \mapsto a(X) \cdot c(X) \quad \text{mód } (X^4 + 1) \tag{2.6}$$

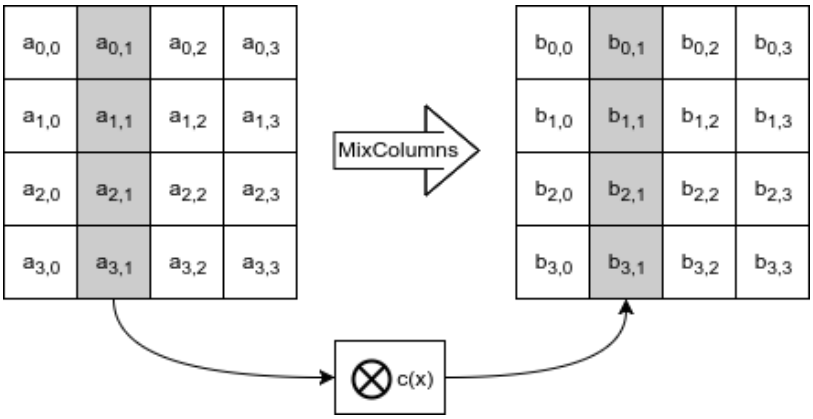


Figura 2.7: Diagrama de la operación $MixColumns$.

AddRoundKey

Esta es la única operación que depende de la llave secreta k. Añade una llave de ronda para intervenir en el resultado de la matriz de estado. Las llaves de ronda son derivadas de la llave secreta k al aplicar el algoritmo de generación de llaves. Las llaves de ronda tienen la misma longitud que los bloques. Esta operación es simplemente una operación XOR bit a bit de la matriz de estado con la llave de ronda en turno. Para obtener el nuevo valor de la matriz de estado se realiza lo siguiente:

$$(matriz\_estado, k_i) \mapsto matriz\_estado \oplus k_i \tag{2.7}$$

Como se tiene una matriz_estado, la llave de ronda (k_i) también es representada como una matriz de bytes con 4 columnas y N_b columnas. Cada una de las N_b palabras de la llave de ronda corresponde a una columna. Entonces se realiza la operación XOR bit a bit sobre las entradas correspondientes de la matriz de estado y la matriz de la llave de ronda.

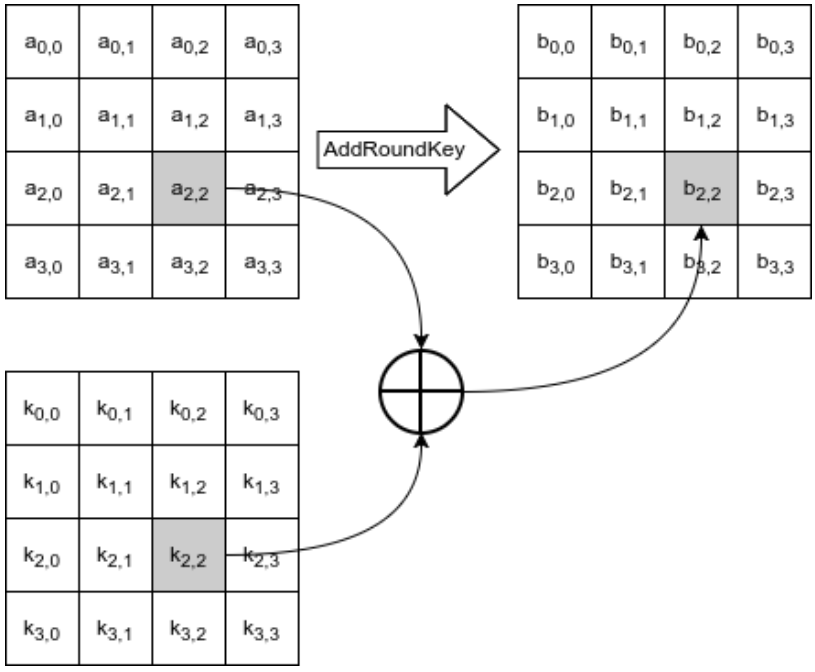


Figura 2.8: Diagrama de la operación *AddRoundKey*.

Esta operación, claro está, es invertible: basta con aplicar la misma operación con la misma llave para revertir el efecto.

2.2.6. Fast Data Encipherment Algorithm (FEAL)

Es una familia de algoritmos que ha tenido una participación crítica en el desarrollo y refinamiento de varias técnicas del criptoanálisis, tales como el criptoanálisis lineal y diferencial. *Fast Data Encipherment Algorithm* (FEAL)-N mapea bloques de texto en claro de 64 bits a bloques de 64 bits de texto cifrado mediante una llave secreta de 64 bits. Es un cifrado Feistel de $n-$rondas parecido a DES, pero con una función f más simple.

FEAL fue diseñado para ser veloz y simple, especialmente para microprocesadores de 8 bits: usa operaciones orientadas a bytes, evita el uso de permutaciones de bit y tablas de consulta. La versión inicial de cuatro rondas (FEAL-4), propuesto como una alternativa rápida a DES, fue encontrado mucho más inseguro de lo planeado; por lo que se propuso realizar más rondas (FEAL-16 y FEAL-32) para compensar y ofrecer un nivel de seguridad parecido a DES; sin embargo, el rendimiento se ve fuertemente afectado mientras el número de rondas aumenta; y, mientras DES puede mejorar su velocidad con tablas de consulta, resulta más complicado para FEAL.

entrada: 64 bits de texto en claro $M = m_1 \dots m_{64}$;
llave de 64 bits $K = k_1 \dots k_{64}$.
salida: bloque de texto cifrado de 64 bits $C = c_1 \dots c_{64}$.
inicio
  Calcular 16 subllaves de 16 bits para K.
  Definir $M_L = m_1 \dots m_{32}; M_R = m_{33} \dots m_{64}$.
  $(L_0, R_0) \leftarrow (M_L, M_R) \oplus ((K_8, K_9), (K_{10}, K_{11}))$
  $R_0 \leftarrow R_0 \oplus L_0$.
  para_todo i desde 1 hasta 8:
    $L_i \leftarrow R_{i-1}$
    $R_i \leftarrow L_{i-1} \oplus f(R_{i-1}, K_{i-1})$
  fin
  $L_8 \leftarrow L_8 \oplus R_8$
  $(R_8, L_8) \leftarrow (R_8, L_8) \oplus ((K_{12}, K_{13}), (K_{14}, K_{15}))$
  $C \leftarrow (R_8, L_8)$.
fin

Pseudocódigo 2.5: FEAL-8, cifrado.

Para descifrar se utiliza el mismo algoritmo, con la misma llave K y el texto cifrado $C = (R_8, L_8)$ se utiliza como la entrada M; sin embargo, la generación de llaves se hace al revés: las subllaves $((K_{12}, K_{13}), (K_{14}, K_{15}))$ se utilizan para la $\oplus$ inicial, las $((K_8, K_9), (K_{10}, K_{11}))$ para la $\oplus$ final y en las rondas se utiliza de la subllave K_7 a la K_0.

FEAL con una llave de 64 bits puede ser generalizado a un número n de rondas rondas con n par, aunque se recomienda $n = 2^x$.

2.2.7. International Data Encryption Algorithm (IDEA)

Cifra bloques de 64 bits utilizando una llave de 128 bits. Este cifrado está basado en una generalización de la estructura Feistel y consiste en 8 rondas idénticas seguidas por una transformación. Cada ronda r utiliza 6 subllaves $K_i^{(r)}$ $(1 \leq i \leq 6)$ de 16 bits que se encargan de transformar una entrada X de 64 bits en una salida de cuatro bloques de 16-bits, que son utilizados como entrada en la siguiente ronda. La salida de la ronda 8 tiene como entrada la transformación de salida que, al emplear cuatro llaves adicionales $K_i^{(9)}$ $(1 \leq i \leq 4)$, produce los datos cifrados $Y = (Y_1, Y_2, Y_3, Y_4)$.

---

entrada: 64−bits de datos en claro $M = m_1 \dots m_{64}$;
llave de 128−bits $K = k_1 \dots k_{128}$.
salida: bloque cifrado de 64−bits $Y = (Y_1, Y_2, Y_3, Y_4)$.
inicio
Calcular las subllaves $K_1^{(r)}, \dots, K_6^{(r)}$ para las rondas $1 \leq r \leq 8$ **y** $K_1^{(9)}, \dots, K_4^{(9)}$ para la transformación de **salida**.
$(X_1, X_2, X_3, X_4) \leftarrow (m_1 \dots m_{16}, m_{17} \dots m_{32}, m_{33} \dots m_{48}, m_{49} \dots m_{64})$
donde X_i almacena 16 bits.
para_todo r desde 1 hasta 8:
a) $X_1 \leftarrow X_1 \times K_1^{(r)} \mod 2^{16} + 1$
$X_4 \leftarrow X_4 \times K_4^{(r)} \mod 2^{16} + 1$
$X_2 \leftarrow X_2 + K_2^{(r)} \mod 2^{16}$
$X_3 \leftarrow X_3 + K_3^{(r)} \mod 2^{16}$
b) $t_0 \leftarrow K_5^{(r)} \times (X_1 \oplus X_3) \mod 2^{16} + 1$
$t_1 \leftarrow K_6^{(r)} \times (t_0 + (X_2 \oplus X_4)) \mod 2^{16} + 1$
$t_2 \leftarrow t_0 + t_1 \mod 2^{16}$
c) $X_1 \leftarrow X_1 \oplus t_1$
$X_4 \leftarrow X_4 \oplus t_2$
$a \leftarrow X_2 \oplus t_2$
$X_2 \leftarrow X_3 \oplus t_1$
$X_3 \leftarrow a$
fin
Realizar la transformación de **salida**:
$Y_1 \leftarrow X_1 \times K_1^{(9)} \mod 2^{16} + 1$
$Y_4 \leftarrow X_4 \times K_4^{(9)} \mod 2^{16} + 1$
$Y_2 \leftarrow X_3 + K_2^{(9)} \mod 2^{16}$
$Y_3 \leftarrow X_2 + K_3^{(9)} \mod 2^{16}$
fin

Pseudocódigo 2.6: IDEA, cifrado.

El descifrado se realiza con el mismo algoritmo de cifrado, pero utilizando como entrada los datos cifrados Y como entrada M. Se usa la misma llave K; aunque las subllaves sufren una modificación al ser generadas, pues se utiliza una tabla y se realizan las operaciones contrarias (inverso de la adición y el inverso del producto).

Descartando los ataques a las llaves débiles, no hay un mejor ataque publicado para el *International Data Encryption Algorithm* (IDEA) de 8 rondas que el de la búsqueda exhaustiva en el espacio de llave. Por lo que la seguridad está ligada a la creciente debilidad de su tamaño de bloque relativamente pequeño.

2.2.8. Secure And Fast Encryption Routine (SAFER)

El cifrado *Secure And Fast Encryption Routine* (SAFER) K-64 es un cifrado por bloques de 64 bits iterativo. Consiste en r rondas idénticas seguidas por una transformación. Originalmente se recomendaban 6 rondas seguidas, sin embargo, ahora se utiliza una generación de claves ligeramente modificada y el uso de 8 rondas (máximo 10). Ambas generaciones de llaves expanden la llave de 64 bits en $2r + 1$ subllaves, cada una de 64 bits (dos por cada ronda y una más para la transformación de salida).

Este cifrado consiste completamente en operaciones de bytes, por lo que es adecuado para procesadores con tamaños de palabra pequeños, como los chips de tarjetas.

entrada: $r, 6 \leq r \leq 10$; 64−bits de datos en claro $M = m_1 \dots m_{64}$; $K = k_1 \dots k_{64}$.
salida: bloque cifrado de 64−bits $Y = (Y_1, \dots, Y_8)$.
inicio
  Calcular las subllaves $K_1, \dots, K_{2r+1}$
  $(X_1, X_2, \dots X_8) \leftarrow (m_1 \dots m_8, m_9 \dots m_{16}, \dots, m_{57} \dots m_{64})$
  para_todo i desde 1 hasta r:
    a) Para $j = 1, 4, 5, 8 : X_j \leftarrow X_j \oplus K_{2i-1}[j]$
       Para $j = 2, 3, 6, 7 : X_j \leftarrow X_j + K_{2i-1}[j] mod\ 2^8$
    b) Para $j = 1, 4, 5, 8 : X_j \leftarrow S[X_j]$
       Para $j = 2, 3, 6, 7 : X_j \leftarrow S_{inversa} X_j$
    c) Para $j = 1, 4, 5, 8 : X_j \leftarrow X_j + K_{2i}[j] mod\ 2^8$
       Para $j = 2, 3, 6, 7 : X_j \leftarrow X_j \oplus K_{2i}[j]$
    d) Para $j = 1, 3, 5, 7 : (X_j, X_{j+1}) \leftarrow f(X_j, X_{j+1})$.
    e) $(Y_1, Y_2) \leftarrow f(X_1, X_3), (Y_3, Y_4) \leftarrow f(X_5, X_7)$,
       $(Y_5, Y_6) \leftarrow f(X_2, X_4), (Y_7, Y_8) \leftarrow f(X_6, X_8)$.
       Para j desde 1 hasta 8: $X_j \leftarrow Y_j$
    f) $(Y_1, Y_2) \leftarrow f(X_1, X_3), (Y_3, Y_4) \leftarrow f(X_5, X_7)$,
       $(Y_5, Y_6) \leftarrow f(X_2, X_4), (Y_7, Y_8) \leftarrow f(X_6, X_8)$.
       Para j desde 1 hasta 8: $X_j \leftarrow Y_j$.
  fin
  Para $j = 1, 4, 5, 8 : Y_j \leftarrow X_j \oplus K_{2r+1}[j]$.
  Para $j = 2, 3, 6, 7 : Y_j \leftarrow X_j + K_{2r+1}[j]$ mód 2^8.
fin

Pseudocódigo 2.7: SAFER K-64, cifrado.

Para descifrar, se utiliza la misma llave K y las subllaves K_i que fueron utilizadas al cifrar. Cada paso del cifrado se hace en orden inverso, del último al primero; comenzando con una transformación de entrada utilizando la llave K_{2r+1} para deshacer la transformación de salida, se sigue con las rondas de descifrado

utilizando las llaves de K_{2r} a K_1, invirtiendo los pasos cada ronda.

2.2.9. RC5

Este cifrado por bloques tiene una arquitectura orientada a palabras (ya sea $w = 16, 32, 64$bits) y tiene una descripción muy compacta adecuada tanto para hardware como para software. Tanto la longitud b de la llave y el número de rondas r es variable; aunque se recomiendan 12 rondas para 32 bits y 16 para cuando se tienen palabras de 64.

```
entrada: 2w−bits de datos en claro M = (A, B); r;
     llave K = K[0]...K[b − 1]
salida: 2w−bits de datos cifrados C.
inicio
  Calcular 2r + 2 subllaves K_0, ..., K_{2r+1}
  A ← A + K_0 mód 2^w, B ← B + K_1 mód 2^w
  para_todo i desde 1 hasta r:
    A ← ((A ⊕ B) ↩ B) + K_{2i} mód 2^w
    B ← ((B ⊕ A) ↩ A) + K_{2i+1} mód 2^w
  fin
  Regresar C ← (A, B)
fin
```

Pseudocódigo 2.8: RC5, cifrado.

Para descifrar, RC5 utiliza el siguiente algoritmo.

```
entrada: 2w−bits de datos cifrados C = (A, B); r;
     llave K = K[0]...K[b − 1]
salida: 2w−bits de datos en claro M.
inicio
  Calcular 2r + 2 subllaves K_0, ..., K_{2r+1}
  A ← A + K_0 mód 2^w, B ← B + K_1 mód 2^w
  Para i desde r hasta 1:
    B ← ((B − K_{2i+1} mód 2^w) ↪ A) ⊕ A
    A ← ((A − K_{2i} mód 2^w) ↪ B) ⊕ B
  fin
  Regresar M ← (A − K_0 mód 2^w, B − K_1 mód 2^w)
fin
```

Pseudocódigo 2.9: RC5, descifrado.

2.2.10. Modos de operación

La información que aquí se presenta se puede consultar a mayor detalle en [6] y [1].

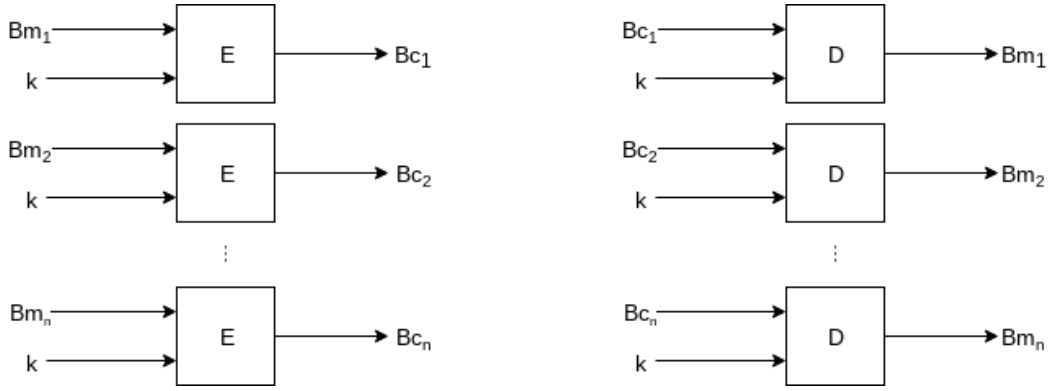
Por sí solos, los cifrados por bloques solamente permiten el cifrado y descifrado de bloques de información de tamaño fijo; donde, en la mayoría de los casos, los bloques son de menos de 256 bits, lo cual es equivalente a alrededor de 8 caracteres. Es fácil darse cuenta de que esta restricción no es ningún tema menor: en la gran mayoría de las aplicaciones, la longitud de lo que se quiere ocultar es arbitraria.

Los modos de operación permiten extender la funcionalidad de los cifrados por bloques para poder aplicarlos a información de tamaño irrestricto: reciben el texto original (de tamaño arbitrario) y lo cifran, ocupando en el proceso un cifrado por bloques.

Un primer enfoque (y quizás el más intuitivo) es partir el mensaje original en bloques del tamaño requerido y después aplicar el algoritmo a cada bloque por separado; en caso de que la longitud del mensaje no sea múltiplo del tamaño de bloque, se puede agregar información extra al último bloque para completar el tamaño requerido. Este es, de hecho, el primero de los modos que se presentan a continuación, el *Electronic Codebook* (ECB); su uso no es recomendado, pues es muy inseguro cuando el mensaje original es simétrico a nivel de bloque. También se enlistan otros tres modos, los cuales junto con ECB, son los más comunes.

Electronic Codebook (ECB)

La figura 2.9 muestra un diagrama esquemático de este modo de operación. El algoritmo recibe a la entrada una llave y un mensaje de longitud arbitraria: la llave se pasa sin ninguna modificación a cada función del cifrado por bloques; el mensaje se debe de partir en bloques ($M = Bm_1||Bm_2||\ldots||Bm_n$).



(a) Cifrado. (b) Descifrado.

Figura 2.9: Modo de operación ECB.

entrada: llave k; bloques de mensaje $Bm_1, Bm_2 \ldots Bm_n$.
salida: bloques de mensaje cifrado $Bc_1, Bc_2 \ldots Bc_n$.
inicio
  para_todo Bm
    $Bc_i \leftarrow$ E_k(Bm_i)

```
  fin
  regresar Bc
fin
```

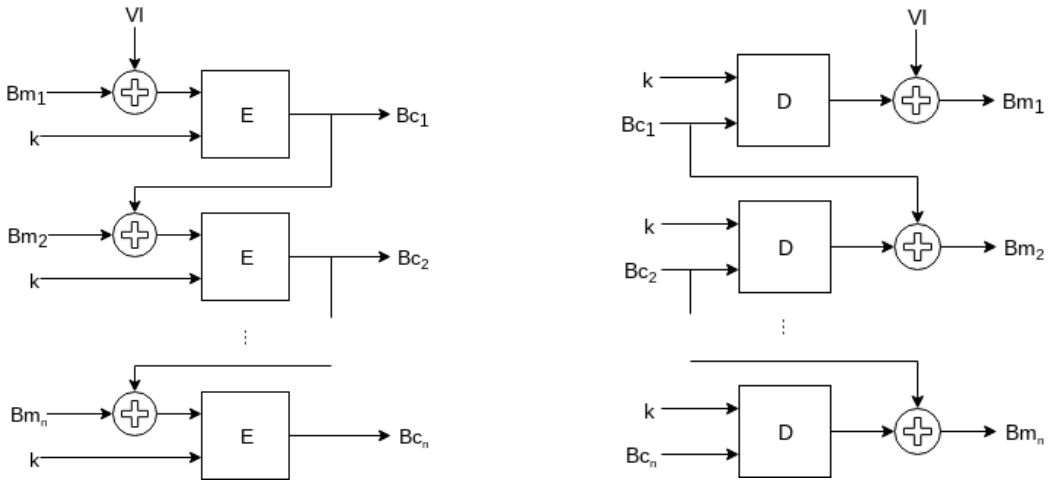
Pseudocódigo 2.10: Modo de operación ECB, cifrado.

```
entrada: llave k; bloques de mensaje cifrado Bc_1, Bc_2 ... Bc_n.
salida:  bloques de mensaje original B_1, B_2 ... B_n.
inicio
  para_todo Bc
    Bm_i ← D_k(Bc_i)
  fin
  regresar Bm
fin
```

Pseudocódigo 2.11: Modo de operación ECB, descifrado.

Cipher-block Chaining (CBC)

En *Cipher-block Chaining* (CBC) la salida del bloque cifrador uno se introduce (junto con el siguiente bloque del mensaje) en el bloque cifrador dos, y así en sucesivo. Para poder replicar este comportamiento en todos los bloque cifradores, este modo de operación necesita un argumento extra a la entrada: un vector de inicialización. De esta manera la salida del bloque i depende de todos los bloques anteriores; esto incrementa la seguridad con respecto a ECB.



(a) Cifrado. (b) Descifrado.

Figura 2.10: Modo de operación CBC.

En la figura 2.10 se muestran los diagramas esquemáticos para cifrar y descifrar; en los pseudocódigos

2.12 y 2.13 se muestran unos de los posibles algoritmos a seguir. Es importante notar que mientras que el proceso de cifrado debe ser forzosamente secuencial (por la dependencias entre salidas), el proceso de descifrado puede ser ejecutado en paralelo.

```
entrada: llave k; vector de inicialización VI;
         bloques de mensaje Bm_1, Bm_2 ... Bm_n.
salida:  bloques de mensaje cifrado Bc_1, Bc_2 ... Bc_n.
inicio
  Bc_0 ← VI                    // El vector de inicialización
  para_todo Bm                 // entra al primer bloque.
    Bc_i ← E(k, Bm_i ⊕ Bc_{i-1})
  fin
  regresar Bc
fin
```

Pseudocódigo 2.12: Modo de operación CBC, cifrado.

```
entrada: llave k; vector de inicialización VI;
         bloques de mensaje cifrado Bc_1, Bc_2 ... Bc_n.
salida:  bloques de mensaje original Bm_1, Bm_2 ... Bm_n.
inicio
  Bc_0 ← VI
  para_todo Bc
    Bm_i ← D_k(Bc_i) ⊕ Bc_{i-1}
  fin
  regresar Bm
fin
```

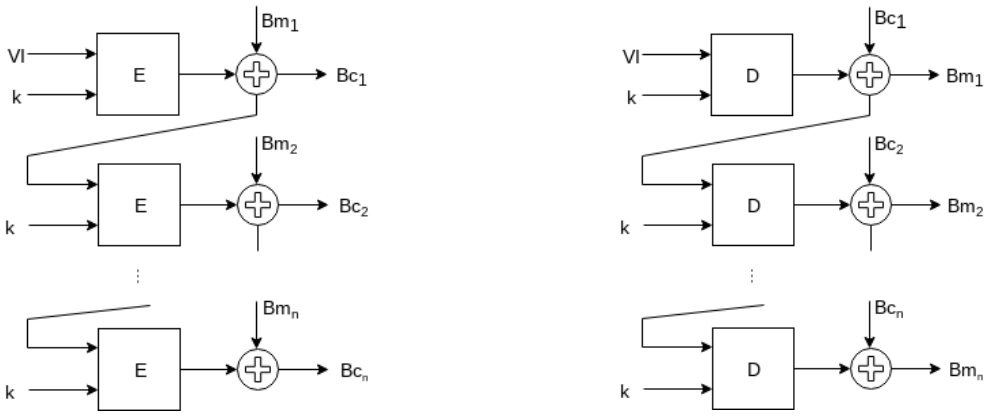
Pseudocódigo 2.13: Modo de operación CBC, descifrado.

Cipher Feedback (CFB)

Al igual que la operación de cifrado de CBC, ambas operaciones de *Cipher Feedback* (CFB) (cifrado y descifrado) están encadenadas bloque a bloque, por lo que son de naturaleza secuencial. En este caso, lo que se cifra en el primer paso es el vector de inicialización; la salida de esto se opera con un `xor` sobre el primer bloque de texto en claro, para obtener el primer bloque cifrado (figura 2.11).

Esta distribución presenta varias ventajas con respecto a CBC: las operaciones de cifrado y descifrado son sumamente similares, lo que permite ser implementadas por un solo algoritmo (pseudocódigo 2.14); tanto para cifrar como para descifrar solamente se ocupa la operación de cifrado del algoritmo a bloques subyacente. Estas ventajas se deben principalmente a las propiedades de la operación `xor` (ecuación 2.8).

$$A \oplus B = C \quad \Rightarrow \quad A = B \oplus C \tag{2.8}$$



(a) Cifrado. (b) Descifrado.

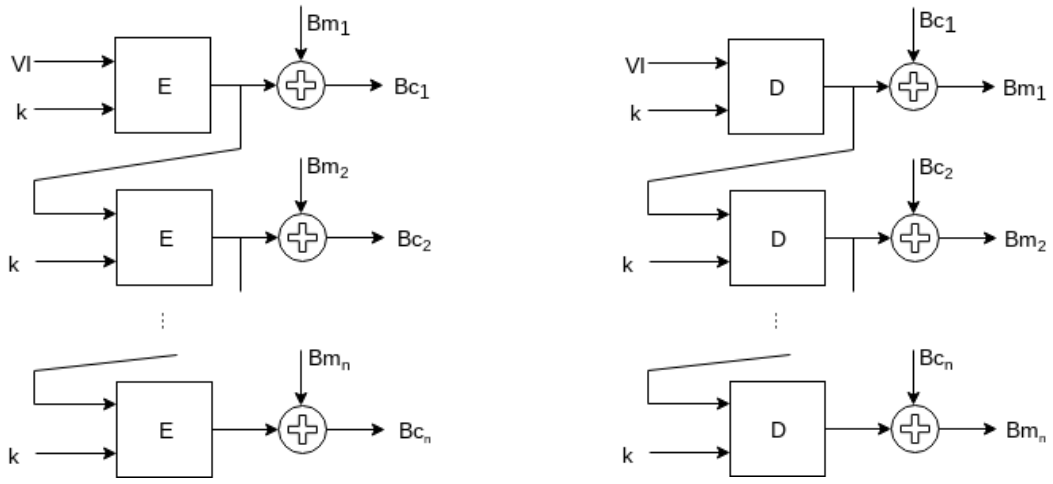
Figura 2.11: Modo de operación CFB.

entrada: llave k; vector de inicialización VI;
bloques de mensaje (cifrado **o** descifrado) $Bm_1, Bm_2 \dots Bm_n$.
salida: bloques de mensaje (cifrado **o** descifrado) $Bc_1, Bc_2 \dots Bc_n$.
inicio
  $Bc_0 \leftarrow VI$
  para_todo Bm
    $Bc_i \leftarrow C_k(Bc_{i-1}) \oplus Bm_i$
  fin
  regresar Bc
fin

Pseudocódigo 2.14: Modo de operación CFB(cifrado y descifrado).

Output Feedback (OFB)

Este modo es muy similar al anterior (CFB), salvo que la retroalimentación va directamente de la salida del cifrador a bloques. De esta forma, nada que tenga que ver con el texto en claro llega al cifrado a bloques; este solamente se la pasa cifrando una y otra vez el vector de inicialización.



(a) Cifrado. (b) Descifrado.

Figura 2.12: Modo de operación *Output Feedback* (OFB).

entrada: llave k; vector de inicialización VI;
bloques de mensaje (cifrado **o** descifrado) $Bm_1, Bm_2 \ldots Bm_n$.
salida: bloques de mensaje (cifrado **o** descifrado) $Bc_1, Bc_2 \ldots Bc_n$.
inicio
  auxiliar $\leftarrow VI$
  para_todo Bm
    auxiliar $\leftarrow$ E_k(auxiliar)
    $Bc_i \leftarrow$ auxiliar $\oplus$ Bm_i
  fin
  regresar Bc
fin

Pseudocódigo 2.15: Modo de operación OFB(cifrado y descifrado).

2.3. Cifrados de flujo

La información de esta sección (junto con las subsecciones contenidas) puede ser encontrada con mayor detalle en [1], [7] y [8].

A diferencia de los cifrados de bloque, que trabajan sobre grupos enteros de bits a la vez, los cifrados de flujo trabajan sobre bits individuales, cifrándolos uno por uno. Una manera de verlos es como cifrados por bloques con un tamaño de bloque igual a 1.

Un cifrado de flujo aplica transformaciones de acuerdo a un flujo de llave: una secuencia de símbolos pertenecientes al espacio de llaves. El flujo de llave puede ser generado tanto de manera aleatoria, como por un algoritmo pseudoaleatorio que reciba a la entrada, o bien una semilla, o bien una semilla y algunos bits del texto cifrado.

Entre las ventajas de los cifrados de flujo sobre los cifrados de bloque se encuentra el hecho de que son más rápidos en hardware y más útiles cuando el buffer es limitado o se necesita procesar la información al momento de llegada. La propagación de los errores es limitada o nula, por lo que también son más apropiados en casos en los que hay probabilidades altas de errores en la transmisión.

Los cifrados de bloques funcionan sin ninguna clase de memoria (por sí solos); en contraste, la función de cifrado de un cifrado de flujo puede variar mientras se procesa el texto en claro, por lo cuál tienen un mecanismo de memoria asociado. Otra denominación para estos cifrados es *de estado*, por que la salida no depende solamente del texto en claro y de la llave, sino que también depende del estado actual.

2.3.1. Clasificación

Una clasificación común es en *síncronos* y en *autosincronizables*. A continuación se describen a grandes rasgos ambos modelos.

Síncronos

Un cifrado de flujo síncrono es aquel en el que el flujo de la llave es generado de manera independiente del texto en claro y del texto cifrado. Se puede definir un modelo general con las siguientes tres ecuaciones.

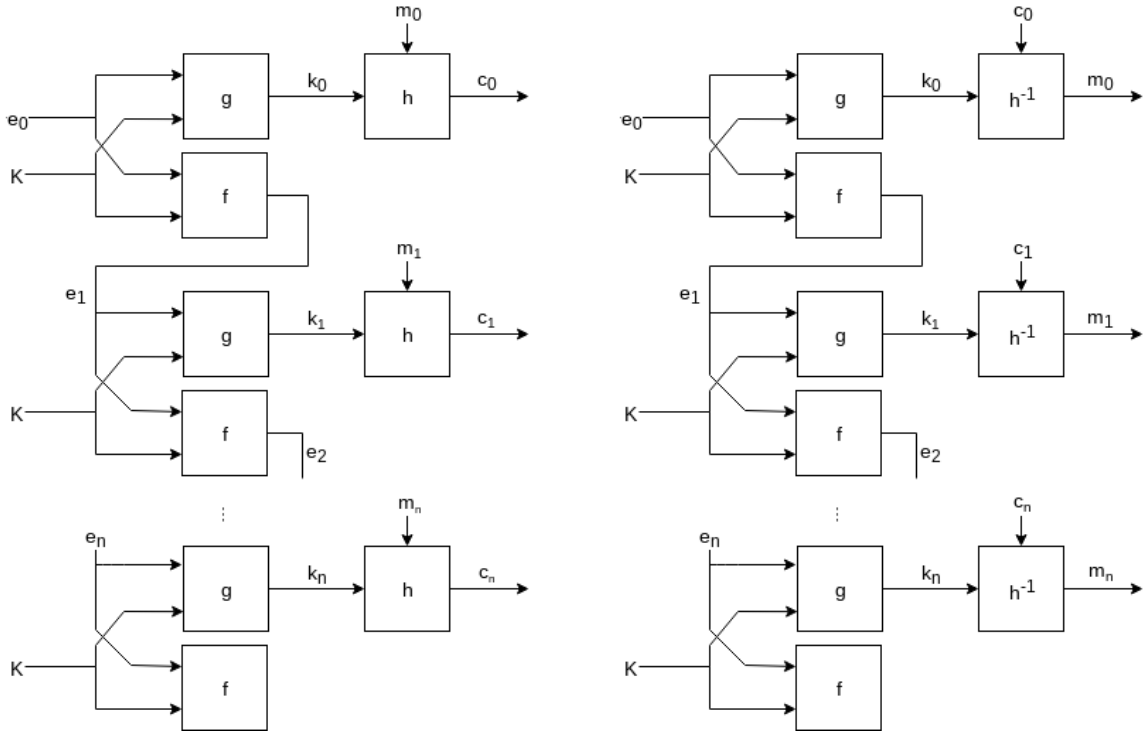
$$e_{i+1} = f(e_i, K) \tag{2.9}$$

$$k_i = g(e_i, K) \tag{2.10}$$

$$c_i = h(k_i, m_i) \tag{2.11}$$

La letra e representa el estado del cifrado, K es la llave, k es la salida del flujo de llave, c es el texto cifrado y m es el texto en claro. La función de la ecuación 2.9 (f) es la que describe el cambio de estado; este se determina a partir del estado actual y de la llave. En la ecuación 2.10 se describe la acción del flujo de llave (g): para determinar el próximo símbolo se emplea solamente el estado actual y la llave. La tercera ecuación (2.11, h) describe la acción de combinar el flujo de la llave con el mensaje, y así obtener el texto cifrado.

En la figura 2.13 se describe de manera gráfica las operaciones de cifrado y descifrado; estas guardan muchas similitudes con el modo de operación OFB (sección 2.2.10), con la única excepción de que este trabaja con bloques del tamaño del cifrado subyacente. En otras palabras, si se definiera el tamaño del bloque (y en consecuencia el tamaño del vector de inicialización) como 1, entonces OFB sería un cifrado de flujo síncrono.



(a) Cifrado. (b) Descifrado.

Figura 2.13: Esquema general de un cifrado de flujo síncrono.

El nombre de esta categoría proviene del hecho de que ambos entes del proceso comunicativo (emisor y receptor) deben encontrarse sincronizados (usar la misma llave y encontrarse en la misma posición) para que la comunicación tenga éxito: si se insertan dígitos extras al mensaje cifrado, la sincronización se pierde. Los cifrados de flujo síncronos no tienen propagación de error: aunque ciertos bits sean modificados (pero no borrados) durante su transmisión, el resto del mensaje sigue siendo descifrable.

Autosincronizables

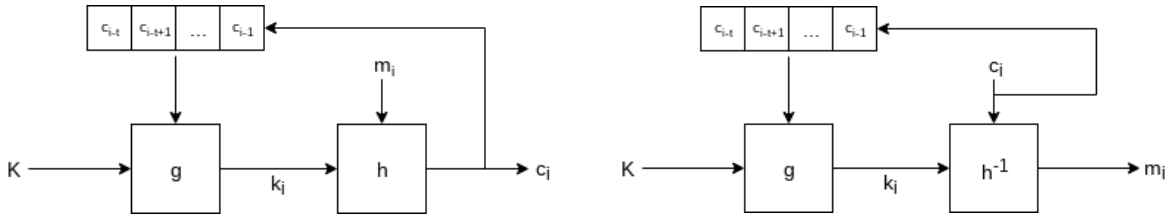
En esta clasificación se engloban a aquellos cifrados cuyo flujo de llave es resultado de la propia llave original y de cierto número previo de dígitos cifrados. Las ecuaciones que describen su comportamiento son las siguientes.

$$e_{i+1} = (c_{i-t}, c_{i-t+1}, \ldots, c_{i-1}) \tag{2.12}$$

$$k_i = g(e_i, K) \tag{2.13}$$

$$c_i = h(k_i, m_i) \tag{2.14}$$

La notación es la misma que en las ecuaciones 2.9, 2.10 y 2.11. En este caso, el próximo estado depende de t (el tamaño de la ventana) dígitos cifrados anteriormente. En la figura 2.14 se describe de manera gráfica el proceso de cifrado y descifrado.



(a) Cifrado. (b) Descifrado.

Figura 2.14: Esquema general de un cifrado de flujo autosincronizable.

En una antítesis de la categoría anterior, el nombre de esta indica que no es necesario que el emisor y el receptor estén sincronizados: si se llegan a perder bits en la transmisión, el esquema es capaz de autosincronizarse, pues el flujo de la llave depende de cierto número de bits anteriores. A esta categoría también se le conoce como «asíncrona».

La propagación de los errores depende del tamaño de ventana (el número t de bits previos utilizados para calcular la próxima llave), si se modifica un bit, entonces los próximos t serán incorrectos.

2.3.2. RC4

RC4 es un cifrado de flujo diseñado por Ron L. Rivest en 1987 para la empresa RSA. Es usado en varios protocolos de seguridad comunes: *Secure Sockets Layer* (SSL)/*Transport Layer Security* (TLS), *Wired Equivalent Privacy* (WEP) y *WiFi Protected Access* (WPA); los dos últimos son parte del estándar *Institute of Electrical and Electronic Engineers* (IEEE) 802.11 para comunicaciones *Local Area Network* (LAN) inalámbricas. RC4 se mantenía como secreto de compañía hasta que, en septiembre de 1994, fue filtrado

de forma anónima en Internet.

En el pseudocódigo 2.16 se describe el proceso de cifrado del algoritmo. S es un vector de estado; T es un vector temporal.

```
entrada: llave k; mensaje original m_1,m_2...m_n.
salida:  mensaje cifrado c_1,c_2...c_n.
inicio
  /* Inicialización */
  para_todo i entre 0 y 255:
    S[i] ← i
    T[i] ← K[i  mód  longitud_de_llave]
  fin

  /* Permutación inicial */
  j ← 0
  para_todo i entre 0 y 255:
    j = (j + S[i] + T[i])  mód  256
    intercambiar(S[i], S[j])
  fin

  /* Proceso de cifrado */
  i, j ← 0
  para_todo m:
    i ← (i + 1)  mód  256
    j ← (j + S[i])  mód  256
    intercambiar(S[i], S[j])
    k ← S[(S[i] + S[j])  mód  256]
    c ← m ⊕ k
  fin
  regresar c
fin
```

Pseudocódigo 2.16: Proceso de cifrado de RC4.

Se han hecho varias publicaciones que analizan métodos para atacar RC4, ninguna de las cuales presenta algo práctico cuando se utiliza una llave mayor a 128 bits. Sin embargo, en [9] se reporta un problema más serio sobre la implementación que se hace de RC4 en el protocolo WEP; este problema en particular no ha demostrado afectar a otras aplicaciones que usan RC4.

2.3.3. El proyecto eSTREAM

La información aquí expuesta puede ser consultada a mayor detalle en [10], [11] y [12].

El proyecto eSTREAM fue un esfuerzo de la comunidad *European Network of Excellence in Cryptology*

Perfil 1	**Perfil 2**
HC-128 Rabbit Salsa20/12 Sosemanuk	Grain v1 MICKEY v2 Trivium

Tabla 2.1: Finalistas del proyecto eSTREAM

(ECRYPT) para promover el diseño de cifrados de flujo eficientes y compactos. Como resultado, se publicó un portafolio en abril de 2008, el cual ha estado bajo continuas actualizaciones desde entonces; actualmente cuenta con siete algoritmos (tabla 2.1).

El portafolio se divide en dos perfiles: el primero contiene algoritmos adecuados para aplicaciones de software con requerimientos de rapidez de procesamiento muy altos; el segundo se enfoca en aplicaciones de hardware con pocos recursos disponibles.

El proyecto se inició después de que Adi Shamir se preguntara si realmente había necesidad de los cifrados de flujo, en una conferencia de RSA en 2004. El principal argumento a favor fue que, para la gran mayoría de los casos, el uso de AES (sección 2.2.5) con una configuración de flujo es una solución adecuada. Este último punto de vista iba generalmente acompañado de la creencia de que era imposible diseñar cifrados de flujo seguros (un proyecto anterior similar, *New European Schemes for Signatures, Integrity and Encryption* (NESSIE), terminó sin resultados después de todos los criptoanálisis hechos). Por otra parte, como argumentos en contra, Shamir identificó dos áreas en las que los cifrados de flujo ofrecen ventajas respecto a los cifrados por bloques:

1. Cuando se requieren tiempos de procesamiento excepcionalmente rápidos (perfil 1).
2. Cuando los recursos disponibles son muy pocos (perfil 2).

Las palabras de Shamir fueron ampliamente difundidas, y más tarde en ese mismo año, ECRYPT lanzó el proyecto eSTREAM, cuyo principal objetivo fue ampliar el conocimiento sobre el análisis y el diseño de cifrados de flujo. Después de un periodo de estudio, se lanzó una convocatoria que generó un interés considerable: antes del 29 de abril de 2005 (la fecha límite) se recibieron 34 propuestas; algunas de las cuales intentaban cumplir con los dos perfiles a la vez (lamentablemente no sobrevivieron mucho tiempo).

2.4. Funciones hash

La información presentada a continuación puede consultarse con más profundidad en las siguientes referencias [1], [2], [13], [14].

Se refiere al conjunto de funciones computacionalmente eficientes que mapean cadenas binarias de una longitud arbitraria a cadenas binarias de una longitud fija, llamadas valores hash.

Matemáticamente, una función hash es una función

$$\begin{aligned} h : \{0,1\}^* &\longrightarrow \{0,1\}^n \\ m &\longmapsto h(m) \end{aligned} \tag{2.15}$$

La longitud de n suele ser entre 128 y 512 bits. Las funciones hash h tienen las siguientes propiedades:

1. Compresión: h mapea una entrada x (cuya longitud finita es arbitraria) a una salida $h(x)$ de longitud fija n.
2. Facilidad de cómputo: dada x y h, $h(x)$ es calculada ya sea sin necesitar mucho espacio, tiempo de cómputo, o requiere pocas operaciones, etcétera.

De manera general, las funciones hash se pueden dividir en dos categorías: las que no utilizan llave y su único parámetro es la entrada x, y las que necesitan una llave secreta k y la entrada x.

Sea una función hash sin llave h con entradas x, x' y salidas y y y', respectivamente. A continuación se listan algunas de las propiedades que puede tener:

1. Resistencia de preimagen: no es computacionalmente factible para una salida específica y encontrar una entrada x' que dé como resultado el mismo valor hash $h(x') = y$ si no se conoce x. Esta propiedad también es llamada *de un sentido*.
2. Resistencia de segunda preimagen: no es computacionalmente factible encontrar una segunda entrada x' que tenga la misma salida que una entrada específica x: $x \neq x'$ tal que $h(x) = h(x')$. Esta propiedad también es conocida como *de débil resistencia a colisiones*.
3. Resistencia a las colisiones: no es computacionalmente factible encontrar dos entradas distintas x, x' que lleven al mismo valor hash, o sea, $h(x) = h(x')$. A diferencia de la anterior, la selección de ambas entradas no está restringida. Esta propiedad también es conocida como *de gran resistencia a colisiones*.

Una función hash h que cumple con las propiedades de resistencia de preimagen y resistencia de segunda preimagen es conocida como una función hash de un solo sentido o *One-Way Hash Function* (OWHF). Las que cumplen con la resistencia de segunda preimagen y resistencia a las colisiones son conocidas como

funciones hash resistentes a colisiones o *Collision-Resistant Hash Function* (CRHF). Aunque casi siempre las funciones CRHF cumplen con la resistencia de preimagen, no es obligatorio que lo hagan.

Algunos ejemplos de las funciones OWHF son el *Secure Hash Algorithm* (SHA)-1 y el *Message Digest 5* (MD5). En los esquemas de firma electrónica, se obtiene el valor hash del mensaje ($h(m)$) y se pone en el lugar de la firma. Los valores hash también son utilizados para revisar la integridad de las llaves públicas y, al utilizarse con una llave secreta, las funciones criptográficas hash se convierten en códigos de autenticación de mensaje (*Message Authentication Code* (MAC), por sus siglas en inglés), una de las herramientas más utilizadas en protocolos como SSL e IPSec para revisar la integridad de un mensaje y autenticar al remitente.

Una de las aplicaciones más conocidas de las funciones hash es la de cifrar las contraseñas: en un sistema, en vez de almacenar la contraseña *clave*, se guarda su valor hash $h(clave)$. Así, cuando un usuario ingresa su contraseña, el sistema calcula su valor hash y lo compara con el que se tiene guardado. Realizar esto ayuda a evitar que las contraseñas sean conocidas para los usuarios con privilegios, como pueden ser los administradores.

2.4.1. Integridad de datos

Las funciones criptográficas hash también son conocidas como funciones *procesadoras de mensajes* y el valor hash $h(m)$ de un mensaje m dado es llamado *huella* de m; ya que es una representación compacta de m y, dada la resistencia a la segunda preimagen, la huella es prácticamente única. Si el mensaje fuese modificado, el valor hash sería distinto; por lo que si se tienen almacenados los valores hash, basta con calcular su valor $h(m)$ y compararlo con el que se tiene guardado para detectar modificaciones. Por esta razón, las funciones hash también son llamadas códigos de detección de modificaciones (como el *Modification Detection Code 2* (MDC-2)).

2.4.2. Firmas

Sea (n, e) la llave pública RSA y d el exponente decodificador secreto de Alice. En el esquema básico de firma RSA, Alice puede firmar mensajes que estén codificados por números $m \in \{0, \ldots, n-1\}$. Para firmar m, aplica el algoritmo de descifrado y obtiene la firma $\sigma = m^d \ mod \ n$ de m. Normalmente, n es un número de 1024 bits y Alice puede firmar una cadena de bits m tal que, cuando es interpretada como número, sea menor que n. Esto es una cadena de, máximo, 128 caracteres *American Standard Code for Information Interchange* (ASCII): la mayoría de los documentos que se desean firmar suelen ser mucho más grandes. Este problema existe en todos los esquemas de firma digital y usualmente es resuelto al aplicar una función hash resistente a colisiones h. De esta forma, primero se obtiene el valor hash del mensaje $h(m)$ y esto es lo que se firma en lugar del mensaje mismo (m):

$$\sigma = h(m)^d \mod n \tag{2.16}$$

Los mensajes que tengan el mismo valor hash tienen la misma firma. En este caso, es primordial que la función hash h sea resistente a colisiones para garantizar el no repudio. De otra manera, Alice podría firmar el mensaje m y después decir que había firmado un mensaje distinto (n). La resistencia a segundas preimágenes previene que un atacante Eve tome un mensaje m firmado por Alice, genere un mensaje nuevo n y utilice σ como una firma válida de Alice para n.

2.4.3. Message Digest-4 (MD4)

En la década de 1990 esta función hash fue diseñada por Ronald Rivest. Tiene entradas de longitud arbitraria y la longitud de la salida procesada es de 128 bits. El *Message Digest 4* (MD4) fue innovador y clave en el diseño para los algoritmos venideros de esta clase (como el MD5).

2.4.4. RIPEMD

Esta función hash, publicada en 1996, está basada en MD4 y fue diseñada por Hans Dobbertin y otros. Consiste en dos formas equivalentes de la función de compresión de MD4. El algoritmo original (RIPEMD-160) devuelve bloques *procesados* de 160 bits; cuando en 1996 Hans descubrió una colisión en dos rondas, se desarrollaron nuevas versiones mejoradas: RIPEMD-128, RIPE-256, RIPE-320; las cuales dan bloques procesados de 128, 256 y 320 bits respectivamente.

2.4.5. *Secure Hash Algorithm (SHA)*

El algoritmo SHA fue publicado por NIST y *National Security Agency* (NSA) en 1993; este algoritmo produce bloques de 160 bits y fue desarrollado para reemplazar al MD4; sin embargo, poco después de haber sido publicado tuvo que ser quitado por problemas de seguridad. Actualmente, SHA es conocido como SHA-0.

En 1995, SHA-0 fue reemplazado por SHA-1; tiene una salida de la misma longitud que su predecesor y es una de las funciones hash más populares. Hay que destacar que la seguridad que brinda esta función es limitada, pues tiene el mismo nivel que un cifrado por bloques de 80 bits.

En 2002 NIST publicó tres funciones hash más: SHA-256, SHA-384 y SHA-512; esta familia de funciones hash es conocida como SHA-2 y fue desarrollada para cubrir la necesidad de una llave más grande para poder empatar su tamaño con AES. Dos años más tarde, una nueva función hash fue agregada a la familia SHA-2: SHA-224.

Finalmente, en 2008, NIST inició un concurso para buscar al SHA-3 y en 2012 anunció al ganador: Keccak, una función hash desarrollada por Guido Bertoni, Joan Daemen, Michael Peeters y Gilles Van Assche. Esta función tiene una construcción completamente distinta a las familias anteriores.

2.5. Códigos de Autenticación de Mensaje (MAC)

La información presentada a continuación puede consultarse con más profundidad en las siguientes referencias [1], [2], [15].

Las funciones hash con llave cuyo propósito específico es la autenticación de origen y garantizar la integridad de datos del mensaje son llamadas MAC. Estas funciones tienen como entrada una llave secreta k y un mensaje de longitud arbitraria y dan como resultado un mensaje de longitud n.

$$h_k : \{0,1\}^* \longrightarrow \{0,1\}^n \tag{2.17}$$

Las funciones MAC son la técnica simétrica estándar utilizada tanto para la autenticación como para la protección de la integridad de los mensajes. Dependen de unas llaves secretas que son compartidas entre las partes que se van a comunicar; cada una de las partes puede producir el MAC correspondiente para un mensaje dado. Como se explica a continuación, los MAC pueden ser obtenidos mediante cifradores de bloque, cifradores de flujo o de funciones hash criptográficas.

El algoritmo MAC más usado basado en un cifrador de bloque utiliza el modo de operación CBC (véase sección 2.2.10). Cuando DES es utilizado como el cifrado de bloque E, el tamaño de bloque es de 64 bits y la llave MAC es de 56 bits.

Otra manera de construir MAC es mediante un algoritmo de *Message Digest Cipher* (MDC) que incluya una llave secreta k como parte de la entrada. Un ejemplo de esto es el algoritmo MD5-MAC; donde la función de compresión depende de la llave secreta k, que interviene en todas las iteraciones.

El algoritmo *Message Authenticator Algorithm* (MAA) fue diseñado en 1938 específicamente para obtener MAC en máquinas de 32 bits. El tiempo de ejecución es directamente proporcional a la longitud del mensaje y alrededor de cuatro veces más largo que el MD4

Capítulo 3

Análisis y diseño

3.1. Generación de *tokens*

3.1.1. Requerimientos

En [16], el *Payment Card Industry* (PCI) *Security Standard Council* (SSC) divide a los *token*s en reversibles e irreversibles. A su vez, los reversibles se dividen en criptográficos y en no criptográficos; mientras que los irreversibles se dividen en autenticables y no autenticables (figura 3.1).

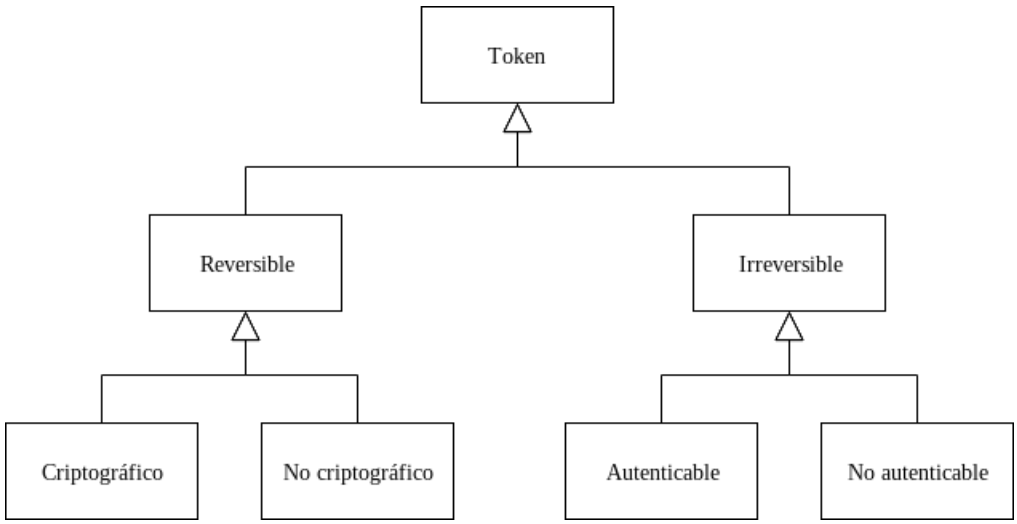


Figura 3.1: Clasificación de los *token*s.

En esta sección se explica cada una de las categorías y se analizan los requerimientos que deben tener. Para comenzar, se enlistan los requerimientos aplicables a todos los *token*s, sin importar en qué categoría estén:

REQUTO-01 Resistencia a texto claro conocido.

Un atacante con acceso a múltiples pares de *token*s y *Personal Account Number* (PAN) no debe de ser capaz de determinar otros PAN a partir de solamente *token*s. En otras palabras, los *token*s deben ser resistentes a ataques con texto en claro conocido (sección 2.1.2).

REQUTO-02 Resistencia a sólo texto cifrado.

Recuperar un PAN a partir de un *token* debe de ser computacionalmente no factible (resistencia a ataques con sólo texto cifrado, sección 2.1.2).

REQUTO-03 Detección de anomalías.

Se deben de implementar disparadores que permitan detectar irregularidades en el sistema (anomalías, funcionamientos erróneos, comportamientos sospechosos). El producto debe registrar dichos eventos y

avisar al personal correspondiente.

REQUTO-04 Distinción entre *token*s y PAN.
Se debe contar con un mecanismo para distinguir entre *token*s y PAN. Los proveedores del servicio de tokenización deben compartir este mecanismo con la entidad (o entidades) que usa los *token*s.

REQUTO-05 Mapeos de *token* a *token* prohibidos.
No se debe poder pasar de un primer *token* válido a un segundo, también válido; forzosamente debe existir un estado intermedio: del primer *token* se pasa al PAN correspondiente (operación de detokenización) y de este se pasa al segundo *token*.

REQUTO-06 Protección contra vulnerabilidades comunes.
Se deben implementar medidas en contra de las vulnerabilidades de seguridad más comunes ([17], requerimiento 5.2). Algunas de estas medidas pueden ser el uso de herramientas de análisis de código estático, o el uso de lenguajes de programación especializados.

REQUTO-07 Primitivas criptográficas usadas.
Las primitivas criptográficas que se usen deben estar basadas en estándares nacionales (Estados Unidos) o internacionales (e. g. AES).

Los *token*s irreversibles no pueden, bajo ninguna circunstancia, ser reconvertidos al PAN original. Esta restricción aplica tanto para cualquier entidad en el entorno del negocio (comerciante, proveedor de *token*s, banco) como para cualquier posible atacante. Dados un PAN y un *token*, los identificables permiten validar cuando el primero fue utilizado para la creación del segundo, mientras que los no identificables, no.

La clasificación del PCI SSC con respecto a los reversibles resulta un poco confusa (esto ya ha sido señalado antes, [18]). Establece que los criptográficos son generados utilizando criptografía fuerte, el PAN nunca se almacena, solamente se guarda una llave; los no criptográficos guardan la relación entre *token*s y PAN en una base de datos. El problema está en que no se menciona *cómo* generar los no criptográficos. A pesar del nombre, los métodos más comunes para esta categoría ocupan primitivas criptográficas (e. g. generadores pseudoaleatorios); además de que, en una implementación real, para poder cumplir con el PCI *Data Security Standard* (DSS), la propia base de datos debe de estar cifrada [19].

PCI DSS define cuatro *dominios* de seguridad para el proceso de tokenización:

1. **Generación de *token*s.** Para cada clase tokenizadora, este dominio se encarga de definir consideraciones para generación segura de *token*s. Cubre los dispositivos, procesos, mecanismos y algoritmos que son utilizados para crear los *token*s.

2. **Mapeo de *token*s.** Este dominio, que se refiere al mapeo de los *token*s con su PAN origen, aplica solamente a los procesos de tokenización reversibles. Entre otras cosas, provee guías respecto a control

de acceso necesarios para las peticiones de tokenización.

3. **Bóveda de datos de tarjeta.** Como el dominio pasado, solo aplica a las implementaciones de tokenización reversibles. Cubre el cifrado obligado del PAN y los controles de acceso necesarios para entrar a la *Card Data Vault* (CDV).

4. **Manejo criptográfico de llaves.** Define las buenas prácticas para el manejo criptográfico de las llaves y las operaciones realizadas con ellas por el producto tokenizador.

Irreversibles

REQUTO-08 Sobre la generación de *token*s.

El mecanismo utilizado para la generación de los *token*s no es reversible (o improbable).

SUBREQUTO-08/1 Sobre el mecanismo generador.

El proceso para crear *token*s clasificado como irreversible debe asegurar que el mecanismo, proceso o algoritmo utilizado para crear el *token* no sea reversible. Si una función hash (véase sección 2.4) es utilizada, esta debe ser una primitiva criptográfica y utilizar una llave secreta k tal que el mero conocimiento de la función hash no permita la creación de un oráculo.

SUBREQUTO-08/2 Contenido en claro.

Los *token*s no deben contener dígitos en claro del PAN original, excepto que estos dígitos sean una coincidencia.

SUBREQUTO-08/3 Creación de un diccionario.

La creación de una tabla o *diccionario* de *token*s estáticos debería ser imposible, o, al menos, al punto de satisfacer que la probabilidad de predecir correctamente el PAN debería ser menor que $\frac{1}{10^6}$.

SUBREQUTO-08/4 Sobre el proceso de autenticación.

Si un *token* autenticable no reversible es utilizado, el proceso de autenticación no debe revelar información suficiente para realizar búsquedas, excepto una exhaustiva (PAN por PAN) y se deben implementar controles para detectar estas últimas.

REQUTO-09 Sobre el manejo adecuado de llaves.

Se deben seguir las buenas prácticas criptográficas respecto a la administración de llaves.

SUBREQUTO-09/1 Sobre el ciclo de vida.

La llave tokenizadora debe seguir la política de los ciclos de llaves descritos en el ISO/IEC 115681.

SUBREQUTO-09/2 Descripción del periodo criptográfico activo.

La política sobre el tiempo de vida de la llave debe incluir una descripción sobre el periodo

criptográfico activo de la llave tokenizadora en cuestión.

SUBREQUTO-09/3 Sobre la destrucción de las llaves.

El proveeddor debe incorporar una función que permita la destrucción de sus llaves criptográficas sin tener que alterar o abrir el dispositivo.

Criptográficos reversibles

REQUTO-10 Seguridad de la administración de llaves.

la administración de las llaves criptográficas debe ser segura.

SUBREQUTO-10/1 Exportar llave en claro prohibido.

Las llaves usadas para generar *token*s no se deben poder exportar en claro desde el programa.

SUBREQUTO-10/2 Entropía de generación de llaves.

La fuente generadora de llaves debe tener, al menos, 128 bits de entropía.

SUBREQUTO-10/3 Llaves de uso único.

Las llaves criptográficas usadas para generar *token*s no deben ser usadas para ningún otro fin.

REQUTO-11 Probabilidad de adivinar relaciones.

La probabilidad de adivinar la relación entre un *token* y un PAN debe de ser menor que 1 en 10^6.

SUBREQUTO-11/1 Distribución uniforme.

Para un PAN dado, todos los *token*s deben ser equiprobables; esto es, el mecanismo tokenizador no debe exhibir tendencias probabilísticas que lo expongan a ataques estadísticos.

SUBREQUTO-11/2 Permutación aleatoria.

El método de tokenización debe actuar como una familia de permutaciones aleatoria desde el espacio de PAN al espacio de *token*s.

SUBREQUTO-11/3 Cambio de llave.

Un cambio en la llave se debe ver reflejado en un cambio en el *token* resultado.

SUBREQUTO-11/4 Cambio de PAN.

Un cambio en el PAN se debe ver reflejado en un cambio en el *token* resultado.

REQUTO-12 Prácticas para la administración de llaves.

La administración de llaves criptográficas debe llevarse a cabo de acuerdo al estándar del NIST [20] y a ISO/IEC 11770.

SUBREQUTO-12/1 Ciclo de vida de las llaves.

Las llaves para *tokenizar* deben seguir una política de ciclo de vida como la descrita en ISO/IEC 11568-1.

SUBREQUTO-12/2 Descripción de periodos para cada llave.

La política de la vida útil de cada llave debe incluir una descripción del periodo activo.

SUBREQUTO-12/3 Permitir la destrucción de todas las llaves.

El sistema debe permitir la destrucción de las llaves sin necesidad de una manipulación externa.

REQUTO-13 Sobre la longitud de las llaves.

Las llaves para *tokenizar* deben tener una fuerza efectiva de, al menos, 128 bits. Cualquier llave utilizada para proteger o para derivar la llave del token debe de ser de igual o mayor fuerza efectiva.

REQUTO-14 Independencia estadística.

Si el espacio de llaves es usado para producir *token*s es dos contextos distintos (e. g. para distintos comerciantes), estas deben ser estadísticamente independientes.

No criptográficos reversibles

REQUTO-15 Generación y almacenamiento de tokens.

La generación de un *token* debe realizarse independientemente de su PAN, y la relación entre un PAN y su *token* sólo tiene que estar almacenada en la base de datos (CDV) establecida.

REQUTO-16 Probabilidad de encontrar un PAN.

La probabilidad de encontrar un PAN a partir de su respectivo *token* debe de ser menor que 1 en 10^6.

SUBREQUTO-16/1 Distribución equiprobable.

Para un PAN dado, todos sus *token*s respectivos deben ser equiprobables, esto es que el sistema *tokenizador* no debe exhibir patrones probabilísticos que lo vulneren a un ataque estadístico.

SUBREQUTO-16/2 Permutaciones aleatorias.

El método de tokenización debe actuar como una familia de permutaciones aleatoria en el espacio efectivo de los PANs al espacio de *token*s.

SUBREQUTO-16/3 Parámetros de tokenización.

El método de tokenización debe incluir parámetros tales que, un cambio en estos parámetros resulte en un *token* diferente; por ejemplo, un cambio en la instancia del proceso debe derivar en una secuencia de *token*s distintos, incluso cuando es usada la misma secuencia de *token*s.

SUBREQUTO-16/4 Reflejo de cambios.

Al cambiar parte de un PAN, debe cambiar su *token* resultante.

REQUTO-17 Distribución imparcial.

El proceso de generación de *token*s debe garantizar una distribución de *token*s imparcial, esto significa que la probabilidad de cualquier par PAN/*token* debe ser igual.

REQUTO-18 Instancias estadísticamente independientes.

Si varias o diferentes instancias de la bases de datos (CDV) son usadas, cada una de estas debe ser estadísticamente independientes.

REQUTO-19 Asignación de tokens.

La asignación de un *token* a un PAN debe realizarse por medio de una búsqueda de datos o un índice dentro de la base de datos (CDV), y no por medio de métodos criptográficos.

SUBREQUTO-19/1 Asignación probabilísticamente independiente.

El PAN y el *token* debe ser probabilísticamente independientes. Cualquier método lógico o matemático no debe ser usado para *tokenizar* el PAN o *detokenizar* el *token*.

REQUTO-20 Controles de acceso basados en roles.

Para obtener el PAN de su *token* asociado, o acceder a la base de datos (CDV), se deben de requerir controles de acceso basados en roles o *Role-Based Access Controls* (RBACs) por sus siglas en inglés.

REQUTO-21 Cifrado de la base de datos.

Dentro de la base de datos (CDV), los PAN deben ser cifrados con una llave de mínimo 128 bits de fuerza efectiva.

REQUTO-22 Seguridad de la administración de llaves.

Todas la operaciones sobre la administración de las llaves criptográficas deben realizarse en un dispositivo criptográfico seguro y aprobado.

REQUTO-23 Administración de llaves de acuerdo al estándar.

La administración de las llaves criptográficas debe realizarse de acuerdo con el estándar NIST/ISO; por ejemplo el NIST SP 800-57, el ISO/IEC 11770 o el NIST SP 800-130.

Primitivas criptográficas

En esta sección se resumen los requerimientos mínimos que debe de tener cualquier primitiva criptográfica que se use dentro del sistema *tokenizador* (anexo C de [16]).

En la tabla 3.1 se colocan los tamaños mínimos de llaves y modos de operación (sección 2.2.10)

asociados para las primitivas criptográficas de los «algoritmos criptográficos»[1] permitidos. Estos son: AES (sección 2.2.5), RSA (sección 2.1.3), *Elliptic Curve Crypthosystem* (ECC) y *Digital Signature Algorithm* (DSA)/*Diffie-Hellman* (DH). Se hace especial énfasis es que *Triple DES* (TDES) no está permitido.

Algoritmo	**Tamaño de llave**	**Modo de operación**
AES	128	CTR, OCB, CBC, OFB, CFB
RSA	3072	RSAES-OAEP
ECC	256	ECDH, ECMQV, ECDSA, ECIES
DSA/DH	3072/256	DHE

Tabla 3.1: Longitudes de llave mínimas y modos de operación permitidos para algoritmos criptográficos

La tabla 3.2 enlista los algoritmos hash (sección 2.4) permitidos. Para evitar introducir fallas de seguridad a través de las funciones hash, estas deben proveer al menos tantos bits de seguridad como el algoritmo criptográfico usado, y en cualquier caso, no menos de 128 bits (lo que deja fuera a MD4 y MD5).

Bits de seguridad	**Algoritmo hash**
128	SHA-256
128	SHA3-256
192	SHA3-384
256	SHA-512
256	SHA3-512

Tabla 3.2: Algoritmos hash permitidos

El número de bits de entropía utilizados para los generadores de números aleatorios debe de ser mayor o igual al número de bits de seguridad utilizados para las primitivas anteriores. Cuando se utilicen generadores determinísticos, estos deben seguir las recomendaciones del NIST en [21].

3.1.2. Lista de posibles algoritmos

En esta sección se presentan todos los algoritmos para generar *token*s encontrados. Además de una descripción del funcionamiento, cada algoritmo se clasifica según las categorías establecidas por el PCI SSC expuestas en la sección anterior.

[1] El PCI SSC parece dividir a las primitivas criptográficas en *algoritmos criptográficos*, *funciones hash* y *generadores de números pseudoaleatorios*; esto resulta confuso dado que las tres categorías pertenecen al campo de estudio de la criptografía.

A Cryptographic Study of Tokenization Systems

Por claridad de contenidos, se establecen las siglas *ACSTS* para referirse a este algoritmo en futuras referencias.

En [18] se analiza formalmente el problema de la generación de *token*s y se propone un algoritmo que no está basado en *Format Preserving Encryption* (FPE). Hasta antes de la publicación de este documento, los únicos métodos para generar *token*s cuya seguridad estaba formalmente demostrada eran los basados en FPE.

El algoritmo propuesto usa primitivas criptográficas para generar *token*s aleatorios y almacena en una base de datos (CDV) la relación original de estos con los PAN. Es, por tanto, un algoritmo tokenizador reversible, y también, pese a la contradiccón con el nombre, no criptográfico. En el pseudocódigo 3.1 se muestra el proceso de tokenización, mientras que en 3.2 está la detokenización.

```
entrada: PAN p; información asociada d; llave k
salida:  token
inicio
  S1 ← buscarPAN(p)
  S2 ← buscarInfoAsociada(d)
  si S1 y S2 = 0:
    t ← RN(k)
    insertar(t, p, d)
  sino:
    t ← S1
  fin
  regresar t
fin
```

Pseudocódigo 3.1: *ACSTS*, método de tokenización

```
entrada: token t; información asociada d; llave k
salida:  PAN
inicio
  S1 ← buscarToken(t)
  S2 ← buscarInfoAsociada(d)
  si S1 y S2 = 0:
    regresar error
  sino:
    p ← S1
  fin
  regresar p
fin
```

Pseudocódigo 3.2: *ACSTS*, método de detokenización

La mayor parte del proceso de tokenización y toda la detokenización son bastante fáciles de comprender; lo único que queda por esclarecer es la función generadora de *token*s aleatorios RN_k. Idealmente, esta función debe regresar un elemento uniformemente aleatorio del espacio de *token*s. La propuesta que se hace en [18] para instanciar esta función se presenta en el pseudocódigo 3.3. Aquí, la variable *contador* mantiene un estado del algoritmo (mantiene su valor a lo largo de las distintas llamadas); el espacio de *token*s contiene cadenas de longitud fija μ de un alfabeto AL cuya cardinalidad es l; el número de bits necesarios para enumerar a todo el alfabeto se guardan en $\lambda = \lceil \log_2 l \rceil$.

```
entrada: llave k
salida:  token
inicio
  X ← f(k, contador)
  X_1, X_2, ..., X_m ← cortar(X, λ)
  t ← ""
  i ← 0
  mientras |t| ≠ μ:
    si entero(X_i) ≤ l:
      t ← t + entero(X_i)
    fin
    i ← i + 1
  fin
  contador ← contador + 1
  regresar t
fin
```

Pseudocódigo 3.3: *ACSTS*, generación de *token*s aleatorios

En el pseudocódigo 3.3 lo primero que se hace es utilizar una primitiva criptográfica f para generar una cadena binaria (hablaremos sobre este punto más adelante); esta cadena (nombrada X) se parte en subcadenas de λ bits; después se itera de manera consecutiva sobre estas subcadenas binarias, si la representación entera de la i-ésima está en el rango del alfabeto de los *token*s, entonces se concatena al token resultado, sino, se pasa a la siguiente subcadena. La longitud de la cadena regresada por f debe ser, aproximadamente, $3\mu\lambda$: dado que se espera que el comportamiento de f sea equiprobable, entonces el ciclo correra un promedio de 2μ veces.

Existen varios candidatos viables para f: un cifrado de flujo (sección 2.3), pues el flujo de llave de estos produce cadenas de aspecto aleatorio; un cifrado por bloques (sección 2.2), utilizando un modo de operación de contador (sección); un *True Random Number Generator* (TRNG) para obtener secuencias de bits verdaderamente aleatorias.

Por último hay que aclarar que el algoritmo presentado en el pseudocódigo 3.1 debe recibir un par de modificaciones más: al momento de generar un *token* debe existir una validación que verifique que este sea único (para evitar que dos PAN tengan un mismo *token*); la base de datos debe estar cifrada, por lo que,

antes de hacer inserciones y después de hacer consultas, deben existir las operaciones correspondientes.

3.2. API web

3.3. Tienda en línea

Bibliografía

[1] Alfred Menezes, Paul C. van Oorschot y Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996. ISBN: 0-8493-8523-7.

[2] Hans Delfs y Helmut Knebl. *Introduction to Cryptography - Principles and Applications*. Information Security and Cryptography. Springer, 2007. ISBN: 978-3-540-49243-6. DOI: 10.1007/3-540-49244-5. URL: https://doi.org/10.1007/3-540-49244-5.

[3] Claude E. Shannon. "Communication theory - Exposition of fundamentals". En: *Trans. of the IRE Professional Group on Information Theory (TIT)* 1 (1953), págs. 44-47. DOI: 10.1109/TIT.1953.1188568. URL: https://doi.org/10.1109/TIT.1953.1188568.

[4] Whitfield Diffie y Martin E. Hellman. "New directions in cryptography". En: *IEEE Trans. Information Theory* 22.6 (1976), págs. 644-654. DOI: 10.1109/TIT.1976.1055638. URL: https://doi.org/10.1109/TIT.1976.1055638.

[5] Ronald L. Rivest, Adi Shamir y Leonard M. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". En: *Commun. ACM* 21.2 (1978), págs. 120-126. DOI: 10.1145/359340.359342. URL: http://doi.acm.org/10.1145/359340.359342.

[6] Debrup Chakraborty y Francisco Rodríguez-Henríquez. "Block Cipher Modes of Operation from a Hardware Implementation Perspective". En: *Cryptographic Engineering*. Ed. por Çetin Kaya Koç. Springer, 2009, págs. 321-363. ISBN: 978-0-387-71816-3. DOI: 10.1007/978-0-387-71817-0_12. URL: https://doi.org/10.1007/978-0-387-71817-0_12.

[7] William Stallings. *Cryptography and network security - principles and practice (6. ed.)* Pearson, 2014. ISBN: 978-0-13-335469-0.

[8] Alan G. Konheim. *Computer Security and Cryptography*. Wiley, 2007. ISBN: 978-0-471-94783-7.

[9] Scott R. Fluhrer, Itsik Mantin y Adi Shamir. "Weaknesses in the Key Scheduling Algorithm of RC4". En: *Selected Areas in Cryptography, 8th Annual International Workshop, SAC 2001 Toronto, Ontario, Canada, August 16-17, 2001, Revised Papers*. Ed. por Serge Vaudenay y Amr M. Youssef. Vol. 2259. Lecture Notes in Computer Science. Springer, 2001, págs. 1-24. ISBN: 3-540-43066-0. DOI: 10.1007/3-540-45537-X_1. URL: https://doi.org/10.1007/3-540-45537-X_1.

[10] Matthew J. B. Robshaw y Olivier Billet, eds. *New Stream Cipher Designs - The eSTREAM Finalists*. Vol. 4986. Lecture Notes in Computer Science. Springer, 2008. ISBN: 978-3-540-68350-6. DOI: 10.1007/978-3-540-68351-3. URL: https://doi.org/10.1007/978-3-540-68351-3.

[11] Christophe De Cannière y Bart Preneel. "Trivium". En: *New Stream Cipher Designs - The eSTREAM Finalists*. Ed. por Matthew J. B. Robshaw y Olivier Billet. Vol. 4986. Lecture Notes in Computer Science. Springer, 2008, págs. 244-266. ISBN: 978-3-540-68350-6. DOI: 10.1007/978-3-540-68351-3_18. URL: https://doi.org/10.1007/978-3-540-68351-3_18.

[12] Steve Babbage, Christophe De Cannière, Anne Canteaut y col. "The eSTREAM Portfolio (rev. 1)". En: (2008). URL: http://www.ecrypt.eu.org/stream/portfolio_revision1.pdf.

[13] Alaa Hussein Al-Hamami y Ghossoon M. Waleed al-Saadoo. *Handbook of research on threat detection and countermeasures in network security*. 1.ª ed. IGI Global, 2015.

[14] Prakash C. Gupta. *Cryptography and Network Security*. PHI Learning, 2015.

[15] Dhiren R. Patel. *Information security - Theory and Practice*. Prentice-Hall of India, 2008.

[16] Payment Card Industry Security Standards Council. *Tokenization Product Security Guidelines – Irreversible and Reversible Tokens*. 2015. URL: https://www.pcisecuritystandards.org/documents/Tokenization_Product_Security_Guidelines.pdf.

[17] Payment Card Industry Security Standards Council. *Payment Application Data Security Standard - Requirements and Security Assessment Procedures - Version 3.0*. 2013. URL: https://www.pcisecuritystandards.org/minisite/en/docs/PA-DSS_v3.pdf.

[18] Sandra Diaz-Santiago, Lil María Rodríguez-Henríquez y Debrup Chakraborty. "A cryptographic study of tokenization systems". En: *Int. J. Inf. Sec.* 15.4 (2016), págs. 413-432. DOI: 10.1007/s10207-015-0313-x. URL: https://doi.org/10.1007/s10207-015-0313-x.

[19] Payment Card Industry Security Standards Council. *Data Security Standard - Version 3.2*. 2016. URL: https://www.pcisecuritystandards.org/documents/pci_dss_v3-2.pdf.

[20] Elaine Barker. *NIST Special Publication 800-57 - Recommendation for Key Management*. 2016. URL: http://dx.doi.org/10.6028/NIST.SP.800-57pt1r4.

[21] Elaine Barker y John Kelsey. *NIST Special Publication 800-90A - Recommendation for Random Number Generation Using Deterministic Random Bit Generators*. 2015. URL: http://dx.doi.org/10.6028/NIST.SP.800-90Ar1.

[22] Payment Card Industry Security Standards Council. *Data Security Standard (DSS) and Payment Application Data Security Standard (PA-DSS) - Glossary of Terms, Abbreviations, and Acronyms - Version 1.2*. 2008. URL: https://www.pcisecuritystandards.org/pdfs/pci_dss_glossary.pdf.

Glosario

token

Valor representativo que se usa en lugar de información valiosa. 1, 42–50, 67, 69

autenticación de origen

Tipo de autenticación donde se corrobora que una entidad es la fuente original de la creación de un conjunto de datos en un tiempo específico. Por definición, la *autenticación de origen* incluye la integridad de datos, pues cuando se modifican los datos, se tiene una nueva fuente. 1, 39

biyección

Dicho de las funciones que son inyectivas y suprayectivas al mismo tiempo; en otras palabras, que todos los elementos del conjunto de salida tengan una imagen distinta en el conjunto de llegada y a cada elemento del conjunto de llegada le corresponde un elemento del conjunto de salida. 1, 16, 60, *véase también* función

cifrado iterativo

Cifrado de bloque que involucra la repetición secuencial de una función interna llamada función de ronda. Los parámetros incluyen en número de rondas, el tamaño de bloque y el tamaño de llave. 1, 14, *véase también* ronda

codominio

Una función mapea a los elementos de un conjunto A con elementos de un conjunto B; A es el dominio y B es el *codominio*. 1, *véase también* función

computacionalmente no factible

Se dice que una tarea es *computacionalmente no factible* si su costo (medido en términos de espacio o de tiempo) es finito pero ridículamente grande. 1, 42

criptografía fuerte

De acuerdo al PCI SSC (en [22]), es la criptografía basada en algorítmos probados y aceptados en la industria, junto con longitudes de llaves fuertes y buenas prácticas de administración de llaves. 1, 43

criptología

Estudio de los sistemas, claves y lenguajes secretos u ocultos. 1, 8, 9

distribución de probabilidad

Una distribución de probabilidad P en S es una secuencia de números positivos p_1, p_2, ... p_n que sumados dan 1. Donde p_i se interpreta como la probabilidad de s_i de ocurrir. 1

dominio

El *dominio* de una función $f(x)$ es el conjunto de valores para los cuales la función está definida. 1, 58, 60, 61, *véase también* función

entropía

Definida para una función de probabilidad de distribución discreta, mide cuánta información en promedio es requerida para identificar muestras aleatorias de esa distribución. 1, 14, 45, *véase también* función

equiprobable

Distribución de probabilidad en la que todos los elementos tienen la misma ocurrencia. 1, 46, 50, *véase también* distribución de probabilidad

estadísticamente independiente

Dicho de la ocurrencia de dos eventos E_1 y E_2: si $P(E_1 \cap E_2) = P(E_1)P(E_2)$ entonces E_1 y E_2 son *estadísticamente independientes* entre sí. Es importante notar que si esto ocurre, entonces $P(E_1|E_2) = P(E_1)$ y $P(E_2|E_1) = P(E_2)$, es decir, la ocurrencia de uno no tiene ninguna influencia en las probabilidades de ocurrencia del otro. 1, 46, 47, *véase también* probabilidad condicional

fuerza efectiva

Para un espacio de llaves K, su *fuerza efectiva* es $\log_2 |K|$ (el logaritmo base dos de su cardinalidad). 1, 46, 47

función

Regla entre dos conjuntos A y B de manera que a cada elemento del conjunto A le corresponda un único elemento del conjunto B. 1, 58–61

imagen

Suponga que se tiene $x \in X$ y $y \in Y$ tal que $f(x) = y$; se dice entonces que y es la *imagen* de x bajo f, o que x es preimagen de y. 1, 60, 61, *véase también* preimagen

integridad de datos

Propiedad en la que los datos no han sido alterados sin autorización desde que fueron creados, transmitidos o almacenados por una fuente autorizada. Operaciones que insertan, eliminan, modifican o reordenan bits invalidan la *integridad de los datos*. La *integridad de los datos* incluye que los datos estén completos y, cuando los datos son divididos en bloques, cada bloque cumplir con lo mencionado anteriormente. 1, 39

inyectiva

Una función $f : D_f \to C_f$ es *inyectiva* (o uno a uno) si a diferentes elementos del dominio le corresponden diferentes elementos del codominio; se cumple para dos valores cualesquiera $x_1, x_2 \in D_f$ que $x_1 \neq x_2 \implies f(x_1) \neq f(x_2)$. 1, 58, *véase también* función

modo de operación

Construcción que permite extender la funcionalidad de un cifrado a bloques para operar sobre tamaños de información arbitrarios. 1, 13, 25–29, 47, 48, 61, 67–69

máquina de Turing

Se considera como una cinta infinita dividida en casillas, cada una de las cuales contiene un símbolo. Sobre dicha cinta actúa un dispositivo que puede adoptar distintos estados y que, en cada instante, lee un símbolo de la casilla sobre la que está situado; dependiendo del símbolo leído y del estado en el que se encuentra, la máquina realiza las siguientes tres acciones: primero, pasa a un nuevo estado; segundo, imprime un símbolo en el lugar del que acaba de leer; y, tercero, se desplaza hacia la derecha, hacia la izquierda o se detiene. 1

oráculo

Se refiere a una máquina abstracta utilizada para estudiar problemas de decisión. Puede verse como una máquina de Turing con una caja negra (llamada oráculo) que puede resolver ciertos problemas de decisión u obtener el valor de una función en una sola operación. 1, 44, *véase también* máquina de Turing

permutación

Sea S un conjunto finito de elementos. Una *permutación* p en S es una biyección de S a sí misma (i. e. $p : S \to S$). 1, 10, 45, 46, *véase también* biyección

preimagen

Suponga que se tiene $x \in X$ y $y \in Y$ tal que $f(x) = y$; se dice entonces que x es *preimagen* de y, o que y es la imagen de x bajo f. 1, 35, 36, 59, *véase también* imagen

primitiva criptográfica

Algoritmos criptográficos que son usados con frecuencia para la construcción de protocolos de seguridad. En [1] (figura 1.1) se clasifican en tres categorías principales: de llave simétrica, de llave pública y sin llave. 1, 44, 47–50

probabilidad condicional

Sean E_1 y E_2 dos eventos, con $P(E_2) \geq 0$. La *probabilidad condicional* se denota por $P(E_1|E_2)$, y es igual a

$$P(E_1|E_2) = \frac{P(E_1 \cap E_2)}{P(E_2)}$$

Esto mide la probabilidad de que ocurra E_1 sabiendo que ya ocurrió E_2. 1

ronda

Bloque compuesto por un conjunto de operaciones que es ejecutado múltiples veces. Las *rondas* son definidas por el algoritmo de cifrado. 1, 14, 16–18, 21–24, 58

suprayectiva

Una función $f : D_f \to C_f$ es *suprayectiva* si todo elemento de su codominio C_f es imagen de por lo menos un elemento de su dominio D_f: $\forall b \in C_f \ \exists a \in D_f$ tal que $f(a) = b$. 1, 58, *véase también* función

vector de inicialización

Cadena de bits de tamaño fijo que sirve como entrada a muchas primitivas criptográficas (e. g. algunos modos de operación). Generalmente se requiere que sea generado de forma aleatoria. 1, 26–28, 31

Siglas y acrónimos

AES *Advanced Encryption Standard*. 11, 17, 34, 37, 43, 48

ASCII *American Standard Code for Information Interchange*. 36

CBC *Cipher-block Chaining*. 26, 27, 39, 48, 67, 69

CDV *Card Data Vault*. 44, 46, 47, 49

CFB *Cipher Feedback*. 27, 28, 48, 67, 69

CRHF *Collision-Resistant Hash Function*. 36

CTR *Counter Mode*. 48

DES *Data Encryption Standard*. 11, 16, 17, 21, 39, 48, 66

DH *Diffie-Hellman*. 48, 64

DHE *DH Ephemeral*. 48

DSA *Digital Signature Algorithm*. 48, 64

DSS *Data Security Standard*. 43

ECB *Electronic Codebook*. 25, 26, 67, 69

ECC *Elliptic Curve Crypthosystem*. 48

ECDH *Elliptic Curve DH*. 48

ECDSA *Elliptic Curve DSA*. 48

ECIES *Elliptic Curve Integrated Encryption Scheme*. 48

ECMQV *Elliptic Curve Menezes-Qu-Vanstone*. 48

ECRYPT *European Network of Excellence in Cryptology*. 33

FEAL *Fast Data Encipherment Algorithm*. 21

FIPS *Federal Information Processing Standard*. 16

FPE *Format Preserving Encryption*. 49

IDEA *International Data Encryption Algorithm*. 23

IEC *International Electrotechnical Commission*. 44–47

IEEE *Institute of Electrical and Electronic Engineers*. 32

ISO *International Organization for Standaridization*. 44–47

LAN *Local Area Network*. 32

MAA *Message Authenticator Algorithm*. 39

MAC *Message Authentication Code*. 36, 39

MD4 *Message Digest 4*. 37, 39, 48

MD5 *Message Digest 5*. 36, 37, 48

MDC *Message Digest Cipher*. 39

MDC-2 *Modification Detection Code 2*. 36

NESSIE *New European Schemes for Signatures, Integrity and Encryption*. 34

NIST *National Institute of Standards and Technology*. 17, 37, 38, 45, 47, 48

NSA *National Security Agency*. 37

OAEP *Optimal Asymmetric Encryption Padding*. 48

OCB *Offset Codebook*. 48

OFB *Output Feedback*. 29, 31, 48, 67, 69

OWHF *One-Way Hash Function*. 35, 36

PAN *Personal Account Number*. 42–47, 49, 50

PCI *Payment Card Industry*. 42, 43, 48, 58

RAE Real Academia Española. 8

RBACs *Role-Based Access Controls*. 47

RSA *Ron Rivest, Adi Shamir, Leonard Adleman*. 12, 32, 34, 36, 48, 65, 69

RSAES *RSA Encryption Scheme*. 48

SAFER *Secure And Fast Encryption Routine*. 23

SHA *Secure Hash Algorithm*. 36–38, 48

SSC *Security Standard Council*. 42, 43, 48, 58

SSL *Secure Sockets Layer*. 32, 36

TDES *Triple DES*. 48

TLS *Transport Layer Security*. 32

TRNG *True Random Number Generator*. 50

WEP *Wired Equivalent Privacy*. 32, 33

WPA *WiFi Protected Access*. 32

Lista de figuras

Lista de tablas

Lista de pseudocódigos