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
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
$\{0,1\}^r$	Cadena de bits de longitud r
$\{0,1\}^*$	Cadena de bits de longitud arbitraria
mód	Operación módulo
$\oplus$	Operación XOR

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 palabra criptografía proviene de las etimologías griegas *Kriptos* (ocultar) y *Graphos* (escritura), y es definida por la Real Academia Española 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 cumplir 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. 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 así a la siguiente clasificación.

1. **Ciphertext-only attack**

En este ataque el adversario sólo es capaz de obtener la información cifrada, y tratara 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. **Known-plaintext attack**

   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. **Chosen-plaintext attack**

   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. **Adaptively-chosen-plaintext attack**

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

5. **Chosen and adaptively-chosen-ciphertext attack**

   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 usado 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 secretos. Mientras que la criptografía moderna es la que se inició después la publicación de la *Teoría de la información* por Claude Elwood Shannon, 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 siguiente figura.

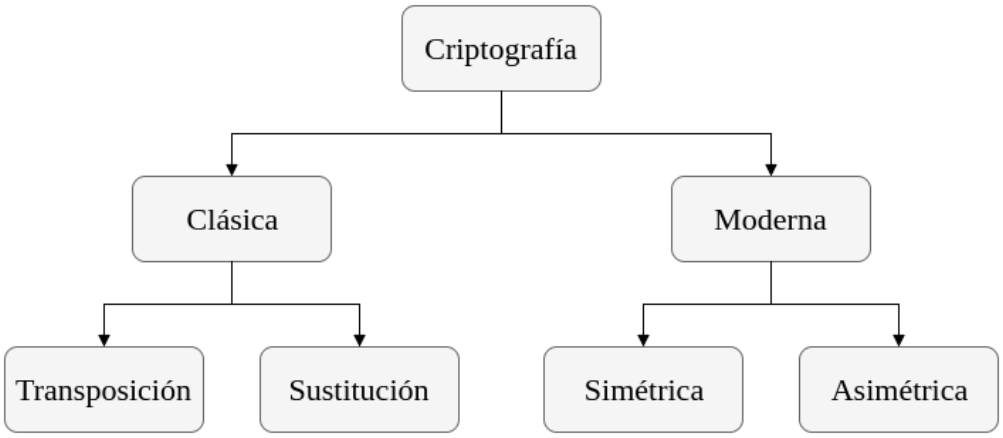


Figura 2.1: Clasificación de la criptografía.

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 correspondiente 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.

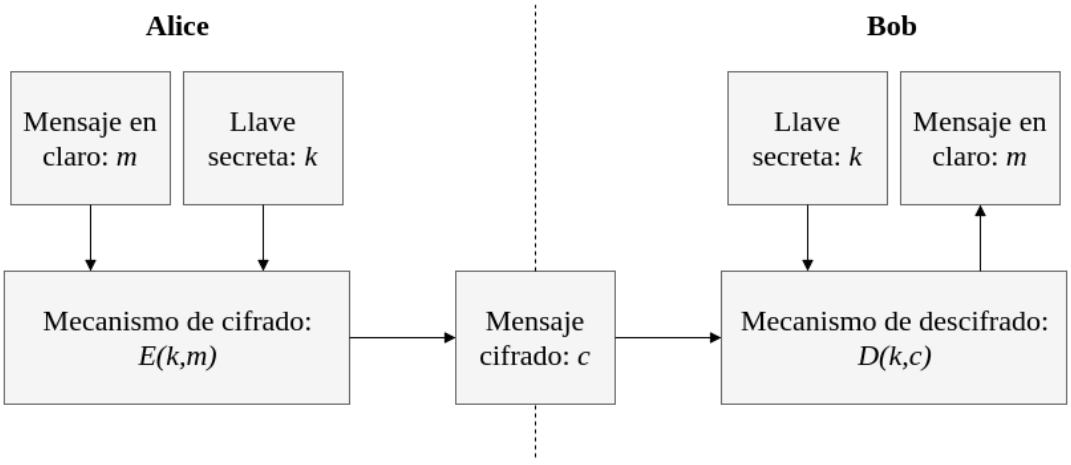


Figura 2.2: Canal de comunicación con criptografía simétrica.

En la figura anterior 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.

Entre los beneficios de este tipo de criptografía está su utilidad para cifrar archivos personales, su relativa facilidad de uso y para garantizar la confidencialidad e integridad debido al uso de una llave, y su rapidez, pero en contraparte, su uso genera problemas para organizar y compartir las llaves secretas de una forma segura y eficiente.

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 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 siguiente figura.

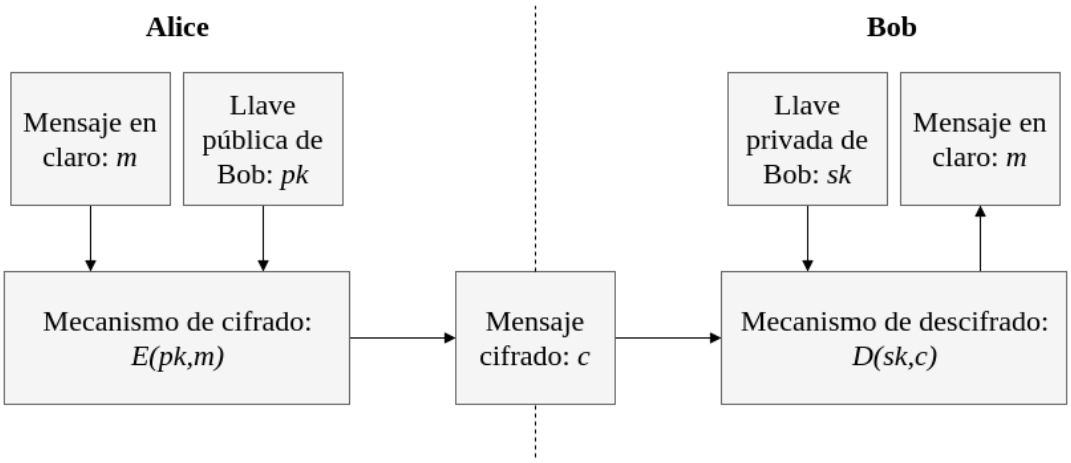


Figura 2.3: Canal de comunicación con criptografía asimétrica.

Entre los uso que se le da a esta criptografía está el mantener la distribución de llaves privada segura, y establecer métodos que garantizan la autenticación y el no repudio, como por ejemplo en las firmas y

certificados digitales.

2.2. Cifrados por bloques

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

$$E : \{0,1\}^r \times \{0,1\}^n \longrightarrow \{0,1\}^n (k,m) \longmapsto E(k,m) \tag{2.1}$$

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. 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 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:

$$L_i = R_{i-1} \tag{2.2}$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i) f(R_{i-1}, K_i) = P(S(E(R_{i-1}) \oplus K_i)) \tag{2.3}$$

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 ... m_64;
         llave de 64 bits K = k_1 ... k_64.
salida:  bloque de texto cifrado de 64 bits C = c_1 ... c_64.
inicio
  Calcular 16 subllaves K_i de 48 bits partiendo de K.
  (L_0, R_0) ← IP(m_1 m_2 ... m_64
  Para i desde 1 hasta 16: Calcular L_i y R_i mediante las ecuaciones
  mostradas anteriormente.
  Obtener f(R_{i-1}, K_i) así:
    a) Expandir R_{i-1} = r_1 r_2 ... r_32 de 32 a 48 bits
       usando E: T ← E(R_{i-1}).
    b) T' ← T ⊕ K_i. Donde T' es representado
       como ocho cadenas de 6 bits cada una (B_1, ..., B_8).
    c) T'' ← (S_1(B_1), S_2(B_2), ... S_8(B_8))
    d) T''' ← P(T'')
  fin
  b_1 b_2 ... b_64 ← (R_16, L16).
  C ← IP^{-1}(b_1 b_2 ... b_64)
fin
```

Pseudocódigo 2.1: 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).

2.2.4. 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 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 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 = c_1 ... c_64.
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, k_0)
Para i desde 1 hasta num_rondas:
  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 2.2: 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.

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.

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).

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:

$$c(X) = 03X^3 + 01X^2 + 01X + 02 \tag{2.4}$$

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

$$a(X) \mapsto a(X) \cdot c(X) mod(X^4 + 1) \tag{2.5}$$

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:

$$(matriz\_estado, k_i) \mapsto matriz\_estado \oplus k_i \tag{2.6}$$

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.

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

2.2.5. 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. 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 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 2.3: 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 $N-$ rondas con N par, aunque se recomienda $N = 2^x$.

2.2.6. 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 la ronda 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 2.4: 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 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.

2.2.7. Secure And Fast Encryption Routine (SAFER)

El cifrado 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 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 2.5: 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.

2.2.8. 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 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 2.6: 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 2.7: RC5, descifrado.

2.2.9. Modos de operación

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[1], 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. Se formaliza este concepto definiendo a un cifrado por bloques como una función C (ecuación 2.7) y a un modo de operación como una función M (ecuación 2.8).

$$C(L, B) \rightarrow Bc \tag{2.7}$$

En donde L es la llave y B es el bloque a cifrar; ambos con un tamaño definido: $L \in \{0,1\}^k$ (k es el tamaño de la llave) y $B \in \{0,1\}^n$ (n es el tamaño de bloque). Bc representa al bloque cifrado, el cuál también tiene longitud n.

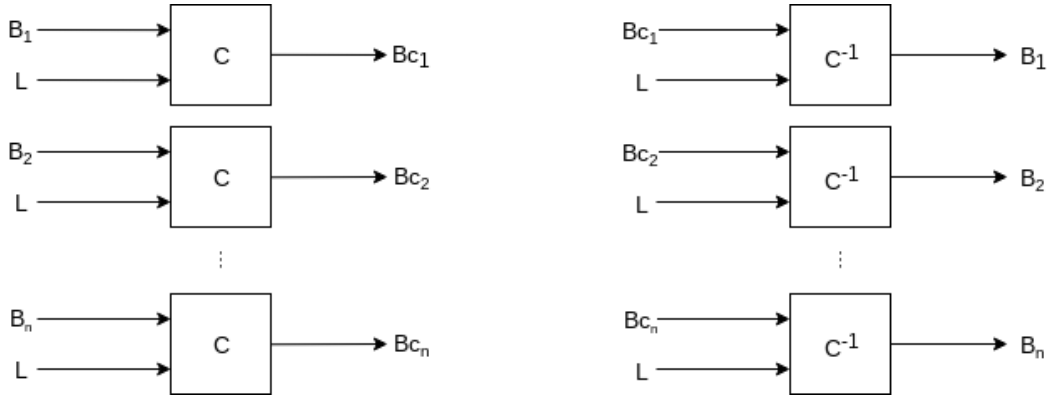
$$M(L, T) \rightarrow Tc \tag{2.8}$$

En este caso L es la misma que en 2.7, T y Tc son el texto original y el texto cifrado, respectivamente, y ambos son de longitud arbitraria: $T, Tc \in \{0,1\}^*$.

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 presenta 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 [1]. También enlistamos otros tres modos, los cuales junto con ECB, son los más comunes.

Electronic Codebook (ECB)

La figura 2.4 muestra un diagrama esquemático de este modo de operación. Según la ecuación 2.8, 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 ($T = B_1||B_2||\ldots||B_n$).



(a) Cifrado. (b) Descifrado.

Figura 2.4: Modo de operación ECB.

```
entrada: llave L; bloques de mensaje B_1, B_2 ... B_n.
salida: bloques de mensaje cifrado Bc_1, Bc_2 ... Bc_n.
inicio
  para_todo B
    Bc_i ← C(L, B_i)
  fin
  regresar Bc
fin
```

Pseudocódigo 2.8: Modo de operación ECB, cifrado.

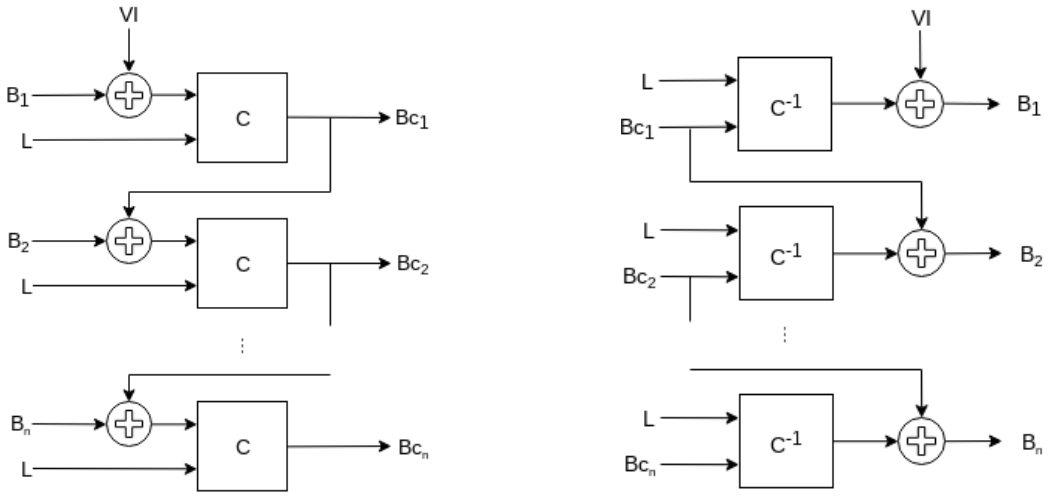
```
entrada: llave L; bloques de mensaje cifrado Bc_1, Bc_2 ... Bc_n.
salida: bloques de mensaje original B_1, B_2 ... B_n.
inicio
  para_todo Bc
    B_i ← C^-1(L, Bc_i)
  fin
  regresar B
fin
```

Pseudocódigo 2.9: Modo de operación ECB, descifrado.

Cipher-block Chaining (CBC)

En 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 2.5: Modo de operación CBC.

En la figura 2.5 se muestran los diagramas esquemáticos para cifrar y descifrar; en los pseudocódigos 2.10 y 2.11 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 L; vector de inicialización VI;
         bloques de mensaje B_1, B_2 ... B_n.
salida:  bloques de mensaje cifrado Bc_1, Bc_2 ... Bc_n.
inicio
  Bc_0 ← VI                    // El vector de inicialización
  para_todo B                  // entra al primer bloque.
    Bc_i ← C(L, B_i ⊕ Bc_{i-1})
  fin
  regresar Bc
fin
```

Pseudocódigo 2.10: Modo de operación CBC, cifrado.

```
entrada: llave L; vector de inicialización VI;
         bloques de mensaje cifrado Bc_1, Bc_2 ... Bc_n.
salida:  bloques de mensaje original B_1, B_2 ... B_n.
inicio
  Bc_0 ← VI
```

```
    para_todo Bc
      B_i ← C^{-1}(L, Bc_i) ⊕ Bc_{i-1}
    fin
    regresar B
fin
```

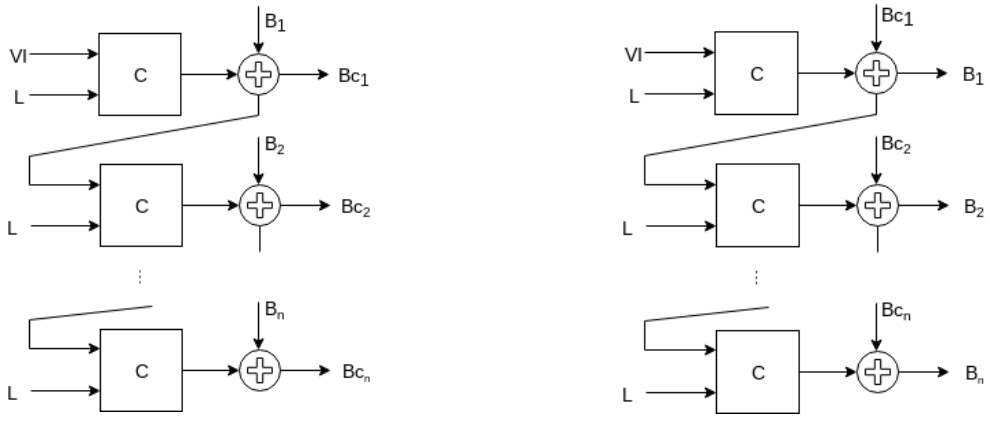
Pseudocódigo 2.11: Modo de operación CBC, descifrado.

Cipher Feedback (CFB)

Al igual que la operación de cifrado de CBC, ambas operaciones de 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 2.6).

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 2.12); 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 2.9).

$$A \oplus B = C \quad \Rightarrow \quad A = B \oplus C \tag{2.9}$$



(a) Cifrado. (b) Descifrado.

Figura 2.6: Modo de operación CFB.

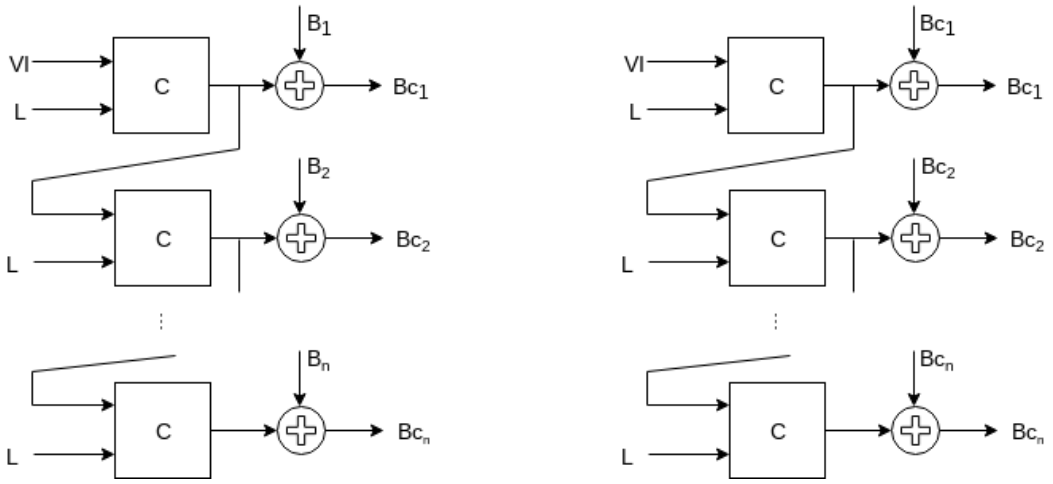
```
entrada: llave L; vector de inicialización VI;
         bloques de mensaje (cifrado o descifrado) B_1, B_2 ... B_n.
```

```
salida: bloques de mensaje (cifrado o descifrado) Bc_1, Bc_2 ... Bc_n.
inicio
  Bc_0 ← VI
  para_todo B
    Bc_i ← C(L, Bc_{i-1}) ⊕ B_i
  fin
  regresar Bc
fin
```

Pseudocódigo 2.12: 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 2.7: Modo de operación OFB.

```
entrada: llave L; vector de inicialización VI;
         bloques de mensaje (cifrado o descifrado) B_1, B_2 ... B_n.
salida: bloques de mensaje (cifrado o descifrado) Bc_1, Bc_2 ... Bc_n.
inicio
  auxiliar ← VI
  para_todo B
    auxiliar ← C(L, auxiliar)
    Bc_i ← auxiliar ⊕ B_i
```

```
    fin
    regresar Bc
fin
```

Pseudocódigo 2.13: Modo de operación OFB(cifrado y descifrado).

2.3. Cifrados de flujo

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 [2].

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.

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

2.3.1. 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.

$$e_{i+1} = f(e_i, L) \tag{2.10}$$

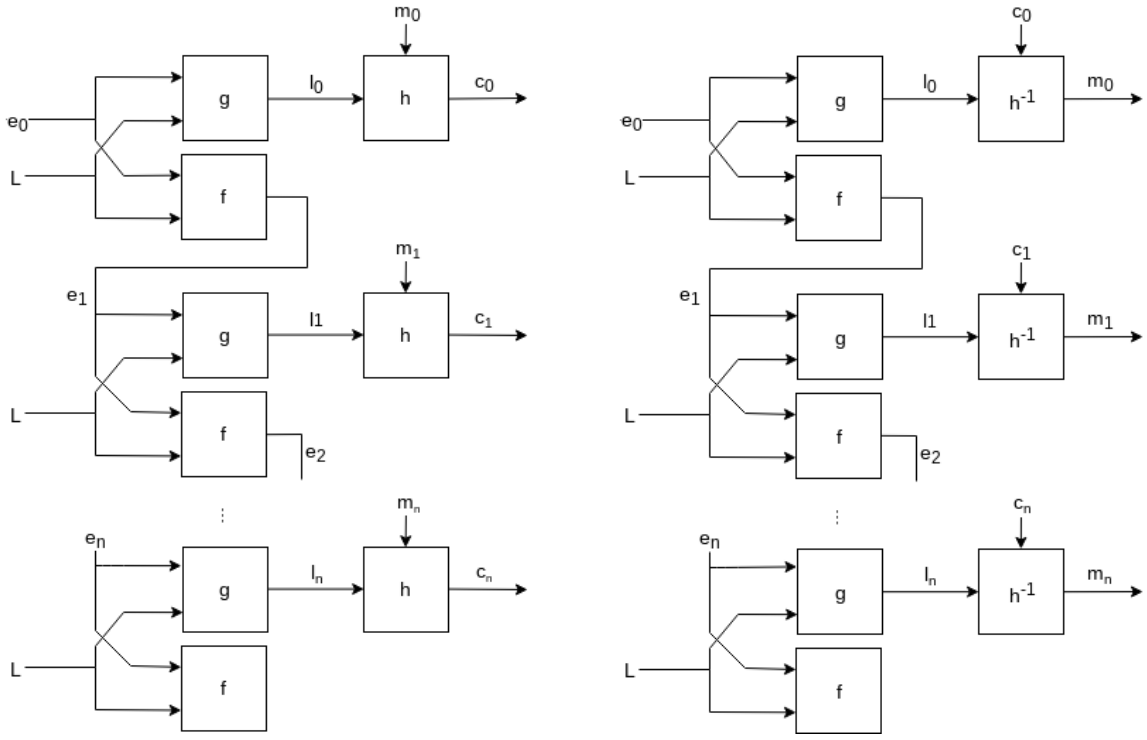
$$l_i = g(e_i, L) \tag{2.11}$$

$$c_i = h(l_i, m_i) \tag{2.12}$$

La letra e representa el estado del cifrado,L es la llave, l es la salida del flujo de llave, c es el texto cifrado y m es el texto en claro. La función de la ecuación 2.10 (f) es la que describe el cambio de estado; este se determina a partir del estado actual y de la llave. En la ecuación 2.11 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 (2.12, h) describe la acción de combinar el flujo de la llave con el mensaje, y así obtener el texto

cifrado.

En la figura 2.8 se describe de manera gráfica las operaciones de cifrado y descifrado; estas guardan muchas similitudes con el modo de operación OFB, 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 2.8: 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.

2.3.2. 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

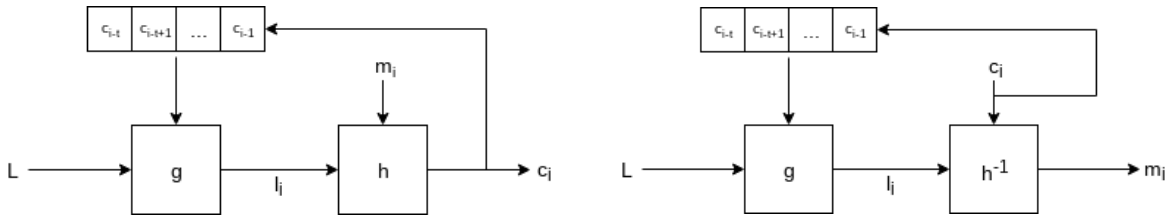
son las siguientes.

$$e_{i+1} = (c_{i-t}, c_{i-t+1}, \ldots, c_{i-1}) \tag{2.13}$$

$$l_i = g(e_i, L) \tag{2.14}$$

$$c_i = h(l_i, m_i) \tag{2.15}$$

La notación es la misma que en las ecuaciones 2.10, 2.11 y 2.12. En este caso, el próximo estado depende de t (el tamaño de la ventana) dígitos cifrados anteriormente. En la figura 2.9 se describe de manera gráfica el proceso de cifrado y descifrado.



(a) Cifrado. (b) Descifrado.

Figura 2.9: 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.

2.4. Funciones hash

La información presentada a continuación puede consultarse con más profundidad en las siguientes referencias **hash˙hussein**, **DBLP:series/isc/DelfsK07**, [2]

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

$$\begin{aligned} h : \{0,1\}^* &\longrightarrow \{0,1\}^n \\ m &\longmapsto h(m) \end{aligned} \tag{2.16}$$

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 OWHF (*one-way hash function*). Las

que cumplen con la resistencia de segunda preimagen y resistencia a las colisiones son conocidas como funciones hash resistentes a colisiones o CRHF (*collision resistant hash function*). Aunque casi siempre las funciones CRHF cumplen con la resistencia de preimagen, no es obligatorio que lo hagan.

Algunos ejemplos de las funciones hash criptográficas son el SHA-1 y el 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 (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 (MDC, por sus siglas en inglés).

2.4.2. Autenticación de mensajes

Otra importante aplicación de las funciones hash es la autenticación del mensaje. Si una función hash es utilizada para la autenticación de mensajes, es llamada código de autenticación de mensajes (MAC, por sus siglas en inglés). MAC es una técnica simétrica estándar muy utilizada para autenticar y proteger la integridad de un mensaje. Dependen de una llave secreta que tienen las partes interesadas y, al contrario de las firmas electrónicas donde solo una persona conoce la llave privada y es capaz de generar la firma, cada uno de los participantes puede producir el MAC válido para un mensaje. La llave privada k es utilizada para parametrizar la función hash.

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

2.4.3. 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 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):

$$\sigma = h(m)^d \quad mod \quad n \tag{2.18}$$

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.

2.4.4. Message Digest-4 (MD4)

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

2.4.5. 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 en 1996 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.

2.4.6. *Secure Hash Algorithm (SHA)*

El algoritmo SHA fue publicado por NIST y 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.

Bibliografía

[1] 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.

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

Glosario

modo de operación

Construcción que permite extender la funcionalidad de un cifrado a bloques para operar sobre tamaños de información arbitrarios. 13, 22–27, 41, 43

ronda

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

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. 23, 25, 26

Siglas

AES

Advanced Encryption Standard. 16, 34

CBC

Cipher-block Chaining. 23–25, 41, 43

CFB

Cipher Feedback. 25, 26, 41, 43

DES

Data Encryption Standard. 14–16, 18

ECB

Electronic Codebook. 22–24, 41, 43

FEAL

Fast Data Encipherment Algorithm. 18

IDEA

International Data Encryption Algorithm. 20

MD4

Message Digest 4. 33

MD5

Message Digest 5. 33

NIST

National Institute of Standards and Technology. 33, 34

NSA

National Security Agency. 33

OFB

Output Feedback. 26, 27, 41, 43

SAFER

Secure And Fast Encryption Routine. 20

SHA

Secure Hash Algorithm. 33, 34

Lista de figuras

Lista de tablas

Lista de pseudocódigos