UNIVERSIDAD AUTÓNOMA METROPOLITANA

UNIVERSIDAD AUTÓNOMA METROPOLITANA - IZTAPALAPA
DIVISIÓN DE CIENCIAS BÁSICAS E INGENIERÍA

Cifrados que preservan formato

Tesis que presenta
Cristina Vargas Puente
Para obtener el grado de
Maestra en Ciencias (Matemáticas Aplicadas e Industriales)

Asesor: **Dr. Horacio Tapia Recillas**

Jurado calificador:

Presidente: Dr. Horacio Tapia Recillas.

Secretario: Dr. José Noé Gutiérrez Herrera.

Vocal: Dra. Sandra Díaz Santiago.

Ciudad de Mexico, 24 de Noviembre del 2017.

Casa abierta al tiempo
UNIVERSIDAD AUTÓNOMA METROPOLITANA

ACTA DE EXAMEN DE GRADO

No. 00172
Matrícula: 2143805852

CIFRADOS QUE PRESERVAN FORMATO

Cristina V. P.

CRISTINA VARGAS PUENTE
ALUMNA

REVISÓ

LIC. JULIO CESAR DE LARA ISASSI
DIRECTOR DE SISTEMAS ESCOLARES

En la Ciudad de México, se presentaron a las 11:00 horas del día 24 del mes de noviembre del año 2017 en la Unidad Iztapalapa de la Universidad Autónoma Metropolitana, los suscritos miembros del jurado:

DR. HORACIO TAPIA RECILLAS
DRA. SANDRA DIAZ SANTIAGO
DR. JOSE NOE GUTIERREZ HERRERA

Bajo la Presidencia del primero y con carácter de Secretario el último, se reunieron para proceder al Examen de Grado cuya denominación aparece al margen, para la obtención del grado de:

MAESTRA EN CIENCIAS (MATEMÁTICAS APLICADAS E INDUSTRIALES)

DE: CRISTINA VARGAS PUENTE

y de acuerdo con el artículo 78 fracción III del Reglamento de Estudios Superiores de la Universidad Autónoma Metropolitana, los miembros del jurado resolvieron:

Aprobar

Acto continuo, el presidente del jurado comunicó a la interesada el resultado de la evaluación y, en caso aprobatorio, le fue tomada la protesta.

DIRECTOR DE LA DIVISIÓN DE CBI

DR. JOSE GILBERTO CORDOBA HERRERA

PRESIDENTE

DR. HORACIO TAPIA RECILLAS

VOCAL

Díaz Santiago

DRA. SANDRA DIAZ SANTIAGO

SECRETARIO

JNoé Gtz H

DR. JOSE NOE GUTIERREZ HERRERA

VIGENCIA

UNIVERSIDAD AUTÓNOMA METROPOLITANA

UNIVERSIDAD AUTÓNOMA METROPOLITANA - IZTAPALAPA
DIVISIÓN DE CIENCIAS BÁSICAS E INGENIERÍA

Cifrados que preservan formato

Tesis que presenta
Cristina Vargas Puente
Para obtener el grado de
Maestra en Ciencias (Matemáticas Aplicadas e Industriales)

Asesor: **Dr. Horacio Tapia Recillas**

Jurado calificador:

Presidente: Dr. Horacio Tapia Recillas.

Secretario: Dr. José Noé Gutiérrez Herrera.

Vocal: Dra. Sandra Díaz Santiago.

Ciudad de Mexico, 24 de Noviembre del 2017.

Agradecimientos

Quiero expresar mis más sinceros agradecimientos a:

Mi familia: Margarita, José Inés, Cesar e Ismael por su incondicional apoyo y, en especial a mi hermano Omar y cuñada Yuri que gracias a su constante motivación pude concluir esta etapa de mi vida.

Mis amigos: Jazmín, Victor, Islette, Ivan, Mario, Liliana, Genaro y Norberto que siempre estuvieron al pendiente de mis progresos, así como sus incondicionales muestras de apoyo. También a aquellos amigos que conocí durante estos dos años: Adriana, Erika, Carlos, Alejandro y Gabriel por los ánimos para seguir con la realización de este trabajo.

Mi asesor el Dr. Horacio Tapia Recillas, por haberme dado la oportunidad de trabajar con él, por su guía y paciencia durante el desarrollo del presente trabajo.

M. en C. Jesus Adolfo Torres Chazaro por su apoyo durante mi primer trimestre de Maestría, así como al Dr. Noé Gutiérrez Herrera y la Dra. Sandra Díaz Santiago por tomarse el tiempo para leer y hacer correcciones a la versión final de este trabajo.

CONACYT, por la beca No. 570918 que me fue proporcionada, porque gracias a ella, fue posible la realización de mis estudios de Maestría así como el desarrollo de este trabajo.

Por último pero no menos importante, a la UAM-I así como el Departamento de Matemáticas, por brindarme todo lo necesario durante mis estudios de Maestría, como por ejemplo la sala de estudio y el laboratorio de Criptografía.

Índice general

Introducción

La introducción en la vida cotidiana de redes de comunicación, en particular de internet, ha abierto nuevas posibilidades para el intercambio de información. Al mismo tiempo, son cada vez mayores las amenazas a la seguridad de la información que se transmite. Es necesario entonces, crear diferentes mecanismos, dirigidos a garantizar la confidencialidad, el no repudio y autenticación de los documentos transmitidos.

Ya que se tienen distintas amenazas a la información cuando ésta es transmitida por medio de redes de comunicación, es importante tomar medidas para enfrentarlas. La seguridad de la información se clasifica en dos grupos:

1. Seguridad física: es la protección del sistema ante amenazas físicas como son: incendios, inundaciones, edificios, cables, etc.

2. Seguridad lógica: es la protección de la información en su propio medio, mediante el enmascaramiento de la misma.

Siendo la seguridad lógica la que nos interesa, pues es en éste aspecto donde se encuentran la *autenticación*, *confidencialidad* y *el no repudio*. Mientras que la autenticación es una técnica que permite la corroboración de la identidad de una entidad ya sea una *persona, computadora, empresa, etc.*, la confidencialidad se refiere a la posibilidad de mantener un documento electrónico inaccesible a todos, excepto a un conjunto de entidades autorizadas.

Por otro lado, el no repudio puede ser respecto al *origen* o *destino*, pues el intercambio de la información involucra a un emisor (origen) y a un receptor (destino). El *no repudio del origen*, es en el que, la persona que envía el mensaje no puede negar ser el emisor de éste, pues el receptor cuenta con las pruebas del envío, mientras que el *no repudio del destino*, es cuando el receptor no puede negar haber recibido el mensaje, porque el emisor cuenta con las pruebas de recepción del mensaje.

Las tres propiedades anteriores son abordadas comúnmente con *técnicas criptográficas* que pueden ser métodos estándar de cifrado como: AES, TDES, DES, funciones

hash, RSA, ElGamal y curvas elípticas. El uso de técnicas criptográficas tiene como propósito incrementar la seguridad en un sistema computarizado. La seguridad, en general, se considera como un aspecto de gran importancia en cualquier entidad que trabaje con sistemas computarizados.

Actualmente parte de la información que se maneja en diversas instancias como por ejemplo las empresas, es almacenada en bases de datos[1] donde el almacenamiento es realizado en tres fases:

1. Extracción de datos de una fuente[2] en particular.
2. Almacenamiento de los datos extraídos en la base de datos.
3. Consulta de la base de datos para obtener información que servirá de ayuda en la toma de decisiones.

Y puesto que los datos de estas bases están en constante riesgo, debido a que existen diversos factores que hacen a los almacenes de datos susceptible a ataques pues:

1. La extracción de datos mediante técnicas usuales, que por lo general son inseguras normalmente es transmitida mediante canales de comunicación también inseguros.
2. El proceso de extracción de datos produce archivos intermedios y carga archivos que contienen información sensible, pero puede que no este bien protegida.
3. Los usuarios a menudo recuperan datos del almacén y crean un "mercado de datos" lo cual conduce a la distribución de copias con datos sensibles.

Es por ello que Brightwell y Smith [1] desarrollan un enfoque de cifrado para la seguridad de las bases de datos que debería de cumplir con ciertos objetivos, algunos de los cuales son:

1. El enfoque debe funcionar con cualquier combinación de bases de datos que esten relacionadas.
2. Correcto funcionamiento en múltiples plataformas tanto de hardware como software.

[1] Una base de datos es un conjunto de datos que pertenecen a un mismo contexto y son almacenados sistemáticamente para su posterior uso.

[2] Se le conoce como fuente a una computadora o conjunto de computadoras donde se encuentra almacenada la información.

3. Debe cifrar y descifrar los datos correctamente en plataformas con diferentes sistemas operativos.

4. Además del cifrado a una base de datos existente, no se debe requerir ninguna cambio en la estructura de la misma.

5. No debe ser dependiente de algún lenguaje de programación en particular.

6. Debe ser a prueba de fallas, es decir, cualquier falla que pudiera suscitarse debe ser tal que el acceso a los datos sea denegado.

Debido que al utilizar los cifrados simétricos, como son el DES y AES el tamaño del dato cifrado es modificado, las aplicaciones en donde se requiere que esto no suceda, como es el caso de las bases de datos cuyos campos están destinados a un tipo de información, generará un gasto extra. Surge así la necesidad de crear cifrados que no modifiquen el tamaño ni cambien el formato del dato.

Brightwell y Smith [1] al parecer son los primeros autores que describen con claridad y de manera más general el problema de cifrado preservando formato (Format Preserving Encryption; FPE) y sus utilidades. Ellos lo llaman *cifrado preservando el tipo*[3] *de dato*, pero Black y Rogaway [2] fueron los que llevaron el problema a la atención de la comunidad criptográfica en 2002, proporcionando definiciones y una serie de soluciones.

Bellare, Ristenpart, Rogaway y Stegers [3] aportaron un enfoque amplio del problema de cifrado que preserva formato, incluyendo definiciones y soluciones empleando lo que ellos llaman redes de Feistel tipo-1 y tipo-2.

En los últimos años el cifrado que preserva formato se ha desarrollado como una herramienta de gran utilidad en criptografía aplicada. Debido a que el objetivo de este tipo de cifrado es: mediante el uso de una llave simétrica[4] K realiza un cifrado determinista (cifrado AES [10]) en X, un texto[5] en claro, para obtener un texto cifrado[6] Y, el cual tiene el mismo formato que X pero no revela información alguna sobre el

[3]Al decir tipo de dato nos estamos refiriendo a la estructura del dato, como puede ser numérico, alfabético, alfanumérerico entre otros.

[4]Secuencia de caracteres y desde el punto de vista computacional una secuencia de bits y/o bytes.

[5]Al decir texto nos estamos refiriendo a un dato (mensaje) cualquiera, recordando que visto desde el punto de vista computacional pueden ser bits y/o bytes. Por lo que al decir texto en claro nos estamos refiriendo al dato (mensaje) original que se desea proteger.

[6]El texto cifrado es el resultado de aplicar un método criptográfico y en éste caso un método determinista. En general el proceso de cifrado es el resultado de aplicar un método criptográfico que cumpla con mantener el dato (mensaje) protegido.

texto sin cifrar más allá de su formato.

El objetivo general de este trabajo es realizar un análisis del esquema de cifrado que preserva formato, las técnicas de éste tipo de cifrado, así como la aplicación de este esquema en una base de datos.

Este trabajo está distribuido de la siguiente manera: En el capítulo I se describe de manera general los esquemas de cifrado: simétricos y asimétricos, así como las funciones hash. Se hace una descripción más detalla en los cifrados simétricos por bloque, pues constituyen la base de los esquemas de cifrado que preservan formato. El capítulo II describe las técnicas principales de los cifrados que preservan formato que son: cifrado de prefijo (Prefix cipher), caminata cíclica (Cycle walking) y, red de Feistel (Feistel Network). En el capítulo III se da la descripción de la aplicación computacional del cifrado que preserva formato, desarrollada para una base de datos cuyos campos contienen número de tarjeta de crédito/débito, número identificador OCR que viene detrás de la credencial para votar emitida por el Instituto Nacional Electoral y número telefónico, sin embargo esta aplicación también se puede llevar a cabo para bases de datos con un mayor número de campos. El capítulo IV contiene la descripción del proceso de tokenización, que es una aplicación de los cifrados que preservan formato. Se presentan las conclusiones y el trabajo a futuro en el capítulo V. Además se cuenta con tres Apéndices, en el apéndice A se realiza un análisis de cómo es que se construye un número de tarjeta de crédito/débito, en el apéndice B se describe la generación de las llaves K_2, K_3 incluyendo el código fuente en lenguaje C para la generación de las mismas, finalmente en el apéndice C se muestran los pseudocódigos del proceso de cifrado y descifrado, con la finalidad de que el lector pueda comprender mejor lo realizado en la aplicación computacional.

Capítulo 1

Esquemas criptográficos

En este capítulo se describen en forma breve los cifrados *simétricos* (y cifrados por bloque simétricos con ajuste), *asimétricos* y las funciones hash, haciendo más énfasis en los *cifrados simétricos por bloque* (y cifrados simétricos por bloque con ajuste), pues son éstos la base de los cifrados que preservan formato. Sin embargo ya que tanto los cifrados asimétricos como las *funciones hash* son importantes para algunas aplicaciones y a lo largo de este trabajo se hará referencia a ellos, se describen de forma general cada uno, y se mencionan algunos ejemplos de donde son utilizados.

1.1. Cifrados simétricos por bloque

Como se mencionó al inicio del capítulo, los cifrados simétricos por bloque como por ejemplo, los cifrados estándar: DES [8] y AES [10], son la base en la construcción de los esquemas de cifrado que preservan formato, éstos son llamados también cifrados de *llave privada* pues utilizan la misma llave [1] para cifrar como para descifrar.

En los cifrados simétricos por bloque [33], el mensaje en claro es dividido en secuencias de caracteres de longitud fija (bloques), cada uno de estos bloques es convertido en un bloque del mensaje cifrado con la misma longitud que del mensaje en claro, mediante un mecanismo que dependen de la *llave*.

Lo anterior se puede expresar de la siguiente manera: dadas las entradas; un mensaje en claro $X \in \{0,1\}^n$ y una llave $k \in \{0,1\}^k$ se produce la salida $Y \in \{0,1\}^n$, quedando así la función 1.1.

[1] En [14] se puede obtener mayor información de como se generan las llaves tanto para cifrados simétricos como asimétricos éstos últimos descritos en 1.3.

$$E : \{0,1\}^k \times \{0,1\}^n \longrightarrow \{0,1\}^n \tag{1.1}$$

Cuando se desea cifrar mensajes más largos que el tamaño del bloque, se utiliza un *modo de operación* [25]. Un *modo* comúnmente combina un algoritmo de cifrado con alguna forma de retroalimentación (*feedback*) y una serie de operaciones, siendo éstas simples pues la seguridad recaerá sobre el algoritmo y no el modo que se está empleando. En la sección 1.1.1 se mencionan los modos que son normalmente empleados cuando se realiza cifrado por bloques.

Los cifrados por bloque son por naturaleza deterministas, es decir, para un mensaje en claro dado, siempre que se utilice la misma llave, se obtendrá el mismo mensaje cifrado.

1.1.1. Modos de operación de los cifrados por bloque

Es importante mencionar algunas de las características generales a considerar respecto los modos de operación que se describirán a lo largo de ésta sección, está información fue obtenida de [25].

Una característica importante, es que las especificaciones del cifrado por bloque simétrico así como de los modos de operación son públicos, por lo cual la seguridad del modo depende en un principio de mantener la llave en secreto.

Debido a que se estará haciendo referencia a la longitud de los bloques de entrada y salida, ésta será denotada con b.

Además del mensaje en claro para los modos de operación CBC (Cipher Block Chaining), CFB (Cipher Feedback) y OFB (Output Feedback) se emplea un vector de inicialización (Initialization Vector; IV) que es utilizado al inicio del cifrado y del correspondiente cifrado del mensaje.

El IV no necesariamente debe ser secreto, pero para los modos CBC y CFB debe ser impredecible. Existen dos métodos recomendados para poder lograrlo, el primero, es aplicar la función de cifrado con la misma llave que se usa para cifrar, a un *nonce* [2].

[2]El *nonce* es un bloque de datos único para cada ejecución de la función de cifrado, por ejemplo, puede ser un contador o un número de mensaje.

Mientras que el segundo método es generar un bloque de datos aleatorios, utilizando un generador de números aleatorios aprobado por FIPS [28].

Por otra parte para el modo OFB un *único* IV debe ser utilizado para cada ejecución, es por ello que el IV o la información suficiente para calcularlo debe ser accesible para cada una de las partes que lleva a cabo la comunicación. Si se llegará a utilizar el mismo IV en este modo, para más de un mensaje, entonces la *confidencialidad* de esos mensajes se puede ver comprometida.

Es importante mencionar que, para poder implementar alguno de estos modos de operación, generalmente se examinará que el IV sea impredecible o único según el modo que se deseé implementar.

Dicho lo anterior, el cifrado de cada uno de los bloques o un segmento de bloque del mensaje en claro dará como resultado bloques o segmentos de bloque de mensaje cifrado.

1. **Modo ECB (Electronic Codebook)**

   El modo ECB es el más sencillo, pues en éste el mensaje en claro es dividido en bloques para que posteriormente cada uno de ellos sea cifrado de forma independiente. En este modo el número total de bits en el mensaje en claro es un múltiplo del tamaño del bloque b, es decir, para un entero n, el número de bits en el mensaje en claro es nb.

   Una desventaja de este modo es que si existen bloques idénticos de mensaje en claro, el bloque del mensaje cifrado será el mismo. Por esta razón se dice que este modo no proporciona confidencialidad y el uso de éste no es recomendado para su uso criptográfico.

   El nombre hace referencia a que un bloque de entrada siempre corresponderá al mismo bloque de salida, siempre y cuando se ha utilizado la misma llave, con lo cual se puede hacer la construcción de lo que se conoce como *libro de códigos*, que contiene la correspondencia entre el mensaje cifrado y el mensaje en claro.

   En la figura 1.1 así como en las ecuaciones 1.2 y 1.3, se muestra el proceso de cifrado y descifrado respectivamente del modo ECB, donde X_j y Y_j corresponden al mensaje en claro y cifrado respectivamente mientras que la E_K y D_K

son las funciones de cifrado y descifrado con llave K fija.

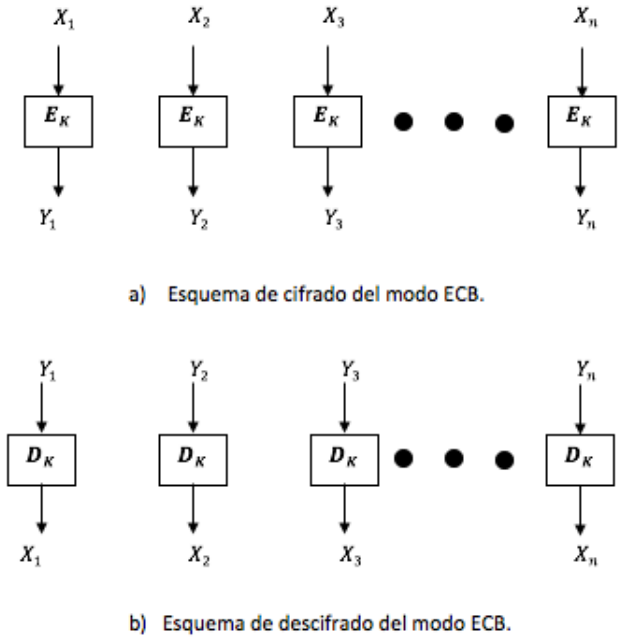


Figura 1.1: Esquemas de cifrado y descifrado del modo ECB

$$Y_j = E_K(X_j) \quad para\ j = 1, 2, \ldots, n. \tag{1.2}$$

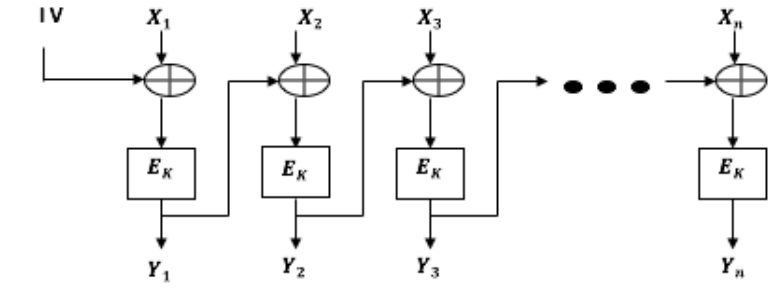
$$X_j = D_K(Y_j) \quad para\ j = 1, 2, \ldots, n \tag{1.3}$$

2. **Modo CBC (Cipher Block Chaining)**

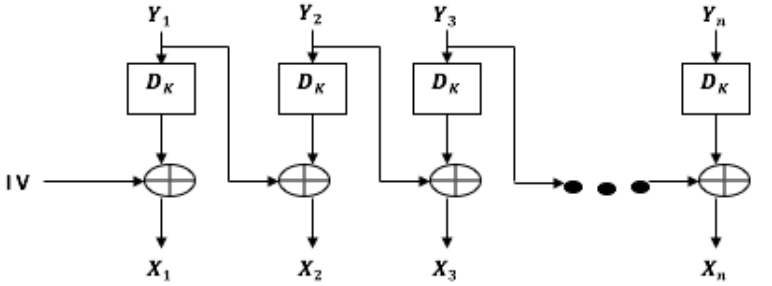
En el modo CBC, el bloque cifrado será utilizado para el proceso de cifrado del siguiente bloque, por lo que cada bloque cifrado no solo depende del anterior bloque cifrado inmediato, sino de todos los bloques cifrados previos. En este modo se aplica la operación XOR entre el bloque del mensaje en claro y el bloque cifrado previo. Para el primer bloque se utiliza un vector de inicialización (IV) con lo que se logra que el mensaje cifrado sea único.

Al igual que en el modo ECB el número total de bits en el mensaje en claro es un múltiplo del tamaño del bloque b. En la figura 1.2 así como en las ecuaciones

1.4 y 1.5 se muestra el proceso de cifrado y descifrado respectivamente, donde IV representa el vector de inicialización, X_j el mensaje en claro, Y_j el mensaje cifrado, con $j = 1, 2, \ldots, n$. E_K y D_K las funciones de cifrado y descifrado respectivamente:



a) Cifrado del modo CBC.



b) Descifrado del modo CBC.

Figura 1.2: Esquemas de cifrado y descifrado del modo CBC

$$\begin{aligned} Y_1 &= E_K(X_1 \oplus IV) \\ Y_j &= E_K(X_j \oplus Y_{j-1}) \quad para\ j = 2, \ldots, n \end{aligned} \tag{1.4}$$

$$\begin{aligned} X_1 &= D_K(Y_1) \oplus IV \\ X_j &= D_K(Y_j) \oplus Y_{j-1} \quad para\ j = 2, \ldots, n \end{aligned} \tag{1.5}$$

3. **Modo CFB (Cipher Feedback)**

   Este modo es de confidencialidad y permite el uso de cifrados por bloques como si fueran cifrados en flujo[3] con lo cual la información puede ser cifrada con longitudes menores a la de los bloques. Al igual que sucede con los cifrados en flujo, si se intercambia o modifica un bit en el mensaje cifrado este bit será modificado en el mensaje en claro.

   A diferencia de los modos descritos previamente, en éste, el número total de bits en el mensaje en claro es un múltiplo de s, donde $1 \leq s \leq b$, es decir, el número total de bits en el mensaje en claro es sb. Por lo que cada segmento del mensaje en claro y mensaje cifrado consiste de s bits, además el valor de s es incorporado al nombre del modo, es decir, 1-bit CFB, 8-bit CFB, 64-bit CFB o 128-bit CFB.

   En el proceso de cifrado, el primer bloque de entrada es el vector de inicialización (IV) al cual se le aplica la función de cifrado (E_K) para de esta manera obtener el primer bloque de salida.

   El primer segmento del mensaje cifrado se produce al realizar la operación XOR entre el primer segmento del mensaje en claro con los s bits más significativos del primer bloque de salida (los $b - s$ bits restantes son descartados). Los $s - b$ bits menos significativos del IV son concatenados con los s bits del primer segmento del mensaje cifrado para de esta manera formar el segundo bloque de entrada.

   Otra forma de ver la generación del segundo bloque de entrada, es que, los bits del primer bloque de entrada realizan un corrimiento circular hacia la izquierda de s posiciones, y entonces el segmento del mensaje cifrado reemplaza los s bits menos significativos.

   Este proceso es repetido con cada uno de los bloques de entrada sucesivos hasta que se produce un segmento de mensaje cifrado desde cada segmento del mensaje en claro, es decir, cada bloque de entrada sucesivo está cifrado para producir un bloque de salida.

[3] Un cifrado en flujo esta diseñado para cifrar un byte a la vez, pero se podría diseñar para hacerlo bit por bit. En éste se realiza la suma XOR entre el flujo del mensaje en claro y la llave.

En la figura 1.3 así como en las ecuaciones 1.6 y 1.7 se muestra el proceso de cifrado y descifrado respectivamente que fueron obtenidos de [25].

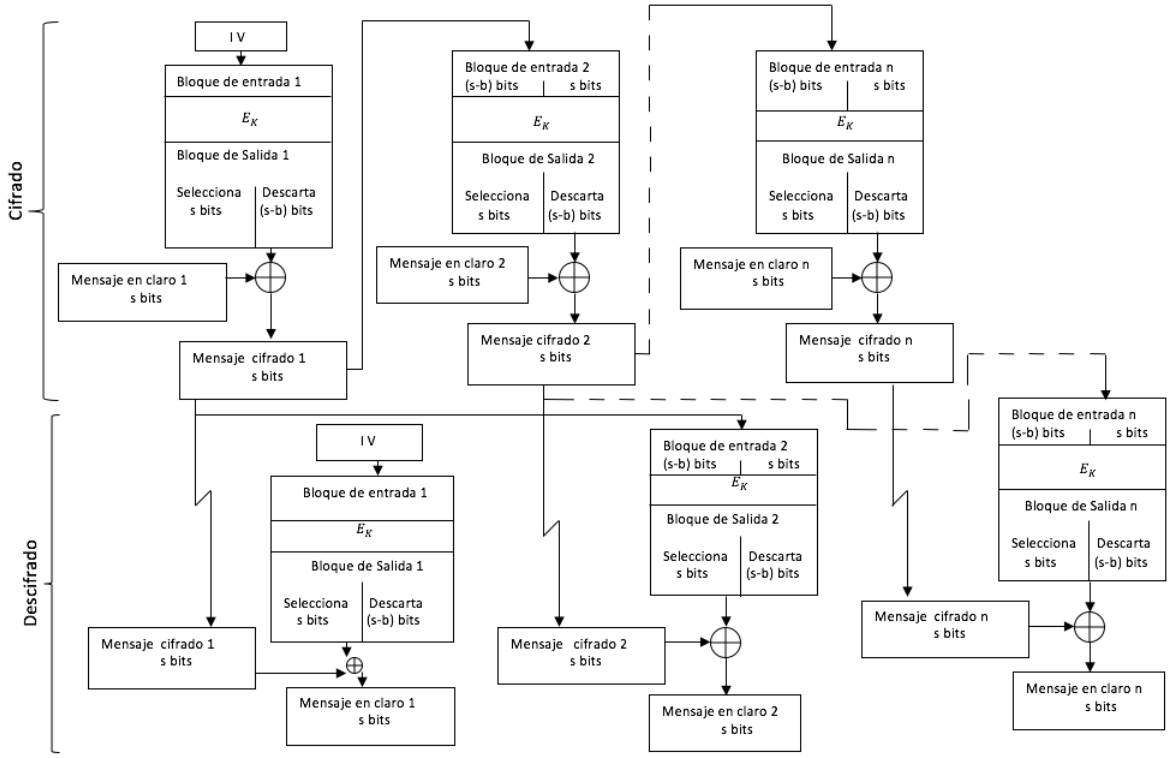


Figura 1.3: Esquemas de cifrado y descifrado del modo CFB

$$\begin{aligned} I_1 &= IV & \\ I_j &= LSB_{b-s}(I_{j-1})|Y_{j-1}^{\#} & para\ j = 2,3,\ldots,n. \\ O_j &= E_K(I_j) & para\ j = 1,2,3,\ldots,n. \\ Y_j^{\#} &= X_j^{\#} \oplus MSB_s(O_j) & para\ j = 1,2,3,\ldots,n. \end{aligned} \tag{1.6}$$

$$\begin{aligned} I_1 &= IV & \\ I_j &= LSB_{b-s}(I_{j-1})|Y_{j-1}^{\#} & para\ j = 2,3,\ldots,n. \\ O_j &= E_K(I_j) & para\ j = 1,2,3,\ldots,n. \\ X_j^{\#} &= Y_j^{\#} \oplus MSB_s(O_j) & para\ j = 1,2,3,\ldots,n. \end{aligned} \tag{1.7}$$

donde I_j representa el j-ésimo bloque de entrada, $Y_{j-1}^{\#}$ representa el $j-1$ segmento de mensaje cifrado, O_j el j-ésimo bloque de salida, $X^{\#}$ el j-ésimo segmento del mensaje en claro, MSB_s representa la cadena de bits que consta de los s bits más significativos de la cadena O_j, LSB_{b-s} representa la cadena de bits que consta de los $b-s$ menos significativos de la cadena I_{j-1}, mientras que $|$ se utiliza para representar la concatenación de cadenas y E_K representa la función de cifrado.

4. **Modo OFB (Output Feedback)**

Este modo es similar al modo CFB con la diferencia de que aquí una cantidad determinada de bits del bloque de salida anterior es trasladada hacia posiciones menos significativas de la cola. Y a diferencia de los otros modos, el tamaño total del mensaje en claro no necesariamente debe de ser un múltiplo del tamaño de bloque.

También es utilizado un vector de inicialización que deber ser único pero no necesariamente se debe mantener en secreto. En la figura 1.4 así como en las ecuaciones 1.8 y 1.9 se muestra el proceso de cifrado y descifrado respectivamente del modo OFB, que se obtuvieron de [25].

$$\begin{aligned} I_1 &= IV \\ I_j &= O_{j-1} && para\ j = 2, \ldots, n \\ O_j &= E_K(I_j) && para\ j = 1, 2, \ldots, n \\ Y_j &= X_j \oplus O_j && para\ j = 1, 2, \ldots, n-1 \\ Y_n^* &= X_n^* \oplus MSB_u(O_n) \end{aligned} \tag{1.8}$$

$$\begin{aligned} I_1 &= IV \\ I_j &= O_{j-1} && para\ j = 2, \ldots, n \\ O_j &= E_K(I_j) && para\ j = 1, 2, \ldots, n \\ X_j &= Y_j \oplus O_j && para\ j = 1, 2, \ldots, n-1 \\ Y_n^* &= X_n^* \oplus MSB_u(O_n) \end{aligned} \tag{1.9}$$

donde I_j representa el j-ésimo bloque de entrada, O_j el j-ésimo bloque de salida, X_n^* el último bloque del mensaje en claro, que puede ser un bloque parcial, Y_n^* el último bloque del mensaje cifrado, que puede ser parcial, MSB_u representa

la cadena de bits que consta de los u bits más significativos de la cadena O_n y E_K la función de cifrado.

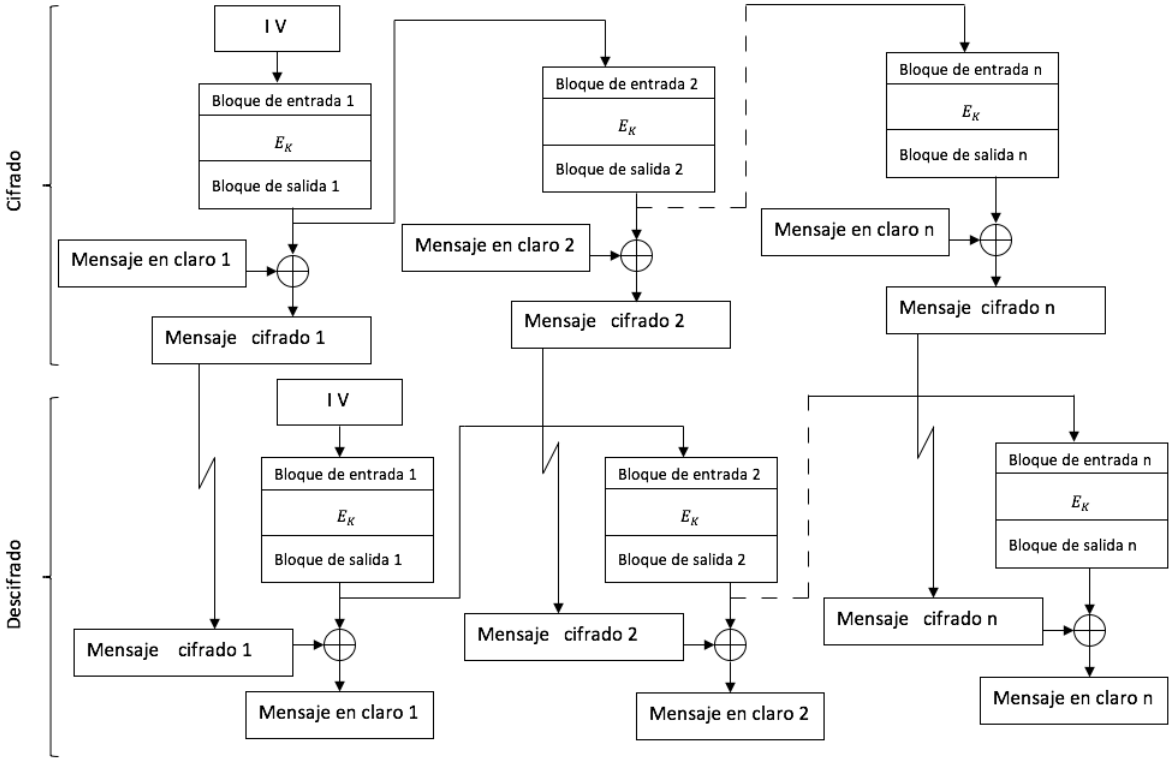


Figura 1.4: Esquemas de cifrado y descifrado del modo OFB

5. **Modo CTR (The Counter Mode)**

Este modo es de confidencialidad, y al igual que el modo OFB el mensaje en claro no necesariamente debe ser un múltiplo del tamaño del bloque.

En este modo se utiliza una secuencia de bloques denominados *contadores* que cumplen con las siguientes característica; cada bloque es distinto, es decir, ningún bloque se repite mientras se este utilizando la misma llave para cifrar sin importar que el mensaje en claro sea distinto; la cantidad de contadores es la misma que los bloques en los que se a dividido el mensaje en claro.

A los *contadores* se les aplica la función de cifrado, obteniendo así *contadores cifrados*, los cuales se suman mediante la operación XOR con el correspondiente

bloque de mensaje en claro, obteniendo así el mensaje cifrado por bloques final.

En este modo un aspecto importante a cumplir es el de *unicidad*, para lograrlo se tiene que considerar lo siguiente: utilizar una *función de incremento*[4] para generar los bloques de contadores desde cualquier bloque de contador inicial, T_1, asegurando así que los bloques de contador no se repitan dentro de un mensaje en claro dado y el bloque de contador inicial, T_1, se debe elegir de tal forma que se garantice que los contadores sean únicos en todos los mensajes cifrados con una llave dada.

En la figura 1.5 así como en las ecuaciones 1.10 y 1.11 se muestra el proceso de cifrado y descifrado respectivamente del modo CTR los cuales se obtuvieron de [25].

$$\begin{aligned} O_j &= E_K(T_j) && para\ j = 1, 2, \ldots, n. \\ Y_j &= X_j \oplus O_j && para\ j = 1, 2, \ldots, n-1. \\ Y_n^* &= X_n^* \oplus MSB_u(O_n). \end{aligned} \tag{1.10}$$

$$\begin{aligned} O_j &= E_K(T_j) && para\ j = 1, 2, \ldots, n. \\ X_j &= Y_j \oplus O_j && para\ j = 1, 2, \ldots, n-1. \\ X_n^* &= Y_n^* \oplus MSB_u(O_n). \end{aligned} \tag{1.11}$$

donde O_j representa el j-ésimo bloque de salida, E_K la función de cifrado que utiliza la llave K, T_j el j-ésimo contador en bloque, Y_j el j-ésimo bloque de texto cifrado, X_j el j-ésimo bloque del mensaje en claro, Y_j^* el último bloque del mensaje cifrado que puede ser un bloque parcial, X_n^* el último bloque del mensaje en claro que puede ser parcial, $MSB_u(O_n)$ la cadena de bits que consiste de los u bits más significativos de la cadena de bits O_n.

[4]Para ver más sobre las funciones de incremento consultar [25].

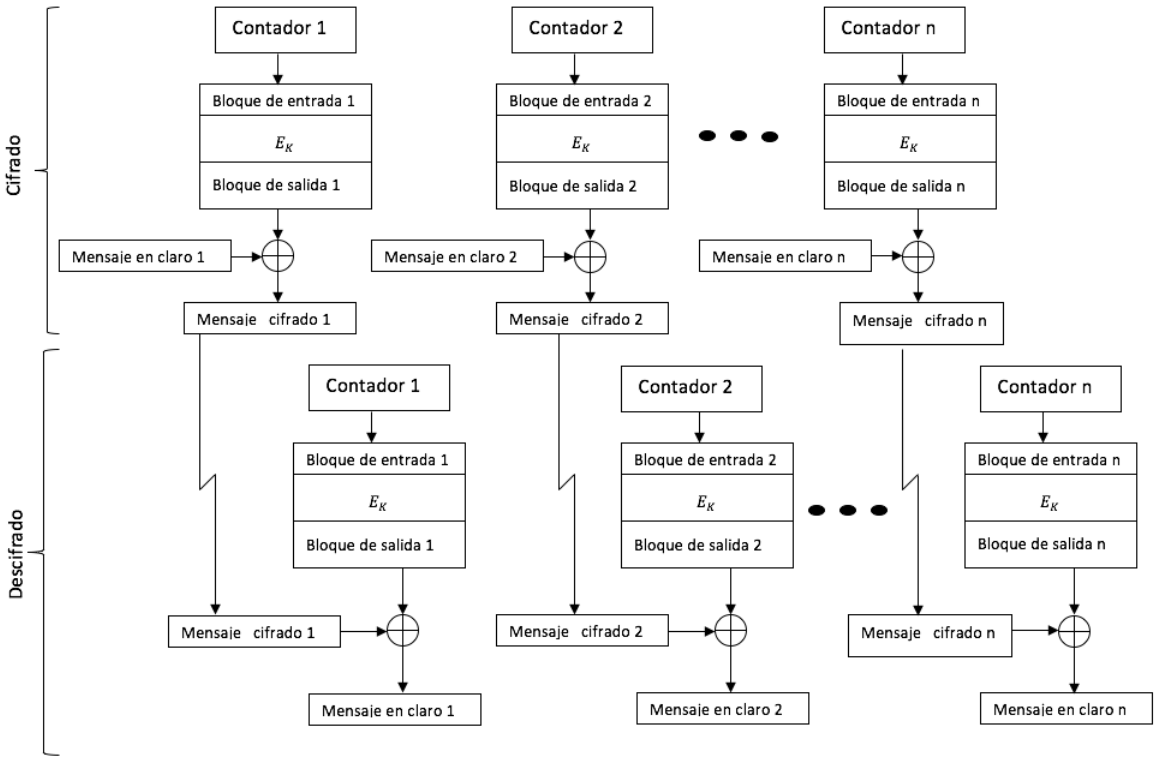


Figura 1.5: Esquemas de cifrado y descifrado del modo CTR

Al momento de describir los cifrados por bloque simétricos surge la importancia de un nuevo diseño en estos cifrados, por lo que en las siguientes secciones se da la descripción de lo que es un *ajuste*, un cifrado por bloque simétrico con ajuste y sus modos de operación.

1.2. Ajuste (Tweak)

Como se menciona en la sección 1.1 un cifrado por bloque estándar tiene la función E definida por la función 1.1.

Debido a que existe un conflicto de que si al mantener la misma llave se puede lograr tanto eficiencia como variabilidad en los cifrados por bloques estándar, se piensa en el diseño de nuevos cifrados por bloques, en los cuales se pueda lograr la variabilidad manteniendo la misma llave, ésto mediante la manipulación del mensaje en claro, del mensaje cifrado o ambas, es así como surgen los *cifrados por bloque con ajuste*.

Los cifrados por bloque con ajuste utilizan un cifrado por bloque estándar (AES, TDES) donde el dominio del cifrado por bloque es ampliado mediante la entrada adicional, el *ajuste* (tweak). El ajuste aporta variabilidad, es decir, variando el ajuste se consigue manejar familias distintas e independientes de cifrados por bloque. Además se debe de asumir que el ajuste no es un valor desconocido para el adversario.

La función que describe a los cifrados por bloque con ajuste (ecuación 1.12) es muy similar a la de un cifrado por bloque estándar (1.1), pues es una función biyectiva, que para una llave $K \in \{0,1\}^k$ fija, un ajuste $T \in \{0,1\}^t$ y un mensaje en claro $X \in \{0,1\}^n$ se produce una única salida $Y \in \{0,1\}^n$.

$$\tilde{E} : \{0,1\}^k \times \{0,1\}^t \times \{0,1\}^n \longrightarrow \{0,1\}^n \tag{1.12}$$

En la figura 1.6 se muestra un esquema comparativo de ambos cifrados por bloque:

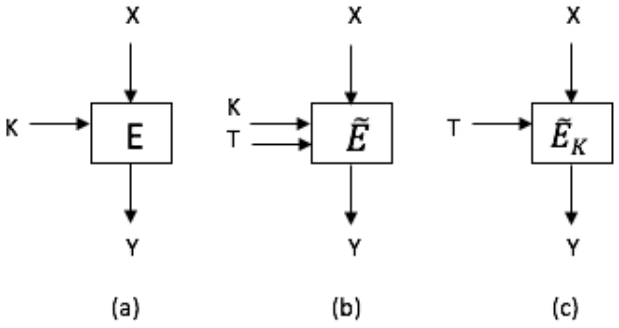


Figura 1.6: (a) Cifrado por bloque estándar: cifrando un mensaje X bajo una llave K para obtener un mensaje cifrado Y mediante la función E. (b) Cifrado por bloque con ajuste (tweak): cifrando un mensaje X bajo una llave K y un ajuste T para obtener un mensaje cifrado Y mediante la función $\tilde{E}$. (c) Otra forma de representar un cifrado por bloque con ajuste.

En el diseño de un cifrado por bloque con ajuste se tienen ciertos objetivos como en cualquier otro cifrado, en primer lugar se quiere que cualquier cifrado por bloque con ajuste que sea diseñado sea lo más eficiente posible pues debe tener la propiedad de que el cambio del ajuste debe ser menos costoso que el cambio de la llave. En segundo lugar también debe ser seguro, lo cual significa que si un adversario tiene el control[5] del ajuste de entrada, se quiere que éste cifrado permanezca de igual forma seguro.

[5]El control del ajuste se refiere a que el adversario conoce el ajuste que se esta utilizando en el momento, más sin embargo, no conoce como es que este ajuste fue generado, por lo que otros mensajes se mantendrán seguros.

Al igual que en los cifrados por bloque estándar, los cifrados por bloque con ajuste tienen modos de operación, los cuales se describen en la siguiente sección acompañándolos de sus correspondientes esquemas y ecuaciones que describen su funcionamiento.

1.2.1. Modos de operación de cifrados por bloque con ajuste

La nueva entrada (el ajuste) en los cifrado por bloques permite nuevos modos de operación [20]. A continuación se describen tres modos posibles.

1. **Encadenamiento de bloque ajustado (Tweak Block Chaining: TBC).**

   Este modo tiene un ajuste inicial T_0 que juega el papel de vector de inicialización, cada bloque del mensaje en claro sucesivo X_i es cifrado bajo el control de la llave de cifrado K y el ajuste T_{i-1} donde $T_i = Y_i$ para $i > 0$, con Y_i el i-ésimo bloque del mensaje cifrado. En este modo cada bloque de mensaje cifrado se convierte en el ajuste para el siguiente bloque de mensaje cifrado.

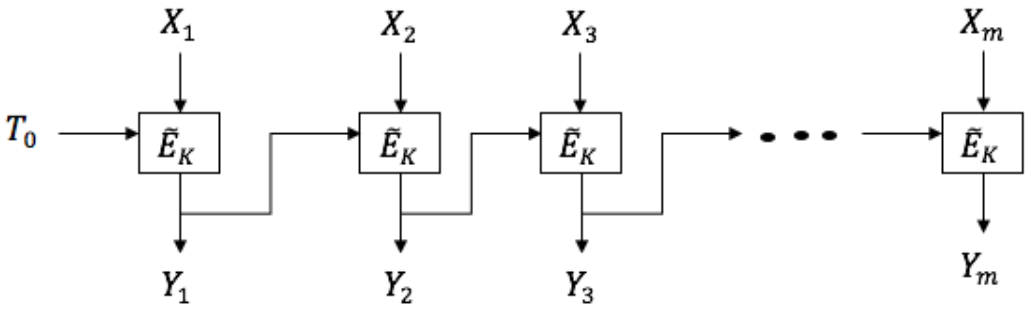


Figura 1.7: Encadenamiento de bloque ajustado (Tweak Block Chaining: TBC)

La siguiente ecuación describe el proceso que sigue el encadenamiento de bloques ajustado presentado en la figura 1.7

$$\begin{aligned} Y_1 &= E_K(T_0, X_1) \\ T_i &= Y_i = E_K(Y_{i-1}, Xi) \qquad\qquad i = 2, 3, \ldots, m. \end{aligned} \tag{1.13}$$

2. **Cadenas Hash ajustadas (Tweak Chain Hash: TCH).**

Este modo se utiliza para crear una función hash[6], sus entradas son: un ajuste T_0 fijo como vector de inicialización, una llave pública K fija en el cifrado por bloque con ajuste y el mensaje en claro X dividido en i bloque con $i = 1, 2, 3, \ldots, m$. El mensaje en claro X debe tener longitud n por lo que algunas ocaciones es rellenado con un 1 y varios ceros para tener dicha longitud.

El primer bloque cifrado Y_0 se obtiene de aplicar $\tilde{E}_K$ al bloque del mensaje en claro X_1 con el ajuste T_0, Y_0 es sumado XOR con X_1 y de esta manera obtener lo que se introducirá a $\tilde{E}_K$ junto con el bloque X_2, este proceso se sigue sucesivamente hasta el bloque X_m, obteniendo así la función hash H. Esto se puede ver en la figura 1.8.

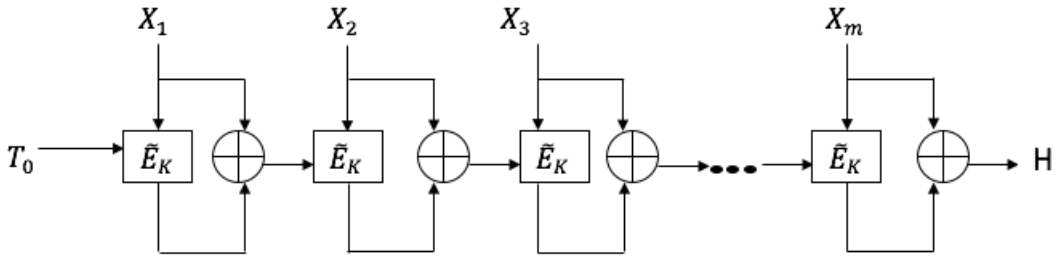


Figura 1.8: Cadenas Hash ajustadas (Tweak Chain Hash: TCH)

La ecuacion 1.14 ilustra el proceso para obtener una cadena hash ajustada mostrado en la figura 1.8.

$$H = \tilde{E}_K(\cdots \tilde{E}_K(\tilde{E}_K(T_0, X_1) \oplus X_1, X_2) \oplus X_2, X_3) \oplus X_3 \cdots, X_m) \oplus X_m \quad (1.14)$$

[6]La función hash se describe en la sección 1.4

3. **Cifrado de autenticación ajustado (Tweakable Authenticated Encryption: TAE).**

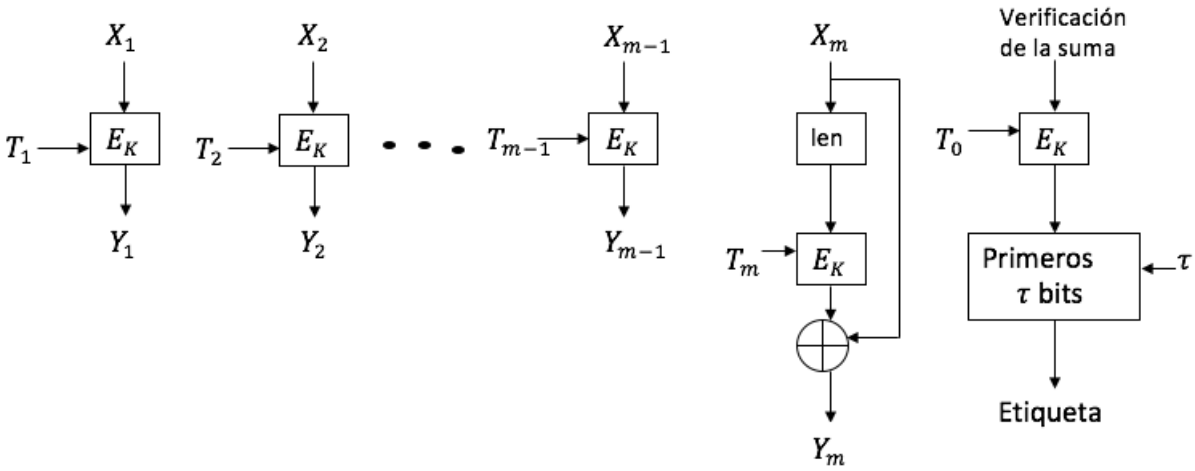


Figura 1.9: Cifrado de autenticación ajustado (Tweakable Authenticated Encryption: TAE)

En la figura 1.9 se muestra el proceso que se sigue para obtener un cifrado de autenticación ajustado para el mensaje X, dicho proceso también se representa mediante la ecuación 1.15.

$$\begin{aligned} Y_i &= E_K(T_i, X_i) \qquad\qquad i = 1, 2, 3, \ldots, m-1. \\ Y_m &= E_K(T_m, \mathsf{len}(X_m)) \oplus X_m. \end{aligned} \tag{1.15}$$

El mensaje en claro X es dividido en $m-1$ bloques de longitud n, es decir, $X_1, \ldots, X_{m-1}$ y un último bloque X_m de longitud r para $0 \leq r \leq n$ (excepto que si $\mid X \mid = 0$ entonces el último y solamente el último bloque tiene longitud 0). Cada bloque del mensaje cifrado Y_i tendrá la misma longitud que X_i.

Como se puede observar en la figura 1.9 el cifrado de autenticación ajustado tiene como entrada, además del mensaje en claro dividido en bloques, un ajuste (tweak) T_i. Para $i > 0$ éste se define como la concatenación de: un *nonce* [7] N de $\frac{n}{2}$ bits, un entero i representado con $\frac{n}{2} - 1$ bits y un bit zero 0, es decir, $T_i = N \mid\mid i \mid\mid 0$. Por otro lado, T_0 se define como la concatenación de N, b, la longitud en bits del mensaje en claro X representado con $\frac{n}{2} - 1$ bits, y un bit

[7] ver sección 1.1.1.

1, por lo que $T_0 = N \,||\, b \,||\, 1$.

La función $\mathsf{len}(X_m)$ produce una representación binaria de $n-$bit de longitud r del último bloque del mensaje en claro. El último bloque del mensaje en claro X_m se rellena con ceros en caso de ser necesario para que de esta manera tenga longitud n antes de realizar la suma. El parámetro τ, puede tomar valores $0 \leq \tau \leq n$ y sirve para especificar la longitud deseada de la etiqueta de autenticación.

Debido a que a lo largo de este capítulo se hace mención de los cifrados asimétricos y funciones hash se cree importante dar una breve descripción de dichos cifrados y funciones en las siguientes secciones.

1.3. Cifrados asimétricos

Los cifrados asimétricos [33] también son llamados cifrados de llave pública, debido a que utilizan un par de llaves, una para cifrar y otra para descifrar. Dichas llaves pertenecen a la entidad que cifra el mensaje en claro, el emisor. La llave que se utiliza para descifrar el mensaje, es pública y utilizada por el receptor del mensaje, mientras que la utilizada para el cifrado permanece privada y solo debe tener acceso a ella el emisor.

Estos cifrados fueron diseñados para evitar el problema del intercambio de la llave, que se tiene en los cifrados simétricos (ver sección 1.1), pues en éstos es necesario que tanto el emisor como el receptor se pongan de acuerdo en la llave que se va a utilizar. La manera de solucionar el intercambio de la llave es mediante la utilización del algoritmo de *Diffie-Hellman* [4], que depende de su eficiencia en la dificultad de calcular logaritmos discretos.

En los cifrados asimétricos la longitud de las llaves es mucho mayor que en la de los simétricos, por ejemplo, mientras en los cifrados simétricos se recomiendan llaves de 128 bits, en la mayoría de los cifrados asimétricos, se recomiendan llaves de al menos 1024 bits. Si se desea consultar más sobre cómo surgió el cifrado asimétrico se puede consultar [4].

Una ventaja de éstos cifrados, es que, se logra la identificación y autenticación del emisor, pues solo él pudo ser quién empleó la llave privada para cifrar el mensaje en claro, al menos que ésta haya sido robada.

Algunos ejemplos de cifrados asimétricos más usados son: ElGamal [23], RSA [22], Curvas elípticas [24]. Los cifrados asimétricos son utilizados en firmas digitales, que

funcionan de forma similar a un a firma escrita, ya que un mensaje que sea firmado con la llave privada por el emisor puede ser verificado por cualquier persona que tenga acceso a la llave pública (receptor) de éste.

1.4. Funciones hash

El objetivo principal de una función hash es la *integridad* de los datos, pues un cambio a cualquier bit en un mensaje M resulta en un cambio en el resultado de la función hash. A las funciones hash que son utilizadas para aplicaciones de seguridad son conocidas como *funciones hash criptográficas*.

Estas funciones tienen como entrada, cadenas (un archivo, dígitos, etc.) de longitudes arbitrarias (grandes pero finitas) que son convertidas a cadenas alfanuméricas de longitud finita fija, obteniendo así un *resumen* de las cadenas de entrada, por lo que son llamadas *funciones de resumen* [33].

Una función hash criptográfica H es un algoritmo [36] en el cual se cumplen las siguientes propiedades:

1. **Tamaño de entrada variable:** H se puede aplicar a un bloque de datos de cualquier tamaño.
2. **Tamaño de salida fijo:** H produce una longitud de salida fija.
3. **Eficiencia:** $H(x)$ es relativamente sencillo de calcular para cualquier x dado, permitiendo así que las implementaciones tanto en hardware como software sean prácticas.
4. **Resistencia a pre-imagen:** Para cualquier valor hash h dado, es computacionalmente inviable encontrar y tal que $H(y) = h$.
5. **Resistencia a segunda pre-imagen:** Para cualquier bloque x dado, es computacionalmente inviable encontrar $y \neq x$ con $H(y) = H(x)$.
6. **Resistencia a colisiones:** Es computacionalmente inviable encontrar cualquier par (x, y) tal que $H(x) = H(y)$.
7. **Pseudoaleatoriedad:** Todas las salidas de H cumplen con las pruebas de aleatoriedad.

La propiedad (4) implica una única dirección, es decir, es fácil generar un resultado hash dado un mensaje, pero es prácticamente imposible generar el mensaje dado el

resultado hash. Por otro lado, la propiedad (5) previene la falsificación.

Una función hash que sea resistente a colisiones también lo será a segundas preimagen, pero inversamente no es necesariamente es cierto. Otro aspecto importante de destacar de esta propiedad, es el siguiente: para un x dado, si resulta computacionalmente fácil encontrar un $y \neq x$ tal que $H(x) = H(y)$ se dice que la función hash tiene una *resistencia débil* a colisiones, por otro lado, si resulta difícil encontrar un par (x, y) tal que $H(x) = H(y)$ entonces se dice que tiene una *resistencia fuerte* a colisiones.

Si una función hash satisface las primeras cinco propiedades se le conoce como *función hash débil*, si además satisface la propiedad 6 entonces es una *función hash fuerte.*

Existe lo que se conoce como código de autenticación de mensaje (Message authentication code; MAC) y código de detección de modificaciones (Modification Detection Codes; MDC). MDC [33] es una función hash sin llave secreta que se utiliza para resolver el problema de integridad del mensaje, pues al mensaje se le aplica un MDC y es enviado junto con el, una vez que el mensaje y el MDC son recibidos por el receptor éste le aplica la función hash al mensaje y comprueba que sea igual al hash que se envío antes.

Por otro lado, los MAC [33] se pueden utilizar para proporcionar autenticación del origen del mensaje, es decir, dada una llave secreta K y un mensaje M, se calcula $h(M, K)$ que es enviado junto con el mensaje, si al llegar a su destino $h(M, K)$ es calculado utilizando la llave secreta K y coincide con el $h(M, K)$ recibido entonces queda comprobado la autenticidad del origen del mensaje M.

El receptor esta seguro que el mensaje no ha sido alterado, ya que si un atacante altera el mensaje pero no el MAC, entonces el cálculo de éste no coincide, además como se asume que el atacante no conoce la llave secreta K, el atacante no puede alterar el MAC para de esta manera lograr una correspondencia con la alteración del mensaje.

Existen dos tipos de MAC: los MAC basados en funciones hash (HMAC) y los basados en cifrados por bloque (Data Authentication Algorithm: DAA y Cipher-Based Message Authentication Code: CMAC) [36]. Los basados en funciones hash son los más utilizados, pues las funciones hash criptográficas como MD5 [33] y SHA generalmente se ejecutan más rápido en un software a diferencia de los cifrados por bloques como el DES.

Sin embargo, las funciones hash SHA no fueron diseñadas para usarse como MAC y no es recomendable utilizarse directamente para ese fin [31], pues éstas no dependen de una llave secreta. Algunos ejemplos de funciones hash son: SHA-1 [36], SHA-256, SHA-384, SHA-512 que producen cadenas de 160, 256, 384 y 512 bits respectivamente y MD5 [33]. Todas las SHA a excepción del SHA-1 se conocen como SHA-2 [32].

SHA-1 y MD5 son muy similares tanto en su estructura como en las operaciones matemáticas básicas que emplean, en la actualidad estas dos funciones hash se consideran inseguras por lo cual se recomienda el uso de SHA-2.

DAA esta basado en el cifrado DES y fue uno de los MAC más utilizados durante varios años, mientras que CMAC fue ampliamente adoptado por el gobierno e industria, éste es seguro bajo un conjunto de criterios de seguridad [31], solo se utiliza para procesar mensaje de una longitud fija de mn bits, donde n es el tamaño de bloque de cifrado y m un número entero positivo fijo.

Capítulo 2

Técnicas de cifrado que preservan formato

En este capítulo se describen las técnicas principales de los cifrados que preservan formato las cuales fueron introducidas por *Black* y *Rogaway* en [2], ilustrándolas con algunos ejemplos. Además de abordar algunas cualidades que debe cumplir un esquema o sistema de cifrado.

Se considera que un sistema criptográfico es perfecto [29] si cumple con las siguientes condiciones:

1. Cada llave empleada debe producir un mensaje cifrado distinto, es decir, si dos llaves distintas producen el mismo resultado entonces estas llaves son equivalentes.

2. Aunque el atacante tenga acceso a un gran número de valores de la tabla[1] le es imposible decir con cual llave fueron obtenidos dichos valores. Esta técnica se conoce como *mensaje en claro conocido, mensaje cifrado conocido*.

3. Si el atacante tiene una gran cantidad de datos en claro, éste no puede decir cual es la relación entre estos y los datos cifrados, es decir, debe de poseer la llave para cifrar y descifrar aun cuando lo desee hacer de forma parcial.

Con las condiciones anteriores se pensó que se podía crear un "algoritmo perfecto" para garantizar la seguridad de los número de tarjeta de crédito/débito, el cual resultó ser poco práctico, pues cada uno de los números de tarjetas serían almacenados en memoria y mezclado hasta producir una tabla para utilizarla como la llave de cifrado,

[1] Con los mensajes cifrados se construye una tabla

siendo una gran desventaja el que dicha tabla puede ser muy grande (aproximadamente 60,000 GB de tamaño) pero con la ventaja de que cumple con cada una de las condiciones mencionadas anteriormente. Debido a ello surgió la pregunta, ¿Cómo generar un algoritmo que funcione como el "algoritmo perfecto" ? con la diferencia de que sea fácil de implementar y tan fuerte como los algoritmos existentes, como el AES.

De ahora en adelante a lo largo de este capítulo cuando se haga mención de una tabla nos estaremos refiriendo a la que fue creada a partir de los valores obtenidos al cifrar un mensaje (o dato) en claro. Se espera que se pueda entender a lo que nos estamos refiriendo con tabla en la descripción de la técnica de cifrado de prefijo.

Una manera de aproximarse a la respuesta de dicha pregunta es resolviendo un problema más pequeño y sencillo, en este problema se puede utilizar el propio algoritmo AES para hacer el mezclado de la tabla y utilizar dicha tabla para cifrar y descifrar, con este planteamiento se puede sustituir el almacenamiento de la tabla por almacenar la llave AES que fue empleada.

Existen tres técnicas principales para cifrados que preservan formato [2]:

1. Cifrado de prefijo (Prefix Cipher).
2. Caminata cíclica (Cycle Walking).
3. Redes de Feistel (Feistel Network).

El objetivo principal de cada una de estas técnicas es regresar el mismo tamaño y tipo de dato que del mensaje en claro, por otro lado es importante mencionar que cada una de estas técnicas trabaja con un mensaje en claro de diferente tamaño debido a cuestiones de seguridad o por cuestiones de desempeño que serán mencionadas en cada una de las secciones.

En cada una de las secciones siguientes se menciona como es la construcción de la técnica a la cual se esta haciendo referencia, así como algunas de las ventajas y desventajas que tiene el utilizar cada una de ellas.

2.1. Cifrado de prefijo (Prefix Cipher)

El cifrado de prefijo es la técnica más sencilla de las mencionadas previamente, con la limitación de que esta trabaja solo con conjuntos pequeños de datos. El cifrado de prefijo está basado en el siguiente algoritmo o serie de pasos [29]:

1. Escriba cada valor[2] (dato) de entrada posible y cífrelo con AES.

2. Ordenar la tabla que se crea con los valores de entrada y sus correspondientes valores de salida (datos cifrado) al aplicar AES, según el orden del alfabeto hexadecimal.

3. El orden de los valores será ahora la columna de salida de la tabla.

Para entender mejor lo dicho en los tres pasos anteriores se hará un ejemplo con la llave K= *2b7e151628aed2a6abf7158809cf4f3c* y el conjunto $X = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, con K obtenida de [27].

Paso 1: Construir una tabla con los datos a cifrar (X) y los datos cifrados con AES (Y) utilizando la llave K.

Entrada	Salida AES
0	1a5abecad799a32683dc8d362277a311
1	96028ebce8e9b1cbbfc9610ab85bcf6d
2	39f9c32cdf3e010e6f431298c3415a50
3	0adc42066ec74a19e2f3f44f9a5c7638
4	aaee6ddecc20eba44b17bba038b6b999
5	acf2f87c48ad1c6845737993b4bf9eca
6	1555b96602fb85d1264bb65342912b9a
7	28e05096e7257b9a3098e1c125121870
8	ca91dd5ac6942b3444cb0b0fb464ccc2
9	c648796c44ae3add42c1fcb4bf44cb19

Tabla I: Tabla del cifrado obtenido de los dígitos [0-9] utilizando AES, con llave K=2b7e151628aed2a6abf7158809cf4f3c.

Los datos representados en la columna de *Salida AES* de la tabla I se obtuvieron al emplear el cifrado AES que se programo durante la realización de la aplicación computacional de éste trabajo y que se muestra en el Capítulo 3.

Paso 2: Ordenar la tabla utilizando la salida AES:

Ya que la salida que se obtiene está expresada en hexadecimal el nuevo orden tendrá en cuenta el orden de dicho alfabeto, para ello nos fijaremos en el primer carácter

[2]Al decir valor de entrada se está haciendo referencia a un dígito a cifrar, pues esta técnica es empleada para cifrar datos como: números de tarjeta, códigos postales.

(de izquierda a derecha) de cada una de las salidas, si existen dos o más salidas cuyo primer carácter sea el mismo, nos fijaremos en el siguiente carácter de cada una y compararemos cual es el menor según el orden del alfabeto hexadecimal, si dichos carácteres volvieran a ser iguales, compararemos los caracteres subsecuentes de cada una de las salidas hasta poder decir cual de las dos salidas es la menor y establecer el nuevo orden.

Como ejemplo observar las salidas de la tabla I para las entradas de los valores *0, 6 - 4,5* así como para los valores *8 y 9*, al ordenar los valores de la tabla I siguiendo lo descrito previamente se obtiene la tabla II.

Entrada	Salida AES
3	0adc42066ec74a19e2f3f44f9a5c7638
6	1555b96602fb85d1264bb65342912b9a
0	1a5abecad799a32683dc8d362277a311
7	28e05096e7257b9a3098e1c125121870
2	39f9c32cdf3e010e6f431298c3415a50
1	96028ebce8e9b1cbbfc9610ab85bcf6d
4	aaee6ddecc20eba44b17bba038b6b999
5	acf2f87c48ad1c6845737993b4bf9eca
9	c648796c44ae3add42c1fcb4bf44cb19
8	ca91dd5ac6942b3444cb0b0fb464ccc2

Tabla II: Tabla del cifrado re-ordenando de acuerdo a la salida AES.

Paso 3: El orden de los valores será ahora la columna de salida de la tabla: se puede descartar los valores de salida AES y ordenar los valores de entrada empleando el nuevo orden con los valores de salida de la tabla cifrada como se muestra en la tabla III.

Tabla de re-ordenamiento									
Entrada	0	1	2	3	4	5	6	7	8	9
Salida	3	6	0	7	2	1	4	5	9	8

Tabla III: Tabla del cifrado re-ordenando de acuerdo a la salida AES.

Obsérvese que esta es una permutación del conjunto X, la cual denotaremos como Π, así el descifrado se puede realizar encontrando el índice del valor cifrado.

$$\prod = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 3 & 6 & 0 & 7 & 2 & 1 & 4 & 5 & 9 & 8 \end{pmatrix}$$

Una de las propiedades interesantes que se puede obtener del cifrado de prefijo es la siguiente: supongamos que el atacante conoce la información que se muestra en la tabla III sin los dos últimos valores de salida, y éste desea conocer el valor correspondiente para la entrada 8, tiene dos posible valores 9 y 8, al menos que se pueda romper el AES o el atacante cuente con la llave utilizada, éste no cuenta con alguna ventaja.

Algunas de las desventajas de esta técnica de cifrado es que se necesitarán N llamadas al cifrado AES para cifrar N dígitos, por lo cual se necesita trabajar con un conjunto de datos pequeño. Además de que se necesita de más tiempo para construir la tabla así como más memoria para almacenarla. Por otro lado, una ventaja es que el proceso de cifrado es rápido para un conjunto pequeño de datos.

2.2. Caminata cíclica (Cycle Walking).

Esta técnica, como su nombre lo indica, es utilizada para cifrar un mensaje en claro repetidamente hasta obtener el mensaje cifrado en el rango deseado y, al igual que en el *cifrado de prefijo*, se utiliza el cifrado por bloque AES o TDES.

Debido a que un cifrado que preserva formato tiene como característica que el mensaje cifrado tenga la misma cantidad de caracteres que el mensaje en claro, esta técnica aplica varias veces el cifrado AES (o TDES), hasta obtener un mensaje cifrado con la misma cantidad de caracteres que el mensaje en claro, es decir, que se encuentre en el rango deseado.

A continuación se muestra un ejemplo de esta técnica cuya función de cifrado no es ni AES ni TDES pero pretende ilustrar el funcionamiento de dicha técnica de una forma sencilla.

Tomemos la función de cifrado $(ax + b)\ mod\ n$, con $a = 3,\ b = 7\ y\ n = 17$, y el conjunto $X = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Al realizar los cálculos correspondientes se puede observar que al cifrar algunos números se necesita aplicar la función de cifrado más de una vez para que el valor cifrado se encuentre dentro del rango requerido, que para nuestro ejemplo son valores entre 0 y 9. A continuación se muestran los cálculos requeridos para los valores de $X = 2$ y $X = 8$ seguidos de la tabla IV con los resultados para cada uno de los valores de X.

$x = 2$,

$c_1 = E_k(2) = (3(2) + 7) = 13 \equiv 13 \; mod \; 17.$

$c_2 = E_K(13) = (3(13) + 7) = 46 \equiv 12 \; mod \; 17.$

$c_3 = E_K(12) = (3(12) + 7) = 43 \equiv 9 \; mod \; 17.$

$x = 8$,

$c_1 = E_K(8) = (3(8) + 7) = 31 \equiv 14 \; mod \; 17.$

$c_2 = E_K(14) = (3(14) + 7) = 49 \equiv 15 \; mod \; 17.$

$c_3 = E_K(15) = (3(15) + 7) = 52 \equiv 1 \; mod \; 17.$

Técnica de cifrado de caminata cíclica										
Valor en claro	0	1	2	3	4	5	6	7	8	9
Valor cifrado	7	3	9	4	2	5	8	6	1	0

Tabla IV: Técnica de cifrado de caminata cíclica para $X = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ con la función de cifrado $(ax + b) \; mod \; n$, para $a = 3, \; b = 7 \; y \; n = 17$.

Una desventaja de esta técnica es que solamente funciona de forma correcta si el tamaño del mensaje en claro es muy cercano al tamaño de bloque del mensaje cifrado, pues en caso contrario se tendrían que hacer demasiadas iteraciones para obtener un mensaje cifrado dentro del rango requerido.

Aunque se pudiera pensar que esta técnica en algunos casos puede no terminar, en [2] se tienen pruebas de que para cualquier caso siempre terminará, pero es posible que existan algunos en los que se requerirá un gran número de llamadas al cifrado AES (o TDES).

Si se desea crear un cifrado que preserva formato utilizando esta técnica es necesario utilizarla junto con una red de Feistel. De esta manera se podrá crear un cifrado por bloque que se aproxime al tamaño deseado, y luego utilizar caminata cíclica para arreglar los últimos valores. Una descripción más detallada se puede ver en [9].

2.3. Redes Feistel

El esquema de cifrado DES y por consecuencia el esquema TDES tienen como característica, que su arquitectura está basada en las llamadas redes de Feistel, además de que los primeros trabajos referentes a los cifrados que preservan formato utilizan lo

que se conoce como redes de Feistel tipo-1 y tipo-2.

El cifrado de Feistel mejor conocido como redes de Feistel fue desarrollado por H. Feistel en 1973 [11]. El primer algoritmo que utilizó éste cifrado en su diseño es el *algoritmo de Lucifer* diseñado por H. Feistel y D. Coppersmith en 1974. Esta técnica es sencilla, siendo popularizada por *Data Encryption Standard*; DES [8].

Existe lo que se conoce como red de Feistel clásica cuya construcción está basada en rondas, la primera de ellas se describe a continuación:

1. Dado un bloque[3] con longitud N (con N par), normalmente de 64 ó 128 bits es dividido en dos subbloques, L y R con igual longitud ($\frac{N}{2}$), donde L es conocido como lado izquierdo y R el lado derecho.

2. Se toma el lado R y se le aplica alguna función de ronda[4] dependiendo de la llave[5] F_K^1, de esta manera se obtiene un nuevo lado R el cual llamaremos R^* cuya longitud es la misma que R.

3. Se realiza la suma XOR entre el lado L y R^* obteniendo de esta manera el nuevo lado derecho $R^{'}$.

4. La parte derecha original (R) se convierte en el nuevo lado izquierdo $L^{'}$.

Cada ronda i subsecuente, trabaja de la misma manera excepto que cada función F_K^i varía con la llave K, pues esta es cambiada en cada una de las rondas, es importante mencionar que se pueden utilizar tantas rondas como se desee. Los pasos descritos previamente se pueden resumir de la siguiente manera.

$$\begin{aligned} L_i^{'} &= R_{i-1} \\ R_i^{'} &= L_{i-1} \oplus f(R_{i-1}, K_i) \end{aligned} \tag{2.1}$$

aquí $f(R_{i-1}, K_i)$ represeanta a F_K^i, pues se le aplica la función de ronda f al lado derecho R_{i-1} con la llave K_i correspondiente.

Además pueden ser representados mediante un diagrama como se muestra en la figura 2.1.

[3] Enteremos por bloque a un conjunto finito de caracteres y/o bits.

[4] Una función de ronda normalmente es una permutación la cual puede ser la misma en cada una de las rondas o ir variando.

[5] La llave es del mismo tamaño del subbloque, es decir, $\frac{N}{2}$.

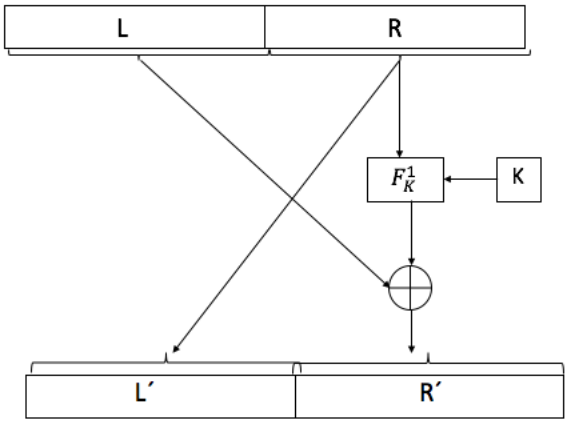


Figura 2.1: Diagrama que muestra la primer ronda de una red de Feistel clásica.

La construcción de una red de Feistel tiene la ventaja de ser reversible, es decir, las operaciones de cifrado y descifrado son idénticas, requiriendo únicamente invertir el orden de las llaves (o subllaves) que fueron utilizadas.

En las dos subsecciones siguientes se explica lo que son las redes de Feistel desequilibrada y alternadas, con lo cual se espera se pueda entender mejor como está construída de manera general una red de Feistel.

2.3.1. Feistel desequilibradas

Es importante resaltar que las redes de Feistel cuya longitud N del bloque es par, son llamadas redes clásicas, por lo cual nos podríamos preguntar: ¿qué pasa en el caso de que esta longitud sea impar?. En este caso existen dos generalizaciones de la construcción de Feistel que se pueden utilizar: *Feistel desequilibradas [12] y Feistel Alternas [13].*

En la red de Feistel clásica tanto la parte derecha (R) como izquierda (L) del bloque tienen la misma longitud, es decir, está equilibrada. En general es posible tener valores intermedios de longitud del lado izquierdo y del lado derecho diferentes, es decir, $a = |L|$ y $b = |R|$ siendo el valor de $a = |L|$ el que cuantifica el desequilibrio y es llamado *división* (split), estas redes se discutieron por primera vez por *Schneier y Kelsey [12] .*

Las redes de Feistel equilibradas se consideran un caso especial de las redes desequilibradas cuya entrada tiene una longitud N par, es decir $a = \frac{N}{2}$. La figura 2.2 (a) muestra una red de Feistel desequilibrada con una *división* de $a = \lfloor \frac{5}{2} \rfloor$, donde $\lfloor \rfloor$ es

la *función piso*[6]. Quedando así una red equilibrada tanto como se puede conseguir con una entrada de longitud impar.

También se pueden considerar esquemas más desequilibrados, como que en un extremo ($a = 1$) que cifra un bloque de n caracteres usando una función de ronda que asigna a $b = N - 1$ caracteres y a $a = 1$ caracteres.

Schneier y Kelsey [12] llaman a una red de Feistel desequilibrada fuente pesada (*source-heavy*) cuando $a < b$, es decir, la longitud de la entrada a la función de ronda es mayor que su salida, en cambio si $a > b$ esta red es llamada objetivo pesado (*target-heavy*) que es cuando la función de ronda expande la entrada. Para redes de fuente pesada (*source-heavy*) los resultados cuantitativos de seguridad mejoran a medida que aumenta el desequilibrio suponiendo que se compensa adecuadamente el desequilibrio con un número suficiente de rondas adicionales. Por otra parte, las consideraciones de seguridad sugieren que uno evita la red de objetivo pesado (*target-heavy*) con una división larga $a = |L|$.

2.3.2. Feistel alternada

Una singularidad de Feistel desequilibrada es la repartición implícita de cadenas intermedias. El método se remota a *Anderson* y *Biham* [15] y de forma independiente a *Lucks* [13]. Existen dos tipos de funciones de ronda que son utilizadas alternadamente las cuales contraen[7] o expanden[8] un elemento[9]. Cada ronda *par* utiliza una de ellas mientras que una ronda *impar* la otra, como se puede ver en la figura 2.2 (b) donde F_K^1, F_K^3 son funciones de compresión, mientras que las funciones $F_K^2,\ F_K^4$ son de expansión.

El operador $\boxplus$ se utiliza para realizar una suma caracter por caracter (*characterwise*) o bloque por bloque (*blockwise*), si se esta trabajando con un alfabeto decimal, es decir, el alfabeto donde a cada carácter le es asignado un número, entonces existen dos posibilidades que podríamos utilizar:

1. La suma caracter por caracter (*characterwise*) que realiza la suma elemento por elemento módulo 10 entre los elementos correspondientes de los bloques de longitud L y R^*. Ejemplo, $456 \boxplus 174 = 520$ pues $4 + 1 = 5\ (mod\ 10)$, $5 + 7 = 2\ (mod\ 10)$ y $6 + 4 = 0\ (mod\ 10)$.

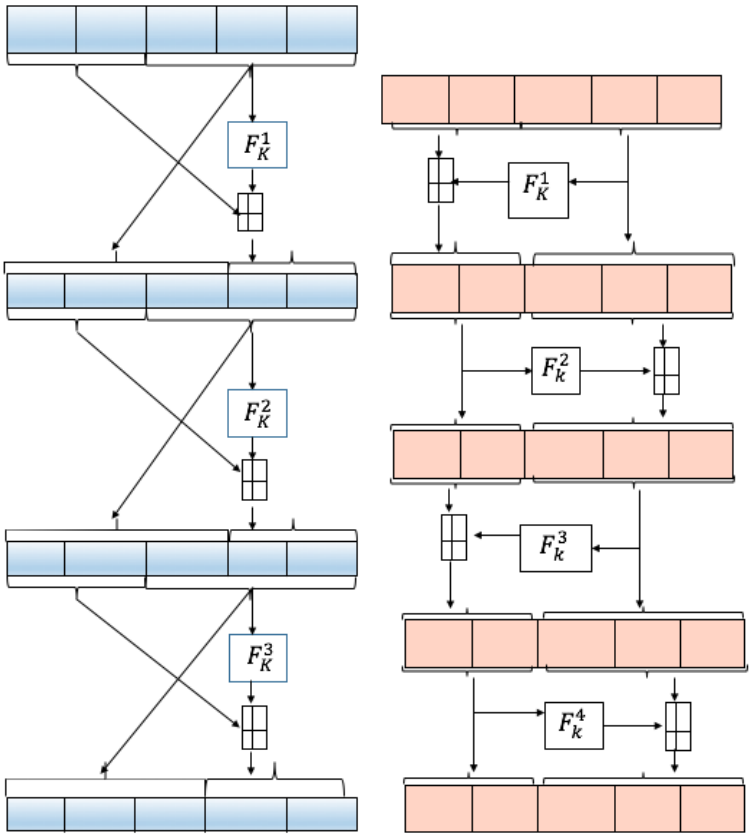
[6]La función piso es aplicada a un número real x devolviendo el mayor número entero menor y no superior a x.

[7]Como por ejemplo una función Hash.

[8]Como lo es la función E utilizada en DES [8].

[9]Un elemento puede ser un carácter, dígito o bit según el alfabeto en el que se este trabajando.

2. La suma bloque por bloque (*blockwise*) que realiza la suma de los dos bloques como si fueran números de igual longitud realizando el acarreo sin contar el último, es decir, sumando dos números de $m-$dígitos utilizando suma ordinaria modulo 10^m. Ejemplo, $456 \boxplus 174 = 630$ pues $456 + 174 = 630$ $(mod\ 10^3)$



(a) Red de Fesitel desequilibrada (b) Red de Feistel alternada

Figura 2.2: Ambas redes están compuestas con tres rondas y la suma que se realiza es bloque por bloque (*blockwise*), además el tamaño del bloque de texto en claro es de longitud N=5 por lo cual se realiza una división lo más equilibrada posible quedando como resultado las longitudes $N_L = 2$ y $N_R = 3$ de los subbloque L y R respectivamente, ambas redes realizan los pasos descritos en la sección 2.3 con la particularidad de que en (a) F_K^1, F_K^2, F_K^3 son funciones de compresión, en (b) las rondas F_K^2, F_K^4 son funciones de expansión mientras que F_K^1, F_K^3 son de compresión.

Es importante mencionar que dependiendo con cual alfabeto se esté trabajando existen tres posibilidades de realizar la suma entre sus elementos, dos de ellas son las descritas previamente, sin embargo si se está trabajando con bits, la suma que se

utilizará es XOR representada por el operador $\oplus$, un esquema de ésto es el de la figura 2.1.

Las redes de Feistel son clasificadas en dos tipos: el tipo-1, son aquellas que utilizan el mismo tipo de función de ronda en cada una de ellas; el tipo-2, son aquellas que alternan el tipo de función en cada una de sus rondas.

Los cifrados basados en redes de Feistel fueron examinados por primera vez por *Luby-Rackoff*. Luby- Rackoff fueron los que introdujeron las permutaciones pseudo-aleatorias [30] gracias a los cifrados por bloques, pues vieron a la llave privada como una determinada permutación de las cadenas.

Una forma de probar la seguridad de estos cifrados consiste en ver si el atacante puede distinguir un cifrado basado en redes de Feistel de una permutación aleatoria. En 2004, *Patarin* [34] mostró que si la construcción de Luby-Rackoff es corrida con suficientes rondas (6, 7 u 8) dependiendo de la seguridad del modelo, el número de rondas con pares de mensaje cifrado/mensaje en claro que necesita un atacante con recursos computacionales ilimitados para poder establecer una diferencia entre un sistema de cifrado Luby-Rackoff y una permutación aleatoria se acerca al máximo teórico, que es la raíz cuadrada del tamaño de todo el mensaje en claro.

Una vez descritas a lo largo de este capítulo cada una de las técnicas de cifrado que preservan formato se muestra en la tabla V los conjunto en los cuales dichas técnicas de cifrado pueden utilizarse, expresádolos en bits y dígitos decimales.

Método	Tamaño del conjunto (bits)	Tamaño del conjunto (decimales)
Cifrado de prefijo	1-20	1-6
Caminata cíclica	50-63	16-20
Red de Feistel	40-240	12-80

Tabla V: Comparación de los conjuntos en los que las técnicas: Cifrado de prefijo, Caminata cíclica y red de Feistel se pueden utilizar, obtenida de [9] .

Además de estas técnicas, existen los métodos: Format-preserving, Feistel based Encryption; FFX, con X el mensaje en claro, propuesto por M. Bellare, Ph. Rogaway y T. Spies [19], así como BPS llamado así por las iniciales sus autores E. Brier, T. Peyrin y J. Stern [16], ambos propuestos como cifrados que preservan formato y enviados al NIST para su estandarización en 2010.

Mientras que el algoritmo del cifrado FFX toma una llave K (128 bits, $K \in \{0,1\}^{128}$, como son las llaves de AES-128), un texto en claro X que es tomado de un alfabeto arbitrario (Chars), y un ajuste T (tweak), el cifrado se realiza de forma determinista obteniendo de esta manera el mensaje cifrado.

Para poder realizar el cifrado y descifrado de FFX se tienen en cuenta los parámetros permitidos que se muestran los apéndices A y B de [19], tales como: radix, que es el número de caracteres que contiene dicho alfabeto, cuyos valores pueden ser 2, 10 y 26 que corresponden a los bits, dígitos y letras del abecedario respectivamente, el número de rondas rnds(n), el grado de desequilibrio [10] split(n) y la función de ronda F en la red de Feistel, los cuales una vez que son seleccionados permanecen fijos hasta que la llave que se está utilizando sea cambiada.

Cabe destacar que FFX es una consecuencia del modo de cifrado de conjunto finito de Feistel (Feistel Finite Set Encryption Mode; FFSEM) [17] que fue presentada al NIST por T. Spies, siendo FFX más general pues utiliza alfabetos no binarios, split no equilibrados, entre otros.

Por otro lado, BPS surge como un algoritmo de cifrado que preserva formato que puede ser muy simple y a la vez flexible, pues está diseñado para cifrar cadenas con caracteres que pueden ser de cualquier conjunto (bits, dígitos, hexadecimales, etc.). Una parte importante de este cifrado esta basado en la propuesta de Black y Rogaway [2] con algunas modificaciones para poder trabajar con cualquier longitud de las cadenas, además de agregar el ajuste (tweak).

El cifrado BPS tiene dos componentes: un *cifrado por bloque interno* que puede ser TDES, AES o una función hash: SHA-2 [32] y, un *modo de operación* para manejar cadenas largas.

Un aspecto importante, es que, el cifrado por bloque interno de BPS está basado en una *red de Feistel*, pues es considerado como un método robusto para llevar a cabo la construcción de cifrados por bloques y es considerada la mejor y más simple opción para resolver el problema del cifrado que preserva formato, que ya había sido notado por los pioneros en este tipo de cifrado, Black y Rogaway [2]. Además de que dicho cifrado por bloque interno tiene como función realizar tanto el proceso de cifrado como de descifrado.

[10] split representa la division de la cual se hablo en la sección de redes de Fiestel.

El proceso en el cifrado por bloque interno es muy similar al modo de cifrado *CBC* (ver sección 1.1.1), con la diferencia de que en lugar de realizar la suma XOR se suma cada carácter por separado módulo s.

Una cualidad de BPS es su *adaptabilidad*, pues uno puede usarlo para cifrar cadenas de 2 a $max_b\ 2^{16}\ s-enteros$, escogidos de cualquier conjunto. Otro aspecto importante es que todos los cifrados por bloque y funciones hash estandarizados (TDES, AES o SHA-2) se puede utilizar como *bloque interno*.

Otra cualidad es su *eficiencia*, pues la llave de entrada para todos los cifrados por bloque interno es constante, lo que implica que se requerirá de solo una llave de cifrado interno por cifrado BPS. Se recomienda utilizar $w = 8$ rondas para la red de Feistel para que el proceso de cifrado sea muy eficiente.

Capítulo 3

Aplicación computacional

En este capítulo, se describen las características de la aplicación computacional para una base de datos cuyos campos contienen: número de cliente, número de tarjeta de crédito/débito, número identificador (OCR) de la credencial del INE y número telefónico; que se realizó para ilustrar el funcionamiento de un esquema de cifrado que preserva formato, basándonos en [26].

Un aspecto importante de mencionar, es que, en [26] los autores describen los pasos a seguir para realizar un cifrado que preserve formato en un número de tarjeta de crédito/débito. Sin embargo en esta aplicación se decidió crear una base de datos con los campos ya mencionados, pero siguiendo las ideas presentadas en [26], con una modificación y agregando algunos pasos.

La razón de utilizar una base de datos con más campos además de los números de tarjeta de crédito/débito es debido a que este tipo de cifrado está pensado para utilizarse en bases de datos en general, como se menciona en la Introducción de este trabajo.

En la mayoría de los artículos que se revisaron referentes a los cifrados que preservan formato, los algoritmos que se describen para poder llevar a cabo estos cifrados están basados en cifrados por bloque, ya sea AES-128 o DES.

Los datos de cada uno de los campos de la base de datos se generaron de la siguiente manera: los números de tarjetas de crédito/débito fueron generados con la información del apéndice A sección A.2 y la tabla II de dicha sección. Mientras que los números identificadores OCR (INE) y números telefónicos fueron generados aleatoriamente siguiendo el hecho de que un número identificador OCR (INE) tiene 13 dígitos y un número telefónico 10.

Se implementó el cifrado AES-128, además de la suma XOR y el código AIKEN (2412) como en [26]. Y como en todo proceso de cifrado se utiliza una llave, para esta aplicación existen tres posibilidades de llaves:

1. K_1=2b7e151628aed2a6abf7158809cf4f3c, la cual es normalmente empleada cuando se muestra el funcionamiento del cifrado AES-128. Un ejemplo de cómo funciona éste cifrado con dicha llave se puede encontrar en [27] y para la descripción del proceso de cifrado de AES ver [10].

2. K_2, llave generada utilizando 4 parámetros, tres de ellos al azar y uno fijo. Para ver como se genera esta llave ver Apéndice B.1.

3. K_3, llave generada utilizando el resumen de la función hash MD5 aplicada a la base de datos en claro y los 4 parámetros que se utilizaron en K_2. La generación de esta llave se puede consultar en el Apéndice B.2 .

Pero, ¿qué es el código AIKEN?, para contestar esta pregunta primero se dará una breve descripción del código que se utiliza normalmente en el proceso de cifrado: codificación binaria decimal BCD (Binary Code Decimal) o código $(8, 4, 2, 1)$ [18].

El código 8421 es un sistema de numeración ponderado, ya que cada posición de una secuencia de dígitos tiene asociado un peso, en este código los pesos son: 2^3, 2^2, 2^1, 2^0, por lo que el bit que se encuentra más a la derecha es el menos significativo pues tiene un peso de 2^0, mientras que el que se encuentra más a la izquierda es el más significativo pues tiene el mayor peso; 2^3. En este sistema de codificación los números son expresados en 4 bits como se muestra en la tabla I.

Código (8421)								
BCD	8421	8421	8421	8421	8421	8421	8421	8421
Decimal	0	1	2	3	4	5	6	7
Binario	0000	0001	0010	0011	0100	0101	0110	1011
Decimal	8	9	10 (A)	11 (B)	12 (C)	13 (D)	14 (E)	15 (F)
Binario	1000	1001	1010	1011	1100	1101	1110	1111

Tabla I: Codificación binaria decimal (8421).

El código AIKEN [21] es similar a éste código pero con un par de diferencias, mientras que el código binario decimal trabaja con los pesos $8, 4, 2, 1$, AIKEN tiene los pesos: $2, 4, 2, 1$ y solo es para los valores de 0-9, obteniendo así la representación mostrada en la tabla II.

Código AIKEN (2421)										
Decimal	0	1	2	3	4	5	6	7	8	9
AIKEN	2421	2421	2421	2421	2421	2421	2421	2421	2421	2421
	0000	0001	0010	0011	0100	1011	1100	1101	1110	1111

Tabla II: Código AIKEN.

A continuación se describen cada uno de los pasos que se sigue en el esquema de cifrado que se muestra en la figura 3.1 diseñado para una base de datos con los campos ya mencionados:

1. **Cifrado AES**: En este paso se utiliza la llave previamente mencionada y el cifrado AES-128 en cada uno de los campos a cifrar de la base de datos, que son: número de tarjeta de crédito/débito, número identificador OCR (INE) y número telefónico, obteniendo así el cifrado correspondiente de 32 caracteres hexadecimales.

2. **Almacenamiento de caracteres del cifrado AES**: Recordemos que nuestra base de datos tiene tres campos diferentes por lo cual se obtendrán tres cifrados AES diferentes, uno por cada campo cifrado.

   Para el campo de número de tarjeta de crédito/débito se almacenan en un arreglo los caracteres en posiciones impares del cifrado AES, comenzando de izquierda a derecha, siendo el primer carácter el que se encuentre en la posición uno. Dicho arreglo es almacenado con el nombre de V1, el cual será utilizado para el proceso de descifrado correspondiente a los números de tarjeta de crédito/débito.

   En el caso de los campos de número identificador OCR (INE) y número telefónico el proceso es un poco diferente, pues debido a que el número identificador OCR (INE) consta de 13 dígitos, solo se trabajarán los primeros 26 de los 32 caracteres resultantes del cifrado AES. De esta manera, se generarán dos arreglos para este campo, el primero almacenará los 26 primeros caracteres en posición impar de izquierda a derecha del cifrado AES correspondiente, mientras que los 6 caracteres restantes se almacenarán en el otro arreglo.

   Por otro lado, como el número telefónico constan de 10 dígitos, un arreglo almacenará los primeros 20 caracteres en posiciones impares del cifrado AES

correspondiente, mientras que los 12 caracteres restantes se almacenarán en el otro arreglo. Este par de arreglos como los generados para el número identificador OCR (INE) serán utilizados en el proceso de descifrado correspondiente a cada campo.

3. **Representación binaria del cifrado AES**: Una vez que se tienen los 32, 26 y 20 caracteres hexadecimales de los números de tarjeta de crédito/débito, número identificador OCR (INE) y número telefónico respectivamente, se utiliza el *código binario decimal* que se muestra en la tabla I para hacer su correspondiente representación binaria, donde cada carácter es representado por 4 bits.

4. **Suma XOR entre pares consecutivos**: Una vez realizada la representación binaria de los 32, 26 y 20 caracteres se obtienen 128, 104 y 80 bits respectivamente. Se toman pares consecutivos de 4 bits realizando la operación XOR entre ellos, obteniendo así 64, 52 y 40 bits respectivamente.

5. **Representación hexadecimal de la suma XOR**: De los 64 bits obtenido previamente se irán tomando de 4 en 4 bits donde cada uno de estos conjuntos de 4 bits corresponderán a un número hexadecimal siguiendo la tabla I. A los números hexadecimales resultantes se les pondrá un identificador A si estos son mayores o iguales a 10, en caso contrario no se les pondrá ninguna identificador. Por ejemplo, si el número fuera 11 se representaría como 11_A. Es importante recordar que de estos 64 bits obtendremos 16 caracteres hexadecimales.

   Se realiza el mismo procedimiento para los 52 y 40 bits, obteniendo así 13 y 10 caracteres hexadecimales respectivamente, los cuales puede ser o no mayor a 10, por lo cual pueden o no tener el identificador A.

   Cada uno de los 16, 13, 10 caracteres hexadecimales resultantes identificados o no con A son almacenados en un archivo para ser utilizado en el proceso de descifrado.

6. **Aplicando el código AIKEN o el código binario decimal**: Si el valor en la representación hexadecimal tiene el identificador A se le aplicará el código AIKEN que se muestra en la tabla II, en caso contrario el código binario decimal mostrado en la tabla I.

El resultado final serán los campos de número de tarjeta de crédito/débito, número identificador OCR (INE) y número telefónico cifrados, cuyo formato es el mismo que

el de los campos en claro (16, 13 y 10 dígitos respectivamente).

Ahora bien, en algún momento se deseará recuperar la base de datos original, es decir, cada uno de los campos cifrados en claro. Además de que en [26] solamente se menciona el proceso de cifrado para el número de tarjeta de crédito/débito, no es mencionado del proceso de descifrado para éste.

Se realizó un análisis al proceso de cifrado para cada uno de los campos de la base de datos, con la finalidad de ver cuáles serían los pasos a seguir en el proceso de descifrado, dicho análisis nos dio como resultado el esquema de descifrado mostrado en la figura 3.2.

Durante el análisis de cual podría ser el resultado de simplemente seguir el proceso de cifrado en sentido contrario y así poder plantear el proceso de descifrado (sin hacer modificación alguna a lo propuesto en [26]), se encontró que no sería posible obtener nuevamente los valores que resultaron de aplicar el proceso de cifrado AES-128, por lo que al aplicar el proceso de descifrado de AES a estos valores no se obtendrían los campos originales (en claro).

Es por ello que se realizó una modificación cuando se aplica el código AIKEN (paso 6), pues este ya no se aplicaría a cada uno de los valores obtenidos, como en [26], sino que ahora solo se aplicaría a los valores mayores o iguales a 10, esto para cada uno de los campos de la base de datos.

Pero, ¿por qué no aplicar a todos los valores el código Aiken?, por ejemplo, si tomamos el 7 de la representación hexadecimal la suma *XOR* (paso 5) en el esquema de cifrado que preserva formato en el campo de número identificador OCR (INE) que se muestra en la tabla V.

Se sabe que la representación binaria de 7 (utilizando el código binario decimal (8421) mostrado en la tabla I) es: *0111*, si aplicáramos el código AIKEN obtenemos nuevamente el *7*, ahora bien, para el proceso de descifrado al *7* le corresponde la representación en $0's$ y $1's$ de *1101* según la tabla II, que como representación binaria le corresponde el valor décimal de *13* (observar la tabla III), obteniendo así un resultado no deseado, pues esperábamos el *7*, por lo que, el proceso de descifrado no nos dará como resultado el número identificador OCR (INE) original (en claro).

Observemos que para realizar el proceso de descifrado en cada uno de los campos de la base de datos correctamente, es necesario obtener nuevamente los 32 caracteres hexadecimales correspondientes al cifrado AES para cada campo, con lo cual podamos

aplicar el descifrado AES y así obtener cada uno de los campos en claro. Es con esta finalidad que se agrega el paso 2: *almacenamiento de caracteres del cifrado AES* a lo propuesto en [26].

Pues mientras que en el proceso de cifrado para el campo de número de tarjeta de crédito/débito se utilizan los 32 caracteres hexadecimales, para los campos del número identificador OCR (INE) y número telefónico solo se trabajarán con los primeros 26 y 20 caracteres hexadecimales respectivamente como se puede observar en V y VI.

Cifrado				
Valor	Décimal	Representación binaria	AIKEN(2421)	Cifrado
7	7	0 1 1 1	0+4+2+1	7
Descifrado				
Valor cifrado		AIKEN(2421)-Representación binaria		Décimal
7		1 1 0 1		13

Tabla III: Cifrado implementando código AIKEN sin hacer distinción alguna, descifrado utilizando la tabla II.

Los siguientes pasos describen el esquema de descifrado que se muestran en la figura 3.2 para una base de datos :

1. **Representación en $0's$ y $1's$.**

   Este paso es el mismo para cada uno de los campos cifrados, aquí no se hace la representación binaria como tal, por lo que se hace referencia a representación en $0's$ y $1's$. Recordemos que en el proceso de cifrado se utiliza tanto la representación binaria con pesos (8,4,2,1) como el código AIKEN con pesos (2,4,2,1).

   Gracias a que en el proceso de cifrado se marcaron con A los valores mayores o iguales a 10, esto nos ayudará a realizar una correcta representación en $0's$ y $1's$, pues se utilizará el código binario decimal o el código AIKEN según corresponda, es decir, si el valor hexadecimal fue marcado con A o no.

   Se utiliza el archivo generado para cada uno de los campos en el proceso de cifrado en el paso de *Representación hexadecimal de la suma XOR*, el análisis del archivo lo realiza internamente el programa presentado en el Apéndice B y en las tablas del proceso de descifrado solo se muestra el resultado obtenido, que es la representación en $0's$ y $1's$.

2. **Obteniendo los valores faltantes para realizar el descifrado de AES.**

Debido a que en el proceso de cifrado de cada uno de los campos de la base de datos se utiliza el cifrado AES-128, para poder obtener nuevamente dichos datos, es importante volver a contar con los 32 caracteres hexadecimales y de esta manera poder hacer uso del proceso de descifrado de AES-128.

Dentro de esta etapa se hace mención del arreglo *V1* para cada uno de los campos, así como del arreglo *ValFal* para los campos de número identificador OCR (INE) y número telefónicos. En el proceso de cifrado se hizo mención de cómo están construidos cada uno de dichos arreglos, estos arreglos son almacenados de forma independiente y se accede a ellos durante el proceso de descifrado.

Pero, ¿Por qué es importante V1?, pues a partir de la suma *XOR* entre la representación binaria de estos valores y la representación en $0's$ y $1's$ podemos obtener los valores faltantes para tener nuevamente los 32 caracteres hexadecimales en el caso del campo de número de tarjeta de crédito/débito.

Para volver a obtener los 32 caracteres hexadecimales para los campos de número identificador OCR (INE) y número telefónico es necesario además de *V1* utilizar el arreglo de *VaFal* correspondientes. A continuación se describen los pasos a seguir para obtener nuevamente los 32 caracteres hexadecimales para cada uno de los campos de la base de datos.

El primer paso, es obtener los valores almacenado en los arreglos V1 correspondientes a cada uno de los campos. Una vez obtenidos dichos valores, se hace la representación binaria correspondiente, siguiendo la tabla I.

Ahora bien, como ya se cuenta con la representación en $0's$ y $1's$ y la representación binaria de los valores almacenados en V1, se realiza la suma XOR entre estas representación y se obtiene la representación hexadecimal correspondiente a la suma XOR.

Para el campo del número de tarjeta de crédito/débito se ordena los valores hexadecimales de V1 y de la Suma XOR. Se inicia con el primer valor hexadecimal de V1 seguido del primer valor hexadecimal de la suma XOR y así sucesivamente alternando entre estos dos conjuntos.

Por otro lado, para los campos de número identificador OCR (INE) y número telefónico además de ir alternando entre los valores hexadecimales de V1 y de la suma XOR, al final se concatenan los valores almacenados en el arreglo *ValFal* correspondiente. De esta manera obtendremos nuevamente los 32 caracteres hexadecimales del cifrado AES correspondiente a cada campo de la base de datos.

3. **Descifrado AES.** En este paso se le aplica el descifrado AES a los 32 caracteres hexadecimales obtenidos en el paso anterior, siguiendo los pasos descritos en [10].

En las tablas IV a VI y VII a IX se muestran los resultados de aplicar el esquema de cifrado y descifrado respectivamente, a un registro de la base de datos.

Es importante mencionar que dicha aplicación fue programada en lenguaje C, las características de la computadora son: procesador de 2.7 GHz Intel Core i7, memoria de 8GB 1333 MHz DDR3, con sistema operativo OS X Yosemite.

Los programas realizados tanto para el esquema de cifrado como de descifrado pueden correr en cualquier computadora siempre que tenga instalado un compilador de lenguaje C (por ejemplo, para Windows: DEV C++).

Los archivos utilizados para el proceso de cifrado son: una base de datos (BaseDeDatos.txt) cuyos campos almacenan números de tarjeta de crédito/débito, números identificadores OCR (INE) y números telefónicos, así como un archivo con la llave (key.txt) a utilizar, estos archivos deben estar en la misma carpeta que el programa para realizar el cifrado.

El proceso de descifrado está diseñado para que al igual que en el proceso de cifrado se lea la llave (key.txt) y calcule las subllaves necesarias para realizar el descifrado. Se lee la base de datos cifrada (BaseDeDatosFPE.txt) generada en el proceso de cifrado así como los archivos que son necesarios (FPEauxI.txt, FPEauxT.txt, FPEauxTEL.txt, V1_T.txt, V1_I.txt, V1_TEL.txt, VFAll.txt y VFalT.txt) para poder realizar el correcto descifrado.

La base de datos utilizada para ver el correcto funcionamiento de los programas cuenta con 100 registros (clientes) cada uno con sus correspondientes campos de número de tarjeta de crédito/débito, número identificador OCR (INE) y número telefónico, pero en general la base de datos puede tener la cantidad de registros (clientes) que se desee.

Un aspecto de seguridad a considerar es que tanto el archivo de la llave como los generados al momento de compilar el programa de cifrado deben ser almacenados en una carpeta a la cual solo pueda tener acceso el personal autorizado.

Otro aspecto importante es que si por algún motivo una persona no autorizada tuviera acceso a los archivos donde se almacena V1 y ValFal (valores faltantes), el intruso contaría con información importante para aplicar el descifrado de AES, pero no contaría con la información de qué papel juegan esos valores, es decir, cuales son sus correspondiente posiciones en el cifrado AES, por lo que un ataque que podría implementar es de *fuerza bruta*.

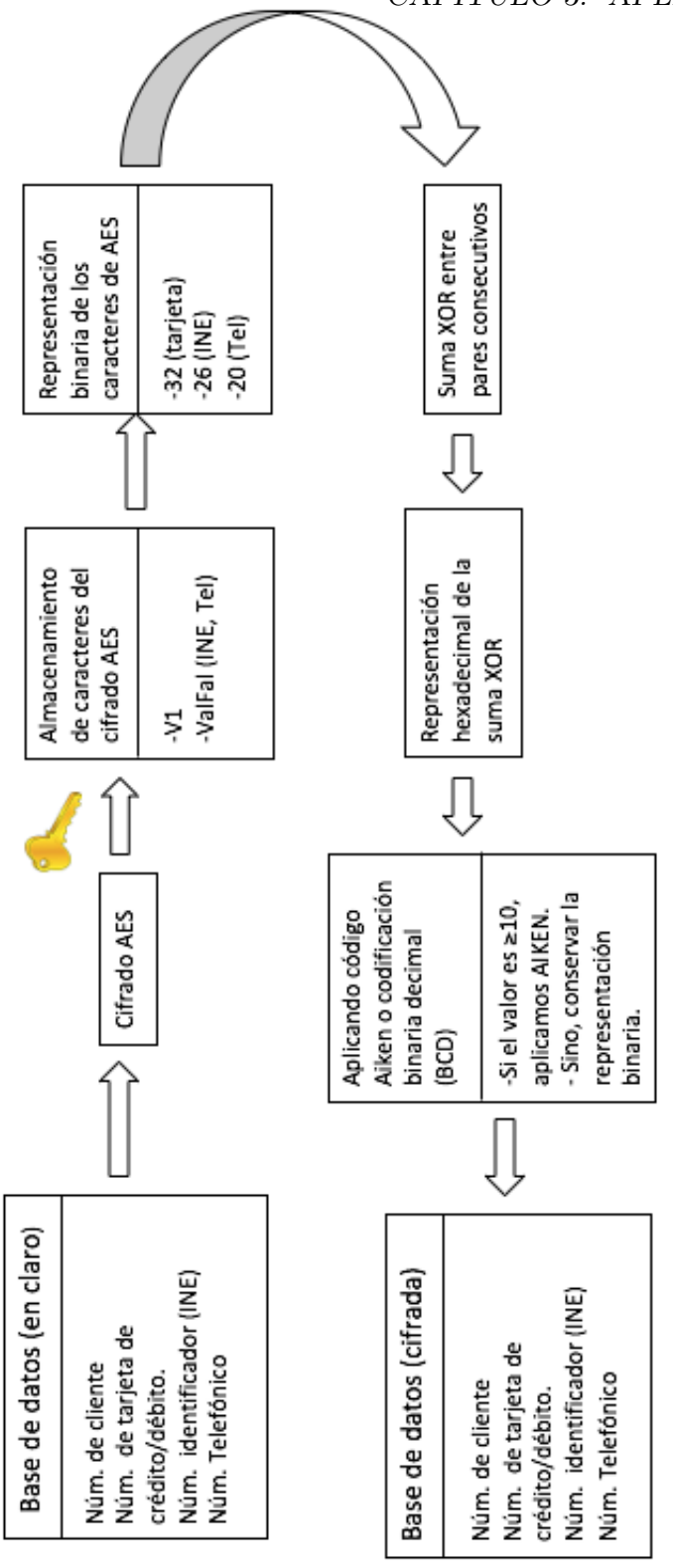


Figura 3.1: Esquema de cifrado que preserva formato para una base de datos cuyos campos son: número de cliente, número de tarjeta de crédito/débito, número identificador OCR (INE) y número telefónico.

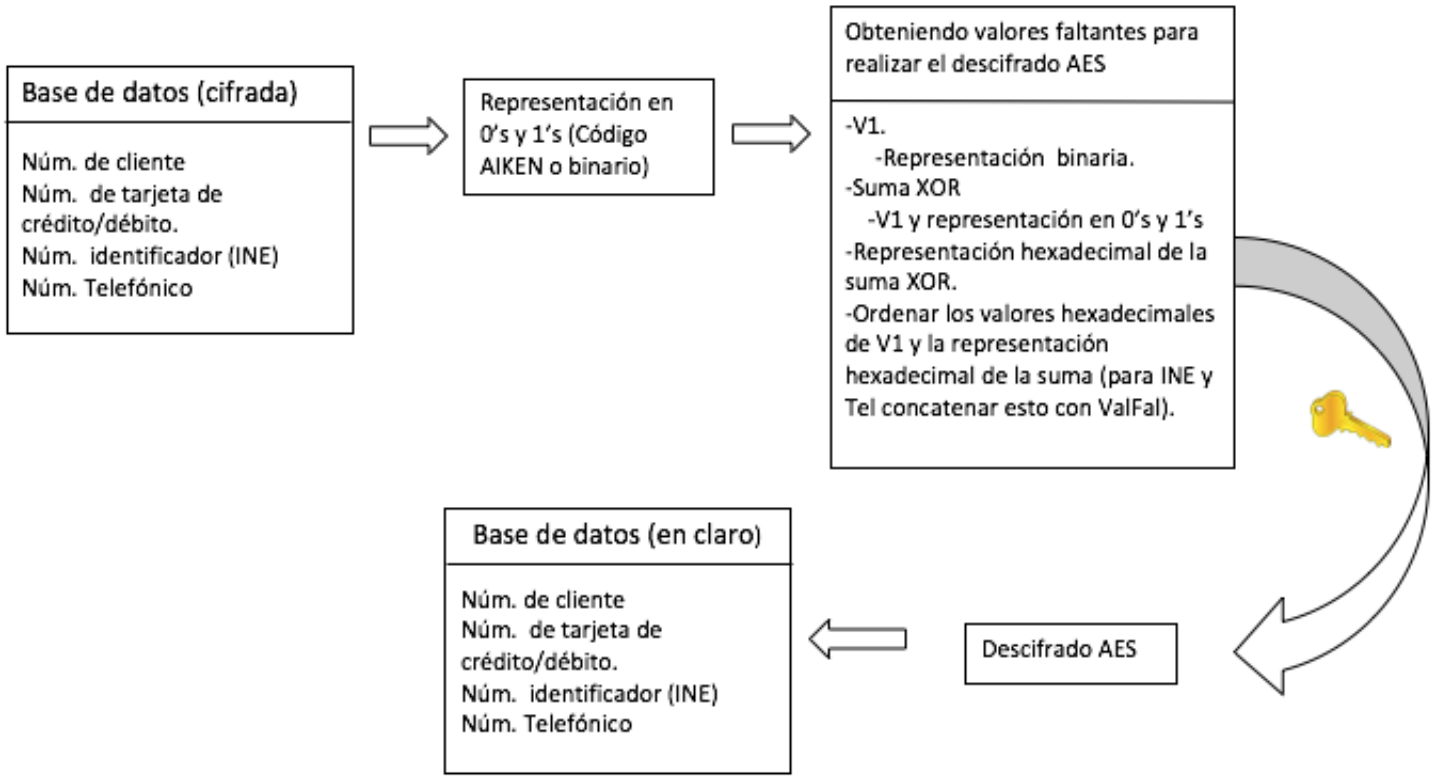


Figura 3.2: Esquema de descifrado que preserva formato para una base de datos cuyos campos son: número de cliente, número de tarjeta de crédito/débito, número identificador OCR (INE) y número telefónico.

Algoritmo de cifrado que preserva formato en el campo de tarjeta de crédito/débito
Datos del cliente Número de tarjeta de crédito/débito: 4547 9032 7725 0129
Cifrado AES
ea ff 83 67 56 f3 6c 98 8d 6e de 18 04 5a 6b 17
Almacenamiento de caracteres del cifrado AES
V1: e f 8 6 5 f 6 9 8 6 d 1 0 5 6 1
Representación binaria del cifrado AES
1110 1010 1111 1111 1000 0011$\cdots$0101 1010 0110 1011 0001 0111
Suma XOR en cada par consecutivo (EX-OR)
0100 0000 1011$\cdots$1111 1101 0110
Representación hexadecimal de la suma XOR
4 0 $11_A\cdots15_A$ 13_A 6
Aplicando el código AIKEN o codificación binaria decimal (BCD) Si el valor es ≥ 10 aplicamos el código AIKEN, sino conservar la representación binaria
BINARIO $2^3 2^2 2^1 2^0$ 0+4+0+0 | BINARIO $2^3 2^2 2^1 2^0$ 0+0+0+0 | AIKEN 2421 2+0+2+1 | ... | AIKEN 2421 2+4+0+1 | BINARIO $2^3 2^2 2^1 2^0$ 0+4+2+0
Número de tarjeta cifrado 4051 3641 5839 4976

Tabla IV: Cifrado del número de tarjeta con la llave K: 2b7e151628aed2a6abf7158809cf4f3c

Algoritmo de cifrado que preserva formato en el campo del número identificador OCR (INE)
Datos del cliente Número identificado OCR (INE): 0012348207752
Cifrado AES
ad 69 29 95 00 f1 19 0b cd 56 b8 3c 27 24 ba b9
Almacenamiento de caracteres del cifrado AES
V1(OCR): a 6 2 9 0 f 1 0 c 5 b 3 2 Últimos 6 caracteres del cifrado AES (ValFal(OCR)): 24 ba b9
Representación binaria de los primeros 26 caracteres del cifrado AES
1010 1101 0110 1001 0010 1001$\cdots$1011 1000 0011 1100 0010 0111
Suma XOR entre pares consecutivos (EX-OR)
0111 1111 1011$\cdots$0011 1111 0101
Representación hexadecimal de la suma XOR
7 15_A $11_A\cdots15_A$ 5
Aplicando el código AIKEN o codificación binaria decimal (BCD) Si el valor es ≥ 10 aplicamos el código AIKEN, sino conservar la representación binaria
BINARIO AIKEN AIKEN ... AIKEN BINARIO $2^3 2^2 2^1 2^0$ 2421 2421 ... 2421 $2^3 2^2 2^1 2^0$ 0+4+2+1 2+4+2+1 2+0+2+1 ... 2+4+2+1 0+4+0+1
OCR cifrado 7956088513395

Tabla V: Cifrado del número OCR con la llave K: 2b7e151628aed2a6abf7158809cf4f3c

Algoritmo de cifrado que preserva formato en el campo de número telefónico
Datos del cliente Número telefónico: 5568923671
Cifrado AES
b1 a4 0e 64 48 39 f7 f4 05 5e 45 b5 63 0a 65 6f
Almacenamiento de caracteres del cifrado AES
V1(Tel): b a 0 6 4 3 f f 0 5 Últimos 10 caracteres del cifrado AES (ValFal(Tel)): 45 b5 63 0a 65 6f
Representación binaria de los primeros 20 caracteres del cifrado AES
1011 0001 1010 0100 0000 1110$\cdots$11111 10100 0000 0101 0101 1110
Suma XOR entre pares consecutivos (EX-OR)
1010 1110 1110$\cdots$1011 0101 1011
Representación hexadecimal de la suma XOR
10_A 14_A $14_A\cdots11_A$ 5 11_A
Aplicando el código AIKEN o codificación binaria decimal (BCD) Si el valor es ≥ 10 aplicamos el código AIKEN, sino conservar la representación binaria
AIKEN AIKEN AIKEN ... BINARIO AIKEN 2421 2421 2421 ... $2^3 2^2 2^1 2^0$ 2421 2+0+2+0 2+4+2+0 2+4+2+0 ... 0+4+0+1 2+0+2+1
Número telefónico cifrado 4882648555

Tabla VI: Cifrado del número telefónico con la llave K: 2b7e151628aed2a6abf7158809cf4f3c

Algoritmo de descifrado que preserva formato en el campo de tarjeta de crédito/débito
Datos del cliente Número de tarjeta de crédito/débito (cifrado) 4051 3641 5839 4976
Representación en $0's$ y $1's$
Representación usando código Aiken y binario 1010 0000 0101 1000 1001 1110$\cdots$0011 1001 0100 1111 1101 0110
Obteniendo los valores faltantes para realizar el descifrado de AES
Valores almacenados (V1) e f 8 6 5 f 6 9 8 6 d 1 0 5 6 1 Representación binaria de V1 1110 1111 1000 0110 0101 1111$\cdots$01101 0001 0000 0101 0110 0001 Suma XOR con entre la representación binaria de V1 y la representación de $0's$ y $1's$ de la tarjeta 1010 1111 0011 0111 0110 0011$\cdots$1110 1000 0100 1010 1011 0111 Representación hexadecimal de la suma XOR 10 15 3 7 6 3 $\cdots$ 14 8 4 10 11 7 Ordenando los valores hexadecimales de V1 y la representación hexadecimal de la suma XOR 14 10 15 15 8 3 6 7 5 6 15 3$\cdots$13 14 1 8 0 4 5 10 6 11 1 7 Representación hexadecimal eaff836756f3$\cdots$de18045a6b17
Aplicamos el descifrado de AES
Número de tarjeta de crédito/débito original (descifrado) 4547 9032 7725 0129

Tabla VII: Descifrado del número de tarjeta con la llave K: 2b7e151628aed2a6abf7158809cf 4f3c

Algoritmo de descifrado que preserva formato en el campo OCR
Datos del cliente Número identificador OCR (INE) cifrado 7956088513395
Representación en $0's$ y $1's$
Representación usando código Aiken y binario 0111 1111 1011 1100$\cdots$0011 0011 1111 0101
Obteniendo los valores faltantes para realizar el descifrado de AES
Valores almacenados (V1(OCR)) a 6 2 9 0 f 1 0 c 5 b 3 2 Representación binaria de V1(OCR) 1010 0110 0010 1001$\cdots$0101 1011 0011 0010 Suma XOR con entre la representación binaria de V1(OCR) y la representación de $0's$ y $1's$ del OCR 1101 1001 1001 0101$\cdots$0110 1000 1100 0111 Representación hexadecimal de la suma XOR 13 9 9 5$\cdots$6 8 12 7 Ordenando los valores hexadecimales de V1(OCR) y la representación hexadecimal de la suma XOR concatenados con los valores faltantes (ValFal(OCR)) 10 13 6 9 2 9 9 5 0$\cdots$ 5 6 11 8 3 12 2 7 2 4 11 10 11 9 Representación hexadecimal ad6929950$\cdots$56b83c2724bab9
Aplicamos el descifrado de AES
Número identificador OCR (INE) original (descifrado) 0012348207752

Tabla VIII: Descifrado del OCR con la llave K: 2b7e151628aed2a6abf7158809cf4f3c

Algoritmo de descifrado que preserva formato en el campo de número telefónico
Datos del cliente Número telefónico (cifrado) 4882648555
Representación en $0's$ y $1's$
Representación usando código Aiken y binario 1010 1110 1110 0010 1100 1010 1000 1011 0101 1011
Obteniendo los valores faltantes para realizar el descifrado de AES
Valores almacenados (V1(Tel)) b a 0 6 4 3 f f 0 5 Representación binaria de V1(Tel) 1011 1010 0000 0110 0100 0011 1111 1111 0000 0101 Suma XOR con entre la representación binaria de V1(Tel) y la representación de $0's$ y $1's$ del número telefónico 0001 0100 1110 0100 1000 1001 0111 0100 0101 1110 Representación hexadecimal de la suma XOR 1 4 14 4 8 9 7 4 5 14 Ordenando los valores hexadecimales de V1(Te)l y la representación hexadecimal de la suma XOR concatenados con los valores faltantes (ValFal(Tel)) 11 1 10 4 0 14 6 4 4 8 3 9 15 7 15 4 0 5 5 14 4 5 11 5 6 3 0 10 6 5 6 15 Representación hexadecimal ad6929950$\cdots$56b83c2724bab9
Aplicamos el descifrado de AES
Número telefónico original (descifrado) 5568923671

Tabla IX: Descifrado del número telefónico con la llave K: 2b7e151628aed2a6abf7158809cf4f 3c

Capítulo 4

Tokenización

Dentro de las aplicaciones que se pueden incluir para los cifrados que preservan formato se encuentra la creación de lo que se conoce como *token*, es por ello que en este capítulo se incluye una breve descripción de éstos.

Para describir lo que es un token, primero es necesario definir el proceso mediante el cual se genera, *tokenización* [35]. Dependiendo de la aplicación en donde será utilizado un token, se define el proceso de tokenización, pues en algunas de ellas se requerirá que cumpla con más propiedades que en otras.

Definición 4.0.1. *La tokenización se puede definir como un proceso que transforma un dato sensible en un nuevo dato llamado token de modo que:*

1. *El token contenga suficiente información acerca del correspondiente dato en claro de manera que ésta pueda ser de utilidad en el futuro.*

2. *El token no revele información sensible contenida en el correspondiente dato en claro.*

Algunas de las aplicaciones en donde se aplican los tokens que cumplen con la definición anterior son:

- *Almacenamiento seguro (contraseñas)*: un ejemplo, es cuando son implementadas medidas para el control de acceso, pues en lugar de almacenar las contraseñas en claro son almacenados los token′s asociados a cada una de ellas, por lo que cuando alguien quiera acceder al sistema mediante su contraseña, esta será procesada para obtener su correspondiente token y así realizar la comparación entre éste y el almacenado y no entre contraseñas reales.

- *Tratamiento estadístico*: en estas aplicaciones son utilizados para cumplir con las leyes de protección de datos.

Un aspecto importante de esta definición y ejemplos de utilización, es que, no se requiere la propiedad de *reversibilidad*, es decir, que a partir del token no se puede recuperar toda la información del dato en claro asociado.

Es por ello que surge una segunda definición, que es utilizada en las aplicaciones que requieren de dicha propiedad, como es el caso de los datos bancarios.

Definición 4.0.2. *La tokenización es el proceso de transformación que modifica un dato sensible en un nuevo dato llamado token, de modo que:*

1. *El proceso de transformación es reversible e incluye como entrada una llave K, requiriendo la transformación inversa de la misma.*

2. *El token tiene esencialmente el mismo formato que el dato en claro.*

3. *El token no revela información sensible del dato en claro asociado.*

Con está definición es necesario además que el proceso de tokenización cumpla con los siguientes requerimientos:

1. Reversibilidad: si se intenta descifrar un token con la llave secreta correcta se debe obtener el dato en claro correspondiente.

2. Seguridad: en este aspecto se suele utilizar el modelo *real-random*, que esencialmente se quiere que un adversario que se enfrente a una máquina que realice el proceso *real* de tokenización no pueda distinguir si en un momento dado se ha cambiado dicho proceso por el proceso *random*. El proceso aleatorio lo único que hace es fijar una permutación aleatoria del conjunto de cadenas con el formato adecuado y una vez que es llamado devuelve el resultado de evaluar el dato en claro con la permutación aleatoria.

   Al utilizar una permutación aleatoria se elimina cualquier dependencia que pudiera existir con el dato en claro. Mientras que en el proceso de tokenización real, el token se construye a partir del dato en claro. Si los dos procesos son indistinguibles se puede demostrar que el adversario no será capaz de distinguir si se enfrenta a un proceso real o random.

Con la última definición y los requerimientos se puede definir el método de tokenización como un esquema que se construye sobre un cifrado que preserva formato.

Un ejemplo de cómo se obtiene un token de un *número de tarjeta de crédito/débito* se muestra en la figura 4.1 utilizando el esquema general (rank-then-encipher + ajuste):

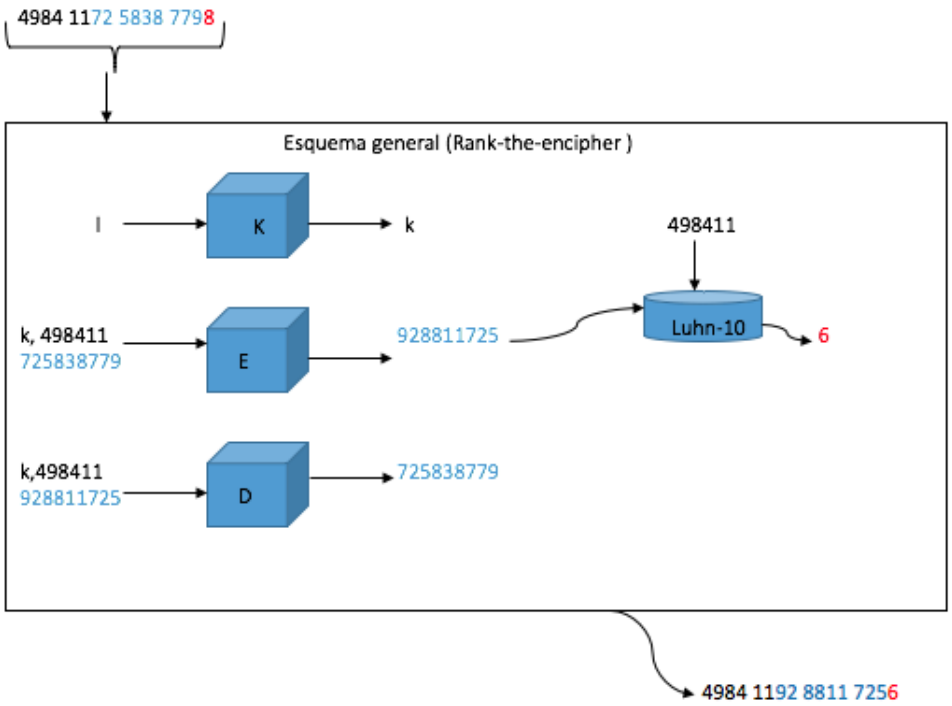


Figura 4.1: El esquema general (Rank-theh-encipher + ajuste) consta de 3 algoritmos: K algoritmo generador de llaves, E algoritmo de cifrado y D algoritmo de descifrado. Y como complemento se utiliza el algoritmo Luhn para crear un número de tarjeta de crédito/débito "válido".

En este esquema, nuestro dato en claro, es decir X, serán los dígitos que se encuentran de la posición 7 a la 15 del número de tarjeta, mientras que el ajuste, los primeros seis dígitos del número de tarjeta.

En el esquema general (rank-then-encipher + ajuste), se tienen tres algoritmos con sus correspondientes valores de entrada y salida. El algoritmo K tiene como entrada una cadena l que al ser introducida a K se genera una *llave pseudo-aleatoria* k. El algoritmo E tiene como entradas: la llave k, el ajuste y X, obteniendo así el dato cifrado, una nueva cadena de 9 dígitos.

El algoritmo D se utiliza para descifrar, es decir, obtener X nuevamente, tiene como entrada la llave k, el ajuste y la cadena de 9 dígitos resultantes del algoritmo E.

En el esquema *rank-the-encipher* + *ajuste* se utiliza el algoritmo de Luhn-10 el cual se describió en el capítulo A, para crear un número de tarjeta de crédito/débito "valido", pues se obtiene el dígito de validación, en este caso, 6.

Capítulo 5

Conclusiones

En la realización de este trabajo se estudiaron diferentes artículos que se encuentran en la literatura referente a los *cifrados que preservan formato* logrando así entender cuál es la importancia de estos, así como la diferencia que tienen con los *cifrados simétricos* (AES, DES). Pues mientras que en los cifrados simétricos no les importa que la información cifrada sea modificada totalmente en cuanto a su formato, los cifrados que preservan formato, como su nombre lo indica, busca que la información cifrada tenga el mismo formato que la información en claro (original).

Otro aspecto importante de estos cifrados, es su utilización en bases de datos, pues al emplearlos en ellas se le puede engañar al enemigo, ya que este podría pensar que la información de la base de datos cifrada es la información en claro, pues cada campo de la base de datos cifrada tendrá el mismo formato de la base de datos en claro (original).

En este trabajo se presenta una contribución original con la realización del esquema de cifrado que preserva formato para base de datos. Y si solo nos interesará cifrar un número de tarjeta de crédito/débito, se hizo una contribución basada en [26] pues en este trabajo se puede volver a obtener el número de tarjeta de crédito/débito en claro nuevamente, lo que no sucede en [26].

Además de que se crearon dos formas de obtener llaves de 128 bits pseudo-aleatorias descritas en los apéndices B.1 y B.2, las cuales son utilizadas en el proceso de cifrado y descifrado de la base de datos.

Como trabajo futuro, podemos señalar una modificación importante en la base de datos a la cual se le aplicará el esquema de cifrado/descifrado que preserva formato aquí presentado, que son campos que no solo contengan dígitos.

Apéndice A

Tarjetas de crédito/débito

Debido a que en el capítulo siguiente uno de los campos de la base de datos que se está empleando es el número de tarjeta de crédito/débito, en este capítulo se ilustrará cómo es que los números de tarjetas de crédito/débito son construidos y verificados para comprobar que sean *válidos*. Por otro lado, se trabajó con un conjunto de datos que al ser analizados se pudo obtener información útil permitiendo simular cómo es que se pueden obtener números de tarjeta *válidos*, esto mediante el uso de *Mathematica*.

Debido a que el manejo de los número de las tarjetas de crédito/débito es considerada como información sensible, el manejo de ellos debe de hacerse de manera cuidadosa. Es por ello que se utiliza criptografía que preserva formato, pues gracias a este esquema si los números de tarjeta caen en entidades no deseadas estos darán la apariencia que son los números originales cuando no es así.

Otra característica a considerar es el almacenamiento de estos números en bases de datos, las cuales tienen sus campos definidos con un tamaño y tipo de dato en particular por lo cual realizar el cifrado con un esquema tradicional no cumple con esta característica.

Todas las tarjetas de crédito/débito a partir de 1989 se tuvieron que adaptar a la norma ISO/TEC 7812 [6], la cual identifica la posición y significado de todos los dígitos del número de una tarjeta. El primer dígito esta en relación con la tabla I.

Enfocándonos en las tarjetas de crédito y débito de México [7] podemos observar que el número de estas en su mayoría cuentan con 16 dígitos, pero existen algunas con 15 dígitos como AMERICAN EXPRESS y 13 dígitos como VISA.

Dígito	
0	Norma ISO/TC 68 y otras funciones de la industria
1	Líneas aéreas
2	Líneas aéreas y otras tareas de la industria
3	Viajes y ocio
4	Servicios bancarios y financieros
5	Servicios bancarios y financieros
6	Venta y banca
7	Petróleo
8	Telecomunicaciones y otras áreas de la industria
9	Asignación nacional

Tabla I: Valores posibles para el primer dígito del número de las tarjetas de crédito/débito y su significado. Esta tabla fue obtenida de [7].

Los primeros siete dígitos en la tarjeta se corresponden con el número de emisor de la empresa o entidad financiera y el país. Por cuestiones de seguridad, la información completa de los emisores y países se mantiene confidencial aunque si se conocen los principales valores de los dígitos de las tarjetas como por ejemplo:

1. La tarjeta VISA comienza con 4, si son VISA Electron [7], sus primeros cuatro dígitos están en el conjunto $\{4026, 4175, 4508, 4844, 4912, 4917\}$.
2. MasterCard [7] tiene asignados los valores del conjunto $\{51, 52, 53, 54, 55\}$ para los primeros dos dígitos.

Del dígito 8 al 16 del número de la tarjeta, son el código interno de la entidad financiera (Banco) para asociar la tarjeta al cliente y corresponden con sus propios criterios de numeración. Cabe mencionar que el último dígito es el que se encarga de verificar que la tarjeta sea válida, para ello se utiliza el algoritmo de Luhn-10 [5] el cual se describirá en la siguiente subsección.

Los códigos de control de errores son diseñados a menudo para aplicaciones en las cuales se espera que ciertos tipos de errores se produzcan, tales como: teclear un número o entrar en una página web entre otros, destacando el escribir un dígito incorrecto o cambiar dos consecutivos.

A.1. Algoritmo de Luhn-10

Este algoritmo fue creado por Luhn [5], de IBM, con patente concedida en agosto de 1960. El algoritmo de Luhn detecta cualquier error en un *único* dígito pues verifica

un número de tarjeta contra su dígito de verificación. Este algoritmo nos dice que dado un número que contenga solamente dígitos [0-9], una tarjeta de crédito/débito es válida sí y solamente si la suma de sus dígitos es igual a cero módulo 10. Este algoritmo consiste de los siguientes pasos:

1. Dado un número de tarjeta, cada dígito en posición impar de izquierda a derecha, debe ser multiplicado por 2.

2. A cada dígito en posición par comenzando de izquierda a derecha no se le realiza modificación.

3. Si el número obtenido al realizar el paso (1) está comprendido por dos dígitos, estos son sumados entre sí para obtener un nuevo número con un solo dígito, es decir, si el resultado es 10 entonces 10: $1 + 0 = 1$.

4. Una vez obtenidos los nuevos dígitos se realiza la suma entre ellos módulo 10.

A continuación se expresa de forma matemática lo descrito anteriormente:

Definición A.1.1. Algoritmo de Luhn

Sea $TC=\mathbb{Z}_{10}^{16}$ *, donde* $\mathbb{Z}_{10}$ *es el anillo de enteros módulo 10. Sea* $N \in TC$*, es decir,* $N = n_1n_2n_3...n_{16}$*. Para* $i = 0, 1, 2, ..., 7$*,* $j = 1, 2, 3, ..., 8$ *comenzando de izquierda a derecha:*

Sean y_k *los nuevos valores de* n_k *para* $k = 1, 2, 3, ..., 16$ *entonces:*

1. *Para* n_{2i+1}*,* $y_{2i+1} = 2n_{2i+1}$*.*

   Si $y_{2i+1} = 10 + k_i$*,* $0 \leq k_i \leq 9$ *entonces* $y_{2i+1} = k_i + 1$*.*

2. *Para* n_{2j}*,* $y_{2j} = n_{2j}$*.*

Si $\sum_{i=0}^{7} y_{2i+1} + \sum_{j=1}^{8} y_{2j} \equiv 0$ *(mód 10) el número de la tarjeta es válido.*

Un ejemplo para ilustrar el funcionamiento del algoritmo de Luhn es el siguiente:

Ejemplo **A.1.** *Sea 3986 X295 5728 1742 el número de la tarjeta en el cual se tiene duda sobre el dígito que ocupa la posición de X. Se comienza identificando que posición ocupa X, es decir, si es un dígito en la posición par o impar. Como se puede observar fácilmente X es el quinto dígito del número de la tarjeta por lo que su posición*

es impar. Siguiendo el algoritmo de Luhn descrito previamente se realiza lo siguiente:

Llamaremos A a la suma de todos los dígitos en posición impar y B a la suma de los restantes dígitos.

$$\begin{aligned} A = \sum_{i=0}^{7} 2n_{2i+1} &= 2(3) + 2(8) + 2X + 2(9) + 2(5) + 2(2) + 2(1) + 2(4) \\ &= 6 + 7 + 2X + 9 + 1 + 4 + 2 + 8 \\ &= 2X + 37 \\ B = \sum_{j=1}^{8} n_{2j} &= 9 + 6 + 2 + 5 + 7 + 8 + 7 + 2 = 46 \end{aligned}$$

Entonces $A + B = 2X + 83 \equiv 0 \ (mod\ 10)$.

En la segunda igualdad de A es importante recordar el paso (3) del algoritmo de Luhn, para entender de donde salen los valores de 7, 9 y 1 que vienen de la multiplicación de 2 con 8, 9 y 5 respectivamente.

De la relación anterior tenemos que obtener el valor de X[1] *tal que la relación anterior se cumpla. En este ejemplo*[2] *si X=8 se tiene que* $2(8) + 83 = 16 + 83 = 7 + 83 \equiv 0 \ mod\ 10$. *El primer termino de la segunda igualdad proviene de aplicar el paso dos del algoritmo de Luhn y recordemos que el segundo termino de la igualdad, 83 se obtiene de suma 37 y 46 de A y B respectivamente por lo cual no se utilizará el algoritmo de Luhn. Por lo tanto el número de la tarjeta para que esta sea válida es: 3986 8295 5728 1742.*

El algoritmo de Luhn-10 no es perfecto en la detección de dígitos intercambiados, pues el intercambio de dos de ellos consecutivos generalmente da como resultado un fallo en la suma de comprobación, pero hay casos en los cuales no son detectados, como por ejemplo el intercambio de 9 y 0 no afecta en nada a la suma de comprobación.

También este algoritmo se podría pensar como un código detector de un solo error.

Se realizó una recopilación de algunos números tanto de tarjeta de crédito como débito para implementar en ellas el algoritmo de Luhn-10 y ver que este arrojaba que

[1] Recordemos que los valores que puede tomar un dígito esta entre [0-9].

[2] Donde la solución $X = 8$ es única pues solo esta permitido tomar valores entre [0-9] y este valor es el único que satisface la congruencia.

dichas tarjetas eran válidas. Por cuestiones de seguridad no se darán los dígitos del número de las tarjetas y también cabe mencionar que los resultados obtenido son solo para ilustrar mejor los resultados planteados anteriormente y no se pretende generar números de tarjeta válidos.

A.2. Análisis de números de tarjetas de crédito/débito

En este trabajo se analizaron números de tarjetas de débito/crédito construidos de las siguiente manera:

$$X_1X_2X_3X_4 \quad X_5X_6X_7X_8 \quad X_9X_{10}X_{11}X_{12} \quad X_{13}X_{14}X_{15}X_{16}$$

donde cada $X_i \in \{0, 1, 2, \cdots, 9\},\ i = 1, 2, 3, \ldots, 16$.

Dado un conjunto de números de tarjeta de crédito/débito procedentes de la misma entidad financiera (Banco) se observó que los únicos dígitos que diferían en cada uno de los números eran $X_{12}, X_{13}, X_{14}, X_{15}$ esto sin contar el dígito de verificación. Analizando este hecho se plantea la ecuación lineal (A.1) con dichos valores de tal forma que al aplicar el algoritmo de Luhn esta ecuación nos de los dígitos para los cuales el número de tarjeta sea *válido*.

$$x + 2y + z + 2u \equiv 0 \ (mod\ 10) \tag{A.1}$$

Donde $x, y, z, u \in \{0, 1, 2, \cdots, 9\}$ y representan los dígitos en las posiciones 12, 13, 14 y 15 respectivamente.

Una manera de resolver este problema fue realizando un programa en Mathematica donde se introdujo $x = -z - y - 2u$ obteniendo 1005 soluciones para esta ecuación, algunos de los resultados se muestran en la tabla II.

Valores Posibles			
(x, y, z, u)	(x, y, z, u)	(x, y, z, u)	(x, y, z, u)
$\{5, 0, 1, 2\}$	$\{7, 0, 3, 5\}$	$\{9, 0, 1, 0\}$	$\{4, 0, 6, 5\}$
$\{3, 3, 3, 9\}$	$\{4, 3, 4, 3\}$	$\{1, 1, 1, 3\}$	$\{5, 0, 5, 5\}$
$\{7, 0, 7, 8\}$	$\{9, 1, 3, 3\}$	$\{3, 0, 3, 2\}$	$\{9, 1, 3, 3\}$

Tabla II: Posibles valores de los dígitos restantes obtenidos resolviendo la ecuación $x + z + 2y + 2u = 0$ en $\mathbb{Z}_{10}$.

Algunas de las soluciones de la tabla II serán utilizadas en el capítulo siguiente para el campo de número de tarjeta de crédito/débito de la base de datos que se desea cifrar.

Apéndice B

Generación de llaves K_2 y K_3

B.1. Generación de la llave K_2

Las llaves K_2 y K_3 son llaves pseudo-aleatorias de 128 bits (32 caracteres hexadecimales). En la generación de la llave K_2 se utilizan los parámetros *par1, par2, clave y parLlave*. Mientras que para la llave K_3 se utilizan estos mismos parámetros agregando uno más, *mD5*, que almacena el resumen (una cadena de 32 caracteres hexadecimales) obtenido de aplicar la función hash MD5 a la base de datos en claro con la finalidad de hacer un poco más aleatoria la llave, pues distinta base de datos en claro será distinto resumen almacenado en el parámetro mD5.

A continuación se describen cada uno de los parámetros utilizados en la generación de la llave K_2.

- *par1* es una cadena de 8 dígitos, un ejemplo de la cadena que se puede utilizar para este parámetro es el de la fecha, como por ejemplo, 26072017, pues esta información será conocida tanto por la persona que cifre como la que descifre la base de datos. De igual forma se puede utilizar cualquier cadena con 8 dígitos siempre y cuando ambas partes la conozcan.

- *par2* es una cadena de 3 dígitos, por ejemplo, si fuera una base de datos de una institución bancaria o comercial, se podría considerar el número de sucursal, pues este valor será conocido por las partes interesadas.

- el parámetro *clave* es una cadena de 5 dígitos.

- *parLlave* es la cadena $\{3, 12, 9, 13, 14, 6, 1, 5, 5, 0, 4, 2, 0, 1, 4, 10\}$ la cual fue construida en base a tres nombre y una fecha.

Los caracteres almacenados en cada uno de los parámetros son re-ordenados para obtener una nueva cadena de 32 caracteres hexadecimales que corresponderán a la llave K_2. Los caracteres en posiciones impares hasta la posición 15 de la nueva cadena, serán los caracteres que se encuentran en las posiciones pares del parámetro *parLlare*.

Los caracteres en posiciones pares (hasta la posición 15) de la nueva cadena, serán los caracteres almacenados en el parámetro *par1*. Para las posiciones de 16 a 32 de la nueva cadena, las posiciones pares tendrán los caracteres de las posiciones impares del parámetro *parLlave*. Mientras que para las posiciones impares, los carácteres de los parámetros *par2 y clave* se irán alternando.

En el siguiente ejemplo se ilustra la construcción de la llave K_2.

Ejemplo **B.1.** *parLlave*=$\{3, 12, 9, 13, 14, 6, 1, 5, 5, 0, 4, 2, 0, 1, 4, 10\}$*, par1: 30062017, par2: 089, clave: 37892. entonces*

K_3= 33090146125041074120133685709281910 2 y expresada como cadena hexadecimal se obtiene la llave K_3= 33090e6125041074c0d36857092819a2.

Un aspecto importante de mencionar es que tanto la entidad que cifra como la que desea recuperar la base de datos en claro (descifra) tienen que conocer los parámetros *par1, par2, clave* sin necesidad de transmitirlos por un canal de comunicación que pueda ser inseguro, sino que deben ser valores a los que solo se puedan tener acceso las entidades involucradas.

```
/*-------------------------------------------------------------------------------
Archivo: Llave.c
Autora: Cristina Vargas Puente
Descripcion: Este programa genera una llave de 128 bits (32 caracteres hexadecimales
    ), con los parametros par1, par2, clave y parLlave que representan una cadena
    decimal de 8, 3, 5 y 16 caracteres respectivamente.
-------------------------------------------------------------------------------*/
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<string.h>
int main(){
        remove("key2.txt");
        char par1[9],par2[4],clave[6];
        int parLlave[16]={3,12,9,13,14,6,1,5,5,0,4,2,0,1,4,10},n1,n2,n3;
        int j=0,k=0,l=0, o=1,s=0,m=0,s1=17,s2=19;
        int llavefinal[32]={0},auxpar1[8]={0},auxpar2[3]={0},auxclave[5]={0};

        printf("Introduce el parametro 1: ");
        gets(par1);
        n1=strlen(par1);
        while(n1<8 || n1>8){
                printf("La informacion es incorrecta\n");
                printf("Introduce el parametro 1: ");
```

```
        gets(par1);
        n1=strlen(par1);
}
printf("Introduce el parametro 2: ");
gets(par2);
n2=strlen(par2);
while(n2<3 || n2>3){
        printf("La informacion es incorrecta\n");
        printf("Introduce el parametro 2: ");
        gets(par2);
        n2=strlen(par2);
}
printf("Clave: ");
gets(clave);
n3=strlen(clave);
while(n3<5 || n3>5){
        printf("La informacion es incorrecta\n");
        printf("Clave: ");
        gets(clave);
        n3=strlen(clave);
}
printf("El parametro 1 es: %s, parametro 2: %s, clave %s\n",par1, par2,
    clave);
for(int i=0;i<8;i++){
        auxpar1[i]=par1[i]-48;
}
for(int i=0;i<3;i++){
        auxpar2[i]=par2[i]-48;
}
for(int i=0;i<5;i++){
        auxclave[i]=clave[i]-48;
}
for(int i=1;i<16;i=i+2){
        llavefinal[i]=parLlave[j];
        j=j+2;
}
for(int i=0;i<16;i=i+2){
        llavefinal[i]=auxpar1[l];
        l++;
}
for(int i=16;i<32;i=i+2){
        llavefinal[i]=parLlave[o];
        if(s1<26){
                llavefinal[s1]=auxpar2[s];
                s1=s1+4;
        }if(s2<28){
                llavefinal[s2]=auxclave[m];
                s2=s2+4;
        }
        llavefinal[29]=auxclave[3];
        llavefinal[31]=auxclave[4];
        s++;
        m++;
        o=o+2;
}
freopen("key2.txt","a",stdout);
for (int i=0; i<32; i++){
                if(llavefinal[i] <= 9){
                        printf("%d",llavefinal[i]);
                }
                else if(llavefinal[i]==10){
```

```
                                printf("a");
                        }
                        else if(llavefinal[i]==11){
                                printf("b");
                        }
                        else if(llavefinal[i]==12){
                                printf("c");
                        }
                        else if(llavefinal[i]==13){
                                printf("d");
                        }
                        else if(llavefinal[i]==14){
                                printf("e");
                        }
                        else if(llavefinal[i]==15){
                                printf("f");
                        }
                }
        fclose(stdout);
        return 0;
}
```

B.2. Generación de la llave K_3

Como se menciona en B.1, en la generación de K_3 se utiliza la función hash MD5 de la base de datos en claro, la cual es almacenada en el parámetro *mD5*.

Los valores almacenados en los parámetros *par1, par2 y clave* son concatenados con los valores almacenados en el parámetro *parLlave*, generando así una cadena hexadecimal de 32 caracteres, la cual es sumada con el resumen obtenido de MD5 (que es una cadena hexadecimal de 32 caracteres) parámetro *mD5*, módulo 16, obteniendo así una nueva cadena hexadecimal de 32 caracteres que corresponderá a la llave K_3.

A continuación se ilustra la generación de la llave K_3 con un ejemplo.

Ejemplo **B.2.** *parLlave*=$\{3, 12, 9, 13, 14, 6, 1, 5, 5, 0, 4, 2, 0, 1, 4, 10\}$, *par1: 30062017, mD5*=$\{d, 7, c, f, 5, 2, 0, 6, a, d, d, 0, 5, f, c, 2, b, c, 0, 9, c, 6, c, 9, c, c, b, e, 6, a, 9, e\}$, *par2: 089, clave: 37892 entonces:*

Concatenando los valores de los parámetros par1, par2, clave *y* parLlave*:*
$\{3, 7, 8, 3, 0, 0, 6, 2, 0, 1, 7, 0, 8, 9, 9, 2, 3, 12, 9, 13, 14, 6, 1, 5, 5, 0, 4, 2, 0, 1, 4, 10\}$

Sumando módulo 16, la cadena anterior con la cadena almacenada en mD5:
$K_3 = \{0e425268ae40d854e896acde1cf06bd8\}$

```
/*-------------------------------------------------------------------------------
Archivo: Llave2.c
Autora: Cristina Vargas Puente
Descripcion: Este programa genera una llave de 128 bits (32 caracteres hexadecimales
    ), con los parametros par1, par2, clave, parLlave y mD5 que representan cadenas
    de 8, 3, 5, 16, 32 digitos respectivamente.
-------------------------------------------------------------------------------*/
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<string.h>
int main(){
      remove("key3.txt");
      char par1[9],par2[4],clave[6];
      int parLlave[16]={3,12,9,13,14,6,1,5,5,0,4,2,0,1,4,10},n1,n2,n3;
      int a=10, b=11, c=12, d=13, e=14, f=15;
      int mD5[32]={d,7,c,f,5,2,0,6,a,d,d,0,5,f,c,2,b,c,0,9,c,6,c,9,c,c,b,e,6,a,9,e};
      int j=0,k=0,l=0, prellave[32]={0}, pl=0;;
      int llavefinal[32]={0},auxpar1[8]={0},auxpar2[3]={0},auxclave[5]={0};
      int llaveCasi[16]={0}, ll1=0, ll2=0, ll3=0;

         printf("Introduce el parametro 1:  ");
         gets(par1);
         n1=strlen(par1);
         while(n1<8 || n1>8){
                 printf("La informacion es incorrecta\n");
                 printf("Introduce el parametro 1: ");
                 gets(par1);
                 n1=strlen(par1);
         }
         printf("Introduce el parametro 2: ");
         gets(par2);
         n2=strlen(par2);
         while(n2<3 || n2>3){
                 printf("La informacion es incorrecta\n");
                 printf("Introduce el parametro 2: ");
                 gets(par2);
                 n2=strlen(par2);
         }
         printf("Clave: ");
         gets(clave);
         n3=strlen(clave);
         while(n3<5 || n3>5){
                 printf("La informacion es incorrecta\n");
                 printf("Clave: ");
                 gets(clave);
                 n3=strlen(clave);
         }
         printf("El parametro 1 es: %s, parametro 2: %s, clave: %s \n",par1, par2,
             clave);
         for(int i=0;i<8;i++){
                 auxpar1[i]=par1[i]-48;
         }
         for(int i=0;i<3;i++){
                 auxpar2[i]=par2[i]-48;
         }
         for(int i=0;i<5;i++){
                 auxclave[i]=clave[i]-48;
         }
```

```
        for(int i=0;i<16;i++){
                if(i<3){
                        llaveCasi[i]=auxclave[i];
                        ll3++;
                }else if(i<11){
                        llaveCasi[i]=auxpar1[ll1];
                        ll1++;
                }
                else if(i< 14){
                        llaveCasi[i]=auxpar2[ll2];
                        ll2++;
                }
                else{
                        llaveCasi[i]=auxclave[ll3];
                        ll3++;
                }
        }
        for(int i=0;i<32;i++){
                if(i<16){
                        prellave[i]=llaveCasi[i];
                }else{
                        prellave[i]=parLlave[pl];
                        pl++;
                }
        }
        for(int L=0;L<32;L++){
                llavefinal[L]=(prellave[L]+mD5[L])%16;
        }
        printf("\n");

        freopen("key3.txt","a",stdout);
        for (int i=0; i<32; i++){
                if(llavefinal[i] <= 9){
                        printf("%d",llavefinal[i]);
                }else if(llavefinal[i]==10){
                        printf("a");
                }else if(llavefinal[i]==11){
                        printf("b");
                }else if(llavefinal[i]==12){
                        printf("c");
                }else if(llavefinal[i]==13){
                        printf("d");
                }else if(llavefinal[i]==14){
                        printf("e");
                }else if(llavefinal[i]==15){
                        printf("f");
                }
        }
        fclose(stdout);

        return 0;
}
```

Apéndice C

Pseudocódigos del cifrado y descifrado

Este apéndice muestra los pseudocódigos para realizar el proceso de cifrado y descifrado de la aplicación computacional desarrollada en este trabajo.

Pseudocódigo de cifrado.

```
for ya < − 0 to ya<NUMTARJETAS do
    cifrado < −AES_K(NumTarjeta);
    cifradoOCR < −AES_K(NumeroOCR);
    cifradoTel< −AES_K(NumTel);

//Pasando a binario la representación decimal

    for ren< − 0 to ren<32 do
        for i< − 0 to i<4 do
            Pot2< −pow(2,(3-i));
            if (cirado[ren][i]=1) then
                cifrado[ren]< −(cifrado[ren]-Pot2);
            else
                Auxcifrado[ren][i] < − 0;
            end if
        end for
        j1++;
    end for
```

```
    for renOCR< −0 to renOCR<26 do
        for i< −0 to i<4 do
            Pot2OCR< − pow(2,(3-i));
            if (cifradoOCR[renOCR]>=Pot2OCR) then
                AuxCifradoOCR[renOCR][i]¡-1;
                cifradoOCR[renOCR]< −(cifradoOCR[renOCR]-Pot2OCR);
            else
                AuxcifradoOCR[renOCR][i]< − 0;
            end if
        end for
        j1OCR++;
    end for

    for renTel< − 0 to renTel<20) do
        for (i< − 0 to i<4 do
            Pot2Tel< − pow(2,(3-i));
            if (cifradoTel[renTel])>=Pot2Tel then
                AuxcifradoTel[renTel][i]< −1;
                cifradoTel[renTel]< −(cifradoTel[renTel]-Pot2Tel);
            else
                AuxCifradoTel[renTel][i]< − 0;
            end if
            j1Tel++;
        end for
    end for

//Haciendo la operación suma por pares XOR
    for rensum < −0 to rensum<16 do
        for colsum < −0 to colsum< 4 do
            sumbinCif[rensum][colsum]< −(AuxCifrado[2*rensum][colsum];
                                        +AuxCifrado[(2*rensum)+1][colsum])mod 2;
        end for
    end for

    for rsumOCR< −0 to rsumOCR<13 do
        for csumOCR< −0 to csumOCR< 4 do
            sbinCifOCR[rsumOCR][csumOCR]< −(AuxCifradoOCR[2*rsumOCR][csumOCR];
                            +AuxCifradoOCR[(2*rsumOCR)+1][csumOCR])mod 2;
        end for
    end for

    for rsumTel < − 0 to rsumTel<13 do
        for csumTel < −0 to csumTel < 4 do
```

```
            sbinCifTel[rsumTel][csumTel] < − (AuxCifradoTel[2*rsumTel][csumTel];
                                +AuxCifradoTel[(2*rsumTel)+1][csumTel])mod 2;
        end for
    end for

//Condicional para Aplicar el codigo AIKEN
    for i< −0 to i< 16 do
        auxpowE < − 0;
        for j< −0 to j< 4 do
            auxpowE < − auxpowE+(sumbinCif[i][j]*pow(2,3-j));
        end for
        DecEXOR[i] < − auxpowE;
    end for

    for i< −0 to i< 13 do
        auxpowEOCR< −0;
        for j< −0 to j< 4 do
            auxpowEOCR< −auxpowEOCR+(sbinCifOCR[i][j]*pow(2,3-j));
        end for
        DecEXOROCR[i]< −auxpowEOCR;
    end for

    for i< −0 to i< 10 do
        auxpowETel < − 0
        for j< −0 to j< 4 do
            auxpowETel < − auxpowTel+(sbinCifTel[i][j]*pow(2,3-j));
        end for
        DecEXORTel[i] < − auxpowETel;
    end for

//Aplicando el codigo AIKEN
    for m1< −0 to m1<16 do
        if (DecEXOR[m1]>=10) then
            l1< − 0;
            codigo24[m1]< − ((2*sumbinCif[m1][l1])+(4*sumbinCif[m1][l1])
                                +(2*sumbinCif[m1][li+2])+sumbinCif[m1][l1+3]);
            Aikenaux[Ai]< −codigo21[m1];
            Aikenaux[vA]< − escribe(“A”);
        else
            codigo24[m1]< − DecEXOR[m1];
            Aikenaux[Ai]< − codigo24[m1];
            Aikenaux[vA]< − escribe(“*”);
        end if
```

```
        vA< −vA+2;
        Ai< −Ai+2;
    end for

    for m1< −0 to m1<13 do
        if (DecEXOROCR[m1]>=10) then
            l1< − 0;
            codigo24OCR[m1]< − ((2*sbinCifOCR[m1][l1])+(4*sbinCifOCR[m1][l1])
                                +(2*sbinCifOCR[m1][l1+2])+sbinCifOCR[m1][l1+3]);
            AikenauxOCR[AiOCR]< −codigo24OCR[m1];
            AikenauxOCR[vAOCR]< − escribe("A");
        else
            codigo24OCR[m1]< − DecEXOROCR[m1];
            AikenauxOCR[AiOCR]< − codigo24OCR[m1];
            AikenauxOCR[vAOCR]< −escribe("*");
        end if
        vAOCR< −vAOCR+2;
        AiOCR< −AiOCR+2;
    end for

    for m1< −0 to m1<13 do
        if (DecEXOTel[m1]>=10) then
            l1< − 0;
            codigo24Tel[m1]< − ((2*sbinCifTel[m1][l1])+(4*sbinCifTel[m1][l1])
                                +(2*sbinCifTel[m1][l1+2])+sbinCifTel[m1][l1+3]);
            AikenauxTel[AiTel]< −codigo24Tel[m1];
            AikenauxTel[vATel]< − escribe("A")
        else
            codigo24Tel[m1]< − DecEXORTel[m1];
            AikenauxTel[AiTel]< − codigo24Tel[m1];
            AikenauxTel[vATel]< − escribe("*");
        end if
        vATel< −vATel+2;
        AiTel< −AiTel+2;
    end for

create archive (VFalI.txt)
    for i < − 26 to i< 32 do
        if (AuxCifrado1OCR[i]<=9) then
            escribe(AuxCifrado1OCR[i]);
        else if (AuxCifrado1OCR[i]== 10) then
            escribe("a");
        else if (AuxCifrado1OCR[i]== 11) then
```

```
            escribe("b");
        else if (AuxCifrado1OCR[i]== 12) then
            escribe("c");
        else if (AuxCifrado1OCR[i]== 13) then
            escribe("d");
        else if (AuxCifrado1OCR[i]== 14) then
            escribe("e");
        else if (AuxCifrado1OCR[i]== 15) then
            escribe("f");
        end if
    end for
close archive

create archive (VFalT.txt)
    for i < − 10 to i< 32 do
        if (AuxCifrado1OCR[i]<=9) then
            escribe(AuxCifrado1Tel[i]);
        else if (AuxCifrado1Tel[i]== 10) then
            escribe("a");
        else if (AuxCifrado1Tel[i]== 11) then
            escribe("b");
        else if (AuxCifrado1Tel[i]== 12) then
            escribe("c");
        else if (AuxCifrado1Tel[i]== 13) then
            escribe("d");
        else if (AuxCifrado1Tel[i]== 14) then
            escribe("e");
        else if (AuxCifrado1Tel[i]== 15) then
            escribe("f");
        end if
    end for
close archive

create archive(V1_t.txt)
    for (v1 < − 0; v1 < 32; v1< − v1+2) do
        if (AuxCifrado1[v1]<= 9) then
            escribe(AuxCifrado1[v1]);
        else if (AuxCifrado1[v1]==10) then
            escribe("a");
        else if (AuxCifrado1[v1]==11) then
            escribe("b");
        else if (AuxCifrado1[v1]==12) then
            escribe("c");
```

```
        else if (AuxCifrado1[v1]==13) then
            escribe("d");
        else if (AuxCifrado1[v1]==14) then
            escribe("e");
        else if (AuxCifrado1[v1]==15) then
            escribe("f");
        end if
    end for
close archive

create archive(V1_I.txt)
    for (v1 < − 0; v1 < 26; v1< − v1+2) do
        if (AuxCifrado1OCR[v1]<= 9) then
            escribe(AuxCifrado1OCR[v1]);
        else if (AuxCifrado1OCR[v1]==10) then
            escribe("a");
        else if (AuxCifrado1OCR[v1]==11) then
            escribe("b");
        else if (AuxCifrado1OCR[v1]==12) then
            escribe("c");
        else if (AuxCifrado1OCR[v1]==13) then
            escribe("d");
        else if (AuxCifrado1OCR[v1]==14) then
            escribe("e");
        else if (AuxCifrado1OCR[v1]==15) then
            escribe("f");
        end if
    end for
close archive

create archive(V1_TEL.txt)
    for (v1 < − 0; v1 < 20; v1< − v1+2) do
        if (AuxCifrado1Tel[v1]<= 9) then
            imprime(AuxCifrado1Tel[v1]);
        else if (AuxCifrado1Tel[v1]==10) then
            escribe("a");
        else if (AuxCifrado1Tel[v1]==11) then
            escribe("b");;
        else if (AuxCifrado1Tel[v1]==12) then
            escribe("c");
        else if (AuxCifrado1Tel[v1]==13) then
            escribe("d");
        else if (AuxCifrado1Tel[v1]==14) then
```

```
            escribe("e ");
        else if (AuxCifrado1Tel[v1]==15) then
            escribe("f");
        end if
    end for
close archive

create archive(FPEauxT.txt)
    for F < − 0 to F<32 do
        if (Aikenaux[F])==65 then
            imprime("A");
        else if (Aikenaux[F]==42) then
            escribe("*");
        else
            escribe(Aikenaux[F]);
        end if
    end for
close archive

create archive(FPEauxI.txt)
    for F < − 0 to F<26 do
        if (AikenauxOCR[F])==65 then
            escribe ("A");
        else if (AikenauxOCR[F]==42) then
            escribe("*");
        else
            escribe(AikenauxOCR[F]);
        end if
    end for
close archive

create archive(FPEauxTEL.txt)
    for F < − 0 to F<20 do
        if (AikenauxTel[F])==65 then
            escribe("A");
        else if (AikenauxTel[F]==42) then
            escribe("*");
        else
            escribe(AikenauxTel[F]);
        end if
    end for
close archive
```

```
create archive(BaseDeDatosFPE.txt)
    imprime ya+1
    for m1< −0 to m1¡16 do
        escribe(codigo[m1]);
    end for
    for n1< −0 to n1< 13 do
        escribe(codigo24OCR[n1]);
    end for
    for l1< −0 to l1< 10 do
        escribe(codigo24Tel[l1]);
    end for
close archive

end for
```

Pseudocódigo de descifrado.

```
Leer(BaseDeDatosFPE), Leer(FPEauxTEL), Leer(FPEauxI), Leer(FPEauxT);
Leer(VFalI), Leer(VFalT), Leer(V1_t), Leer(V1_I), Leer(V1_TEL), Leer(key);

numTarjetaFPE< −BaseDeDatosFPE(NumTarjeta);
numOCRFPE< − BaseDeDatosFPE(NumeroOCR);
numTelFPE< − BaseDeDatos(NumTel);

estructura tarjetasD
v1T< − V1_t;
v1OCR< − V1_I;
v1Tel< − V1_Tel;
fPET< − FPEauxT
fPEOCR< − FPEauxI;
fPETel< − FPEauxTEL;
vaFalOCR< − VFalI;
vaFalTel< − VFalT;
fin estructura


for D< −0 to D<NUMERODETARJETASD do
//Quitando los valores ASCII de FPE y v1

    for AS< −0; AS< 32; AS< − AS+2 do
        DmenCifT[AS]< − tarjetasD[D].fPET[AS]-48;
        DmenCifOCR[AS]< − tarjetasD[D].fPEOCR[AS]-48;
        DmenCifT[AS2]< − tarjetasD[D].fPET[AS2];
```

```
        DmenCifOCR[AS2]< − tarjetasD[D].fPET[AS2];
        AS2< − AS2+2;
    end for
    for As< −0; AS< 20; AS< −AS+2 do
        DmenCifTel[AS]< − t¡rjetasD[D].fPETel[AS]-48;
        DmenCiTel[AS2Tel]< −tarjetasD[D].fPETel[AS2Tel];
    end for
    for  AV< −0 to AV<16 do
        if (tarjetasD[D].v1T[AV]=='a') then
            DecV1T[AV]< −10;
        else if (tarjetasD[D].v1T[AV]=='b') then
            DecV1T[AV]< −11;
        else if (tarjetasD[D].v1T[AV]=='c') then
            DecV1T[AV]< −12;
        else if (tarjetasD[D].v1T[AV]=='d') then
            DecV1T[AV]< −13;
        else if (tarjetasD[D].v1T[AV]=='e') then
            DecV1T[AV]< −14;
        else if (tarjetasD[D].v1T[AV]=='f') then
            DecV1T[AV]< −15;
        else
            DecV1T[AV]< − tarjetasD[D].v1T[AV]-48;
        end if
    end for
```

//Se realiza el mismo procedimiento para obtener las matrices DecV1OCR y DecV1Tel así como DValFalTel y DValFal.

```
//Analizando las posiciones pares de FPE
    for (I< −0; I<32; I< −I+2) do
        if (tarjetasD[D].fPET=='A') then
            if (tarjetasD[D].fPET[I1]=='0') then
                DesAikenBINT[JA][0]< −AIKEN[0][0];
                DesAikenBINT[JA][1]< −AIKEN[0][1];
                DesAikenBINT[JA][2]< −AIKEN[0][2];
                DesAikenBINT[JA][3]< −AIKEN[0][3];
            else if (tarjetasD[D].fPET[I1]=='1') then
                DesAikenBINT[JA][0]< −AIKEN[1][0];
                DesAikenBINT[JA][1]< −AIKEN[1][1];
                DesAikenBINT[JA][2]< −AIKEN[1][2];
                DesAikenBINT[JA][3]< −AIKEN[1][3];
            else if (tarjetasD[D].fPET[I1]=='2') then
                DesAikenBINT[JA][0]< −AIKEN[2][0];
```

```
            DesAikenBINT[JA][1]< −AIKEN[2][1];
            DesAikenBINT[JA][2]< −AIKEN[2][2];
            DesAikenBINT[JA][3]< −AIKEN[2][3];
        else if (tarjetasD[D].fPET[I1]=='3') then
            DesAikenBINT[JA][0]< −AIKEN[3][0];
            DesAikenBINT[JA][1]< −AIKEN[3][1];
            DesAikenBINT[JA][2]< −AIKEN[3][2];
            DesAikenBINT[JA][3]< −AIKEN[3][3];
        else if (tarjetasD[D].fPET[I1]=='4') then
            DesAikenBINT[JA][0]< −AIKEN[4][0];
            DesAikenBINT[JA][1]< −AIKEN[4][1];
            DesAikenBINT[JA][2]< −AIKEN[4][2];
            DesAikenBINT[JA][3]< −AIKEN[4][3];
        else if (tarjetasD[D].fPET[I1]=='5') then
            DesAikenBINT[JA][0]< −AIKEN[5][0];
            DesAikenBINT[JA][1]< −AIKEN[5][1];
            DesAikenBINT[JA][2]< −AIKEN[5][2];
            DesAikenBINT[JA][3]< −AIKEN[5][3];
        else if (tarjetasD[D].fPET[I1]=='6') then
            DesAikenBINT[JA][0]< −AIKEN[6][0];
            DesAikenBINT[JA][1]< −AIKEN[6][1];
            DesAikenBINT[JA][2]< −AIKEN[6][2];
            DesAikenBINT[JA][3]< −AIKEN[6][3];
        else if (tarjetasD[D].fPET[I1]=='7') then
            DesAikenBINT[JA][0]< −AIKEN[7][0];
            DesAikenBINT[JA][1]< −AIKEN[7][1];
            DesAikenBINT[JA][2]< −AIKEN[7][2];
            DesAikenBINT[JA][3]< −AIKEN[7][3];
        else if (tarjetasD[D].fPET[I1]=='8') then
            DesAikenBINT[JA][0]< −AIKEN[8][0];
            DesAikenBINT[JA][1]< −AIKEN[8][1];
            DesAikenBINT[JA][2]< −AIKEN[8][2];
            DesAikenBINT[JA][3]< −AIKEN[8][3];
        else if (tarjetasD[D].fPET[I1]=='9') then
            DesAikenBINT[JA][0]< −AIKEN[9][0];
            DesAikenBINT[JA][1]< −AIKEN[9][1];
            DesAikenBINT[JA][2]< −AIKEN[9][2];
            DesAikenBINT[JA][3]< −AIKEN[9][3];
        else
        end if
    end if
    for Ca< −0 to Ca< 4 do
        PA< −pow(2,(3-Ca));
```

```
            if (DmenCifT[I1])¿=PA then
                DesAikenBINT[JA][CA]< − 1;
                DmenCifT[I1]< − ((DmenCifT[I1])-PA);
            else
                DesAikenBINT[JA][Ca]< −0;
            end if
        end for
        I1< − I1+2;
        JA< − I1+1;
    end for
```

// Esto se repite para el OCR y Tel obteniendo las matrices DesAikenBINOCR y DesAikenBINTel.

```
    for  GH< −0 to GH<16 do
        auxDecV1T[GH]< − DecV1T[GH];
    end for
    for  GH< −0 to GH<13 do
        auxDecV1OCR[GH]< − DecV1OCR[GH];
    end for
    for  GH< −0 to GH< 10 do
        auxDecV1Tel[GH]< − DecV1Tel[GH];
    end for

    for R< −0 to R<16 do
        Po< −0;
        for Co< −0 to Co<4 do
            Po< −pow(2,(3-Co));
            if ((auxDecV1T[R])>=Po then
                DesBINV1T[R][Co]< −1;
                auxDecV1T[R]< −(auxDecV1T[R]-Po);
            else
                DesBINV1T[R][Co]< −0;
            end if
        end for
    end for

    for R< −0 to R<13 do
        Po< −0;
        for Co< −0 to Co<4 do
            Po< −pow(2,(3-Co));
            if ((auxDecV1OCR[R])>=Po then
                DesBINV1OCR[R][Co]< −1;
                auxDecV1OCR[R]< −(auxDecV1OCR[R]-Po);
```

```
            else
                DesBINV1OCR[R][Co]< −0;
            end if
        end for
    end for

    for R< −0 to R<10 do
        Po< −0;
        for Co< −0 to Co<4 do
            Po< −pow(2,(3-Co));
            if ((auxDecV1Tel[R])>=Po then
                DesBINV1Tel[R][Co]< −1;
                auxDecV1Tel[R]< −(auxDecV1Tel[R]-Po);
            else
                DesBINV1Tel[R][Co]< −0;
            end if
        end for
    end for

//Obteniendo el complemento para AES en binario

    for O< −0 to 16 do
        for P< −0 to 4 do
            ComAESBINT[O][P]< −(DesAikenBINT[O][P]
                                +DesBinV1T[O][P])mod 2;
        end for
    end for

    for O< −0 to 13 do
        for P< −0 to 4 do
            ComAESBINOCR[O][P]< −(DesAikenBINOCR[O][P]
                                +DesBinV1OCR[O][P])mod 2;
        end for
    end for

    for O< −0 to 10 do
        for P< −0 to 4 do
            ComAESBINTel[O][P]< −(DesAikenBINTel[O][P]
                                +DesBinV1Tel[O][P])mod 2;
        end for
    end for
```

```
//En decimal

    for H< −0 to H<16 do
        auxPo< −0
        for L< −0 to L<4 do
            auxPo< −auxPo+(ComAESBINT[H][L]*pow(2,3-L));
        end for
        DecComT[H]< −auxPo;
    end for

    for H< −0 to H<13 do
        auxPo< −0
        for L< −0 to L<4 do
            auxPo< −auxPo+(ComAESBINOCR[H][L]*pow(2,3-L));
        end for
        DecComOCR[H]< −auxPo;
    end for

    for H< −0 to H<10 do
        auxPo< −0
        for L< −0 to L<4 do
            auxPo< −auxPo+(ComAESBINTel[H][L]*pow(2,3-L));
        end for
        DecComTel[H]< −auxPo;
    end for

//Matriz final para la tarjeta, OCR y telefono
kl< −0;
    for (K< −1; K<32; K< −K+2) do
        ComAEST[K]< −DecComT[kl];
        kl< − kl+1;
    end for
n< −0;
    for (N< −0; N<32; N< −N+2) do
        ComAEST[N]< −DecV1T[n];
        n< − n+1;
    end for

kl< −0;
    for (K< −1; K<26; K< − K+2) do
        ComAESOCR[K]< −DecComOCR[kl];
        kl< − kl+1;
    end for
```

```
n< −0;
    for (N< −0; N<26; N< −N+2) do
        ComAESOCR[N]< −DecV1OCR[n];
        n< − n+1;
    end for
FAL< −0;
    for nF< −26 to nF<32 do
        ComAESOCR[nF]< −DValFal[FAL];
        FAL< −FAL+1;
    end for
kl< −0;
    for (K< −1; K<20; K< − K+2) do
        ComAESTel[K]< −DecComTel[kl];
        kl< − kl+1;
    end for
n< −0;
    for (N< −0; N<20; N< −N+2) do
        ComAESTel[N]< −DecV1Tel[n];
        n< − n+1;
    end for
FAL< −0;
    for nF< −20 to nF<32 do
        ComAESTel[nF]< −DValFalTel[FAL];
        FAL< −FAL+1;
    end for

    for Z< −0 to Z<32 do
        hmenDT[Z]< − ComAEST[Z];
        hmenDOCR[Z]< − ComAESOCR[Z];
        hmenDTel[Z]< − ComAESTel[Z];
    end for

//Aplicando el descifrado AES

NumTarjetaOr< − DesAES_K(hmenDT);
NumOCROr< − DesAES_K(hmenDOCR);
NumTelOR< − DesAES_K(hmenDTel);
end for
```

Bibliografía

[1] M. Brightwell, H. Smith, *Using datatype-preserving encryption to enhance data ware-house security.* 20th National Information Systems Security Conference Proceedings (NISSC), pp. 141−149, 1997.
`csrc.nist.gov/nissc/1997/proceedings/141.pdf`

[2] J. Black, Ph. Rogaway, *Ciphers with arbitrary finite domains.* RSA Data Security Conference, Cryptographer′s Track (RSA CT '02). LNCS Vol. 2271, pp. 114−130, Springer, 2002.

[3] M. Bellare, T. Ristenpart, Ph. Rogaway, T. Stegers, *Format-Preserving Encryption*, Selected Areas in Cryptography (SAC 2009), LNCS 5867, Springer, Also ePrint report 2009/251, 2009.

[4] W. Diffie, M. Hellman, *New Directions in Cryptography*, IEEE Transactions on Information Theory, Vol. IT-22, No.6, pp. 644-651, 1976.
`https://www-ee.stanford.edu/~hellman/publications/24.pdf`

[5] R. Tervo, *Secrets of the LUHN-10 Algorithm- An Error Detection Method*, Depertment of Electrical and Computer Engineering, University of New Brunswick, Fredericton, NB, Canada, 2002.
`http://www.ee.unb.ca/tervo/ee4253/luhn.shtml`

[6] ISO/IEC 7812-1:2006. Identification cards-Identification of issuers- Part 1: Numbering system.
`http://www.iso.org/iso/catalogue$\_$detail.htm?csnumber=39698`

[7] https://www.bbva.com/es/significan-los-numeros-las-tarjetas-credito-debito/

[8] R. Kammer, W. Mehuron, *Data Encryption Standard (DES)*, FIPS 46-3, 1999. `http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf`

[9] T. Spies, *Format Preserving Encryption*, A Voltage Security White Paper.
https://pdfs.semanticscholar.org/23fe/f4a9beccb9ef4f064a50eee3366246474bf7.pdf

[10] J. Daemen, V. Rijmen, *The Design of Rijndael, AES-The Advanced Encryption Standard*, Springer-Verlag Berlin Heidelberg, 2002.
https://autonome-antifa.org/IMG/pdf/Rijndael.pdf

[11] H. Feistel, *Cryptography and Computer Privacy*, Scientific American, Vol. 228, Number 5, pp. 15-23, 1973.
http://www.apprendre-en-ligne.net/crypto/bibliotheque/feistel/

[12] B. Schneier, J. Kelsey, *Unbalanced Feistel Networks and Block-Cipher Design*, Fast Software Encryption, Springer-Verlag, 1996.
www.schneier.com/cryptography/paperfiles/paper-unbalanced-feistel.pdf

[13] S. Lucks, *Faster Luby-Rackoff Ciphers.*, Proceeding of the Third International Workshop on Fast Software Encryption, pp. 189-203, Springer-Verlag, 1996.
https://link.springer.com/content/pdf/10.1007/3-540-60865-6_53.pdf

[14] E. Barker, A. Roginsky, *Recommendation for Criptographic Key Generation*, NIST Special Publication 800-133, 2012.
nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-133.pdf

[15] R. Anderson, E. Biham, *Two practical and provably secure block ciphers: Bear and LION*, Fast Software Encryption, Springer-Verlag, 1996.

[16] E. Brier, T. Peyrin, J. Stern, *BPS: a Format-Preserving Encryption Proposal*, Ingenico, 2010.
http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/bps/bps-spec.pdf

[17] T. Spies, *Feistel finite set encryption.* NIST submission, 2008.
http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ffsem/ffsem-spec.pdf

[18] Learn About electronics Digital Electronics.
http://www.learnabout-electronics.org/Digital/dig16.php

[19] M. Bellare, Ph. Rogaway, T. Spies, *The FFX Mode of Operation for Format-Preserving Encryption*, Draft 1.1, 2010.
http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ffx/ffx-spec.pdf

[20] M. Liskov, R. Rivest, D. Wagner, *Tweakable Block Ciphers*, Journal of Cryptology, 2010.
https://people.eecs.berkeley.edu/~daw/papers/tweak-crypto02.pdf

[21] Different Types of Binary Codes.
http://www.electronicshub.org/binary-codes/#8421_Code_or_BCD_Code

[22] R. Rivest, A. Shamir, L. Adleman, *A Method for Obtaining Digital Signatures and Public Key Cryptosystems*, Communications of the ACM,Vol. 21, No. 2, pp.120-16, 1978.
https://people.csail.mit.edu/rivest/Rsapaper.pdf

[23] T. ElGamal, *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*, IEEE Transactions on Information Theory, Vol. IT-31, No. 4, pp. 469-472, 1985.
http://caislab.kaist.ac.kr/lecture/2010/spring/cs548/basic/B02.pdf

[24] D. Hankerson, A. Menezes, S. Vanstone, *Guide to Elliptic Curve Cryptography*, Springer-Verlag, ISBN: 0-387-95273-X, 2004.
http://diamond.boisestate.edu/~liljanab/MATH308/GuideToECC.pdf

[25] M. Dworkin, *Recommendation for Block Cipher Modes of Operation: Methods and Techniques*, NIST Special Publication 800-38A, 2001.
nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf

[26] K. Chitra, S. Vidhya, *Efficient Fpe Algorithm For Encrypting Credit Card Numbers*, IOSR Journal of Computer Engineering (IOSR-JCE), Vol. 14, Issue 6, pp. 23-29, 2013.
iosrjournals.org/iosr-jce/papers/Vol14-issue6/D01462329.pdf?id=7545

[27] National Institute of Standards Technology, *Announcing the Advanced Encryption Standard (AES)*, NIST FIPS 197, 2001.
http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

[28] A. Rukhin, J. Soto, J. Nechvatal, A Statical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications, NIST Special Publication 800-22, Revision 1a.

[29] Voltage Security, *How to Encrypt a Credit Card Number*, A Deep Dive into the Technology That Makes it Happen, Technical brief.
https://www.voltage.com/resource/how-to-encrypt-a-credit-card-number-a-deep-dive-into-the-technology-that-makes-it-happen-%EF%BF%BC/

[30] M. Luby, C. Rackoff, *How to construct pseudorandom permutations and pseudorandom functions*, SIAM Journal on Computing, Vol. 17, Number. 2, pp. 373-203, 1988.
github.com/emintham/Papers/blob/master/Luby,Rackoff-%20How%20to%20Construct%20Pseudorandom%20Permutations%20from%20Pseudorandom%20Functions.pdf

[31] M. Bellare, J. Kilian, P. Rogaway, *The Security of the cipher Block Chaining Message Authentication Code*, Journal of Computer and System Sciences, 2000.

[32] Federal Information Processing Standards Publication, *Secure Hash Standard (SHS)*, FIPS PUB 180-4, 2015.
http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

[33] A. Menezes, P. Oorschot, S. Vanstone, *Handbook of APPLIED CRYPTOGRAPHY*, pp.223-271, 283-312, 321-376, CRC Press, 2001.
http://cacr.uwaterloo.ca/hac/

[34] J. Patarin, *Generic Attacks on Feistel Schemmes*, Extended Version, at Asiacrypt'2001.
https://eprint.iacr.org/2008/036.pdf

[35] F. García, R. Criado, M. Gonzalez, A. Perez, M. Romance, *Tokenización: Una revisión al cifrado preservando el formato para el caso de datos bancarios*, XII Reunion Española sobre Criptología y Seguridad de la información, 2012.
http://recsi2012.mondragon.edu/es/programa/recsi2012_submission_10.pdf

[36] W. Stallings, *Cryptography and Network Security*, Principles and practice, Fifth Edition, pp. 327-393, Prentice Hall, 2011.