UNIVERSIDAD AUTÓNOMA METROPOLITANA IZTAPALAPA
DIVISIÓN DE CIENCIAS BÁSICAS E INGENIERÍA

SEGURIDAD Y EFICIENCIA DE ALGUNAS VARIANTES DEL CRIPTOSISTEMA IDEA

Tesis que presenta
Pedro José Sobrevilla Moreno
Para obtener el grado de
Maestro en Ciencias
(Matemáticas Aplicadas e Industriales)

Asesor:	Dr. José Noé Gutiérrez Herrera
Jurado calificador:	
Presidente:	Dr. Guillermo Morales Luna
Secretario:	Dr. José Noé Gutiérrez Herrera
Vocal:	Dr. Miguel Alfonso Castro García

Ciudad de México, 22 de septiembre de 2016

UNIVERSIDAD AUTÓNOMA METROPOLITANA IZTAPALAPA
DIVISIÓN DE CIENCIAS BÁSICAS E INGENIERÍA

SEGURIDAD Y EFICIENCIA DE ALGUNAS VARIANTES DEL CRIPTOSISTEMA IDEA

Tesis que presenta
Pedro José Sobrevilla Moreno
Para obtener el grado de
Maestro en Ciencias
(Matemáticas Aplicadas e Industriales)

J Noé Gt 2016
Dr. José Noé Gutiérrez

Dr. Guillermo Morales Luna

Asesor: Dr. José Noé Gutiérrez Herrera

Jurado calificador:
Presidente: Dr. Guillermo Morales Luna
Secretario: Dr. José Noé Gutiérrez Herrera
Vocal: Dr. Miguel Alfonso Castro García

Ciudad de México, 22 de septiembre de 2016

Índice general

Resumen

Recordemos que el objetivo de la criptografía es el de asegurar la privacidad de la información cuando ésta se transmite a través de un canal inseguro. A lo largo de la historia se han desarrollado diferentes sistemas criptográficos tanto de llave pública, como de llave de privada. Siendo los más usados el RSA, DES, AES, PES e IDEA. Este trabajo de investigación consiste en el estudio del sistema de cifrado de datos PES, el cual fue creado como posible sustituto del DES, sin embargo al ser susceptible al criptoanálisis diferencial sus creadores, Lai y Massey [2], se vieron obligados a presentar la versión mejorada IDEA. Además se analiza la seguridad de PES e IDEA contra los ataques por criptoanálisis diferencial y lineal. Siendo el objetivo principal de la tesis hacerle modificaciones al sistema IDEA con la idea de probar si el cifrado sigue siendo seguro y si estas son convenientes. Además se reporta un intento de implementación del ataque diferencial al sistema PES.

Agradecimientos

A mis padres por todo el esfuerzo y sacrificio que hicieron por mi, así como toda la paciencia y el cariño que siempre me han tenido. Gracias también por todo lo que me enseñaron y aconsejaron, sin ustedes nada de estos sería posible. Gracias por siempre impulsarme a mejorar.

A mis hermanas, por todas sus enseñanzas, su ayuda, su cariño, por el apoyo que siempre me han dado y por su paciencia cuando más la necesitaba. Gracias por hacer de mi una mejor persona.

A mi familia por siempre estar al pendiente y ser un apoyo más.

A mis amigos que también son mis hermanos por su incansable apoyo, ayuda y compañía a pesar de las dificultades que a veces represento. Gracias en especial a los que de alguna u otra manera me ayudaron e impulsaron a concluir la maestría y sobre todo este trabajo de tesis.

A mi asesor el Dr. José Noé Gutiérrez Herrera por su apoyo, sus consejos su tiempo y su eterna paciencia durante la maestría y la realización de esta tesis.

A Josué David Torres Covarrubias por tu amistad y por donar tu tiempo y ayudarme con el código del criptoanálisis diferencial al sistema PES, sin tu aportación no hubiera sido posible.

A mis sinodales el Dr. Guillermo Morales Luna y el Dr. Miguel Alfonso Castro García por aceptar leer mi trabajo y por sus comentarios puntuales, sin los cuales esta tesis estaría incompleta.

Al Consejo Nacional de Ciencia y Tecnología (CONACyT) por su patrocinio a lo largo de mis estudios de maestría.

Introducción

Desde los inicios de la humanidad ha sido necesario comunicar información de forma privada entre los emisores y receptores interesados. La criptografía se ocupa de este problema y otros relacionados. De particular importancia resulta hoy en día el problema de cifrado de datos, para transmitir información de forma segura a través de un canal inseguro, como lo es internet. Ya en la época de los antiguos egipcios una forma de asegurar la privacidad de la información era cifrándola de alguna manera, de tal modo que la recuperación del mensaje únicamente fuera posible conociendo cierta información secreta denominada llave. La llave podía ser un número, cierta forma de reacomodar los caracteres del mensaje, una regla de sustitución de símbolos, entre otras muchas opciones de cifrado.

El sistema de cifrado *Data Encryption Standard* (DES), uno de los criptosistemas más utilizado y analizado, fue publicado el 17 de marzo de 1975, adoptado como estándar en Estados Unidos el 15 de enero de 1977, con una vida útil, originalmente, de entre 10 y 15 años. Por esta razón desde la segunda mitad de la década de 1980 aparecieron propuestas para reemplazarlo. El *International Data Encryption Algorithm* (IDEA) fue una de dichas alternativas de cifrado.

El sistema *Proposed Encryption Standard* (PES) se dio a conocer en 1990, sin embargo, en 1991 se pudo romper gracias a la aparición del criptoanálisis diferencial. Esto quiere decir que se logró recuperar un mensaje cifrado con PES sin conocer la llave utilizada. Una ligera modificación impidió que PES fuera atacable por este método de criptoanálisis. A dicha modificación se le conoce actualmente como IDEA, la cual fue publicada en 1991. Por otro lado tanto PES como IDEA, cumplen dos propiedades fundamentales para la seguridad de cualquier sistema criptográfico, la confusión y la difusión. En la primera, si hay un cambio de un solo bit de texto en claro entonces estadísticamente la mitad de los bits del texto cifrado deben cambiar y viceversa, si hay un cambio en un bit de texto cifrado, la mitad de los bits de texto en claro deben cambiar. Mientras que en la segunda se busca que cada bit del texto cifrado dependa de varios bits de llave.

En el cifrado IDEA, tanto el mensaje en claro como el texto cifrado son bloques de 64 bits, mientras que la llave (secreta) es de 128 bits. IDEA es un cifrado iterado que consta de 8 rondas, mas una transformación de salida. En el proceso se usan las siguientes tres operaciones de grupos:

1. La suma XOR de dos bloques de 16 bits, denotada $\oplus$

2. La suma de enteros módulo 2^{16}, la operación se denota como $\boxplus$

3. La multiplicación de enteros módulo $2^{16}+1$, donde el bloque compuesto sólo de ceros se trata como el entero 2^{16}. La operación se denota como $\odot$

IDEA basa sus seguridad en el uso de los grupos $(\mathbb{Z}_{2^{16}}, \oplus)$, $(\mathbb{Z}_{2^{16}}, \boxplus)$ y $(\mathbb{Z}_{2^{16}+1}, \odot)$. Dado que sus operaciones no son compatibles entre sí.

El objetivo del trabajo de tesis es presentar dos modificaciones al sistema de cifrado IDEA, así como analizar tanto las ventajas computacionales de tales modificaciones como su seguridad ante los criptoanálisis lineal y diferencial. El objetivo principal es concluir si es recomendable sustituir IDEA por alguna de las modificaciones propuestas.

En el proyecto se usan los siguiente grupos:

- $\mathbb{Z}_{2^8} \times \mathbb{Z}_{2^8}$.
- $\mathbb{Z}_{2^4} \times \mathbb{Z}_{2^4} \times \mathbb{Z}_{2^4} \times \mathbb{Z}_{2^4}$.

para sustituir el usado en la operación $\boxplus$ y

- $\mathbb{Z}_{2^9} \times \mathbb{Z}_{2^9}$.
- $\mathbb{Z}_{2^5} \times \mathbb{Z}_{2^5} \times \mathbb{Z}_{2^5} \times \mathbb{Z}_{2^5}$.

para sustituir el usado en la operación $\odot$ en IDEA.

Además se incluye un ataque computacional a 3 rondas del cifrado PES, con el cual se busca ejemplificar que aunque en la teoría el sistema de cifrado sea susceptible a cierto tipo de criptoanálisis para todas las rondas de cifrado, esto no quiere decir que en la práctica el ataque se pueda implementar completamente al sistema criptográfico.

Es importante mencionar que aunque los cifrados DES,PES e IDEA son cada vez menos utilizados, desde el punto de vista matemático sigue siendo importante estudiarlos. Ya sea como ejemplos para estudiar ataques, como el criptoanálisis diferencial, o para la creación de nuevos cifrados.

A continuación se presenta una breve exposición del contenido de cada capítulo que contiene esta tesis.

- En el primer capítulo se da una descripción del funcionamiento de los sistemas PES, IDEA e IDEA reducido. Además se da un ejemplo de mini IDEA con el cual se espera mostrar claramente cómo funcionan las rondas y el proceso de cifrado y descifrado de IDEA. Por otra parte se muestran la interacción de las operaciones tanto de IDEA como de PES, pues su incompatibilidad le da robustez a ambos sistemas. También se demuestra que en estos sistemas la operación de cifrado es esencialmente la misma que la de descifrado con excepción de las llaves utilizadas.

- En el segundo capítulo se estudian los criptoanálisis lineal y diferencial. Ambos son ataques probabilistas, el primero base su ataque en la suposición de que el atacante conoce una gran cantidad de parejas de textos en claro y cifrados. En cambio el criptoanálisis diferencial es un ataque a texto en claro elegido. Tanto IDEA como PES son resistentes al ataque lineal pero a diferencia de IDEA, PES es susceptible al criptoanálisis diferencial.

- En el tercer capítulo se presentan la implementación del ataque diferencial a miniPES y a 3 rondas de PES. Ambas implementaciones se realizaron siguiendo el ataque teórico presentado en el capítulo dos.
- En el cuarto capítulo se realizan las modificaciones a IDEA mencionadas anteriormente, se realizan dos pruebas comparando su velocidad con la de IDEA original, y por último se demuestra que ambas son resistentes al criptoanálisis diferencial.

Todos los programas utilizados durante este trabajo de tesis se encuentran en el apéndice al final del escrito.

Capítulo 1

PES e IDEA

El objetivo de cualquier sistema criptográfico es ocultar y proteger información sensible contra personas no autorizadas. Actualmente los esquemas criptográficos se dividen en cifrados de llave pública y en cifrados de llave privada, en este trabajo se estudian únicamente sistemas que pertenecen a la segunda categoría. Algunos ejemplos de cifrados de llave privada son: *Data Encryption Standard* (DES), *Advanced Encryption Standard* (AES) e *International Data Encryption Algortihtm*. En esta tesis se estudia principalmente el cifrado IDEA que fue creado con el objetivo de sustituir al sistema DES, aunque este propósito no se cumplió pues IDEA no compitió para ser el nuevo estándar.

En 1990 el cifrado *Proposed Encryption Standard*, también conocido como PES se dio a conocer por Xuejia Lai y James Massey. Pero este criptosistema resultó vulnerable al criptoanálisis diferencial, el cual será estudiado a profundidad en el Capítulo 2. Por esta razón en 1991 Lai y Massey presentaron una nueva versión de PES, que es conocida como *International Data Encryption Algorithm*. IDEA fue patentado, pero se podía acceder a él libremente para uso no comercial. En el 2012 la patente caducó y por lo tanto ahora es de uso libre. El cifrado IDEA se utiliza como un cifrado opcional en el sistema de correo electrónico seguro `openPGP`, además de en servicios de internet de banda ancha y televisión digital entre otros.

1.1. PES

El cifrado PES (Proposed Encryption Standard) fue creado para fungir como candidato para ser un nuevo estándar de cifrado que pudiera sustituir al DES. En este cifrado tanto el mensaje en claro como el texto cifrado son bloques de 64 bits, mientras que la llave (secreta) es de 128 bits. Recordemos que tanto la confusión como la difusión son de vital importancia para la seguridad de cualquier esquema de cifrado, pues oscurecen la información del mensaje original, ambos conceptos serán estudiados más adelante en este capítulo. Este cifrado logra hacer confusión usando sucesivamente tres operaciones de grupos incompatibles, aplicadas a pares de subbloques de 16 bits. La estructura del cifrado fue elegida de tal forma que logra hacer la difusión necesaria para que el sistema se considere seguro. PES está construido de tal manera que el proceso de descifrado sea el mismo que el de cifrado (una vez que las subllaves

de descifrado sean calculadas a partir de las subllaves de cifrado). PES consta de 8 rondas, más una transformación de salida (véase la Figura 1.1). En el proceso se usan las siguientes tres operaciones de grupos:

1. La suma X-OR de dos bloques de 16 bits, denotada $\oplus$
2. La suma de enteros módulo 2^{16}, la operación se denota como $\boxplus$
3. La multiplicación de enteros módulo $2^{16}+1$, donde el bloque compuesto sólo de ceros se trata como el entero 2^{16}. La operación se denota como $\odot$

Algoritmos de PES

Parte importante en sistemas de cifrado son los algoritmos de generación de subllaves, cifrado, y descifrado. A continuación se presentan los relacionados al sistema PES:

Algoritmo de generación de subllaves

Entrada: llave de 128 bits $K = k_1 \dots k_{128}$
Salida: 52 subllaves de 16 bits $k_1^{(r)}, \dots, k_6^{(r)}$ para la ronda r, $1 \leq r \leq 8$ y $k_1^{(9)}, \dots, k_4^{(9)}$ para la transformación de salida.

1. Se divide la llave K en 8 subbloques de 16 bits: asigna éstos directamente a las primeras 8 subllaves.
2. Hacer lo siguiente hasta que las 52 subllaves sean asignadas: realizar un corrimiento cíclico de K, hacia a la izquierda, por 25 bits (o lugares), partir el resultado en 8 bloques; asignar estos bloques a las siguientes 8 subllaves

Algoritmo de cifrado

Entrada: Texto en claro de 64 bits $M = m_1 \dots m_{64}$
Llave $K = k_1 \dots k_{128}$ de 128 bits
Salida: Texto cifrado de 64 bits en bloques $Y = (y_1, y_2, y_3, y_4)$

1. (Generación de subllaves) Calcular a partir de la llave K las subllaves de 16 bits $k_1^{(r)}, \dots, k_6^{(r)}$ para la ronda r, $1 \leq r \leq 8$ y $k_1^{(9)}, \dots, k_4^{(9)}$ para la transformación de salida.
2. $(X_1, X_2, X_3, X_4) \leftarrow (m_1 \dots m_{16}, m_{17} \dots m_{32}, m_{33} \dots m_{48}, m_{49} \dots m_{64})$ donde cada X_i es un subbloque de 16 bits.
3. Para la ronda, $1 \leq r \leq 8$ hacer:

   a) $X_1 \leftarrow X_1 \odot k_1^{(r)}$, $X_2 \leftarrow X_2 \odot k_2^{(r)}$, $X_3 \leftarrow X_3 \boxplus k_3^{(r)}$, $X_4 \leftarrow X_4 \boxplus k_4^{(r)}$

   b) $t_0 \leftarrow k_5^{(r)} \odot (X_1 \oplus X_3)$, $t_1 \leftarrow k_6^{(r)} \odot (t_0 \boxplus (X_2 \oplus X_4))$, $t_2 \leftarrow t_0 \boxplus t_1$

c) $a \leftarrow X_3 \oplus t_1$, $b \leftarrow X_4 \oplus t_2$

d) $X_3 \leftarrow X_1 \oplus t_1$, $X_4 \leftarrow X_2 \oplus t_2$, $X_1 \leftarrow a$, $X_2 \leftarrow b$

4. Hacer $y_1 \leftarrow X_1 \odot k_1^{(9)}$, $y_4 \leftarrow X_4 \odot k_4^{(9)}$, $y_2 \leftarrow X_3 \boxplus K_2^{(9)}$, $y_3 \leftarrow X_2 \boxplus k_3^{(9)}$

Algoritmo de descifrado

Para PES proceso de descifrado es esencialmente el mismo que el de cifrado, con la diferencia que las subllaves $k_i^{'(r)}$ son calculadas a partir de las subbllaves de cifrado $k_i^{(r)}$ como sigue:

ronda	$k_1^{'(r)}$	$k_2^{'(r)}$	$k_3^{'(r)}$	$k_4^{'(r)}$	$k_5^{'(r)}$	$k_6^{'(r)}$
$1 \leq r \leq 8$	$(k_1^{(10-r)})^{-1}$	$(k_2^{(10-r)})^{-1}$	$-k_3^{(10-r)}$	$-k_4^{(10-r)}$	$k_5^{(9-r)}$	$k_6^{(9-r)}$
$r = 9$	$(k_1^{(10-r)})^{-1}$	$(k_2^{(10-r)})^{-1}$	$-k_3^{(10-r)}$	$-k_4^{(10-r)}$	$-$	$-$

donde k^{-1} denota el inverso multiplicativo de k módulo $2^{16}+1$ y $-k$ el inverso aditivo de k módulo 2^{16}. Más adelante se da la razón de estas asignaciones.

1.2. IDEA

En el cifrado IDEA (International Data Encryption Algorithm), tanto el mensaje en claro como el texto cifrado son bloques de 64 bits, mientras que la llave (secreta) es de 128 bits. Este cifrado logra hacer confusión usando sucesivamente tres operaciones de grupos incompatibles, aplicadas a pares de subbloques de 16 bits. La estructura del cifrado fue escogida de tal manera que logra hacer la difusión necesaria.
IDEA es un cifrado iterado que consta de 8 rondas, más una transformación de salida (véase la Figura 1.2). En el proceso se usan las mismas tres operaciones de grupos que en PES.

Algoritmos de IDEA

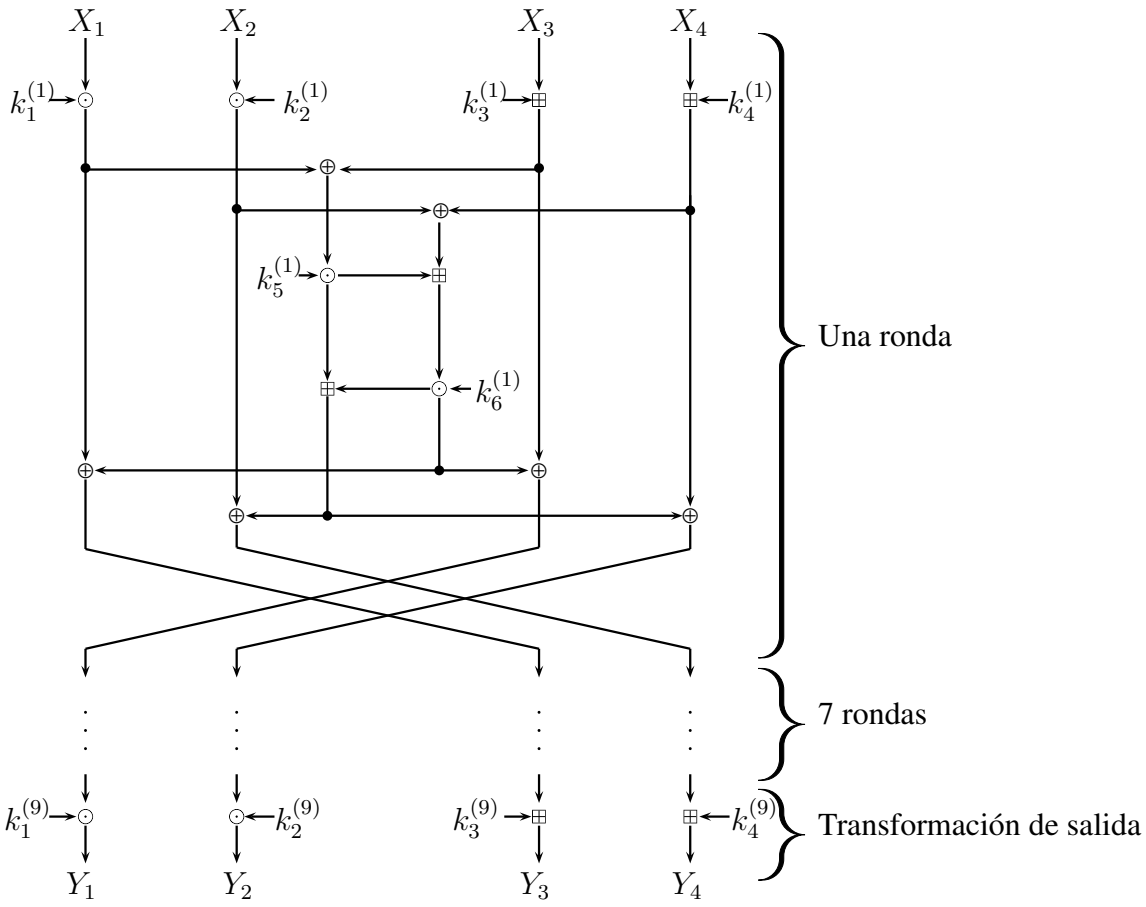
A continuación se presentan los algoritmos de generación de subllaves, cifrado y descifrado de IDEA. Cabe mencionar que el arreglo de de llaves para IDEA es idéntico al de PES.

Algoritmo de generación de subllaves

Entrada: llave de 128 bits $K = k_1 \dots k_{128}$
Salida: 52 subllaves de 16 bits $k_1^{(r)}, \dots, k_6^{(r)}$ para las rondas $1 \leq r \leq 8$ y $k_1^{(9)}, \dots, k_4^{(9)}$ para la transformación de salida.

1. Se divide K en 8 subbloques de 16 bits: asigna éstos directamente a las primeras 8 subllaves.

2. Hacer lo siguiente hasta que las 52 subllaves sean asignadas: hacer un corrimiento cíclico de K, hacia a la izquierda, por 25 bits (o lugares), partir el resultado en 8 bloques; asigna estos bloques a las siguientes 8 subllaves



X_i: Subbloque de 16-bits de texto en claro
Y_i: Subbloque de 16 bits de texto cifrado
$k_i^{(r)}$: Subbloque de llave de 16-bits
$\oplus$: Suma bit a bit módulo 2 de subbloques de 16 bits
$\boxplus$: Suma módulo 2^{16} de enteros de 16 bits
$\odot$: Producto módulo $2^{16}+1$ de enteros de 16 bits con el subbloque cero correspondiendo a 2^{16}

Figura 1.1: Proceso de cifrado del sistema PES

Algoritmo de cifrado

Entrada: Texto en claro de 64 bits $M = m_1 \dots m_{64}$
Llave $K = k_1 \dots k_{128}$ de 128 bits,
Salida: Texto cifrado de 64 bits en bloques $Y = (y_1, y_2, y_3, y_4)$

1. (Generación de subllaves) Calcular a partir de la llave K las subllaves de 16 bits $k_1^{(r)}, \dots, k_6^{(r)}$ para las rondas $1 \leq r \leq 8$ y $k_1^{(9)}, \dots, k_4^{(9)}$ para la transformación de salida.

2. $(X_1, X_2, X_3, X_4) \leftarrow (m_1 \dots m_{16}, m_{17} \dots m_{32}, m_{33} \dots m_{48}, m_{49} \dots m_{64})$ donde cada X_i es un subbloque de 16 bits.

3. Para la ronda, $1 \leq r \leq 8$ hacer:

   a) $X_1 \leftarrow X_1 \odot k_1^{(r)}$, $X_4 \leftarrow X_4 \odot k_4^{(r)}$, $X_2 \leftarrow X_2 \boxplus k_2^{(r)}$, $X_3 \leftarrow X_3 \boxplus k_3^{(r)}$

   b) $t_0 \leftarrow k_5^{(r)} \odot (X_1 \oplus X_3)$, $t_1 \leftarrow k_6^{(r)} \odot (t_0 \boxplus (X_2 \oplus X_4))$, $t_2 \leftarrow t_0 \boxplus t_1$

   c) $X_1 \leftarrow X_1 \oplus t_1$, $X_4 \leftarrow X_4 \oplus t_2$, $a \leftarrow X_2 \oplus t_2$, $X_2 \leftarrow X_3 \oplus t_1$, $X_3 \leftarrow a$

4. Hacer $y_1 \leftarrow X_1 \odot k_1^{(9)}$, $y_4 \leftarrow X_4 \odot k_4^{(9)}$, $y_2 \leftarrow X_3 \boxplus K_2^{(9)}$, $y_3 \leftarrow X_2 \boxplus k_3^{(9)}$

Algoritmo de descifrado

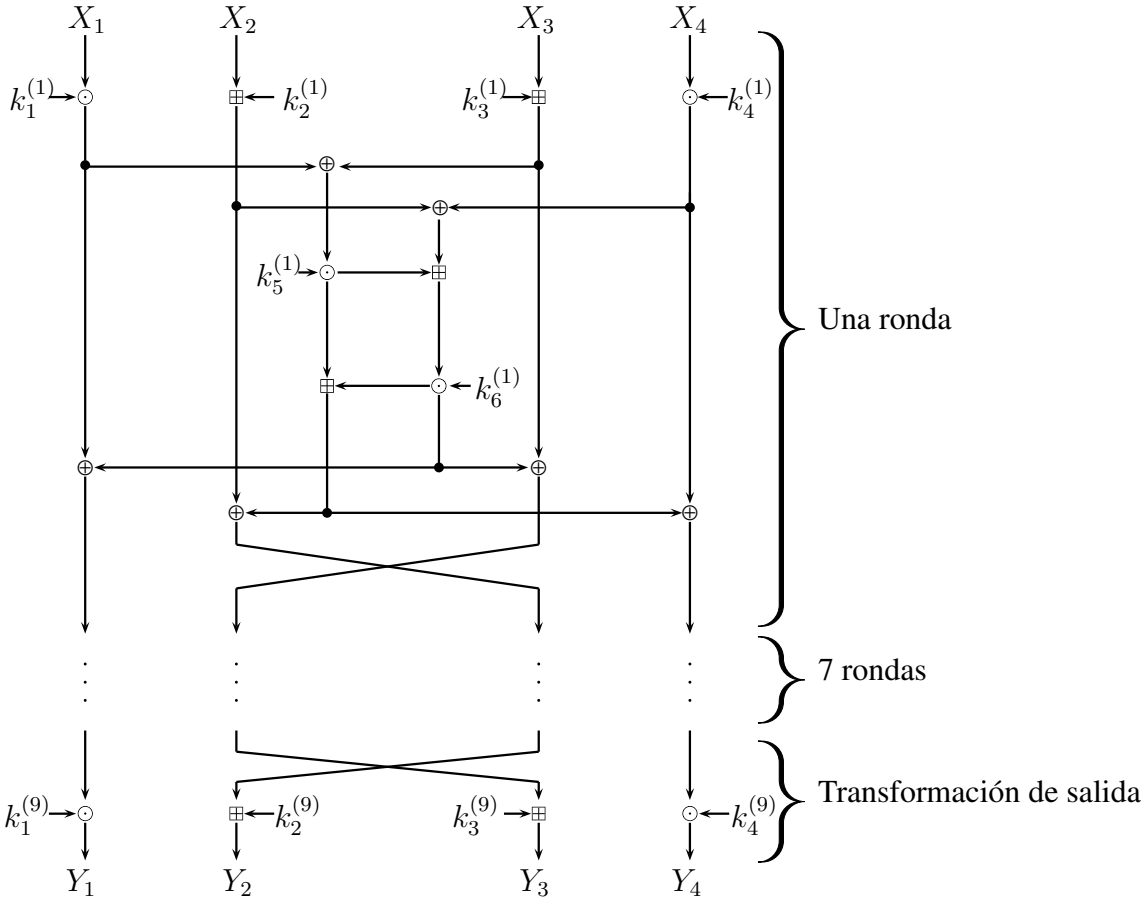
En IDEA el proceso de descifrado es esencialmente el mismo que el de cifrado, con la diferencia que las subllaves $k_i^{'(r)}$ son calculadas a partir de las subllaves de cifrado $k_i^{(r)}$ como sigue:

ronda	$k_1^{'(r)}$	$k_2^{'(r)}$	$k_3^{'(r)}$	$k_4^{'(r)}$	$k_5^{'(r)}$	$k_6^{'(r)}$
$r = 1$	$(k_1^{(10-r)})^{-1}$	$-k_2^{(10-r)}$	$-k_3^{(10-r)}$	$(k_4^{(10-r)})^{-1}$	$k_5^{(9-r)}$	$k_6^{(9-r)}$
$2 \leq r \leq 8$	$(k_1^{(10-r)})^{-1}$	$-k_3^{(10-r)}$	$-k_2^{(10-r)}$	$(k_4^{(10-r)})^{-1}$	$k_5^{(9-r)}$	$k_6^{(9-r)}$
$r = 9$	$(k_1^{(10-r)})^{-1}$	$-k_2^{(10-r)}$	$-k_3^{(10-r)}$	$(k_4^{(10-r)})^{-1}$	$-$	$-$

donde k^{-1} denota el inverso multiplicativo de k módulo $2^{16} + 1$ y $-k$ el inverso aditivo de k módulo 2^{16}.

1.3. IDEA reducido

El sistema reducido de IDEA o mini-IDEA, es una versión de IDEA en la cual se disminuye el número de rondas, en este caso se tendrán 3 además de la transformación de salida. También se reduce la longitud de los bloques de texto en claro y cifrado, de 64 a 16, lo cual deriva en algunos cambios en los grupos de la operaciones usadas en el cifrado. La utilidad de esta versión reducida es poder hacer el estudio del cifrado paso a paso, así como realizar calcular las operaciones rápidamente y con gran facilidad. En el cifrado se usan las siguientes tres operaciones de grupos:



X_i: Subbloque de 16-bits de texto en claro
Y_i: Subbloque de 16 bits de texto cifrado
$k_i^{(r)}$: Subbloque de llave de 16-bits
$\oplus$: Suma bit a bit módulo 2 de subbloques de 16 bits
$\boxplus$: Suma módulo 2^{16} de enteros de 16 bits
$\odot$: Producto módulo $2^{16}+1$ de enteros de 16 bits con el subbloque cero correspondiendo a 2^{16}

Figura 1.2: Proceso de cifrado del sistema IDEA

1. La suma X-OR de dos bloques de 4 bits, denotada $\oplus$
2. La suma de enteros módulo 2^4, la operación se denota como $\boxplus$
3. La multiplicación de enteros módulo $2^4 + 1$, donde el bloque compuesto sólo de ceros se trata como el entero 2^4. La operación se denota como $\odot$

Algoritmos de mini-IDEA

A continuación se presentan los algoritmos de generación de subllaves, cifrado y descifrado de IDEA reducido.

Algoritmo de generación de subllaves

Entrada: llave de 32 bits $K = k_1 \dots k_{32}$
Salida: 22 subllaves de 4 bits $k_1^{(r)}, \dots, k_6^{(r)}$ para las rondas $1 \leq r \leq 3$ y $k_1^{(4)}, \dots, k_4^{(4)}$ para la transformación de salida.

1. Se divide K en 8 subbloques de 4 bits: asignar éstos directamente a las primeras 8 subllaves.
2. Hacer lo siguiente hasta que las 22 subllaves sean asignadas: realizar un corrimiento cíclico de K, hacia a la izquierda, por 6 bits (o lugares), partir el resultado en 8 bloques; asignar estos bloques a las siguientes 8 subllaves

Algoritmo de cifrado

Entrada: Texto en claro de 16 bits $M = m_1 \dots m_{16}$
Llave $K = k_1 \dots k_{32}$ de 32 bits
Salida: Texto cifrado de 16 bits en bloques $Y = (y_1, y_2, y_3, y_4)$

1. (Generación de subllaves) Calcular a partir de la llave K las subllaves de 4 bits $k_1^{(r)}, \dots, k_6^{(r)}$ para las rondas $1 \leq r \leq 3$ y $k_1^{(4)}, \dots, k_4^{(4)}$ para la transformación de salida.
2. $(X_1, X_2, X_3, X_4) \leftarrow (m_1 \dots m_{16}, m_{17} \dots m_{32}, m_{33} \dots m_{48}, m_{49} \dots m_{64})$ donde cada X_i es un subbloque de 16 bits.
3. Para la ronda, $1 \leq r \leq 3$ hacer:
   a) $X_1 \leftarrow X_1 \odot k_1^{(r)}$, $X_4 \leftarrow X_4 \odot k_4^{(r)}$, $X_2 \leftarrow X_2 \boxplus k_2^{(r)}$, $X_3 \leftarrow X_3 \boxplus k_3^{(r)}$
   b) $t_0 \leftarrow k_5^{(r)} \odot (X_1 \oplus X_3)$, $t_1 \leftarrow k_6^{(r)} \odot (t_0 \boxplus (X_2 \oplus X_4))$, $t_2 \leftarrow t_0 \boxplus t_1$
   c) $X_1 \leftarrow X_1 \oplus t_1$, $X_4 \leftarrow X_4 \oplus t_2$, $a \leftarrow X_2 \oplus t_2$, $X_2 \leftarrow X_3 \oplus t_1$, $X_3 \leftarrow a$
4. Hacer $y_1 \leftarrow X_1 \odot k_1^{(4)}$, $y_4 \leftarrow X_4 \odot k_4^{(4)}$, $y_2 \leftarrow X_3 \boxplus K_2^{(4)}$, $y_3 \leftarrow X_2 \boxplus k_3^{(4)}$

Algoritmo de descifrado

El proceso de descifrado es esencialmente el mismo que el de cifrado, con la diferencia que las subllaves $k_i^{'(r)}$ son calculadas a partir de las subllaves de cifrado $k_i^{(r)}$ como sigue:

ronda	$k_1^{'(r)}$	$k_2^{'(r)}$	$k_3^{'(r)}$	$k_4^{'(r)}$	$k_5^{'(r)}$	$k_6^{'(r)}$
$r = 1$	$(k_1^{(5-r)})^{-1}$	$-k_2^{(5-r)}$	$-k_3^{(5-r)}$	$(k_4^{(5-r)})^{-1}$	$k_5^{(4-r)}$	$k_6^{(4-r)}$
$1 \leq r \leq 3$	$(k_1^{(5-r)})^{-1}$	$-k_3^{(5-r)}$	$-k_2^{(5-r)}$	$(k_4^{(5-r)})^{-1}$	$k_5^{(4-r)}$	$k_6^{(4-r)}$
$r = 4$	$(k_1^{(5-r)})^{-1}$	$-k_2^{(5-r)}$	$-k_3^{(5-r)}$	$(k_4^{(5-r)})^{-1}$	$-$	$-$

donde k^{-1} denota el inverso multiplicativo de k módulo $2^4 + 1$ y $-k$ el inverso aditivo de k módulo 2^4.

A continuación se presenta un ejemplo de cómo funciona mini-IDEA.

Ejemplo 1. Antes de iniciar con el proceso de cifrado se presentarán las tablas de inversos de las operaciones $\odot$ y $\boxplus$.

Entero	Inverso aditivo (mód 2^4)
0	0
1	15
2	14
3	13
4	12
5	11
6	10
7	9
8	8
9	7
10	6
11	5
12	4
13	3
14	2
15	1

Entero	Inverso multiplicativo (mód $2^4 + 1$)
$0 = 16 = -1$	$0 = 16 = -1$
1	1
2	9
3	6
4	13
5	7
6	3
7	5
8	15
9	2
10	12
11	14
12	10
13	4
14	11
15	8

Cuadro 1.1: Inversos de la suma (mód 2^4) e inversos del producto (mód $2^4 + 1$)

Ahora si nuestra llave aleatoria en bits es 1110000011010011110011110110011 0, siguiendo el algoritmo de arreglos las subllaves se obtienen como sigue:

Primero la llave original se divide en 8 subbloques de 4 bits y se asigna a las 8 primeras subllaves como sigue

Llave	Cadena de 4 bits	Cadena vista como entero
k_1^1	1110	14
k_2^1	0000	0
k_3^1	1101	13
k_4^1	0011	3
k_5^1	1100	12
k_6^1	1111	15
k_1^2	0110	6
k_2^2	0110	6

Prosiguiendo con el algoritmo la llave se recorre 6 bits hacia la izquierda y se obtiene la cadena 001101001111001111101100110111000, de nueva cuenta se divide en bloques de 4 bits y se asigna a las siguiente 8 subllaves como se muestra a continuación

Llave	Cadena de 4 bits	Cadena vista como entero
k_3^2	0011	3
k_4^2	0100	4
k_5^2	1111	15
k_6^2	0011	3
k_1^3	1101	13
k_2^3	1001	9
k_3^3	1011	11
k_4^3	1000	8

Se repite el proceso con la nueva cadena 0011110011110110011011100000 1101 con la cual se obtienen las últimas 8 subllaves

Llave	Cadena de 4 bits	Cadena vista como entero
k_5^3	0011	3
k_6^3	1100	12
k_1^4	1111	15
k_2^4	0110	6
k_3^4	0110	6
k_4^4	1110	14

Finalmente el arreglo de llaves de cifrado es el siguiente

Ronda/llave	k_1	k_2	k_3	k_4	k_5	k_6
1	14	0	13	3	12	15
2	6	6	3	4	15	3
3	13	9	11	8	3	12
4	15	6	6	14	–	–

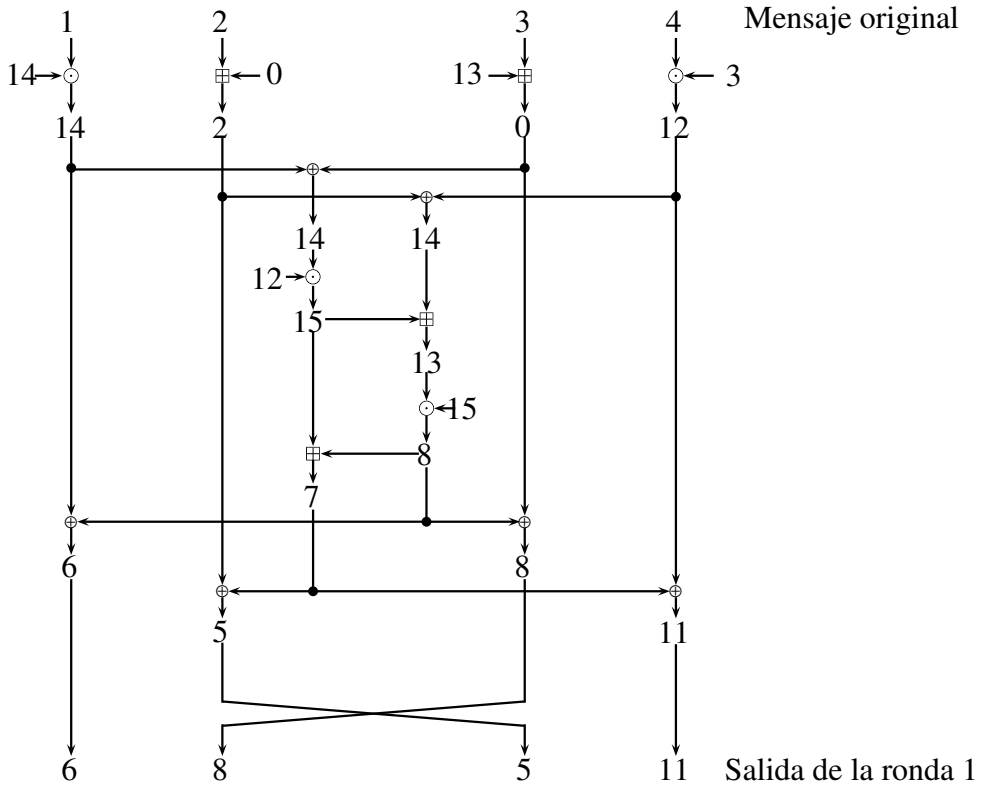
Cuadro 1.2: Llaves de cifrado de mini-IDEA

El arreglo de llaves de descifrado se muestra a continuación

Ronda/llave	k_1'	k_2'	k_3'	k_4'	k_5'	k_6'
1	8	10	10	11	3	12
2	4	5	7	15	15	3
3	3	13	10	13	12	15
4	11	0	3	6	–	–

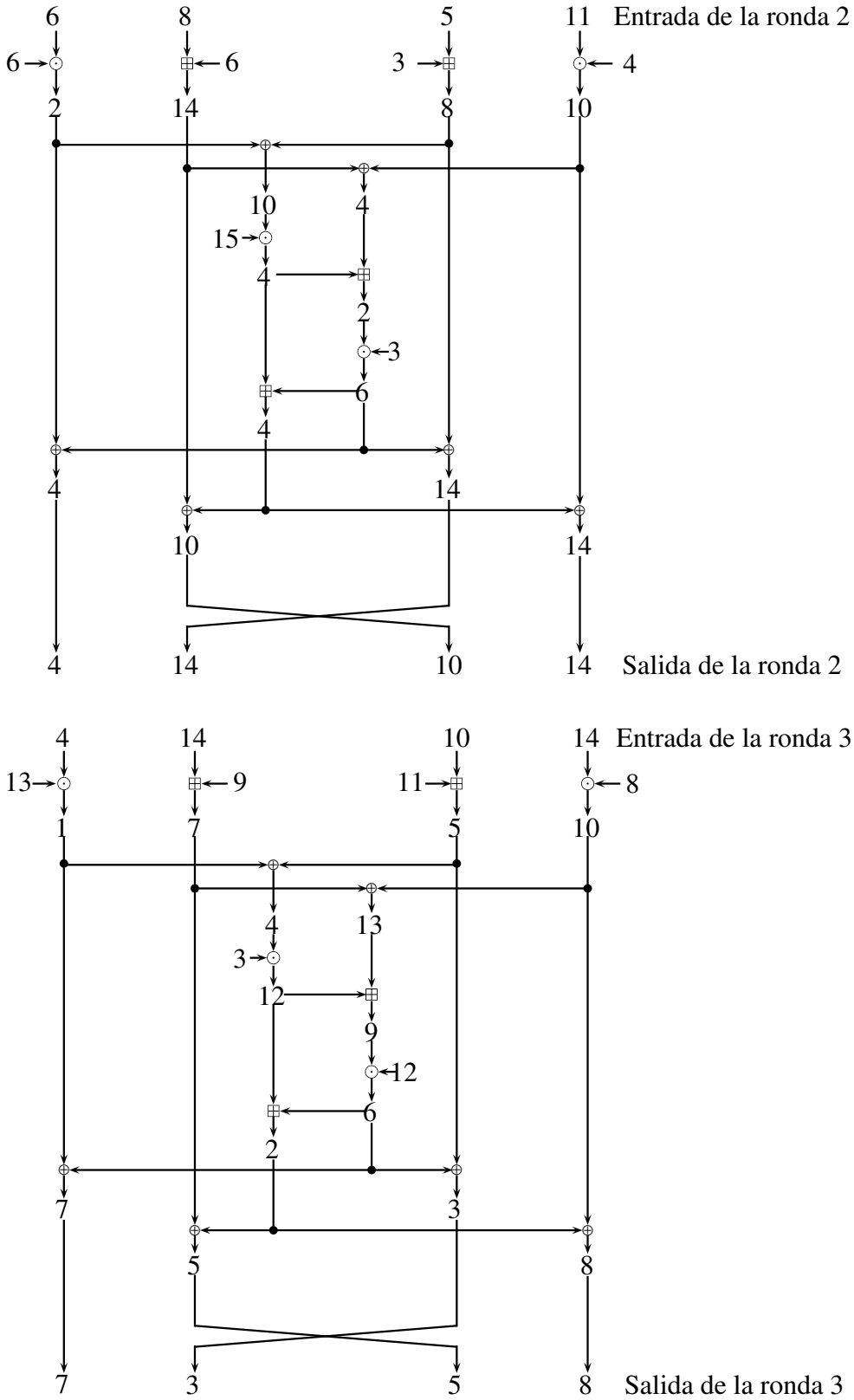
Cuadro 1.3: Llaves de descifrado de mini-IDEA

Una vez que se tienen las llaves se realiza el cifrado del mensaje $(1, 2, 3, 4)$. A continuación se muestra la primera ronda,



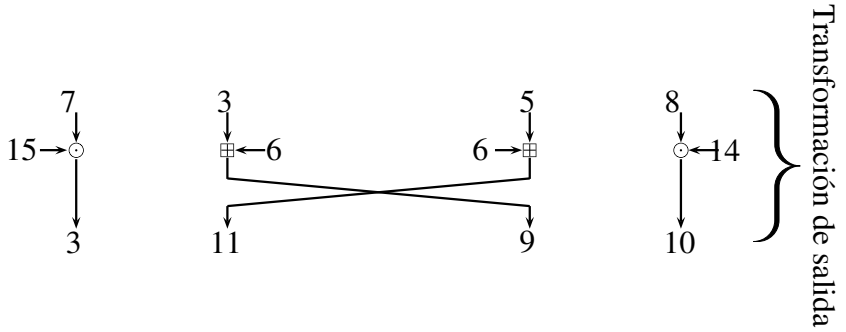
La siguiente imagen es el esquema de la segunda ronda

En seguida se tiene el esquema de la ronda tres

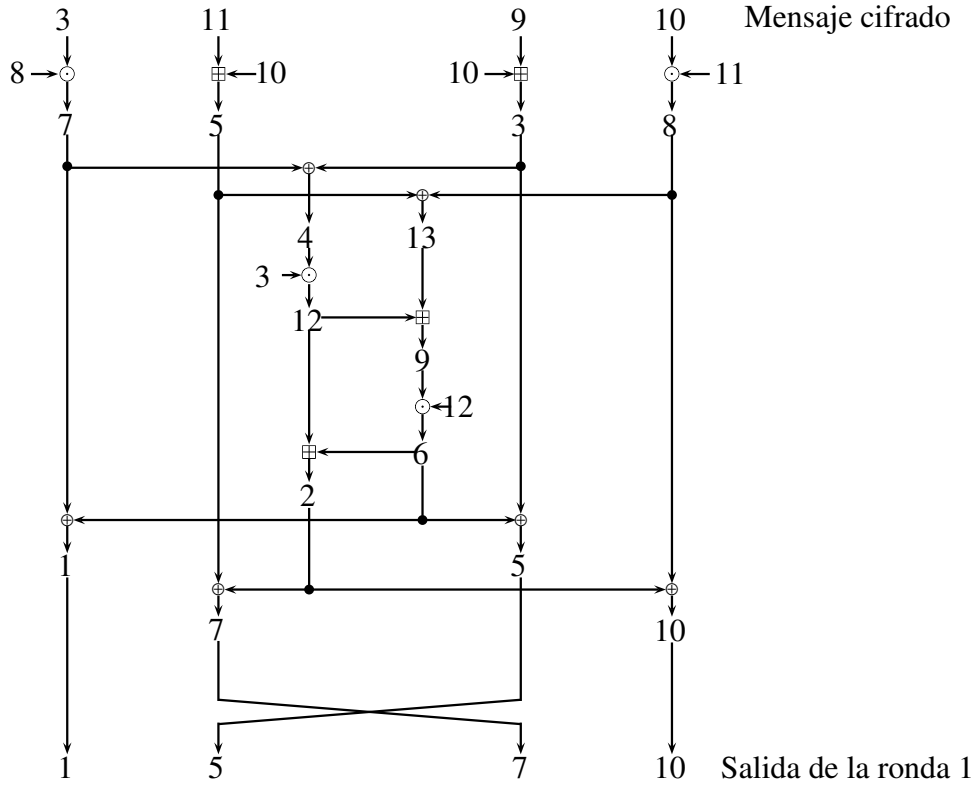


Finalmente se tiene la transformación de salida, de la cual se obtiene el mensaje cifrado

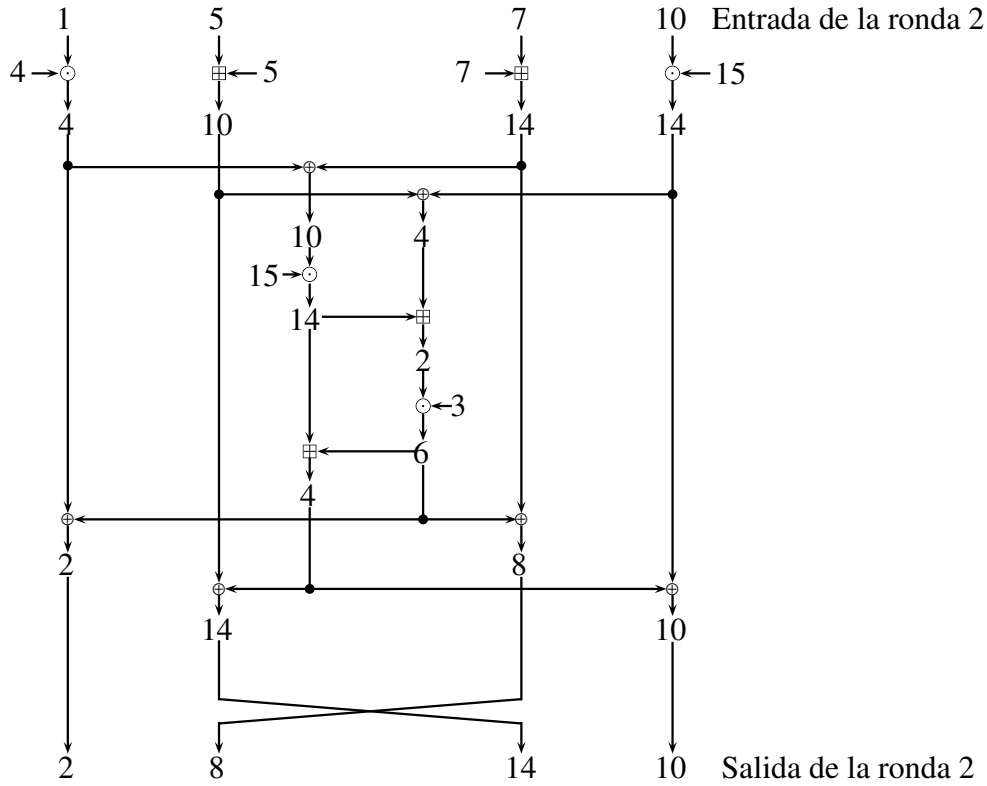
$(3, 11, 9, 10)$



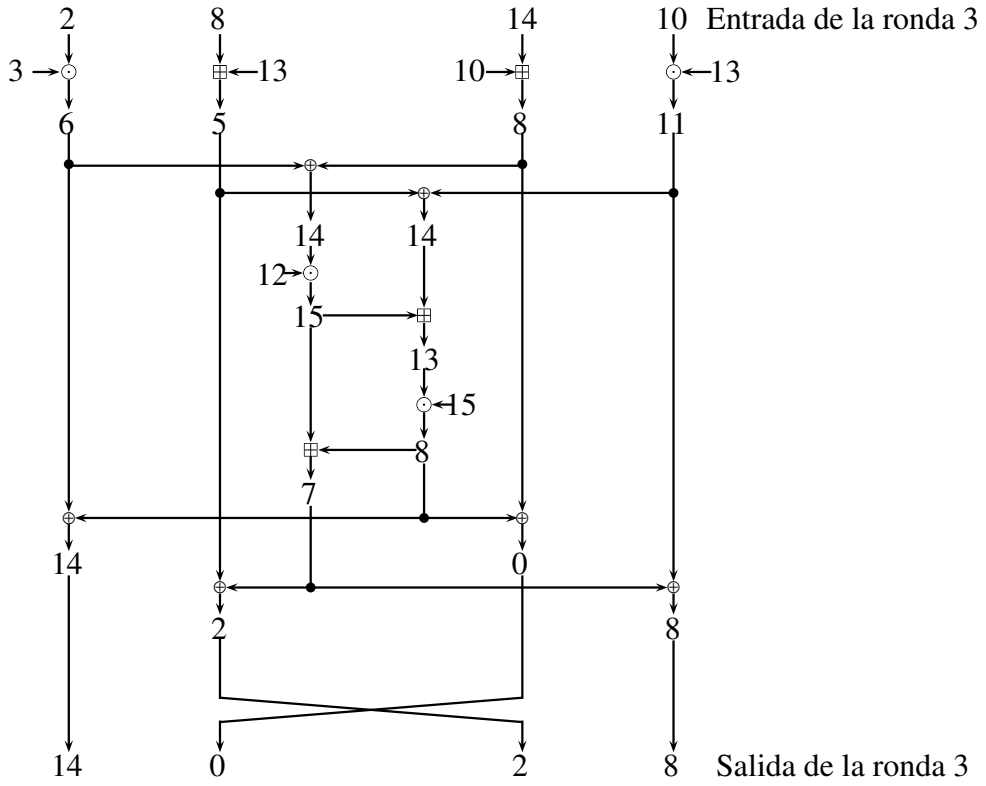
Ahora el descifrado es esencialmente el mismo proceso que el cifrado sólo esta vez se deben usar las llaves del descifrado. En la siguiente imagen se muestra la primera ronda

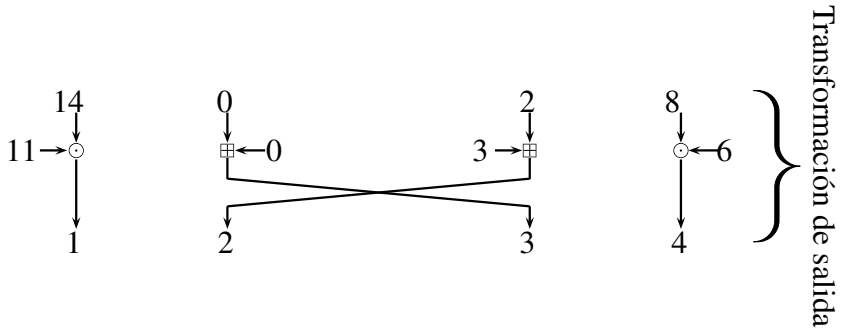


La ronda dos se muestra a continuación



Por último se presentan la tercera ronda y la transformación de salida





De esta forma el se ha recuperado el mensaje original $(1, 2, 3, 4)$ del mensaje cifrado $(3, 11, 9, 10)$.

En el apéndice A.4 se encuentran las funciones que se usan para correr el programa del sistema IDEA reducido.

1.4. Interacción de las operaciones en IDEA y PES

Prueba de isotopismo

Los cifrados IDEA y PES están basados en el concepto de mezclar operaciones de distintos grupos algebraicos con el mismo número de elementos. Se escogieron operaciones de grupos pues la relación estadística de cualesquiera tres variables aleatorias U, V, W relacionadas por una operación de grupo como $W = U * V$ tiene la propiedad del secreto perfecto, tal como el cifrado de Vernam [9]. La interacción de los distintos grupos será considerada en términos de isotopismos de cuasigrupos y en términos de expresiones polinomiales. Se consideran los casos donde $n = 1, 2, 4, 8,$ ó 16 tal que $2^n + 1$ es un número primo, además se toman, $(\mathbb{Z}^*_{2^n+1}, \cdot)$ el grupo multiplicativo de los elementos distintos de cero del campo $\mathbb{Z}_{2^n+1}$, $(\mathbb{Z}_{2^n}, +)$ el grupo aditivo del anillo $\mathbb{Z}_{2^n}$ de enteros módulo 2^n y $(\mathbb{F}^n_2, \oplus)$. Por otro lado, definase la función directa d de $\mathbb{Z}^*_{2^n+1}$ en $\mathbb{Z}_{2^n}$ como

$$d(x) = \begin{cases} x, & \text{si } x \neq 2^n \\ 0, & \text{si } x = 2^n \end{cases}$$

A continuación se presentan las definiciones para cuasigrupo, grupo e isotopismo.

Definición 1. *Sea S un conjunto no vacío y sea $* : S \times S \to S$ una operación binaria, donde $*(a, b)$ se denota como $a * b$. Entonces $(S, *)$ es un cuasigrupo si $\forall\ a, b \in S$ las ecuaciones $a * x = b$ y $y * a = b$ tienen soluciones únicas S.*

Definición 2. *Un grupo es un cuasigrupo en que la operación es asociativa, es decir*

$$a * (b * c) = (a * b) * c$$

para toda a, b y c en S.

Definición 3. *Se dice que los cuasigrupos $(S_1, *_1)$, $(S_2, *_2)$ son isotópicos si existen funciones biyectivas $\theta, \phi, \psi : S_1 \to S_2$ tales que $\theta(x) *_2 \phi(y) = \psi(x *_1 y)\ \forall\ x, y \in S_1$.*

La tripleta (θ, ϕ, ψ) de biyecciones es llamada un isotopismo de $(S_1, *_1)$ en $(S_2, *_2)$. Se tiene que dos grupos son isomorfos si son isotópicos como cuasigrupos y el isotopismo tiene la forma (θ, θ, θ). Además dos grupos son isomorfos si y sólo si son isotópicos. El siguiente teorema establece algunas incompatibilidades entre los grupos $(\mathbb{F}_2^n, \oplus)$, $(\mathbb{Z}_{2^n}, +)$ y $(\mathbb{Z}^*_{2^n+1}, \cdot)$ cuando $n \geq 2$.

Teorema 1. *Para $n \in \{1, 2, 4, 8, 16\}$*

1. *Los cuasigrupos $(\mathbb{F}_2^n, \oplus)$, $(\mathbb{Z}_{2^n}, +)$ no son isotópicos para $n \geq 2$.*

2. *Los cuasigrupos $(\mathbb{F}_2^n, \oplus)$, $(\mathbb{Z}^*_{2^n+1}, \cdot)$ no son isotópicos para $n \geq 2$.*

3. *La tripleta (θ, ϕ, ψ) de biyecciones de $\mathbb{Z}^*_{2^n+1}$ en $\mathbb{Z}_{2^n}$ es un isotopismo si y sólo si existen c_1 y c_2 en $\mathbb{Z}_{2^n}$ y un generador α del grupo cíclico $\mathbb{Z}^*_{2^n+1}$ tales que, para todo x en $\mathbb{Z}^*_{2^n+1}$*

$$\theta(x) - c_1 = \phi(x) - c_2 = \psi(x) - (c_1 + c_2) = \log_\alpha(x), \tag{1.4.1}$$

donde $y = \log_\alpha(x)$ es el entero positivo más pequeño tal que $x \equiv \alpha^y \pmod{2^n + 1}$. Además cuando $n \geq 2$, ninguna de las tres biyecciones en un isotopismo (θ, ϕ, ψ) de $\mathbb{Z}^*_{2^n+1}$ en $\mathbb{Z}_{2^n}$ puede ser la función directa d.

Demostración. 1. Para $n \geq 2$ los grupos $(\mathbb{F}_2^n, \oplus)$ y $(\mathbb{Z}_{2^n}, +)$ no son isomorfos pues el primero no es cíclico y el segundo lo es. Por lo tanto no son isotópicos como cuasigrupos.

2. Para $n = 1, 2, 4, 8, 16$ el grupo $(\mathbb{Z}^*_{2^n+1}, \cdot)$ es cíclico y $(\mathbb{F}_2^n, \oplus)$ no lo es, entonces no son isomorfos. Luego no son isotópicos como cuasigrupos.

3. Supóngase primero que la tripleta (θ, ϕ, ψ) cumple la igualdad (3.1) para todo $x \in \mathbb{Z}^*_{2^n+1}$, entonces para toda x, y en $\mathbb{Z}^*_{2^n+1}$ se tiene que

$$\psi(x \cdot y) = \log_\alpha(x \cdot y) + c_1 + c_2 = \log_\alpha(x) + \log_\alpha(y) + c_1 + c_2 = \theta(x) + \phi(y).$$

De este modo (θ, ϕ, ψ) es un isotopismo.
Ahora si (θ, ϕ, ψ) es un isotopismo de $(\mathbb{Z}^*_{2^n+1}, \cdot)$ en $(\mathbb{Z}_{2^n}, +)$, entonces para todos x, y en $\mathbb{Z}^*_{2^n+1}$, $\theta(x) + \phi(y) = \psi(x \cdot y)$. Sean $\theta_1(x) = \theta(x) - \theta(1)$, $\phi_1(x) = \phi(x) - \phi(1)$ y $\psi_1(x) = \psi(x) - \psi(1)$, luego $(\theta_1, \phi_1, \psi_1)$ también es un isotopismo de $(\mathbb{Z}^*_{2^n+1}, \cdot)$ en $(\mathbb{Z}_{2^n}, +)$. Además $\theta_1(1) = \phi_1(1) = \psi_1(1) = 0$. Ahora si en la ecuación

$$\theta_1(x) + \phi_1(y) = \psi_1(x \cdot y) \tag{1.4.2}$$

se toma el valor $x = 1$ se tiene que $\phi_1(y) = \psi_1(y)$ para toda $y \in \mathbb{Z}^*_{2^n+1}$, y tomando $y = 1$ se sigue que $\theta_1(x) = \psi_1(x)$ para toda $x \in \mathbb{Z}^*_{2^n+1}$, luego las tres funciones θ_1, ϕ_1, ψ_1 son idénticas y podemos escribir 1.4.2 como

$$\psi_1(x) + \psi_1(y) = \psi_1(x \cdot y). \tag{1.4.3}$$

Como ψ_1 es biyectiva existe $\alpha \in \mathbb{Z}^*_{2^n+1}$ tal que $\psi_1(\alpha) = 1$, entonces por 1.4.2 se tiene que $\psi_1(\alpha^k) = k$ para $k = 1, 2, \ldots, 2^n - 1$ y $\psi_1(\alpha^{2^n}) = 0$. Luego α es el generador del

grupo $(\mathbb{Z}^*_{2^n+1}, \cdot)$ el cuál es cíclico pues recordemos que estamos tomando los casos en que $2^n + 1$ es primo y también se tiene que $\psi_1(x) = \log_\alpha(x)$ para toda $x \in \mathbb{Z}^*_{2^n+1}$. Haciendo $c_1 = \theta_1(1)$ y $c_2 = \phi_1(1)$ se obtiene que

$$\theta(x) - c_1 = \phi(x) - c_2 = \psi(x) - (c_1 + c_2) = \log_\alpha(x).$$

Finalmente si alguna de las funciones ϕ, θ ó ψ fuera la función directa d, entonces existiría α en $\mathbb{Z}_{2^n+1}$ y de 1.4.1 se sigue que existe c en $\mathbb{Z}_{2^n}$ tal que

$$d(x) = \log_\alpha(x) + c \ \forall \ x \in \mathbb{Z}^*_{2^n+1} \tag{1.4.4}$$

Pero por la definición de la función directa $d(1) = 1$, entonces $1 = \log_\alpha(1) + c = c$ de esto se sigue que $d(x) = \log_\alpha(x) + 1$. Además para $n \geq 2, d(2) = 2$, luego $2 = \log_\alpha(2) + 1$ y por lo tanto $\alpha = 2$. Ahora del hecho que $d(2^n) = 0$ se sigue que $\log_2(2^n) + 1 = n + 1 = 0$ lo cual es una contradicción pues $n < 2^n - 1$ para $n \geq 2$. Por lo tanto ninguna de las funciones ϕ, θ, ψ pueden ser la función directa.

□

Continuando con el proceso de cifrado de IDEA y de PES, notamos que la multiplicación módulo $2^n + 1$ y la adición módulo 2^n están relacionadas por la función directa y su inverso. En efecto la multiplicación módulo $2^n + 1$ induce la función $g : \mathbb{Z}_{2^n} \times \mathbb{Z}_{2^n} \to \mathbb{Z}_{2^n}$ con regla de correspondencia

$$g(x, y) = d[(d^{-1}(x) \cdot d^{-1}(y)) \text{ mód } 2^n + 1] \ \forall \ x, y \in \mathbb{Z}_{2^n} \tag{1.4.5}$$

De manera similar la suma módulo 2^n induce la función $f : \mathbb{Z}^*_{2^n+1} \times \mathbb{Z}^*_{2^n+1} \to \mathbb{Z}^*_{2^{n+1}}$ definida como

$$f(x, y) = d^{-1}[(d(x) + d(y)) \text{ mód } 2^n] \ \forall \ x, y \in \mathbb{Z}^*_{2^n+1} \tag{1.4.6}$$

A continuación se demostrará la no linealidad de la función f, pues ésta es una función polinomial de grado mayor que uno.

Teorema 2. *Para toda a en $\mathbb{Z}_{2^n+1} - \{0, 2^n\}$, la función $f(a, y)$ es un polinomio en y sobre el campo $\mathbb{Z}_{2^n+1}$ de grado $2^n - 1$. De manera similar para toda a en $\mathbb{Z}_{2^n+1} - \{0, 2^n\}$, la función $f(x, a)$ es un polinomio en x sobre el campo $\mathbb{Z}_{2^n+1}$ de grado $2^n - 1$.*

Demostración. En cualquier campo finito $\mathbb{F} = GF(q)$ con q elementos y para toda α en $\mathbb{F}^*$, donde $\mathbb{F}^* = \mathbb{F} - \{0\}$ es un grupo cíclico bajo el producto, se tiene que

$$\prod_{\substack{j \neq i \\ \alpha_j \neq 0}} (x - \alpha_j)(-\alpha_i) = \begin{cases} \prod_{\substack{j \neq i \\ \alpha_j \neq 0}} (\alpha_i - \alpha_j)(-\alpha_i) = -\prod_{\alpha_i \neq 0} \alpha_i = 1 \text{ si } x = \alpha_i \\ (-\alpha_i)(\alpha_j - \alpha_1) \cdot \ldots \cdot (\alpha_j - \alpha_j) \ldots = 0 \text{ si } x \neq \alpha_i \end{cases}$$

esta igualdad se sigue del hecho que en cualquier campo finito, el producto de todos los elementos distintos de cero es igual -1. De este modo cualquier función $h(x)$ del conjunto

$GF(q) - \{0\}$ en $GF(q) - \{0\}$ se puede escribir como un polinomio sobre $GF(q)$ de grado a lo más $q - 2$ como sigue:

$$h(x) = \sum_{\alpha_i \in GF(q)-\{0\}} f(\alpha_i) \prod_{\alpha_j \neq \alpha_i, 0} (x - \alpha_j)(-\alpha_i). \tag{1.4.7}$$

Ahora encontremos una expresión polinomial para $f(a, y)$. Si $1 \leq y \leq 2^n - a$ entonces $1 \leq 1 + a \leq y + a \leq 2^n$, luego como $a \neq 0$ se tiene que

$$f(a, y) = d^{-1}[(d(a) + d(y)) \text{ mód } 2^n] = d^{-1}(a + y) = a + y. \tag{1.4.8}$$

Por otra lado si $2^n - a \leq y \leq 2^n$ se sigue que $2^n - a + 1 \leq y \leq 2^n$, luego $2^n + 1 \leq y + a \leq 2^n + a$, así $y + a = 2^n + r$ con $1 \leq r \leq a$ y $y \neq 2^n$ por lo que

$$\begin{aligned} f(a, y) &= d^{-1}[(d(a) + d(y)) \text{ mód } 2^n] \\ &= d^{-1}(a + y \text{ mód } 2^n) = d^{-1}(r) = r \\ &= r + (2^n + 1) = (r + 2^n) + 1 = a + y + 1 \end{aligned}$$

Finalmente si $y = 2^n$

$$\begin{aligned} f(a, y) &= d^{-1}[(d(a) + d(y)) \text{ mód } 2^n] \\ &= d^{-1}(a + 0 \text{ mód } 2^n) = d^{-1}(a) = a \\ &= a + (2^n + 1) = (a + 2^n) + 1 = a + y + 1 \end{aligned}$$

De este modo por los tres resultados anteriores y al ser f una función de $GF(2^n + 1) - \{0\}$ en $GF(2^n + 1) - \{0\}$ por (1.4.8) la podemos escribir como:

$$f(a, y) = \begin{cases} a + y, \text{ si } 1 \leq y \leq 2^n - a \\ a + y + 1, \text{ si } 2^n - a \leq y \leq 2^n \end{cases}$$

Ahora podemos reescribir $f(a, y)$ como se muestra enseguida

$$\begin{aligned} f(a, y) &= \textstyle\sum_{i=1}^{2^n - a} (a + i)(-i) \prod_{\substack{j \leq i \\ 1 \leq j \leq 2^n}} (y - j) + \sum_{i=2^n - a + 1}^{2^n} (a + i + 1)(-i) \prod_{\substack{j \leq i \\ 1 \leq j \leq 2^n}} (y - j) \\ &= \left[\textstyle\sum_{i=1}^{2^n} (a + i)(-i) + \sum_{i=2^n - a + 1}^{2^n} (-i)\right] \prod_{\substack{j \leq i \\ 1 \leq j \leq 2^n}} (y - j) \\ &= \left[a \textstyle\sum_{i=1}^{2^n} i + \sum_{i=1}^{2^n} -(i^2) + \sum_{i=2^n - a + 1}^{2^n} (-i)\right] \prod_{\substack{j \leq i \\ 1 \leq j \leq 2^n}} (y - j) \\ &= \tfrac{a(a+1)}{2} \prod_{\substack{j \leq i \\ 1 \leq j \leq 2^n}} (y - j) \end{aligned}$$

Esta última igualdad se da porque $\sum_{i=1}^{2^n} i = \frac{2^n(2^n+1)}{2} = 0$ mód $(2^n + 1)$, de forma similar $\sum_{i=1}^{2^n} i^2 = \frac{2^n(2^n+1)(2^{n+1}+1)}{6} = 0$ mód (2^n+1) y por último se tiene que $\sum_{i=2^n-a+1}^{2^n}(-i) = \sum_{i=1}^{a} i$. Ahora como $\frac{a(a+1)}{2} = 0$ si y sólo si $a = 0$ o $a = -1 = 2^n$, pero por hipótesis estos casos no se incluyen, por lo tanto el grado de $f(a, y)$ es $2^n - 1$. Nótese que $f(x, y) = f(y, x)$ para todas x, y en $\mathbb{Z}_{2^n+1}$, luego para toda $a \notin \{0, 2^n\}$, $f(x, a)$ también es un polinomio de grado $2^n - 1$. □

Ahora se probará que la función g no es lineal pues es una función bilineal, pero antes es necesario el siguiente lema:

Lema 1. *Si $p(x)$ es un polinomio sobre $\mathbb{Z}_{2^n}$ entonces para toda $\beta \in \mathbb{Z}_{2^n}$ se cumple que* $p(2\beta)$ mód $2 = p(0)$ mód 2.

Demostración. Si $p(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ entonces para cualquier $\beta \in \mathbb{Z}_{2^n}$, $p(2\beta) = a_n(2\beta)^n + a_{n-1}(2\beta)^{n-1} + \cdots + a_1(2\beta) + a_0$. Luego $p(2\beta)$ mód $2 = a_0 = p(0)$ mód 2. □

Teorema 3. *Si $n \in \{2, 4, 8, 16\}$, entonces, para todo a en $\mathbb{Z}_{2^n} - \{0, 1\}$, la función $g(a, x) = a \odot x = x \odot a$ no se puede escribir como un polinomio en x sobre el anillo $\mathbb{Z}_{2^n}$.*

Demostración. Sea $n > 1$, entonces para cada entero a, con $1 < a < 2^n$ existe un entero $x_0 \in \{1, 2, \ldots, 2^n\}$ tal que se cumplen las siguientes desigualdades:

$$2^n + 1 < 2ax_0 < 2(2^n + 1) \tag{1.4.9}$$

$$0 \leq 2a(x_0 - 1) \leq 2^n + 1 \tag{1.4.10}$$

Ahora la primera desigualdad se puede escribir como $0 < 2ax_0 - (2^n + 1) < 2^n + 1$, nótese que $2ax_0 - (2^n + 1)$ es impar. Entonces por la definición de g, véase (1.4.5), se tiene que

$$g(a, 2x_0) = d[(d^{-1}(a) \cdot d^{-1}(2x_0) \text{ mód } 2^n + 1]$$

De esta igualdad se sigue sucesivamente que

$$\begin{aligned} g(a, 2x_0) &= \begin{cases} d[(a \cdot 2x_0) \text{ mód } 2^n + 1] \text{ si } 2x_0 \geq 0 \\ d[(a \cdot 2^n) \text{ mód } 2^n + 1] \text{ si } 2x_0 = 0 \text{ ó } 2^n \end{cases} \\ &= \begin{cases} d(a2x_0 - (2^n + 1)) \text{ si } 2x_0 \geq 0 \\ d[(a2^n) - (2^n + 1)] \text{ si } 2x_0 = 0 \text{ ó } 2^n \end{cases} \\ &= \begin{cases} a2x_0 - (2^n + 1) \text{ si } 2x_0 \geq 0 \\ a2^n - (2^n + 1) \text{ si } 2x_0 = 0 \text{ ó } 2^n \end{cases} \\ &= 2ax_0 - (2^n + 1) \end{aligned}$$

luego

$$g(a, 2x_0) \text{ mód } 2 = 1$$

Por otro lado, por la desigualdad 1.4.10 se tiene que $2a(x_0 - 1)$ es un entero par, de esto se sigue que

$$g(a, 2(x_0 - 1)) = d[(d^{-1}(a) \cdot d^{-1}(2(x_0 - 1)) \text{ mód } 2^n + 1]$$

De esta forma se obtienen las siguientes igualdades

$$g(a, 2(x_0 - 1)) = \begin{cases} d[(a \cdot 2(x_0 - 1)) \text{ mód } 2^n + 1] \text{ si } x_0 \neq 1 \\ d[(a \cdot 2^n) \text{ mód } 2^n + 1] \text{ si } x_0 = 1 \end{cases}$$

$$= \begin{cases} d(a2(x_0 - 1) \text{ si } x_0 \neq 1 \\ d[(a2^n) - (k(2^n + 1))] \text{ si } x_0 = 1 \end{cases}$$

$$= \begin{cases} a2(x_0 - 1) \text{ si } x_0 \neq 1 \\ (a - k)2^n - k \text{ si } x_0 = 1 \end{cases}$$

por lo tanto

$$g(a, 2(x_0 - 1)) \text{ mód } 2 = \begin{cases} a2(x_0 - 1) \text{ mód } 2 = 0, \text{ si } x_0 \neq 1 \\ (a - k)2^n - k \text{ mód } 2 = 0, \text{ si } -k \text{ es par} \\ (a - k)2^n - k \text{ mód } 2 = 1, \text{ si } -k \text{ es impar} \end{cases}$$

Luego por el lema anterior se tiene que $g(a, x_0) = a \odot x$ no es un polinomio sobre $\mathbb{Z}_{2^n}$. □

1.5. Características de seguridad de IDEA y PES

Antes de enunciar las características de seguridad de PES e IDEA recordemos dos conceptos importantes para lograr un cifrado robusto: difusión y confusión.

Motivación 1. *La confusión es una operación de encripción donde la relación entre la llave y el texto cifrado es oscurecida.*

Motivación 2. *La difusión es una operación del cifrado donde la influencia de un símbolo del texto en claro abarca varios símbolos del texto cifrado con el objetivo de esconder propiedades estadísticas del texto en claro.*

Para explicar la confusión se requiere de la siguiente definición

Definición 4. *Diremos que las operaciones $\boxdot$ y $\circledast$ son asociativas si satisfacen que*

$$a \boxdot (b \circledast c) = (a \boxdot b) \circledast (a \boxdot c)$$

para cualesquiera a, b, c para las que tenga sentido ambos miembros de la igualdad.

Confusión

La confusión requerida para un cifrado seguro se logra en PES e IDEA mezclando 3 operaciones incompatibles. Además dichas operaciones están ordenadas de tal manera que la salida de un tipo de operación nunca sea la entrada de una operación del mismo tipo. Las tres operaciones son incompatibles por las siguientes razones:

1. En pares las tres operaciones no cumplen la propiedad distributiva

a) Tomemos las operaciones $\boxplus$ y $\odot$ entonces existen $a, b, c \in \{0,1\}^{16}$ de manera que $a \boxplus (b \odot c) \neq (a \boxplus b) \odot (a \boxplus c)$. Por ejemplo los valores $a = b = c = 1$ cumplen con lo anterior.

b) Ahora si se tienen las operaciones $\oplus$ y $\odot$ entonces existen $a, b, c \in \{0,1\}^{16}$ tal que $a \oplus (b \odot c) \neq (a \oplus b) \odot (a \oplus c)$. Por ejemplo los valores $a = 2, b = c = 1$ satisfacen la desigualdad.

c) Para $\odot, \boxplus$ si se toman los enteros $a = 4, b = 2, c = 3$ se cumple que $a \odot (b \boxplus c) \neq (a \odot b) \boxplus (a \odot c)$.

d) Con los valores $a = b = c = 1$ y las operaciones $\oplus, \boxplus$ se tiene que $a \oplus (b \boxplus c) \neq (a \boxplus b) \oplus (a \boxplus c)$

e) Si $a = 2^n, b = c = 1$ y se toman las operaciones $\odot, \boxplus$ entonces $a \odot (b \boxplus c) \neq (a \boxplus b) \odot (a \boxplus c)$

f) Finalmente si $a = 4, b = 2, c = 3$ entonces se cumple que $a \odot (b \oplus c) \neq (a \odot b) \oplus (a \odot c)$

2. En pares las tres operaciones no satisfacen la propiedad asociativa

   a) Para las operaciones $\boxplus, \oplus$ existe $a, b, c \in \{0,1\}^{16}$ tales que $a \boxplus (b \oplus c) \neq (a \boxplus b) \oplus c$, por ejemplo $a = b = c = 1$

   b) Con las operaciones $\oplus, \odot$ y los valores $a = 1, b = c = 2$ se tiene que $a \oplus (b \odot c) \neq (a \oplus b) \odot c$

   c) Por último si $c = 1$ y $b = a = 1$ y las operaciones son $\boxplus, \odot$ entonces $a \boxplus (b \odot c) \neq (a \boxplus b) \odot c$

   Luego, no se puede sustituir el orden de las operaciones para simplificar el análisis.

3. Como se vio en el Teorema 1 las 3 operaciones están conectadas por la función directa y su inversa, lo cual inhibe isotopismos. De este modo no se puede sustituir una operación por otra mediante una función biyectiva en las entradas y en las salidas.

4. Bajo la función directa d y su inverso es posible considerar las operaciones $\odot$ y $\boxplus$ actuando en el mismo conjunto, ya sea en el anillo $\mathbb{Z}_{2^n}$ o en el campo $\mathbb{Z}^*_{2^n+1}$. Sin embargo, al hacer esto se requiere analizar algunas funciones no lineales, pues la multiplicación módulo $2^n + 1$, es una función bilineal sobre $\mathbb{Z}_{2^n+1}$ que corresponde a una función no polinomial en $\mathbb{Z}_{2^n}$. De igual manera la suma módulo 2^n corresponde a un polinomio en dos variables de grado $2^n - 1$ sobre $\mathbb{Z}_{2^n+1}$.

Difusión

La difusión en IDEA y PES está dada por la transformación llamada multiplicación-adición (MA). La MA transforma dos subbloques de 16 bits en dos subbloques de 16 bits controlados por dos subllaves (k_5, k_6). Esta estructura tiene las siguientes propiedades:

1. Para cualesquiera subbloques de llave k_5 y k_6 $MA(\cdot, \cdot, k_5, k_6)$ es una transformación invertible y para cualesquiera U_1 y U_2 $MA(U_1, U_2, \cdot, \cdot)$ también es invertible.

2. MA tiene el efecto de una «difusión completa» pues cada subbloque de salida depende de cada subbloque de entrada y además esta estructura usa el menor número de operaciones (cuatro) requeridas para lograr la difusión completa.

Para demostrar esta última afirmación podemos encontrar definiciones más precisas en la referencia [2], para que la presentación sea más corta nos basta con las que se presentan a continuación:

Definición 5. *Una operación es una función de dos variables en una variable, es decir, para un conjunto A una operación es una función $f : A \times A \to A$.*

Definición 6. *La gráfica computacional de una operación es una gráfica dirigida en la cual los vértices son operaciones, las aristas que van a un vértice son las entradas de las operaciones y las aristas salientes de un vértices son las variables de salida de la operación.*

Definición 7. *Un algoritmo para calcular una operación determina una gráfica computacional donde las variables de entrada son las entradas del algoritmo y las variables de salida son las salidas del algoritmo.*

Definición 8. *Considérese una operación que con la forma*

$$(Y_1, Y_2) = E(X_1, X_2, Z_1, Z_2)\ X_i, Y_i \in \mathbb{F}_2^n,\ Z_i \in \mathbb{F}_2^k$$

y tal que para toda elección de (Z_1, Z_2) $E(\cdot, \cdot, Z_1, Z_2)$ es invertible. A dicha función se le llama un cifrado 2-bloque.

Definición 9. *Se dice que un cifrado 2-bloque tiene difusión completa si cada una de sus dos variables de salida depende activamente de cada variable de entrada.*

Lema 2. *Si un cifrado 2-bloque de la forma anterior tiene difusión completa entonces la gráfica computacional determinada por cualquier algoritmo que calcula el cifrado tiene al menos 4 operaciones.*

Demostración. Sean $Y_1 = E_1(X_1, X_2, Z_1, Z_2)$ y $Y_2 = E_2(X_1, X_2, Z_1, Z_2)$. Como E_1 tiene difusión completa, su gráfica computacional contiene al menos 3 operaciones pues esta función tiene cuatro variables de entrada. Supóngase que E_1 contiene exactamente 3 operaciones. La invertibilidad del cifrado 2-bloque implica que $E_1 \neq E_2$ pues si fueran iguales se tendría que $(Y_1, Y_1) = E(X_1, X_2, Z_1, Z_2)$ pero $E : (\mathbb{F}_2^n)^2 \to (\mathbb{F}_2^n)^2$, $|Im(E)| = 2^n$ y $|Dom(E)| = 2^{2n}$ lo cual es una contradicción. Además la difusión completa requiere que E_2 no iguale a ninguno de los paso intermedios que aparecen en E_1 pues si E_2 fuese alguno de dichos pasos entonces se tendría que E_1 utiliza a lo más dos operaciones lo cual contradice el hecho que E_1 contiene exactamente 3 operaciones. Luego al menos una operación que no aparece en E_1 es requerida en la gráfica computacional de E_2. □

1.5.1. Similitudes entre el cifrado y el descifrado en IDEA

En el sistema IDEA el descifrado tiene esencialmente el mismo proceso que el cifrado, sólo difieren en los subllaves utilizadas. Para mostrar lo anterior se hará un análisis profundo del funcionamiento de IDEA. En lugar de estudiar las 8 rondas y la transformación de salida con las que usualmente se describe, se estudiará IDEA en 17 nuevas rondas agrupadas en pares e impares. Cada ronda tiene una entrada de 64 bits, X, que se divide en 4 subbloques X_1, X_2, X_3, X_4, además la ronda par usa las subllaves k_1, k_2, k_3, k_4, mientras que la impar utiliza las subllaves k_5, k_6.

1.5.2. Ronda impar

En esta ronda se realizan las siguientes operaciones:
$X_1 \leftarrow X_1 \odot k_1$
$X_4 \leftarrow X_4 \odot k_4$
$X_2 \leftarrow X_3 \boxplus k_2$
$X_3 \leftarrow X_2 \boxplus k_3$
Nótese que esta ronda es invertible pues para recuperar las entradas X_1 y X_4 sea realiza la operación $\odot$ con el inverso multiplicativo de k_1 y k_4 (mód $2^{16}+1$) respectivamente. Por otra parte para obtener las entradas X_2 y X_3 se suma el inverso aditivo de k_3 y k_2 respectivamente.

1.5.3. Ronda par

De nuevo se tienen entradas X_1, X_2, X_3, X_4 y además se tiene las llaves k_5 y k_6. Primero se calculan los valores Y_e y Z_e como se describe enseguida. Después se usa la función multiplicación-adición (MA) que tiene como entradas Y_e, Z_e, k_5 y k_6, con salida Y_s y Z_s. Finalmente se utilizan Y_s y Z_s para modificar las entradas de la ronda, X_1, X_2, X_3, X_4.

Esta ronda consiste de las siguientes operaciones:
$Y_e \leftarrow X_1 \oplus X_2$
$Z_e \leftarrow X_3 \oplus X_4$
Aplicar MA:
$Y_s \leftarrow ((k_5 \odot Y_e) \boxplus Z_e) \odot k_6$
$Z_s \leftarrow (k_5 \odot Y_e) \boxplus Y_s)$
Los cálculos de los nuevos X_1, X_2, X_3, X_4 son:
$X_1 \leftarrow X_1 \oplus Y_s$
$X_2 \leftarrow X_2 \oplus Y_s$
$X_3 \leftarrow X_3 \oplus Z_s$
$X_4 \leftarrow X_4 \oplus Z_s$

Nótese que esta descripción coincide con la dada en la sección 1.2 para el cifrado IDEA. Ahora supongamos que la ronda par tiene como entrada X_1, X_2, X_3, X_4 junto con las llaves k_5 y k_6 y tiene salida X_1', X_2', X_3', X_4'. Si a la ronda par se le da entrada X_1', X_2', X_3', X_4' con las llaves k_5 y k_6 la salida serán los subbloques de entrada anteriores, es decir, X_1, X_2, X_3, X_4. Nótese que $X_1' = X_1 \oplus Y_s$ y $X_2' = X_2 \oplus Y_s$. Además al inicio de la ronda se calcula $X_1 \oplus X_2$

con salida Y_e, una entrada de MA. Si se calcula $X_1^{'} \oplus X_2^{'}$ se obtiene

$$(X_1 \oplus Y_s) \oplus (X_2 \oplus Y_s) = X_1 \oplus X_2 = Y_e.$$

De este modo Y_e es la misma ya sea que tomen X_1 y X_2 como entrada o $X_1^{'}$ y $X_2^{'}$. De la misma manera se puede mostrar que $X_3 \oplus X_4 = X_3^{'} \oplus X_4^{'} = Z_e$. Por lo tanto la salida de la función multiplicación-adición es la misma ya sea que se tenga como entrada X_1, X_2, X_3, X_4 o que la entrada sea $X_1^{'}, X_2^{'}, X_3^{'}, X_4^{'}$.
Con lo anterior se demostrará que con la entrada $X_1^{'}, X_2^{'}, X_3^{'}, X_4^{'}$ la ronda par tiene como salida X_1, X_2, X_3, X_4.
Como Y_s y Z_s son las mismas independientemente de cuál sea la entrada de la ronda. La primera salida de la ronda se calcula sumando la primera entrada con Y_s. También sabemos que $X_1^{'} = X_1 \oplus Y_s$. De este modo se tiene que
primera salida = primera entrada $\oplus Y_s$
primera salida = $X_1^{'} \oplus Y_s$
primera salida = $(X_1^{'} \oplus Y_s) \oplus Y_s = X_1$

Por lo tanto con entrada la $X_1^{'}$ la salida de la ronda es X_1. De manera análoga se puede ver que con entrada $X_i^{'}$ la salida de la ronda par es X_i para $i = 2, 3, 4$. Luego la ronda par es su propia inversa, por lo que el cifrado y descifrado en IDEA son idénticos salvo por las llaves utilizadas.

Parte importante de un cifrado es la seguridad, por lo cual es necesario estudiar varias técnicas matemáticas con el objetivo de romper el criptosistema, al estudio de estas técnicas se le conoce con el nombre de criptoanálisis. Dos de estos ataques son el criptonálisis lineal y el criptoanálisis diferencial los cuales se analizan a profundidad en el siguiente capítulo.

Capítulo 2

Criptoanálisis lineal y diferencial

2.1. Criptoanálisis lineal y diferencial

Un cifrado de bloque es una función que transforma bloques de texto en claro de n-bits a bloques de texto cifrado de la misma longitud, a n se le conoce como la longitud de bloque. A continuación se presenta una definición formal,

Definición 10. *Un cifrado de bloque de n-bits es una función $E : V_n \times \mathcal{K} \to V_n$, donde V_n es el conjunto de todos los vectores de n bits, tal que para cada llave $K \in \mathcal{K}$, $E(P, K)$ es una transformación invertible de V_n a V_n, y se denota como $E_K(P)$. La transformación inversa es la función de descifrado, denotado $D_K(C)$. En esta notación $C = E_K(P)$ significa que el texto cifrado C resulta de cifrar el texto en claro P bajo K.*

Para que un criptosistema de bloque se considere seguro, es necesario, entre otras, que no sea vulnerable a ataques por criptoanálisis lineal y diferencial. Por lo cual en esta sección se estudiarán ambos ataques, para comprender su funcionamiento. Además se analizará cuál es su efecto principalmente sobre el sistema IDEA.

2.2. Criptoanálisis lineal

El criptoanálisis lineal es un ataque a texto en claro conocido, es decir que el ataque se basa en la suposición que el atacante posee una gran cantidad de mensajes de texto en claro. Este criptoanálisis consiste en encontrar relaciones lineales probabilísticas entre un subconjunto de bits de texto en claro y de los bits de la sustitución anterior a la última ronda del cifrado, con el objetivo de obtener algunos bits de la llave. Supóngase que un atacante tiene una cantidad grande de textos en claro y sus correspondientes textos cifrados, todos ellos encriptados con la misma llave desconocida K. Ahora para cada pareja de textos en claro y cifrados se descifra el texto cifrado, utilizando todas las posibles llaves candidatas, las cuales se determinan mediante el proceso descrito en los siguientes párrafos. Para cada llave candidata el atacante debe observar si la relación lineal es válida y de ser así se aumenta el contador correspondiente a la llave candidata. Al final del proceso, se espera que aquella llave

candidata que tenga la mayor frecuencia de conteo sea la llave real, es decir la llave utilizada para cifrar los mensajes originales.

Antes de estudiar el criptoanálisis lineal es necesario tener en cuenta ciertos resultados de probabilidad, los cuales se muestran a continuación.

Supóngase que $X_1, X_2, \ldots$ son variables aleatorias independientes que toman valores en el conjunto $\{0, 1\}$. Sean $p_1, p_2, \ldots$ números reales tales que $0 \leq p_i \leq 1$ para todo i y supóngase que $P[X_i = 0] = p_i$ para $i = 1, 2, \ldots$, donde $P[X = x]$ denota la probabilidad de que la variable aleatoria X tome el valor x, luego $P[X_i = 1] = 1 - p_i$ con $i = 1, 2, \ldots$.
Ahora si $i \neq j$ como X_i y X_j son independientes se tiene que:

$$P[X_i = 0, X_j = 0] = p_i p_j$$

$$P[X_i = 0, X_j = 1] = p_i(1 - p_j)$$

$$P[X_i = 1, X_j = 1] = (1 - p_i)(1 - p_j)$$

Considérese la variable aleatoria discreta $X_i \oplus X_j$, es decir $X_i + X_j \pmod 2$. Nótese que $X_i \oplus X_j$ tiene la siguiente distribución de probabilidad:

1. $P[X_i \oplus X_j = 0] = p_i p_j + (1 - p_i)(1 - p_j)$
2. $P[X_i \oplus X_j = 1] = p_i(1 - p_j) + (1 - p_i)p_j$

Esto es así porque en el primer caso $X_i \oplus X_j = 0$ si $X_i = X_j = 0$ o $X_i = X_j = 1$ y en el segundo $X_i \oplus X_j = 0$ si $X_i = 1$ y $X_j = 0$ o si $X_i = 0$ y $X_j = 1$.

Ahora el sesgo, ϵ_i, de una variable aleatoria X_i tal que $P[X_i = 0] = p_i$ está definido como $\epsilon_i = p_i - \frac{1}{2}$ por lo que $-\frac{1}{2} \leq \epsilon_i \leq \frac{1}{2}$ pues $0 \leq p_i \leq 1$. Así se tiene que:

$$P[X_i = 0] = \frac{1}{2} + \epsilon_i \text{ para } i = 1, 2, \ldots$$

$$P[X_i = 1] = \frac{1}{2} - \epsilon_i \text{ para } i = 1, 2, \ldots$$

El siguiente lema nos permite calcular el sesgo de una distribución conjunta

$$X_{i_1} \oplus X_{i_2} \oplus \cdots \oplus X_{i_k}$$

Lema 3. *Sea $\epsilon_{i_1,i_2,\ldots,i_k}$ el sesgo de la variable aleatoria $X_1 \oplus X_2 \oplus \cdots \oplus X_k$ entonces*

$$\epsilon_{i_1,i_2,\ldots,i_k} = 2^{k-1} \prod_{j=1}^{k} \epsilon_{i_j}$$

Demostración. La demostración es por inducción sobre k. Para $k = 1$ el lema es claramente válido.
Para $k = 2$ se tiene que

$$\begin{aligned} P[X_{i_1} \oplus X_{i_2} = 0] &= p_{i_1}p_{i_2} + (1-p_{i_1})(1-p_{i_2}) \\ &= (\epsilon_{i_1} + \tfrac{1}{2})(\epsilon_{i_2} + \tfrac{1}{2}) + (\tfrac{1}{2} - \epsilon_{i_1})(\tfrac{1}{2} - \epsilon_{i_2}) \\ &= \epsilon_{i_1}\epsilon_{i_2} + \tfrac{\epsilon_{i_1}}{2} + \tfrac{\epsilon_{i_2}}{2} + \tfrac{1}{4} + \tfrac{1}{4} - \tfrac{\epsilon_{i_1}}{2} - \tfrac{\epsilon_{i_2}}{2} + \epsilon_{i_1}\epsilon_{i_2} \\ &\tfrac{1}{2} + 2\epsilon_{i_1}\epsilon_{i_2} \end{aligned}$$

luego $\epsilon_{i_1,i_2} = 2\epsilon_{i_1}\epsilon_{i_2}$.
Ahora supóngase que la afirmación del lema es válida para $k = n$, así sólo falta verificarlo para $k = n + 1$.
Como $X_{i_1} \oplus X_{i_2} \oplus \cdots \oplus X_{i_{n+1}} = (X_{i_1} \oplus X_{i_2} \oplus \cdots \oplus X_{i_n}) \oplus X_{i_n}$, por hipótesis de inducción y por el caso $k = 2$ se tiene que

$$X_{i_1} \oplus X_{i_2} \oplus \cdots \oplus X_{i_{n+1}} = 2(2^{n-1}\prod_{j=1}^{n-1}\epsilon_{i_j})\epsilon_{i_n} = 2^n\prod_{j=1}^{n}\epsilon_{i_j}$$

□

Con estos antecedentes en mente se puede iniciar el estudio del criptoanálisis lineal.

Aproximación lineal de cajas-S

Considérese una caja-S, $\pi_s : \{0,1\}^m \to \{0,1\}^n$, no necesariamente una permutación o $m = n$. Sea $X = (x_1, \ldots, x_m)$ una entrada de π_s tal que X se escoge uniformemente en $\{0,1\}^m$, de este modo cada x_i define una variable aleatoria X_i con sesgo $\epsilon_i = 0$ pues $p_i = \frac{1}{2}$. Además estas m variables aleatorias son independientes.

Supongamos adicionalmente que $Y = (y_1, y_2, \ldots, y_n)$ es la salida de π_s, así cada y_i define una variable aleatoria Y_j. Estas n variables aleatorias en general no son independientes entre ellas o de las X_i's. Además se tiene las siguientes identidades

$$P[X_1 = x_1, \ldots, X_m = x_m, Y_1 = y_1, \ldots, Y_n = y_n] = 0 \text{ si } (y_1, y_2, \ldots, y_n) \neq \pi_s(x_1, \ldots, x_m)$$

$$P[X_1 = x_1, \ldots, X_m = x_m, Y_1 = y_1, \ldots, Y_n = y_n] = 2^{-m} \text{ si } (y_1, y_2, \ldots, y_n) = \pi_s(x_1, \ldots, x_m)$$

La segunda identidad se cumple porque $P[X_1 = x_1, \ldots, X_m = x_m] = 2^{-m}$ y porque $P[Y_1 = y_1, \ldots, Y_n = y_n | X_1 = x_1, \ldots, X_m = x_m] = 1$ si $(y_1, y_2, \ldots, y_n) = \pi_s(x_1, \ldots, x_m)$.
Ahora con ayuda de las identidades anteriores el cálculo de sesgo de una variable aleatoria de la forma $X_{i_1} \oplus \cdots \oplus X_{i_k} \oplus Y_{i_1} \oplus \cdots \oplus Y_{i_l}$ se hace relativamente fácil. Para mostrar cómo se hace veamos el siguiente ejemplo.

Ejemplo 1. Se tiene una caja-S definida por una permutación $\pi_s : \{0,1\}^4 \to \{0,1\}^4$. En la siguiente tabla se registran los posibles valores que toman las ocho variables aleatorias $X_1, \ldots, X_4$, $Y_1, \ldots, Y_4$:

X_1	X_2	X_3	X_4	Y_1	Y_2	Y_3	Y_4
0	0	0	0	1	1	1	0
0	0	0	1	0	1	0	0
0	0	1	0	1	1	0	1
0	0	1	1	0	0	0	1
0	1	0	0	1	0	1	0
0	1	0	1	0	1	1	1
0	1	1	0	1	0	1	1
0	1	1	1	0	0	0	0
1	0	0	0	1	0	1	1
1	0	0	1	0	0	1	0
1	0	1	0	1	1	1	0
1	0	1	1	0	1	0	0
1	1	0	0	1	1	0	1
1	1	0	1	0	0	0	1
1	1	1	0	1	0	0	0
1	1	1	1	0	1	1	1

Ahora si por ejemplo se considera $X_1 \oplus X_4 \oplus Y_2$, la probabilidad de que esta variable aleatoria tome el valor 0 se puede determinar contando el número de columnas en la tabla anterior en las cuales $X_1 \oplus X_4 \oplus Y_2 = 0$ y dividiendo después entre 2^4, que son los casos favorables. Se tiene que:

$$P[X_1 \oplus X_4 \oplus Y_2 = 0] = \frac{1}{2}$$

luego

$$P[X_1 \oplus X_4 \oplus Y_2 = 1] = \frac{1}{2}$$

por lo que el sesgo de esta variable aleatoria es 0.

Es posible registrar esta información usando la siguiente notación. Se representa cada una de las variables aleatorias relevantes de la forma

$$\left(\bigoplus_{i=1}^{4} a_i X_i\right) \oplus \left(\bigoplus_{i=1}^{4} b_i Y_i\right)$$

donde $a_i, b_i \in \{0,1\}, i = 1,2,3,4$. Luego, con el objetivo de tener una notación compacta, se toma cada uno de los vectores binarios $\bar{a} = (a_1, a_2, a_3, a_4)$ y $\bar{b} = (b_1, b_2, b_3, b_4)$ tratados como un dígito hexadecimal, llamados suma de entrada y suma de salida respectivamente. De este modo cada una de las 256 variables aleatorias está dada por un único par de números hexadecimales.
Por ejemplo considérese nuevamente la variable aleatoria $X_1 \oplus X_4 \oplus Y_2$, entonces la suma de entrada es $(1,0,0,1)$ la cual es 9 en hexadecimal. mientras que las suma de salida es $(0,1,0,0) = 4_{hex}$.

Para una variable aleatoria con suma de entrada $\bar{a}$ y suma de salida $\bar{b}$, sea $N_L(\bar{a}, \bar{b})$ el número de 8-tuplas binarias $(x_1, x_2, x_3, x_4, y_1, y_2, y_3, y_4)$ tales que $(y_1, y_2, y_3, y_4) = \pi_s(x_1, x_2, x_3, x_4)$ y

$$\left(\bigoplus_{i=1}^{4} a_i X_i\right) \oplus \left(\bigoplus_{i=1}^{4} b_i Y_i\right) = 0$$

De este modo el sesgo de la variable aleatoria con suma de entrada $\bar{a}$ y suma de salida $\bar{b}$ se calcula como $\epsilon(\bar{a}, \bar{b}) = \frac{(N_L(\bar{a},\bar{b})-8)}{16}$. Todos los posibles valores para $N_L(\bar{a}, \bar{b})$ se muestran en la siguiente tabla, llamada de aproximaciones lineales.

$\bar{b}/\bar{a}$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	16	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
1	8	8	6	6	8	8	6	14	10	10	8	8	10	10	8	8
2	8	8	6	6	8	8	6	6	8	8	10	10	8	8	2	10
3	8	8	8	8	8	8	8	8	10	2	6	6	10	10	6	6
4	8	10	8	6	6	4	6	8	8	6	8	10	10	4	10	8
5	8	6	6	8	6	8	12	10	6	8	4	10	8	6	6	8
6	8	10	6	12	10	8	8	10	8	6	10	12	6	8	8	6
7	8	6	8	10	10	4	10	8	6	8	10	8	12	10	8	10
8	8	8	8	8	8	8	8	8	6	10	10	6	10	6	6	2
9	8	8	6	6	8	8	6	6	4	8	6	10	8	12	10	6
A	8	12	6	10	4	8	10	6	10	10	8	8	10	10	8	8
B	8	12	8	4	12	8	12	8	8	8	8	8	8	8	8	8
C	8	6	12	6	6	8	10	8	10	8	10	12	8	10	8	6
D	8	10	10	8	6	12	8	10	4	6	10	8	10	8	8	10
E	8	10	10	8	6	4	8	10	6	8	8	6	4	10	6	8
F	8	6	4	6	6	8	10	8	8	6	12	6	6	8	10	8

Por ejemplo si $\bar{b} = 9$ y $\bar{a} = 4$ entonces buscando en la tabla anterior se tiene que $N_L(9, 4) = 8$. De esta manera podemos calcular el sesgo como sigue $\epsilon(9, 4) = \frac{(N_L(\bar{a},\bar{b})-8)}{16} = \frac{8-8}{16} = 0$.

Una vez que se sabe como encontrar relaciones lineales para cualquier caja-S a continuación se estudia el ataque lineal a un sistema completo de cifrado.

Un ataque lineal a un red de substitución-permutación (SPN)

La Figura 2.1 ilustra la estructura de la aproximación que se usará en cifrado basado en una red de sustitución-permutación. El diagrama se interpreta como sigue: Las flechas corresponden a las variables aleatorias que estarán involucradas en las aproximaciones lineales. Las cajas-S etiquetadas son las que se utilizan en estas aproximaciones, llamadas cajas-S activas.

Esta aproximación involucra cuatro cajas-S activas:
En S_2^1 la variable aleatoria $T_1 = U_5^1 \oplus U_7^1 \oplus U_8^1 \oplus V_6^1$ tiene sesgo $\frac{1}{4}$.
En S_2^2 la variable aleatoria $T_2 = U_6^2 \oplus V_6^2 \oplus V_8^2$ tiene sesgo $-\frac{1}{4}$.
En S_3^3 la variable aleatoria $T_3 = U_6^3 \oplus V_6^3 \oplus V_8^3$ tiene sesgo $-\frac{1}{4}$.

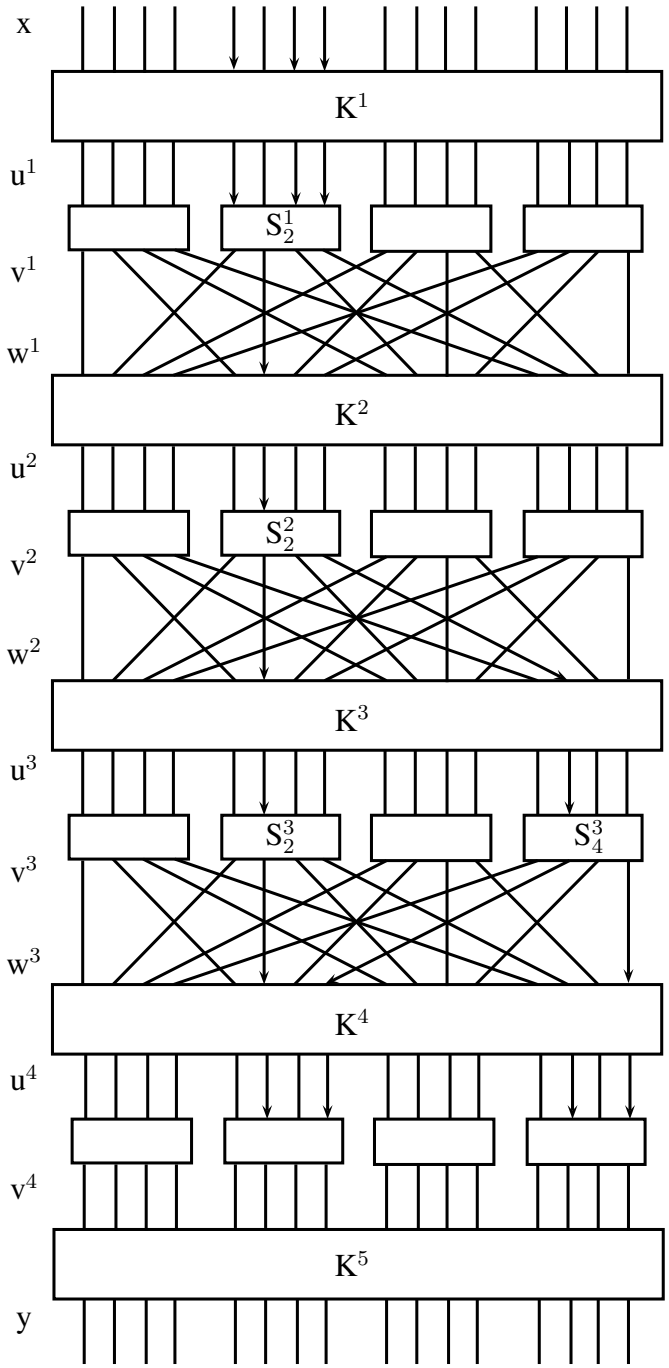


Figura 2.1: Una aproximación lineal a una red de permutación-substitución.

En S_4^3 la variable aleatoria $T_4 = U_{14}^3 \oplus V_{14}^3 \oplus V_{16}^3$ tiene sesgo $-\frac{1}{4}$.

Estas variables aleatorias tienen sesgo grande en valor absoluto. Más aún se prueba que la suma x-or de estas variables aleatorias lleva a cancelar variables aleatorias intermedias. Ahora si suponemos que estas cuatro variables aleatorias son independientes, aunque en realidad no lo sean, entonces se puede calcular el sesgo de su suma x-or. Así se tiene por hipótesis que la variable aleatoria $T_1 \oplus T_2 \oplus T_3 \oplus T_4$ tiene sesgo $2^3(\frac{1}{4})(-\frac{1}{4})^3 = -\frac{1}{32}$.
Además las variables aleatorias T_1, T_2, T_3, T_4 tienen la propiedad de que su suma x-or puede expresarse en términos de bits de texto en claro, y bits de u^4, que es la entrada de la última ronda de cajas-S. Esto se hace como sigue, usando las relaciones del diagrama.

$$T_1 = U_5^1 \oplus U_7^1 \oplus U_8^1 \oplus V_6^1 = X_5 \oplus K_5^1 \oplus X_7 \oplus K_7^1 \oplus X_8 \oplus K_8^1 \oplus V_6^1$$

$$T_2 = U_6^2 \oplus V_6^2 \oplus V_8^2 = V_6^1 \oplus K_6^2 \oplus V_6^2 \oplus V_8^2$$

$$T_3 = U_6^3 \oplus V_6^3 \oplus V_8^3 = V_6^2 \oplus K_6^3 \oplus V_6^3 \oplus V_8^3$$

$$T_4 = U_{14}^3 \oplus V_{14}^3 \oplus V_{16}^3 = V_8^2 \oplus K_{14}^3 \oplus V_{14}^3 \oplus V_{16}^3$$

Calculando $T_1 \oplus T_2 \oplus T_3 \oplus T_4$ se obtiene:

$$X_5 \oplus X_7 \oplus X_8 \oplus V_6^3 \oplus V_8^3 \oplus V_{14}^3 \oplus V_{16}^3 \oplus K_5^1 \oplus K_7^1 \oplus K_8^1 \oplus K_6^2 \oplus K_6^3 \oplus K_{14}^3$$

que tiene sesgo $-\frac{1}{32}$.

Lo siguiente es sustituir los términos V_i^3 por las siguientes expresiones:

$$V_6^3 = U_6^4 \oplus K_6^4$$

$$V_8^3 = U_{14}^4 \oplus K_{14}^4$$

$$V_{14}^3 = U_8^4 \oplus K_8^4$$

$$V_{16}^3 = U_{16}^4 \oplus K_{16}^4$$

Ahora sustituyendo en la suma $T_1 \oplus T_2 \oplus T_3 \oplus T_4$ se obtiene:

$$X_5 \oplus X_7 \oplus X_8 \oplus U_6^4 \oplus U_8^4 \oplus U_{14}^4 \oplus U_{16}^4 \oplus K_5^1 \oplus K_7^1 \oplus K_8^1 \oplus K_6^2 \oplus K_6^3 \oplus K_{14}^3 \oplus K_6^4 \oplus K_8^4 \oplus K_{14}^4 \oplus K_{16}^4$$

Esta expresión sólo tiene bits de texto en claro, bits de u^4 y bits de llave. Supongamos que los bits de llave en la expresión anterior están fijos, entonces la variable aleatoria

$$K_5^1 \oplus K_7^1 \oplus K_8^1 \oplus K_6^2 \oplus K_6^3 \oplus K_{14}^3 \oplus K_6^4 \oplus K_8^4 \oplus K_{14}^4 \oplus K_{16}^4$$

tiene el valor fijo 0 ó 1. De esto se sigue que la variable aleatoria

$$X_5 \oplus X_7 \oplus X_8 \oplus U_6^4 \oplus U_8^4 \oplus U_{14}^4 \oplus U_{16}^4 \tag{2.2.1}$$

tiene sesgo $\pm\frac{1}{32}$ donde el signo del sesgo depende de los valores de los bits de llave desconocidos.
Supongamos ahora que se tienen T parejas de textos en claro y textos cifrados, todas con la misma llave desconocida k. Denotemos este conjunto de T parejas como τ. El ataque nos permitirá obtener los ocho bits de llave $k_{<2>}^5$ y $k_{<4>}^5$, donde $k_{<j>}$ es el bloque j de 4 bits, digamos $k_5^5, k_6^5, k_7^5, k_8^5, k_{13}^5, k_{14}^5, k_{15}^5, k_{16}^5$. Estos son los 8 bits de llave que son sumados x-or con la salida de las cajas-S S_2^4 y S_4^4. Nótese que hay 256 posibilidades para esta lista de ocho bits de llave. A esta 8-tupla binaria se le conoce como *llave candidata*.
Para cada pareja $(x, y) \in \tau$ y para cada subllave candidata es posible calcular un descifrado parcial de y, obteniendo el valor resultante $u_{<2>}^4$ y $u_{<4>}^4$. Luego se calcula el valor

$$x_5 \oplus x_7 \oplus x_8 \oplus u_6^4 \oplus u_8^4 \oplus u_{14}^4 \oplus u_{16}^4 \tag{2.2.2}$$

tomados por la variable aleatoria (2.2.1). Se guarda un arreglo de contadores indexados por las 256 posibles subllaves candidatas, y se incrementa el contador que corresponde a una subllave particular cada vez que (2.2.2) tenga valor 0, el arreglo se inicializa en 0.
Al final de este proceso de conteo se espera que la mayor parte de los contadores tengan un valor cercano a $\frac{T}{2}$, pero que el contador de la subllave candidata correcta tenga un valor cercano a $\frac{T}{2} \pm \frac{T}{32}$. Esto permite identificar los 8 bits de subllave.

Criptoanálisis lineal del sistema IDEA

Hasta el momento no se ha encontrado un criptoanálisis lineal efectivo al sistema IDEA. Pues aunque se pueden encontrar relaciones lineales, estas no sirven para el análisis lineal porque toman los bits menos significativos de las cajas-S, por lo cual el sesgo de las variables aleatorias es igual a 0. En este análisis se toman los bits menos significativos pues la manera de atacar a IDEA es a través de la operación $\boxplus$, que justo en el bit de la derecha se comporta de la misma manera que la suma x-or, lo cual en teoría permitiría un ataque lineal.

En la literatura se encontraron algunos ataques de tipo lineal o a texto en claro conocido a sistemas reducidos de IDEA por ejemplo los que se presentan en [5], [6] y [7].

2.3. Criptoanálisis diferencial

El criptoanálisis diferencial es un ataque con texto en claro elegido. Se supone que X, X^* son elementos de texto en claro que se encriptan usando la misma llave desconocida K, con textos cifrados resultantes Y, Y^* respectivamente y que el atacante tiene una gran cantidad

de cuartetas (X, X^*, Y, Y^*), donde $X' = X \otimes X^{*^{-1}}$ es fijo y $\otimes$ denota una operación de grupo especifica en el conjunto de textos en claro. Ahora para cada una de las cuartetas se empieza descifrando los textos cifrados Y, Y^*, usando todas las posibles llaves candidatas para la última ronda del cifrado. Para cada llave candidata, se calculan los valores de ciertos estados de bit y se determina si su diferencia $\otimes$ tiene un cierto valor, digamos el valor más probable para la entrada $\otimes$ dada. Siempre que esto pase, se incrementa el contador correspondiente de la llave candidata particular. Al final del proceso, se espera que la llave candidata con la mayor frecuencia de conteo tenga los valores correctos para los bits de llave.

Sea $Y = f(X, K)$ una función ronda, en la que para cada subllave de ronda K, $f(-, K)$ establece una correspondencia inyectiva entre la entrada de la ronda, X, y la salida de la ronda, Y. En términos matemáticos $f : \mathcal{X} \times \mathcal{K} \to \mathcal{Y}$ tal que $(X, K) \mapsto f(X, K) = Y$ es una función ronda, donde $\mathcal{X}$ es el conjunto las entradas de una ronda, $\mathcal{K}$ es el conjunto de las subllaves de una ronda y $\mathcal{Y}$ es el conjunto de salidas para una ronda. Considérese un par ordenado de cadenas de bits de longitud m, digamos (X, X^*) entonces se define la diferencia entre dos textos en claro (o dos textos cifrados) de la siguiente manera $X' = X \otimes X^{*^{-1}}$, donde $\otimes$ denota la operación de grupo que actúa sobre los textos en claro y $X^{*^{-1}}$ es el inverso de X^* en el grupo. Además para cualquier $X' \in \{0,1\}^m$, se define $\Delta(X')$ como el conjunto que contiene a todas la parejas ordenadas (X, X^*) que tienen diferencia $\otimes$ igual a X'. Nótese que para cualquier conjunto $\Delta(X')$ contiene 2^m parejas y que

$$\Delta(X') = \{(X, X^*) | X \in \{0,1\}^m, X^* = X \otimes (X')^{-1}\}.$$

Por otra parte se dice que la función ronda $Y = f(X, K)$ es *criptográficamente débil* si dadas unas cuantas tripletas $(X \otimes X^{*^{-1}}, Y, Y')$ es posible (en la mayoría de los casos) determinar la subllave K.

De la pareja de encripciones se obtiene la serie de diferencias

$$Y(0) \otimes Y(0)^{*^{-1}}, Y(1) \otimes Y(1)^{*^{-1}}, \ldots, Y(r) \otimes Y(r)^{*^{-1}} \tag{2.3.1}$$

donde $Y(0) = X$ y $Y(0)^* = X^*$ denota la pareja de textos en claro. De este modo se tiene la igualdad $Y(0) \otimes Y(0)^{*^{-1}} = X \otimes X^{*^{-1}}$ y donde $Y(i)$ y $Y(i)^*$ para $0 < i \leq r$ son las salidas de la i-ésima ronda, las cuales son las entradas de la ronda $i + 1$.

Ahora en el criptoanálisis diferencial se logra recuperar la subllave K de la última ronda escogiendo parejas de texto en claro (X, X^*) con determinada diferencia α tal que la diferencia $Y(r-1) \otimes Y(r-1)^{*^{-1}}$ de la pareja de entradas de la última ronda tome un valor particular β con probabilidad alta. Esto lleva a la siguiente definición.

Definición 11. *La diferencial de la ronda i es una pareja (α, β), donde α es la diferencia de un par de textos en claro X y X' y donde β es la posible diferencia para las salidas, Y y Y' de la ronda i. La probabilidad de la diferencial de la ronda i, (α, β) es la probabilidad condicional de que β sea la diferencial $Y(i) \otimes Y(i)^{*^{-1}}$ de la pareja de textos cifrados después de i rondas dado que la pareja (X, X^*) de textos en claro tiene diferencial $X \otimes X^{*^{-1}} = \alpha$, cuando el texto en claro X y las subllaves $k^{(1)}, \ldots, k^{(i)}$ son independientes y uniformemente distribuidas, es decir $P((Y(i), Y(i)^*) \in \Delta(\beta) | (X, X^*) \in \Delta(\alpha))$.*

A continuación se describe el procedimiento básico del criptoanálisis diferencial:

1. Encuentre una diferencial de la ronda $(r-1)$, (α, β) tal que
$$P((Y(r-1), Y(r-1)^*) \in \Delta(\beta) | (X, X^*) \in \Delta(\alpha))$$
sea máxima o casi máxima.

2. Escoja uniformemente un texto en claro X y calcule X^* tal que la diferencia entre ellos sea α. Solicitar el cifrado de X y X^* con la llave actual K. De los textos cifrados $Y(r)$ y $Y(r)^*$, encontrar cada posible valor, si los hay, de la subllave $k^{(r)}$ de la última ronda, correspondiente a la diferencia anticipada $Y(r-1) \otimes Y(r-1)^{*^{-1}} = \beta$. Contabilizar el número de apariciones de las subllaves $k^{(r)}$.

3. Repetir 2. hasta que uno o más valores de la subllave $k^{(r)}$ se hayan contado significativamente con más frecuencia que otros. Tomar la subllave más frecuentemente contada ó este conjunto pequeño de tales subllaves, como la llave $k^{(r)}$.

Nótese que en el criptoanálisis diferencial la llave es fija y que sólo el texto en claro se puede elegir aleatoriamente, pues X se distribuye uniforme. Sin embargo en el cálculo de la probabilidad diferencial, el texto en claro y todas las subllaves son independientes y uniformemente distribuidas. Al calcular las probabilidades diferenciales para determinar cual diferencial usar en el ataque se emplea la siguiente hipótesis.

Hipótesis de equivalencia estocástica

Para una diferencial de ronda $(r-1)$, (α, β)

$$P((Y(r-1), Y(r-1)^*) \in \Delta(\beta) | (X, X^*) \in \Delta(\alpha)) \approx$$

$$P((Y(r-1), Y(r-1)^*) \in \Delta(\beta) | (X, X^*) \in \Delta(\alpha), k^{(1)} = w_1, \ldots, k^{(r-1)} = w_{r-1})$$

para casi todos los valores de subllave $(w_1, \ldots, w_{r-1})$.

De la descripción del criptoanálisis diferencial y del hecho que hay $2^m - 1$ posibles valores para $Y(r-1) \otimes Y(r-1)^{*^{-1}}$ se tiene que:

Si se supone cierta la hipótesis de equivalencia estocástica, entonces un cifrado de r rondas con subllaves independientes, es vulnerable al criptoanálisis diferencial si y sólo si la función ronda es débil y existe una diferencial de la ronda $(r-1)$, (α, β) tal que

$$P((Y(r-1), Y(r-1)^*) \in \Delta(\beta) | (X, X^*) \in \Delta(\alpha)) \gg 2^{-m}$$

donde m es la longitud del bloque del cifrado.

A continuación se presenta un tipo de cifrados iterados que son de especial interés para el criptoanálisis diferencial. Para dichos cifrados la sucesión (2.3.1) forma una cadena de Markov.

2.3.1. Cifrados de Markov

Recordemos que una sucesión discreta de variables aleatorias $v_0, v_1, \ldots, v_r$ es una cadena de Markov si para $0 \leq i < r$

$$P(v_{i+1} = \beta_{i+1} | v_i = \beta_i, v_{i-1} = \beta_{i-1}, \ldots, v_0 = \beta_0) = P(v_{i+1} = \beta_{i+1} | v_i = \beta_i)$$

Una cadena de Markov es homogénea si $P(v_{i+1} = \beta | v_i = \alpha)$ es independiente de i para toda α y β.

Definición 12. *Un cifrado iterado con función ronda $Y = f(X, K)$ es un cifrado de Markov si hay una operación de grupo $\otimes$ para definir diferencias tal que para toda elección de α y β, ambos distintos del neutro del grupo e, se tiene que*

$$P((Y, Y^*) \in \Delta(\beta) | (X, X^*) \in \Delta(\alpha), X = \gamma)$$

es independiente de γ cuando la subllave k se distribuye uniformemente, o equivalentemente si

$$P((Y, Y^*) \in \Delta(\beta) | (X, X^*) \in \Delta(\alpha), X = \gamma) = P((Y(1), Y(1)^*) \in \Delta(\beta) | (X, X^*) \in \Delta(\alpha))$$

para toda elección de γ cuando la subllave k se distribuye uniforme.

Antes de enunciar el siguiente teorema que relaciona a los cifrados de Markov con las cadenas de Markov, se define el concepto de distribución estacionaria para una cadena de Markov.

Definición 13. *Sea X_n, $n \geq 1$, una cadena de Markov con espacio de estados $\mathcal{E}$ y matriz de transición P. Si*

$$\sum_i \pi(i) P(i, j) = \pi(j), \ j \in \mathcal{E},$$

decimos que π es una distribución estacionaria para la cadena X_n

Teorema 4. *Si un cifrado de r rondas es de Markov y las llaves de las r rondas son independientes y uniformemente distribuidas, entonces la sucesión* (2.3.1) *es una cadena homogénea de Markov. Más aún esta cadena de Markov es estacionaria si $X \otimes X^{*-1}$ se distribuye uniformemente sobre los elementos no neutros del grupo.*

Demostración. Para demostrar que la sucesión de diferencias es una cadena de Markov basta con mostrar que para la segunda ronda se cumple que

$$P((Y(2), Y(2)^*) \in \Delta(\beta_2) | (Y(1), Y(1)^*) \in \Delta(\beta_1), (X, X^*) \in \Delta(\alpha))$$

$$= P((Y(2), Y(2)^*) \in \Delta(\beta_2) | (Y(1), Y(1)^*) \in \Delta(\beta_1))$$

Esto es cierto porque:

$P((Y(2), Y(2)^*) \in \Delta(\beta_2) | (Y(1), Y(1)^*) \in \Delta(\beta_1), (X, X^*) \in \Delta(\alpha))$

$= \sum_\gamma P(Y(1) = \gamma, (Y(2), Y(2)^*) \in \Delta(\beta_2) | (Y(1), Y(1)^*) \in \Delta(\beta_1), (X, X^*) \in \Delta(\alpha))$

$= \sum_\gamma P(Y(1) = \gamma | (Y(1), Y(1)^*) \in \Delta(\beta_1), (X, X^*) \in \Delta(\alpha)) P((Y(2), Y(2)^*)$
$\in \Delta(\beta_2) | (Y(1), Y(1)^*) \in \Delta(\beta_1), Y(1) = \gamma, (X, X^*)) \in \Delta(\alpha))$

$= {}^1 \sum_\gamma P(Y(1) = \gamma | (Y(1), Y(1)^*) \in \Delta(\beta_1), (X, X^*) \in \Delta(\alpha)) P((Y(2), Y(2)^*) \in \Delta(\beta_2) | (Y(1), Y(1)^*) \in \Delta(\beta_1), Y(1) = \gamma)$

$= \sum_\gamma P(Y(1) = \gamma | (Y(1), Y(1)^*) \in \Delta(\beta_1), (X, X^*) \in \Delta(\alpha)) P(Y(2), Y(2)^*) \in \Delta(\beta_2) | (Y(1), Y(1)^*) \in \Delta(\beta_1))$

$= P((Y(2), Y(2)^*) \in \Delta(\beta_2) | (Y(1), Y(1)^*) \in \Delta(\beta_1))$

Ahora la cadena de Markov es homogénea pues en cada ronda se usa la misma función ronda. Además para cualquier llave $K = k$ la función ronda $f(\cdot, k)$ es una biyección del conjunto de textos en claro en el conjunto de textos cifrados. Esta biyección induce una biyección entre las diferentes parejas (X, X^*) de textos en claro y las parejas (Y, Y^*) de textos cifrados. Luego del hecho que X y $X \otimes X^{*^{-1}} \neq e$ son independientes y uniformemente distribuidos implica que (X, X^*) se distribuye uniformemente sobre las distintas parejas de textos en claro. De este modo (Y, Y^*) se distribuye uniformemente sobre las distintas parejas de textos cifrados. Por lo tanto $Y \otimes Y^{*^{-1}} \neq e$ también se distribuye uniforme. Así la distribución uniforme es una distribución estacionaria para esta cadena de Markov. □

Ejemplo 1. *El cifrado DES es un cifrado de Markov bajo la diferencia $X \oplus X^{*^{-1}}$ donde $\oplus$ es la suma x-or.*

Para un cifrado de Markov con subllaves de ronda independientes y uniformemente distribuidas, la probabilidad de una r ronda característica está dada por la ecuación de Chapman-Kolgomorov para una cadena de Markov como:

$$P((Y(1), Y(1)^*) \in \Delta\beta_1, \ldots, (Y(r), Y(r)^*) \in \Delta\beta_r | (X, X^*) \in \Delta(\alpha))$$

$$= P((Y(1), Y(1)^*) \in \Delta\beta_i | (X, X^*) \in \Delta(\beta_{i-1}))$$

De esto se sigue que la probabilidad de una diferencial de la ronda r $(\beta_0, \beta r)$ es:

$$P((Y(r), Y(r)^*) \in \Delta\beta_r | (X, X^*) \in \Delta(\beta_0))$$

$$= \sum_{\beta_1} \sum_{\beta_2} \cdots \sum_{\beta_{r-1}} \prod_{i=1}^{r} P((Y(1), Y(1)^*) \in \Delta\beta_i | (X, X^*) \in \Delta(\beta_{i-1}))$$

Donde las sumas son sobre todos los posibles valores de las diferencias entre elementos distintos, es decir, sobre todos los elementos del grupo excepto el neutro.
Para cualquier cifrado de Markov sea $\prod$ la matriz de probabilidad de transición de la cadena

[1] Esto pasa porque la pareja $(Y(2), Y(2)^*)$ no tiene dependencia de (X, X^*) al estar determinados los valores $Y(1)$ y $(Y(1), Y(1)^*)$ y además porque el cifrado es de Markov.

homogénea de Markov (2.3.1). De esta forma la entrada (i, j) de $\prod$ es

$$P((Y(1), Y(1)^*) \in \Delta(\alpha_j) | (X, X^*) \in \Delta(\alpha_i)),$$

$\alpha_1, \alpha_2, \ldots, \alpha_M$ es un ordenamiento acordado de los M posibles valores de $X \otimes X^{*^{-1}}$ y $M = 2^m - 1$ para un cifrado de m bits. Luego, para cualquier r la entrada (i, j) en $\prod^r$, $p_{ij}^{(r)}$ es igual a

$$P((Y(r), Y(r)^*) \in \Delta(\alpha_j) | (X, X^*) \in \Delta(\alpha_i))$$

es decir $p_{ij}^{(r)}$ es la probabilidad de la diferencia de la ronda r, (α_i, α_j).

La seguridad de los criptosistemas iterados se basa en la creencia de que se puede obtener una función fuerte a partir de iterar funciones criptográficamente débiles suficientes veces. Con respecto a esto y a los cifrados de Markov se tiene el siguiente resultado.

Teorema 5. *Para un cifrado de Markov con longitud de bloque m y con subllaves independientes y uniformemente distribuidas, si la cadena de Markov semi infinita $Y(0) \otimes Y(0)^{*^{-1}}, Y(1) \otimes Y(1)^{*^{-1}}, \ldots$ tiene una distribución de probabilidad de estado estable, es decir, si hay un vector de probabilidad $(p_1, p_2, \ldots, p_M)$ tal que para todo α_i,*

$$\lim_{r \to \infty} P((Y(r), Y(r)^*) \in \Delta(\alpha_j) | (X, X^*) \in \Delta(\alpha_i)) = p_j$$

entonces esta distribución de estado estable debe ser la distribución uniforme, $(\frac{1}{M}, \frac{1}{M}, \ldots, \frac{1}{M})$, es decir

$$\lim_{r \to \infty} P((Y(r), Y(r)^*) \in \Delta(\beta) | (X, X^*) \in \Delta(\alpha)) = \frac{1}{2^m - 1}$$

para toda diferencial, (α, β). De este modo cada diferencial será aproximadamente la misma después de suficientes rondas. Si además se supone cierta la hipótesis de equivalencia estocástica entonces se cumple que para este cifrado de Markov, para casi todas las subllaves, este cifrado es seguro en contra de un ataque diferencial después de suficientes rondas.

Demostración. Por el Teorema 4 la distribución uniforme es una distribución estacionaria para esta cadena de Markov, entonces también es la distribución de estado estable pues se sabe que la existencia de una distribución de estado estable implica que una cadena homogénea de Markov tiene una única distribución estacionaria, la cual es la distribución de estado estable. □

En las siguientes secciones se analiza a profundidad el estudio del criptoanálisis diferencial tanto a PES como a IDEA realizado por Lay y Massey en [2].

2.3.2. Criptoanálisis diferencial del cifrado PES

En esta sección se muestra que el cifrado PES es de Markov, por lo cual PES es susceptible a un ataque por criptoanálisis diferencial. También se lleva acabo dicho criptoanálisis.

Sea $\otimes$ una operación definida para bloques de 64 bits como:

$$X \otimes K_A^{(i)} = (X_1 \odot k_1^{(i)}, X_2 \odot k_2^{(i)}, X_3 \oplus k_3^{(i)}, X_4 \oplus k_4^{(i)})$$

Donde $K_A^{(i)} = (k_1^{(i)}, k_2^{(i)}, k_3^{(i)}, k_4^{(i)})$ y cada $k_j^{(i)}$ es un subbloque de llave de la i-ésima ronda del PES. Nótese que la función $\otimes$ la que aparece en el inciso a) del paso 3) del algoritmo de cifrado del PES.

Lema 4. *PES es un cifrado de Markov bajo la diferencia* $X \otimes X^{*^{-1}}$.

Demostración. Nótese que si $S(1) = X \otimes K_A^{(1)}$ entonces

$$S(1) \otimes S(1)^{*^{-1}} = (X \otimes K_A^{(1)}) \otimes (X^* \otimes K_A^{(1)})^{-1} = X \otimes X^{*^{-1}}.$$

Además $P(S(1) = \lambda | (S(1), S(1)^*) \in \Delta(\beta_0), X = \gamma) = P(S(1) = \lambda | X = \gamma)$ es igual a

$$\begin{aligned} P(X \otimes K_A^{(1)} = \lambda | X = \gamma) &= P(X = \lambda \otimes (K^{(1)})^{-1} | X = \gamma) \\ &= P(\gamma = \lambda \otimes (K_A^{(1)})^{-1}) \\ &= P(K_A^{(1)} = \lambda \otimes \gamma^{-1}), \end{aligned}$$

utilizando la notación $Y_1 = Y(1)$, $Y_1^* = Y(1)^*$, $S_1 = S(1)$, $S_1^* = S(1)^*$, tenemos

$$\begin{aligned} &P((Y_1, Y_1^*) \in \Delta(\beta_1) | (X, X^*) \in \Delta(\beta_0), X = \gamma) \\ &= P((Y_1, Y_1^*) \in \Delta(\beta_1) | (S_1, S_1^*) \in \Delta(\beta_0), X = \gamma) \\ &= \textstyle\sum_\lambda P((Y_1, Y_1^*) \in \Delta(\beta_1), S_1 = \lambda | (S_1, S_1^*) \in \Delta(\beta_0), X = \gamma) \\ &= \textstyle\sum_\lambda P((Y_1, Y_1^*) \in \Delta(\beta_1) | (S_1, S_1^*) \in \Delta(\beta_0), X = \gamma, S_1 = \lambda) P(S_1 = \lambda | (S_1, S_1^*) \in \Delta(\beta_0), X = \gamma) \\ &= \textstyle\sum_\lambda P((Y_1, Y_1^*) \in \Delta(\beta_1) | (S_1, S_1^*) \in \Delta(\beta_0), S_1 = \lambda) P(K_A^{(1)} = \lambda \otimes \gamma^{-1}) \\ &= \frac{1}{2^{64}} \textstyle\sum_\lambda P((Y_1, Y_1^*) \in \Delta(\beta_1) | (S_1, S_1^*) \in \Delta(\beta_0), S_1 = \lambda) \end{aligned}$$

Por tanto esta probabilidad no depende de γ y de ahí que el cifrado es de Markov. □

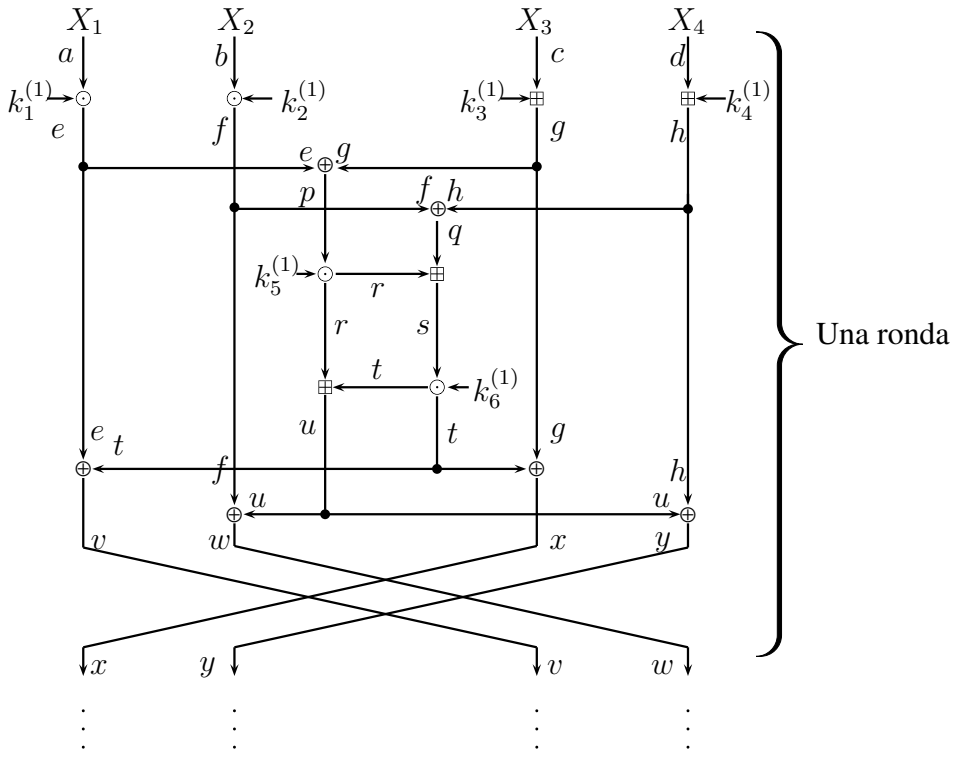
Sabiendo que PES es susceptible a continuación se estudia la propuesta de Lai y Massey de un ataque por criptoanálisis diferencial a PES.

Un ataque diferencial a las $8\frac{1}{2}$ rondas del sistema PES

A continuación se presenta el criptoanálisis diferencial para PES, el cual se lleva acabo calculando las probabilidades de algunas diferenciales de la primera ronda de PES. Esto permite calcular la probabilidad de una diferencial de la ronda $7\frac{1}{2}$ la cual hace posible encontrar la subllave que se usa en la última ronda de PES.

Se denota $\boxminus z$ como el inverso de z (mód 2^{16}), es decir, bajo la operación $\boxplus$, y a z^{-1} como el inverso de z (mód $2^{16}+1$). Ahora sea δ la diferencia entre dos números z_1 y z_2 bajo la operación $\odot$ y ∂ como la diferencia entre z_1 y z_2 bajo la operación $\boxplus$, es decir,

$$\delta z = z_1 \odot z_2^{-1}, \partial z = z_1 \boxminus z_2$$



$\oplus$: Suma bit a bit módulo 2 de subbloques de 16 bits
$\boxplus$: Suma módulo 2^{16} de enteros de 16 bits
$\odot$: Producto módulo $2^{16}+1$ de enteros de 16 bits con el subbloque cero correspondiendo a 2^{16}

Figura 2.2: La primera ronda de PES y la notación usada para el criptoanálisis diferencial.

de este modo para cualquier número k de 16 bits se tiene que

$$(z_1 \odot k) \odot (z_2 \odot k)^{-1} = z_1 \odot k \odot z_2^{-1} \odot k^{-1} = \delta z$$

y

$$(z_1 \boxplus k) \boxminus (z_2 \boxplus k) = z_1 \oplus k \boxminus z_2 \boxminus k = \partial z$$

Ahora si se supone cierto que $\delta z = z_1 \odot z_2^{-1} = 0$ entonces para $z_1, z_2 \neq 1, 0$

$$z_1 = 2^{16} z_2 = (2^{16} + 1) z_2 - z_2 = -z_2 \pmod{2^{16} + 1}$$

luego

$$z_1 + z_2 = 0 \pmod{2^{16} + 1}.$$

Además como z_1 y z_2 son enteros positivos, se tiene que $z_1 + z_2 = 2^{16} + 1$, de este modo si $z_1 = 0 \pmod{2^{16}}$ entonces $z_2 = 1$, así en cualquiera de los casos $z_1 + z_2 = 1 \pmod{2^{16}}$. Por otra parte si $z_1 + z_2 = 1 \pmod{2^{16}}$ entonces $z_1 = 0$ y $z_2 = 1$ o viceversa. Luego

$$\delta z = 0 \iff z_1 + z_2 = 1 \pmod{2^{16}} \tag{2.3.2}$$

Supongamos que la caja-MA de PES tiene entrada (p_1, q_1) y (p_2, q_2) y salida (t_1, u_1) y (t_2, u_2) respectivamente. Más aún supongamos que

$$p_1 + p_2 = 1 \pmod{2^{16}}$$
$$q_1 + q_2 = 0 \pmod{2^{16}}$$

Entonces $\delta p = 0$ por (2.3.2) luego

$$\delta r = r_1 \odot r_2^{-1} = (p_1 \odot k_5^{(1)}) \odot (p_2 \odot k_5^{(1)})^{-1} = p_1 \odot p_2^{-1} = \delta p = 0$$

de esto se sigue que $r_1 + r_2 = 1 \pmod{2^{16}}$. Por lo que

$$s_1 + s_2 = (r_1 \boxplus q_1) + (r_2 \boxplus q_2) = r_1 + r_2 + q_1 + q_2 = 1 \pmod{2^{16}}$$

y se cumple que $\delta s = 0$. De tal manera que $\delta t = 0$ pues

$$\delta t = t_1 \odot t_2^{-1} = (s_1 \odot k_6^1) \odot (s_2 \odot k_6^1)^{-1} = s_1 \odot s_2^{-1} = \delta s = 0$$

luego $t_1 + t_2 = 1 \pmod{2^{16}}$ y por lo tanto

$$u_1 + u_2 = (r_1 \boxplus t_1) + (r_2 \boxplus t_2) = r_1 + r_2 + t_1 + t_2 = 2 \pmod{2^{16}}.$$

Así, se mostró la siguiente relación entre un par de entradas y un par de salidas de la caja MA

$$p_1 + p_2 = 1,\ q_1 + q_2 = 0 \pmod{2^{16}} \implies t_1 + t_2 = 1,\ u_1 + u_2 = 2 \pmod{2^{16}} \tag{2.3.3}$$

Consideremos una ronda del cifrado con subbloques de entrada (a, b, c, d) y subbloques de salida (x, y, v, w), donde además se tiene una pareja de entradas con diferencia

$$(\delta a, \delta b, \partial c, \partial d) = (0, 0, 0, n)$$

donde $n \in S = \{\pm 1, \pm 3, \pm 5, \pm 7\}$.

De este modo $(\delta e, \delta f, \partial g, \partial h) = (0, 0, 0, n)$ pues

$$\delta e = e_1 \odot e_2 = (a_1 \odot k_1^{(1)}) \odot (a_2 \odot k_1^{(1)})^{-1} = \delta a = 0$$

$$\delta f = f_1 \odot f_2 = (b_1 \odot k_2^{(1)}) \odot (b_2 \odot k_2^{(1)})^{-1} = \delta b = 0$$

$$\partial g = g_1 \boxplus g_2 = (c_1 \oplus k_3^{(1)}) \boxminus (c_2 \oplus k_3^{(1)}) = \partial c = 0$$

$$\partial h = h_1 \boxplus h_2 = (d_1 \oplus k_4^{(1)}) \boxminus (d_2 \oplus k_4^{(1)}) = \partial d = n.$$

Por otro lado una ronda del cifrado se puede separar en dos partes, pues e y g se operan para obtener p y por otra parte f y h se operan para obtener q. Empezando con la mitad (a, c) supongamos que

$$e_1 = (\alpha, 10 \ldots 0, \theta)$$

para un número α de $16 - (l + 1)$ bits, donde $l \in \{0, \ldots, 15\}$ y $\theta \in \{0, 1\}$, de este modo hay $l - 1$ ceros consecutivos antes de θ. Tal θ existe con probabilidad 2^{-l} pues podemos fijar α y θ y la probabilidad sólo depende de la elección de l. Luego de (2.3.2) se tiene que como $\delta e = 0$ entonces $e_1 + e_2 = 1 \pmod{2^{16}}$, así e_2 tiene la forma

$$e_2 = (\alpha^c, 10 \ldots 0, \theta^c),$$

pues sabemos que $e_1 + e_2 = 1 \pmod{2^{16}}$ por lo tanto que se requiere que $e_1 + e_2$ sea el número binario de 17 bits $10 \ldots 01$ pero $\theta \in \{0, 1\}$, así el bit menos significativo de e_2 es θ^c, luego la cadena de $l - 1$ bits en e_1 obliga a que e_2 tenga la misma cadena de $l - 1$ bits para obtener un bit de acarreo. De este modo la suma de dichas cadenas intermedias, da como resultado $1, 00 \ldots 0$ con $l - 1$ ceros consecutivos, ahora este bit extra obliga a que los bits más significativos de e_2 sean α^c para obtener el número de 17 bits $10 \ldots 01$. Finalmente se requiere que $\alpha + \beta + 1 = 10 \ldots 0$ donde la cadena de ceros es de $16 - (l + 1)$ bits y esto sólo se consigue si $\beta = \alpha^c$. Además e_2 tiene esta forma con probabilidad 2^{-l}. Por otro lado como $\partial g = 0$ podemos elegir a $g_1 = g_2$, de la forma

$$g_1 = (\beta, 0 \ldots 0, \phi)$$

donde β es un número de $16 - (l + 1)$ bits y $\phi \in \{0, 1\}$. Por consiguiente podemos calcular p_1 y p_2 como sigue:

$$p_1 = e_1 \oplus g_1 = (\alpha \oplus \beta, 10 \ldots 0, \theta \oplus \phi)$$

$$p_2 = e_2 \oplus g_2 = (\alpha^c \oplus \beta, 10 \ldots 0, \theta^c \oplus \phi).$$

Nótese que $\alpha^c \oplus \beta = (\alpha \oplus \beta)^c$ y que $\theta^c \oplus \phi = (\theta \oplus \phi)^c$ luego

$$p_2 = ((\alpha \oplus \beta)^c, 10 \ldots 0, (\theta \oplus \phi)^c)$$

por lo tanto $p_1 + p_2 = 1 \pmod{2^{16}}$ por la misma razón que $e_1 + e_2 = 1 \pmod{2^{16}}$. Ahora por (2.3.3) se sigue que

$$q_1 + q_2 = 0 \pmod{2^{16}} \implies t_1 + t_2 = 1 \pmod{2^{16}}.$$

Así se tiene con probabilidad 2^{-l} que

$$t_1 = (\gamma, 10 \ldots 0, \rho)$$

$$t_2 = (\gamma^c, 10 \ldots 0, \rho^c)$$

para algún número γ de $16 - (l+1)$ bits y $\rho \in \{0, 1\}$. Por lo tanto para l dada, e_i, g_i y t_i los tres tienen la forma antes mencionada con probabilidad 2^{-3l}. Luego e_i, g_i y t_i todas tienen la forma especificada anteriormente para alguna l con probabilidad

$$\sum_{l=1}^{15} 2^{-3l} \approx \frac{1}{7}$$

De este modo con probabilidad $\frac{1}{7}$ se tienen los siguientes resultados. Primero

$$v_1 = e_1 \oplus t_1 = (\alpha \oplus \gamma, 00 \ldots 0, \theta \oplus \rho)$$

$$v_2 = e_2 \oplus t_2 = (\alpha^c \oplus \gamma^c, 00 \ldots 0, \theta^c \oplus \rho^c) = (\alpha \oplus \gamma, 00 \ldots 0, \theta \oplus \rho)$$

así $v_1 = v_2$ y por lo tanto $\partial v = 0$. El segundo resultado es que

$$x_1 = g_1 \oplus t_1 = (\beta \oplus \gamma, 10 \ldots 0, \phi \oplus \rho)$$

$$x_2 = g_2 \oplus t_2 = (\beta \oplus \gamma^c, 10 \ldots 0, \phi \oplus \rho^c)$$

de esto se sigue que $x_1 + x_2 = 1 \pmod{2^{16}}$ y que $\delta x = 0$. Se ha demostrado que si $q_1 + q_2 = 0 \pmod{2^{16}}$ y si $(\delta a, \partial c) = (0, 0)$ entonces

$$(\delta x, \partial v) = (0, 0)$$

con probabilidad $\approx \frac{1}{7}$.

Para la segunda parte del cifrado, se tiene que $\delta f = 0$ y $\partial h \in S$. Ahora si se encuentran f y h tales que $q_1 + q_2 = 0 \pmod{2^{16}}$ entonces dado que $\delta p = 0$, se tendría que $u_1 + u_2 = 2 \pmod{2^{16}}$. Para encontrar una diferencial con alta probabilidad se necesita encontrar (u_1, u_2) con $\delta y = 0$ y $\partial w \in S$, donde

$$y_i = h_i \oplus u_i$$

$$w_i = f_i \oplus u_i.$$

Así es necesario encontrar f, h y u tales que

$$f_1 + f_2 = 1 \pmod{2^{16}},\ h_1 - h_2 \in S,\ u_1 + u_2 = 2 \pmod{2^{16}},$$

y que satisfacen, para q, w e y que

$$q_1 + q_2 = 0 \pmod{2^{16}},\ w_1 - w_2 \in S,\ y_1 + y_2 = 1 \pmod{2^{16}} \tag{2.3.4}$$

Ahora se pueden encontrar la mayoría de las posibles soluciones como sigue: supóngase que

$$\begin{array}{ll} f_1 = (\alpha, \theta_1), & f_2 = (\alpha^c, \theta_2), \\ h_1 = (\beta, \phi_1), & h_2 = (\beta, \phi_2), \\ u_1 = (\gamma, \rho_1), & u_2 = (\gamma^c, \rho_2), \end{array}$$

para números α, β, γ de 12 bits y números θ_i, ϕ_i, ρ_i de 4 bits. Luego como se cumple la igualdad $f_1 + f_2 = (\alpha + \alpha^c, \theta_1 + \theta_2) = 1 \pmod{2^{16}}$ y la segunda parte de la suma es un complemento se requiere que tanto el bit más significativo como el menos de $\theta_1 + \theta_2$ sean 1 y el resto cero, dicho número en decimal es 17 entonces $\theta_1 + \theta_2 = 17$. Por un razonamiento similar como $u_1 + u_2 = 2 \pmod{2^{16}}$ entonces $\rho_1 + \rho_2 = 18$ y como $h_1 - h_2 \in S$ luego $\phi_1 - \phi_2 \in S$. De este modo se requiere que

$$\theta_1 + \theta_2 = 17, \quad \phi_1 - \phi_2 \in S, \quad \rho_1 + \rho_2 = 18. \tag{2.3.5}$$

Ahora por definición,

$$\begin{array}{ll} q_1 = (\alpha \oplus \beta, \theta_1 \oplus \phi_1), & q_2 = (\alpha^c \oplus \beta, \theta_2 \oplus \phi_2), \\ y_1 = (\beta \oplus \gamma, \phi_1 \oplus \rho_1), & y_2 = (\beta \oplus \gamma^c, \phi_2 \oplus \rho_2), \\ w_1 = (\alpha \oplus \gamma, \theta_1 \oplus \rho_1), & w_2 = (\alpha^c \oplus \gamma^c, \theta_2 \oplus \rho_2) = (\alpha \oplus \gamma, \theta_2 \oplus \rho_2). \end{array}$$

Por otra lado como $q_1 + q_2 = 0 \pmod{2^{16}}$, $w_1 - w_2 \in S$, $y_1 + y_2 = 1 \pmod{2^{16}}$ entonces es necesario encontrar soluciones para el siguiente sistema

$$\begin{array}{c} (\theta_1 \oplus \phi_1) + (\theta_2 \oplus \phi_2) = 16 \\ (\phi_1 \oplus \rho_1) + (\phi_2 \oplus \rho_2) = 17 \\ (\theta_1 \oplus \rho_1) + (\theta_2 \oplus \rho_2) \in S \end{array} \tag{2.3.6}$$

Computacionalmente se encontró que hay 514 tripletas que satisfacen (2.3.5) y (2.3.6). En la matriz M que se muestra a continuación se presentan con el número de soluciones que corresponden a cada par de elementos de S, donde los renglones y las columnas están en el siguiente orden $\{-1, 1, -3, 3, -5, 5, -7, 7\}$:

$$M = \begin{pmatrix} 0 & 73 & 36 & 0 & 0 & 27 & 2 & 0 \\ 73 & 0 & 0 & 36 & 27 & 0 & 0 & 2 \\ 36 & 0 & 0 & 0 & 18 & 0 & 0 & 9 \\ 0 & 36 & 0 & 0 & 0 & 18 & 9 & 0 \\ 0 & 27 & 18 & 0 & 0 & 0 & 0 & 0 \\ 27 & 0 & 0 & 18 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 9 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{2.3.7}$$

Esto da una matriz de transición de probabilidad $T_{b,d} = 2^{-12}M$

$$T_{b,d} = \begin{pmatrix} 0 & 0{,}017822 & 0.0087890 & 0 & 0 & 0.0065917 & 0.0004882 & 0 \\ 0.017822 & 0 & 0 & 0.0087890 & 0.0065917 & 0 & 0 & 0.0004882 \\ 0.0087890 & 0 & 0 & 0 & 0.0043945 & 0 & 0 & 0.0021972 \\ 0 & 0.0087890 & 0 & 0 & 0 & 0.0043945 & 0.0021972 & 0 \\ 0 & 0.0065917 & 0.0043945 & 0 & 0 & 0 & 0 & 0 \\ 0.0065917 & 0 & 0 & 0.0043945 & 0 & 0 & 0 & 0 \\ 0.0004882 & 0 & 0 & 0.0021972 & 0 & 0 & 0 & 0 \\ 0 & 0.0004882 & 0.0021972 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

la cual puede escribirse como

$$T_{b,d} = 10^{-7} \begin{pmatrix} 0 & 178222 & 87890 & 0 & 0 & 65917 & 4882 & 0 \\ 178222 & 0 & 0 & 87890 & 65917 & 0 & 0 & 4882 \\ 87890 & 0 & 0 & 0 & 43945 & 0 & 0 & 21972 \\ 0 & 87890 & 0 & 0 & 0 & 43945 & 21972 & 0 \\ 0 & 65917 & 43945 & 0 & 0 & 0 & 0 & 0 \\ 65917 & 0 & 0 & 43945 & 0 & 0 & 0 & 0 \\ 4882 & 0 & 0 & 21972 & 0 & 0 & 0 & 0 \\ 0 & 4882 & 21972 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

De este modo si $p_1 + p_2 = 1 \pmod{2^{16}}$, $(\delta b, \partial d) = (0, n_1)$ entonces $(\delta y, \partial w) = (0, n_2)$ para $n_1, n_2 \in S$ con probabilidad dada por la entrada correspondiente en la matriz $T_{b,d}$.

Finalmente se puede calcular una aproximación de una diferencial de la primera ronda de PES. La cual está dada por la matriz de transición $T_{a,b,c,d} = \frac{1}{7}T_{b,d}$. Luego

$$T_{a,b,c,d} = 10^{-7} \begin{pmatrix} 0 & 25460 & 12555 & 0 & 0 & 9416 & 697 & 0 \\ 25460 & 0 & 0 & 12555 & 9416 & 0 & 0 & 697 \\ 12555 & 0 & 0 & 0 & 6277 & 0 & 0 & 3138 \\ 0 & 12555 & 0 & 0 & 0 & 6277 & 3138 & 0 \\ 0 & 9416 & 6277 & 0 & 0 & 0 & 0 & 0 \\ 9416 & 0 & 0 & 6277 & 0 & 0 & 0 & 0 \\ 697 & 0 & 0 & 3138 & 0 & 0 & 0 & 0 \\ 0 & 697 & 3138 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

y por lo tanto

$$(\delta a, \delta b, \partial c, \partial d) = (0, 0, 0, n_1) \implies (\delta x, \delta y, \partial v, \partial w) = (0, 0, 0, n_2), n_1, n_2 \in S$$

con probabilidad dada por al entra correspondiente de $T_{a,b,c,d}$.

Ahora se puede calcular la matriz de transición de 7 rondas, cuyas entradas dan la probabilidad que, dado que las parejas de entrada de la primera ronda difieren por un valor dado, entonces las diferencias de la salida de la ronda siete y las diferencias de la entrada de las rondas intermedias son todas de la forma requerida. Se denota a esta matriz como $T7_{a,b,c,d}$, la cual se calcula fácilmente pues

$$T7_{a,b,c,d} = T^7_{a,b,c,d} = \frac{1}{(2^{12*7})(7^7)}M^7.$$

El cálculo resulta

$$T7_{a,b,c,d} = 2^{-58} \begin{pmatrix} 0 & 1.223 & 0.533 & 0 & 0 & 0.433 & 0.073 & 0 \\ 1.223 & 0 & 0 & 0.533 & 0.433 & 0 & 0 & 0.073 \\ 0.533 & 0 & 0 & 0.232 & 0.189 & 0 & 0 & 0.032 \\ 0 & 0.533 & 0.232 & 0 & 0 & 0.189 & 0.032 & 0 \\ 0 & 0.433 & 0.189 & 0 & 0 & 0.153 & 0.026 & 0 \\ 0.433 & 0 & 0 & 0.189 & 0.153 & 0 & 0 & 0.026 \\ 0.073 & 0 & 0 & 0.032 & 0.026 & 0 & 0 & 0.004 \\ 0 & 0.073 & 0.032 & 0 & 0 & 0.026 & 0.004 & 0 \end{pmatrix}$$

Recordemos que los renglones y las columnas están en el orden $\{-1, 1, -3, 3, -5, 5, -7, 7\}$. Luego esta matriz se puede reducir a la siguiente

$$T7_{a,b,c,d} = 2^{-58} \begin{pmatrix} 1.223 & 0.533 & 0.433 & 0.026 \\ 0.533 & 0.232 & 0.189 & 0.032 \\ 0.433 & 0.189 & 0.153 & 0.032 \\ 0.073 & 0.032 & 0.032 & 0.004 \end{pmatrix}$$

donde la entrada (i, j) de la matriz da una buena aproximación de la probabilidad de las diferenciales de la ronda 7 de la forma

$$(\delta a^1, \delta b^1, \partial c^1 \partial d^1) = (0, 0, 0, \pm(2i-1)), (\delta a^7, \delta b^7, \partial c^7 \partial d^7) = (0, 0, 0, \mp(-1)^{i+j}(2j-1))$$

donde los superíndices indican el índice de la ronda. Se considera que la aproximación es buena pues la diferencial de la séptima ronda se construye a partir de la diferenciales más probables para la ronda 1, además aunque las probabilidades son pequeñas su valor es suficiente para iniciar el ataque.

De este modo se puede obtener la llave de la siguiente manera. Supóngase que se eligen inicialmente las diferencias $(\delta a^1, \delta b^1, \partial c^1 \partial d^1) = (0, 0, 0, 1)$, luego después de 7 rondas de cifrado se sabe que

$$(\delta a^8, \delta b^8, \partial c^8 \partial d^8) = (\delta e^8, \delta f^8, \partial g^8 \partial h^8) = (0, 0, 0, -1)$$

con probabilidad 1.22×2^{-58}. Como vimos anteriormente para la mitad del cifrado (a, c) se sigue que si $(\delta a, \partial c) = (0, 0)$ entonces $\delta p = 0$ con probabilidad $\sum_{l=1}^{15} 2^{-l} \approx \frac{1}{3}$ y por búsqueda exhaustiva se puede calcular que

$$(\delta b, \partial d) = (0, 1) \implies q_1 + q_2 = 0 \pmod{2^{16}}$$

con probabilidad aproximadamente de $\frac{42}{256}$. Luego se tiene que

$$t_1 + t_2 = 1 \pmod{2^{16}},\ u_1 + u_2 = 2 \pmod{2^{16}}$$

con probabilidad $\approx \left(\frac{1}{3}\right)\left(\frac{42}{256}\right) = \frac{14}{256} = 1.75 \times 2^{-5}$. Después de 7 rondas se obtiene

$$(\delta e^8, \delta f^8, \partial g^8 \partial h^8) = (0, 0, 0, -1)$$

$$p_1^8 + p_2^8 = 1,\ q_1^8 + q_2^8 = 0,\ t_1^8 + t_2^8 = 1,\ u_1^8 + u_2^8 = 2$$

con probabilidad $(1.22\times2^{-58})(1.75\times2^{-5})$ =1.07×2^{-62}. Enseguida se describe cómo se pueden determinar los 96 bits de llave utilizados en este paso hasta obtener el texto cifrado. Si se denota el texto cifrado como $e_i^9, f_i^9, g_i^9, h_i^9$, luego observando un par de textos cifrados resulta

$$(\delta e^9, \delta f^9, \partial g^9 \partial h^9) = (\delta x^8, \delta y^8, \partial v^8 \partial w^8).$$

Como anteriormente, se divide la ronda de cifrado en dos mitades y se considera primero la mitad que termina con (x, v). Para una pareja dada $(\delta x^8, \partial v^8)$, cada tercia (t_1^8, x_1^8, v_1^8) de 48 bits determina todas las demás cantidades en esta mitad del cifrado afuera de la caja MA. Además se deben satisfacer tres restricciones de 16 bits para cumplir las condiciones para las características de las $7\frac{1}{2}$ rondas dadas anteriormente. Así para una pareja $(\delta x^8, \partial v^8)$ dada, se tendrá en promedio una tercia (t_1^8, x_1^8, v_1^8) de 48 bits y por lo tanto un posible valor para los bloques de llave $k_1^{(9)}$ y $k_3^{(9)}$. Sin embargo, algunos valores de $(\delta x^8, \partial v^8)$ darán considerablemente más tercias (t_1^8, x_1^8, v_1^8). Por ejemplo, si $(\delta x^8, \partial v^8) = (0, 0)$ es la salida de la diferencial, entonces un séptimo de toda las tercias (t_1^8, x_1^8, v_1^8) serán posibles. Tales diferencias no ocurren frecuentemente, la anterior tiene probabilidad 2^{-32}. Nótese que estas tercias de 48 bits y los dos bloques de llave de 16 bits pueden ser precalculados para cada valor de $(\delta e^9, \partial g^9)$. Un resultado similar se tiene para $(\delta f^9, \partial h^9)$, es decir que para $(\delta w^8, \partial y^8)$ dado se obtendrá en promedio una tercia (u_1^8, w_1^8, y_1^8) de 48 bits y también un posible valor para un valor probable para los bloques de llave $k_2^{(9)}$ y $k_4^{(9)}$. Combinando los dos resultados, para cada valor $(\delta e^9, \delta f^9, \partial g^9 \partial h^9)$, se obtiene en promedio un valor para $(p_1^8, q_1^8, t_1^8, u_1^8)$ [1] y al invertir la caja MA, se obtiene en promedio un valor para los bloques de llave $k_5^{(8)}$ y $k_6^{(8)}$. De este modo para cada pareja de texto cifrado y en claro, se obtiene en promedio un posible valor para los 96 bits de llave, es decir, un valor particular para los 96 bits de llave ocurre con probabilidad de 2^{-96} por par de cifrado.

Si una llave ocurre con probabilidad p, entonces en 2^N parejas de cifrado, la llave ocurre k veces con probabilidad

$$\binom{2^N}{k} p^k (1-p)^{2^N-k} \approx \frac{(2^N p)^k}{k!} e^{-2^N p}$$

recordando que la distribución binomial se puede aproximar como una exponencial, véase [8]. Luego, en 2^N parejas de cifrado, una llave incorrecta ocurrirá dos o más veces con probabilidad $\frac{1}{2}(2^{2(N-96)})$. Si se cifra todo el espacio de mensajes $(N = 64,\ 2^N p = 2^{-32})$, entonces una llave errónea sucederá dos o más veces con probabilidad

$$1 - P(\text{la llave errónea ocurre 0 ó 1 vez}) = 1 - e^{-2^{-32}} - 2^{-32} e^{-2^{-32}} \approx \frac{1}{2^{-64}},$$

de tal manera que este evento ocurre para 2^{31} de las llaves de 96 bits. La llave correcta, sin embargo, aparece con esta probabilidad siempre que la diferencial de $7\frac{1}{2}$ no ocurra, y

[1] Esto pasa pues de las tercias (t_1^8, x_1^8, v_1^8) y (u_1^8, w_1^8, y_1^8), se obtiene $e_1^8 = v_1^8 \oplus t_1^8$ y $g_1^8 = x_1^8 \oplus t_1^8$ luego $p_1^8 = e_1^8 \oplus g_1^8 = v_1^8 \oplus x_1^8$, y por un razonamiento similar se obtiene que $q_1^8 = w_1^8 \oplus y_1^8$

con probabilidad $p = 1.07 \times 2^{-62}$ en caso contrario. Por lo tanto en 2^{64} parejas de cifrado ($2^N p = 4{,}28$) la llave correcta aparecerá menos de dos veces con probabilidad

$$e^{-4.28} + 4.28e^{-4.28} \approx 0.073,$$

así que el evento que la llave ocurra más de un vez es altamente probable (alrededor del 93 % de las veces). Para encontrar la llave, se puede intentar todos los 96 bits de llave que ocurren más de una vez en este procedimiento, hay aproximadamente 2^{31} de tales llaves. Sin embargo, como las subllaves están determinadas por la llave de 128 bits, hay 2^{32} llaves que dan origen a cada subllave de 96 bits de la última ronda, de este modo hay 2^{63} llaves que dan subllaves de la última ronda que ocurren más de una vez. Como la llave real ocurre con una probabilidad mucho más alta que cualquiera de las llaves falsas, se esperaría encontrarla antes de haber intentado con muchas llaves falsas. De este modo se tendrían que probar tan sólo las 2^{32} llaves que dan la subllave correcta de 96 bits, y tal vez unas pocas subllaves más. Luego la búsqueda de la llave en la práctica se verá reducida a casi 2^{32}.

2.3.3. Criptoanálisis diferencial de IDEA

En esta sección se demostrará que el sistema IDEA es resistente al criptoanálisis diferencial. Primero se enuncia un resultado que relaciona la simetría de la matriz de transición de un cifrado de Markov y su eficacia en contra del ataque diferencial. Después se mostrarán algunas relaciones útiles de la caja MA y se encontrará la probabilidad de una de las diferenciales más probables de IDEA. Finalmente se demuestra que la matriz de transición de probabilidad es no simétrica haciendo a IDEA suficientemente seguro en contra del criptoanálisis diferencial.

Antes de describir el proceso del criptoanálisis de IDEA se presenta el siguiente resultado, el cual es una requerimiento práctico para la seguridad de un cifrado de Markov.

Proposición 2.3.1. *La matriz de transición de probabilidad de un cifrado de Markov debe ser no simétrica.*

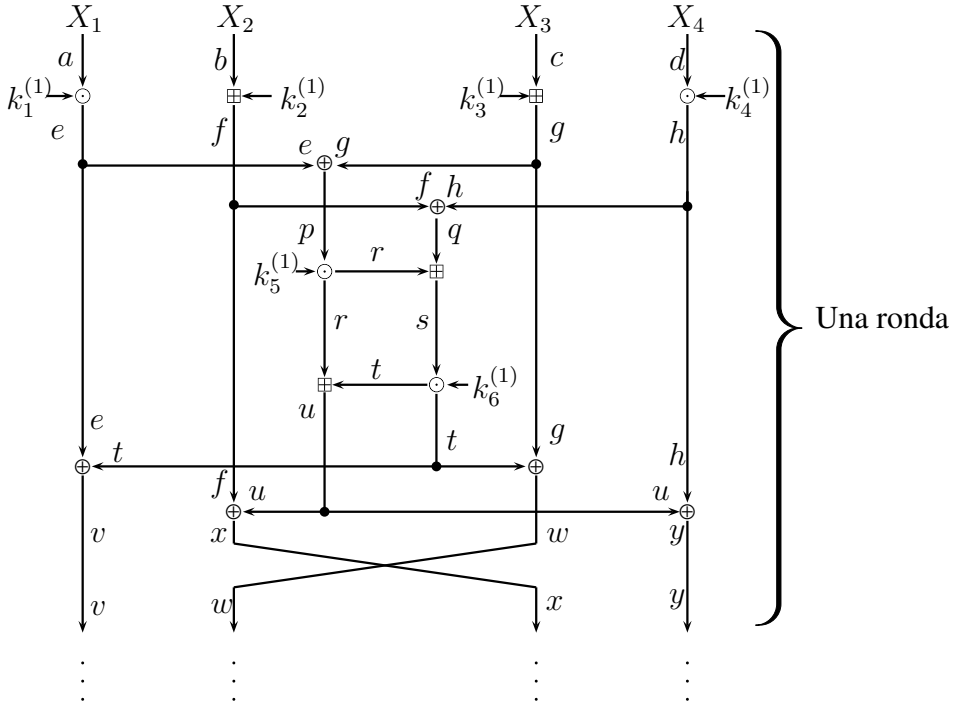
Demostración. Suponga que la matriz de transición es simétrica. Entonces, cada par de diferenciales de una ronda de la forma (a, b) y (b, a), donde a es la entrada diferencia y b es la salida diferencia, tendrán la misma probabilidad $p_{ab} = p_{ba}$. Suponga ahora que (a, b) es la diferencial más probable de una ronda. Entonces $(a, b, a, b, \ldots, a, b, a)$ será la más probable característica de la ronda $2i$, recordemos que la característica de la ronda i se define como una i-tupla $(\alpha, \beta_1, \ldots, \beta_i)$ considerada como un posible valor de $(\Delta X, \Delta Y(1), \ldots, \Delta Y(i))$. Para i pequeña y en el caso de que p_{ab} es significativamente grande, (a, a) será una diferencial de la ronda $2i$ con probabilidad alta. Esta probabilidad puede ser bien aproximada por la probabilidad correspondiente a la característica de la ronda $2i$, es decir, $p_{aa}^{(2i)} \approx p_{ba}^{2i}$. De manera similar, (a, b) será una diferencial de la ronda $(2i+1)$ con probabilidad alta. Esta probabilidad se puede aproximar por la probabilidad de la característica de la ronda $(2i+1)$ más probable $(a, b, a, b, \ldots, a, b)$, i.e., $p_{ba}^{(2i+1)} \approx p_{ba}^{2i+1}$. Luego, la concatenación de la diferencial de una ronda consigo misma $r-1$ veces produce la más probable característica de la ronda r, la cual da una diferencial de la ronda r con alta probabilidad. De este modo la no simetría de la matriz de transición previene tal concatenación de diferenciales de una ronda con probabilidad alta. □

A continuación se presenta el criptoanálisis de IDEA y para comenzarlo es necesario definir las diferencias de las distintas operaciones en el cifrado. En este caso se utilizaron las siguientes:

$$\delta x = x \odot (x^*)^{-1}$$

$$y$$

$$\partial x = x \boxplus (-x^*) = x \boxminus x^*$$



$\oplus$: Suma bit a bit módulo 2 de subbloques de 16 bits
$\boxplus$: Suma módulo 2^{16} de enteros de 16 bits
$\odot$: Producto módulo $2^{16}+1$ de enteros de 16 bits con el subbloque cero correspondiendo a 2^{16}

Figura 2.3: La primera ronda de IDEA y la notación usada para el criptoanálisis diferencial.

Para iniciar el criptoanálisis se necesita encontrar algunas relaciones útiles entre las operaciones del cifrado. En el caso de IDEA se encuentra la siguiente:

Proposición 2.3.2. *Si la función dada por la estructura MA se escribe como*

$$(t, u) = MA(p, q; k_5, k_6) \tag{2.3.8}$$

entonces para cada elección de llave (k_5, k_6)*, las entradas* (p, q)*,* (p^*, q^*) *y las salidas* (t, u)*,* (t^*, u^*) *de la caja* MA *para la misma llave* (k_5, k_6) *satisfacen la siguiente relación:*

$$\delta p = 1, \partial q = 0 \Leftrightarrow \delta t = 1, \partial u = 0. \tag{2.3.9}$$

Demostración. Supondremos primero que $\delta p = 1$ y $\partial q = 0$, entonces se tiene que

$$\delta r = (p \odot k_5) \odot (p^* \odot k_5)^{-1} = p \odot (p^*)^{-1} = \delta p = 1$$

y

$$\partial s = (q \boxplus r) \boxminus (q^* \boxplus r^*) = r \boxminus r^*$$

pero como $\delta r = 1$ se sigue que $r = r^*$ y así $\partial s = 0$. La siguiente relación en la caja MA nos dice que

$$\delta t = (s \odot k_6) \odot (s^* \odot k_6)^{-1} = s \odot (s^*)^{-1}$$

pero $\partial s = s \boxminus s^* = 0$, luego $\delta t = s \odot (s^*)^{-1} = 1$, y $s = s^*$. Finalmente

$$\partial u = (t \boxplus r) \boxminus (t^* \boxplus r^*) = (t \boxminus t^*) \boxplus (r \boxminus r^*) = 0,$$

esta última igualdad se da porque $\delta t = t \odot (t^*)^{-1} = 1$.

Para la otra implicación se tiene que $t = t^*$ y $u = u^*$ pues $\delta t = 1$ y $\partial u = 0$. Luego como

$$\partial u = (t \boxplus r) \boxminus (t^* \boxplus r^*) = (t \boxminus t^*) \boxplus (r \boxminus r^*) = 0$$

$$y$$

$$\delta t = (s \odot k_6) \odot (s^* \odot k_6)^{-1} = s \odot (s^*)^{-1} = 1$$

entonces $r = r^*$ y $s = s^*$, de este modo $\delta r = 1$ y $\partial s = 0$. Ahora $\partial q = 0$ porque tenemos las siguientes igualdades $\partial s = (q \boxplus r) \boxminus (q^* \boxplus r^*) = q \boxminus q^* = \partial q = 0$. Por último se tiene que

$$\delta r = (p \odot k_5) \odot (p^* \odot k_5)^{-1} = p \odot (p^*)^{-1} = \delta p = 1$$

de este modo $\delta p = 1$ y $\partial q = 0$. □

Ahora las diferenciales más probables en IDEA tienen diferencia de entrada $(1, l, 0, 0)$ y diferencia de salida $(1, 0, j, 0)$ con $l, j \in \{1, 3, \ldots, 2^n - 1\}$, es decir en los enteros impares entre 1 y $2^n - 1$. Pero para mostrar la resistencia de IDEA en contra del criptoanálisis diferencial es suficiente con probar que su matriz de transición es no simétrica, para lo cual sólo calcularemos la probabilidad de la diferencial más sencilla y en base a ella veremos que la matriz es no simétrica.

Proposición 2.3.3. *Para IDEA se cumple que la diferencia de entrada* $(1, 1, 0, 0)$ *tiene diferencia de salida* $(1, 0, 1, 0)$ *con probabilidad*

$$P((Y, Y^*) \in \Delta(1, 0, 1, 0) | (X, X^*) \in \Delta(1, 1, 0, 0)) = 2^{-30}.$$

Demostración. Suponga que se tienen un par de entradas (a, b, c, d) y (a^*, b^*, c^*, d^*) con diferencia

$$(\delta a, \partial b, \partial c, \delta d) = (1, 1, 0, 0).$$

Luego

$$(\delta e, \partial f, \partial g, \delta h) = (1, 1, 0, 0)$$

pues $\delta e = e \odot (e^*)^{-1} = (a \odot k_1) \odot (a^* \odot k_1)^{-1}) = a \odot (a^*)^{-1} = \delta a$ y de manera similar se puede obtener la relación entre δd y δh. En el caso de ∂f se tiene que
$\partial f = f \boxminus f^* = (b \boxplus k_2) \boxminus (b^* \boxplus k_2) = b \boxminus b^*$, análogamente se deduce que $\partial g = \partial c$.

Ahora como $\delta e = 1$ y $\partial g = 0$ se cumple que $e = e^*$ y $g = g^*$ pues recordemos que $\delta e = e \odot (e^*)^{-1}$ y $\partial g = g \boxminus g^*$. Del siguiente paso del cifrado se tiene que $p = e \oplus g$ y $p^* = e^* \oplus g^* = e \oplus g$, luego $p = p^*$ y $r = r^*$ pues $\delta r = \delta p$. Por otra parte como $\delta v = 1$ se sigue que $t = t^*$ porque $\delta v = (e \oplus t) \odot (e^* \oplus t^*) = 1$ luego, $(e \oplus t) = (e^* \oplus t^*)$ pero $e = e^*$ por lo tanto $t = t^*$. Además como $\partial u = (r \boxplus t) \boxminus (r^* \boxplus t^*)$, $r = r^*$ y $t = t^*$ entonces $\partial u = 0$ y $u = u^*$. Luego (4.4.5) implica que $\partial q = 0$. Así si la diferencia de la entrada $(1, 1, 0, 0)$ tiene diferencia de salida $(1, 0, 1, 0)$ entonces $\partial q = 0$. A continuación se probará que

$$P(\partial q = 0 | \partial f = 1, \delta h = 0) = 2^{-15}. \tag{2.3.10}$$

Para demostrar esto se requieren los siguientes resultados

$$\delta h = 0 \Leftrightarrow h = (H, 10 \ldots 0, \phi) \text{ y } h^* = (H^c, 10 \ldots 0, \phi^c)$$

donde H es un número de $16 - (l + 1)$ bits, $l \in \{0, 1, \ldots, 15\}$ y $\phi \in \{0, 1\}$, de este modo hay $l - 1$ ceros antes de ϕ

$$y$$

$$\partial f = 1 \Leftrightarrow f = (F, 10 \ldots 0) \text{ y } f^* = (F, 01 \ldots 1)$$

con F un número de $16 - (l + 1)$ bits y $l \in \{0, 1, \ldots, 16\}$ para $l < 15$.
Notese que $h \oplus h^* = (1 \ldots 1, 0 \ldots 0, 1)$ y que $f \oplus f^* = (0 \ldots 0, 1 \ldots 1)$. Además por el esquema del cifrado se tiene que

$$q = q^* \Leftrightarrow f \oplus h = f^* \oplus h^* \Leftrightarrow f \oplus f^* = h \oplus h^*,$$

de tal manera que $h \oplus h^* = f \oplus f^* = (0 \ldots 0, 1)$. Usando el programa que aparece en el apéndice $A.5$ se encontró que hay sólo cuatro de tales h: $\{0, 1, 2^{15}, 2^{15} + 1\}$ y 2^{15} de números f cuyo bit menos significativo es 1. Esto es, de los 2^{32} posibles valores para f, h, hay exactamente 2^{17} elecciones que dan como resultado $\partial q = 0$, de este modo se cumple (2.3.10). Ahora si se tiene que $h \in \{0, 1, 2^{15}, 2^{15} + 1\}$, $f \in \{1, 3, 5, \ldots, 2^{16} - 1\}$, $\partial f = 1$, $\delta h = 0$ y $u = u^*$,

$$\partial x = 1 \text{ y } \delta y = 0 \Leftrightarrow u \in \{0, 2^{15}\}.$$

Si se supone cierto que $\partial x = 1$ y $\delta y = 0$ usando de nueva cuenta el programa del apéndice $A.5$ se tiene que $u \in \{0, 2^{15}\}$. Si $u = 0$ entonces $\partial x = (u \oplus f) \boxminus (u \oplus f^*) = f \boxminus f^* = 1$ y $\delta y = (u \oplus h) \odot (u \oplus h^*)^{-1} = h \odot (h^*)^{-1} = 0$. Ahora el entero $2^{15} = 10 \ldots 0$ tiene 15 ceros

después del 1, luego como $f = (F, 10\ldots0)$ y $f^* = (F, 01\ldots1)$, son de esta forma entonces u sólo afecta el último dígito de f y f^* pero ambos tienen el mismo valor para este bit, entonces u no cambia la diferencia entre f y f^*, de esto se sigue que $\partial x = 1$. Para encontrar el valor de δy, por (2.3.2) basta con demostrar que $y+y^* = 1 \pmod{2^{16}}$. Ahora $y+y^* = (u\oplus h)+(u\oplus h^*)$, $h = (H, 10\ldots0, \phi)$ y $h^* = (H^c, 10\ldots0, \phi^c)$ por lo que u afecta únicamente a el último bit de H y H^c, de tal modo que la suma entre h y h^* no cambia. Pero por hipótesis $h + h^* = 1$ pues $\delta h = 0$, de este modo $\delta y = 0$.

Por lo tanto se ha demostrado que, para e, f, g, h y k_5, k_6 elegidas uniformemente e independientes, si la entrada diferencia es $(1, 1, 0, 0)$ tiene como salida diferencia $(1, 0, 1, 0)$ entonces

$$h \in \{0, 1, 2^{15}, 2^{15}+1\}, f \in \{1, 3, \ldots, 2^{16}-1\} \text{ y } u \in \{0, 2^{15}\}.$$

Nótese que en el evento anterior u toma valores en $\mathbb{F}_2^{16}$ uniformemente e independiente de f y h, de tal manera que

$$P((Y, Y^*) \in \Delta(1,0,1,0) | (X, X^*) \in \Delta(1,1,0,0)) = \left(\frac{2^{15}}{2^{16}}\right)\left(\frac{2^2}{2^{16}}\right)\left(\frac{2^1}{2^{16}}\right) = 2^{-1}2^{-14}2^{-15} = 2^{-30}.$$

□

Se debe notar que la probabilidad de estas diferenciales condicionadas a un valor especifico de subllave $(k_1, k_2, k_3, k_4, k_5, k_6)$ es invariante a la elección del valor de la subllave.

Finalmente se puede probar que IDEA es resistente al criptoanálisis diferencial pues su matriz de transición es no simétrica.

Proposición 2.3.4. *La matriz de transición de IDEA es no simétrica pues*

$$P((Y, Y^*) \in \Delta(1,0,1,0) | (X, X^*) \in \Delta(1,1,0,0)) = 2^{-30},$$

y

$$P((Y, Y^*) \in \Delta(1,1,0,0) | (X, X^*) \in \Delta(1,0,1,0)) = 0.$$

Demostración. Ya se demostró que se cumple la primera igualdad, así que sólo se probará la segunda. Como $\delta e = 1 = \delta v$ entonces $\delta v = (e \oplus t) \odot (e^* \oplus t^*)^{-1} = 1$ luego $e \oplus t = e^* \oplus t^*$, de esto se sigue que $e \oplus e^* = t \oplus t^*$ y $t = t^*$, por lo tanto $\delta t = 1$. Por otro lado $\partial f = \partial x = 0$ por lo que $u = u^*$ pues $\delta x = (f \oplus u) \boxminus (f^* \oplus u^*)$ luego $f \oplus f^* = u \oplus u^*$ pero $f = f^*$ y así $u = u^*$ y $\partial u = 0$. De estas implicaciones se tiene que $1 = \delta t = (s \odot Z_6) \odot (s^* \odot Z_6)^{-1}) = s \odot (s^*)^{-1} = \delta s$ y que $\partial u = (r \boxplus t) \boxminus (r^* \boxplus t^*) = 0$ de ahí que $r \boxplus t = r^* \boxplus t^*$, por lo tanto $r = r^*$. Así se tiene que $\delta p = \delta r = 1$ y $\partial q = (r \boxplus s) \boxminus (r^* \boxplus s^*) = (r \boxminus r^*) \boxplus (s \boxminus s^*) = 0$. Pero por la diferencia de entrada se tiene que $\delta e = 1$ y $\partial g = 1$, lo que implica que $p \neq p^*$ pues aunque $e = e^*$, $g \neq g^*$ y por lo tanto $p = e \oplus g \neq e \oplus g^* = p^*$. Luego la matriz de transición es no simétrica. □

Una vez estudiado el criptoanálisis diferencial y sus efectos sobre el sistema PES, el siguiente paso es hacer una implementación computacional del ataque, la cual se explica en el siguiente capítulo.

Capítulo 3

Implementación de un ataque por criptoanálisis diferencial del sistema PES

Como parte del estudio del criptoanálisis diferencial se propuso implementar dicho ataque al sistema PES, el cual no ha arrojado resultados favorables hasta el momento para PES de $2\frac{1}{2}$ rondas. La implementación que se realizó fue para encontrar las últimas cuatro llaves del cifrado, basándose en la propuesta por Massey y Lai. Primero se aplicó para el cifrado mini-PES, en el cual el ataque fue exitoso, y se pudo recuperar la llave en su totalidad. Posteriormente se intentó aplicar el mismo algoritmo para dos rondas y media de PES de 64 bits, pero fue infructuoso. Esto debido a que obtener las tercias propuestas en el criptoanálisis, para luego usarlas con el fin de recuperar las llaves de cifrado tiene un costo elevado computacionalmente. Al notar esta deficiencia en el programa se obtuvieron las tercias por medio de su forma combinatoria. A pesar de que esto redujo notablemente los tiempos de cálculo no se registraron resultados favorables para dos rondas y media. Por lo tanto cualquier intento de llevar el ataque a más rondas no será exitoso por el momento.

A continuación se exponen brevemente la manera en la que se hizo la implementación.

3.1. Ataque por criptoanálisis diferencial a miniPES

En el caso del miniPES se buscaron las tercias t_1^8, x_1^8, v_1^8 con tres iteraciones como se muestra en el algoritmo 1. Estas iteraciones son costosas computacionalmente pero al trabajar con elementos de 4 bits, tales operaciones se pueden realizar relativamente rápido en cualquier computadora.

Para las tercias u_1^8, w_1^8, y_1^8 se necesitaron las siguientes 4 iteraciones, pues para calcular las u_2 que son útiles en el ataque se requieren los valores de h_1 y h_2, como se describe en el algoritmo 2.

Algoritmo 1 Encuentra las tercias t_1^8, x_1^8, v_1^8

Salida: Los arreglos $X1, V1, T1$ con las tercias requeridas

```
contador ← 0
para x1 = 0 hasta x1 < 2^4 hacer
  para v1 = 0 hasta v1 < 2^4 hacer
    para t1 = 0 hasta t1 < 2^4 hacer
      x2 ← (1 − x1) (mód 2^4)
      v2 ← v1
      t2 ← x1 xor x2 xor t1
      e1 ← v1 xor t1
      e2 ← v2 xor t2
      difft ← (t1 + t2) (mód 2^4)
      diffe ← (e1 + e2) (mód 2^4)
      si difft = 1 y diffe = 1 entonces
        X1[contador] ← x2
        V1[contador] ← v2
        T1[contador] ← t2
        contador ← contador + 1
      fin si
    fin para
  fin para
fin para
```

Una vez que se tienen las tercias se crea una llave aleatoria. Y los siguientes pasos se repetirán hasta encontrar las llaves correctas:

1. Son inicializados los arreglos que servirán para contar las llaves exitosas durante el ataque.

2. Se calculan las parejas de textos en claro con sus respectivas parejas textos cifrados que cumplen las diferencias necesarias.

3. Son realizados los cálculos para obtener las llaves candidatas y se buscan las que más repeticiones obtuvieron y se verifica si son las llaves buscadas.

Finalmente se buscan el resto de los bits de llave aleatoriamente y el ataque requiere alrededor de 8 minutos en ser exitoso, véase los algoritmos 3 y 4.

Algoritmo 2 Encuentra las tercias u_1^8, w_1^8, y_1^8

Salida: Los arreglos $Y1, W1, U1$ con las tercias requeridas

```
contador ← 0
para h1 = 0 hasta h1 < 2^4 hacer
  h2 ← h1 − 1
  H[h1] ← h2
fin para
para y1 = 0 hasta y1 < 2^4 hacer
  para w1 = 0 hasta w1 < 2^4 hacer
    para u1 = 0 hasta u1 < 2^4 hacer
      para h1 = 0 hasta h < 2^4 hacer
        y2 ← ((1 − y1) (mód 2^4)
        w2 ← (w1 + 1) (mód 2^4)
        u2 ← y1 xor y2 xor h1 xor H[h1]
        f1 ← u1 xor w1
        f2 ← u2 xor w2
        diffu ← (u1 + u2) (mód 2^4)
        diffF ← (f1 + f2) (mód 2^4)
        si diffF = 1 y diffu = 2 entonces
          Y1[contador] ← y2
          W1[contador] ← w2
          U1[contador] ← u2
          contador ← contador + 1
        fin si
      fin para
    fin para
  fin para
fin para
```

Algoritmo 3 Creación de textos cifrados con diferencia de entrada $(0, 0, 0, 1)$

Entrada: Llave de cifrado K, Función de cifrado PES
Salida: Arreglos de texto cifrados TC y $TC2$

$numClaro \leftarrow$ cantidad deseada de textos en claro
para $i = 0$ **hasta** $i < 2^4$ **hacer**
  para $j = 0$ **hasta** $j < 2^4$ **hacer**
    $llave1[i][j] \leftarrow 0$
    $llave2[i][j] \leftarrow 0$
  fin para
fin para
para $i = 0$ **hasta** $i = numClaro$ **hacer**
  // Arreglo aleatorio de textos en claro
  para $j = 0$ textbfhasta $j < 4$ **hacer**
    $TP1[i][j] \leftarrow nmeroaleatorio \pmod{2^4}$
  fin para
fin para
para $i = 0$ **hasta** $i = numClaro$ **hacer**
  // Se cifran los textos en claro
  para $j = 0$ textbfhasta $j < 4$ **hacer**
    $TC[i][j] \leftarrow PES(TP1[i][j], K)$
  fin para
fin para
para $i = 0$ **hasta** $i = numClaro$ **hacer**
  // Se crean textos en claro condiferencia de entrada es (0,0,0,1)
  $TP2[i][0] \leftarrow (1 - TP1[i][0]) \pmod{2^4}$
  $TP2[i][1] \leftarrow (1 - TP1[i][1]) \pmod{2^4}$
  $TP2[i][2] \leftarrow TP1[i][2]$
  $TP2[i][3] \leftarrow (TP1[i][3] + 1) \pmod{2^4}$
fin para
para $i = 0$ **hasta** $i = numClaro$ **hacer**
  // Se cifra el segundo arreglo de textos en claro
  para $j = 0$ textbfhasta $j < 4$ **hacer**
    $TC2[i][j] \leftarrow PES(TP2[i][j], K)$
  fin para
fin para

Algoritmo 4 Busqueda de las llaves originales

Entrada: Función *prod* para calcular producto módulo $2^4 + 1$, arreglos de textos cifrados TC y $TC2$, subllaves de cifrado de la última ronda $k9[0], k9[1], k9[2], k9[3]$

Salida: subllaves originales $maxllave1, maxllave2, maxllave3, maxllave4$

mientras $k9[0] \neq maxllave1$ **o** $k9[2] \neq maxllave2$ **o** $k9[1] \neq maxllave3$ **o** $k9[3] \neq maxllave4$ **hacer**
  para $i = 0$ **hasta** $i = numClaro$ **hacer**
    para $j = 0$ **hasta** $j < 2^4$ **hacer**
      $z1 \leftarrow prod(TC1[i][0], aee[X1[j]])$
      $z2 \leftarrow (TC1[i][2] - V1[j]) \pmod{2^4}$
      $llave1[z1][z2] \leftarrow llave1[z1][z2] + 1$
    fin para
  fin para
  para $i = 0$ **hasta** $i = numClaro$ **hacer**
    para $j = 0$ **hasta** $j < 2^4$ **hacer**
      $z3 \leftarrow prod(TC1[i][1], aee[Y1[j]])$
      $z4 \leftarrow (TC1[i][3] - W1[j]) \pmod{2^4}$
      $llave2[z3][z4] \leftarrow llave2[z3][z4] + 1$
    fin para
  fin para
  $max \leftarrow -1$
  $maxllave1 \leftarrow 0$
  $maxllave2 \leftarrow 0$
  para $i = 0$ **hasta** $i = numClaro$ **hacer**
    para $i = 0$ **hasta** $i = numClaro$ **hacer**
      si llave1[i][j]>max **entonces**
        $max \leftarrow llave1[i][j]$
        $maxllave1 \leftarrow i$
        $maxllave2 \leftarrow j$
      fin si
    fin para
  fin para
  $max \leftarrow -1$
  $maxllave3 \leftarrow 0$
  $maxllave4 \leftarrow 0$
  para $i = 0$ **hasta** $i = numClaro$ **hacer**
    para $i = 0$ **hasta** $i = numClaro$ **hacer**
      si llave1[i][j]>max **entonces**
        $max \leftarrow llave2[i][j]$
        $maxllave3 \leftarrow i$
        $maxllave4 \leftarrow j$
      fin si
    fin para
  fin para
fin mientras

3.2. Ataque por criptoanálisis diferencial a $2\frac{1}{2}$ rondas de PES

Como se mencionó al inicio de este capítulo el ataque utilizado para el sistema miniPES no se puede implementar para el sistema completo de PES debido a que el tamaño de los arreglos estáticos de C no tienen la capacidad suficiente para guardar las tercias t_1^8, x_1^8, v_1^8, así como para las tercias u_1^8, w_1^8, y_1^8 debido a la cantidad de elementos en $\mathbb{Z}_{2^{16}}$ y $\mathbb{Z}_{2^{16}+1}$. Por lo tanto se requirió el uso de arreglos dinámicos en C, con el fin de evitar desbordamientos de memoria y acelerar el procedimiento. Además las tercias t_1^8, x_1^8, v_1^8 se calcularon de manera combinatoria pues recordemos que son de la forma $t_1 = (\gamma, 10\ldots0, \rho)$, $v_1 = (\alpha \oplus \gamma, 00\ldots0, \theta \oplus \rho)$ y $x_1 = (\beta \oplus \gamma, 10\ldots0, \phi \oplus \rho)$, véase el algoritmos 5 y 6.

Algoritmo 5 Genera las parejas (x, t)

Salida: Parejas (x, t)

$maskB \leftarrow 0x8000$ // El número binario 1000000000000000 el cual se irá recorriendo para obtener los numeros requeridos
$xs[1] \leftarrow$ apuntador a entero de tamaño 2^2 el tamaño de un entero.
$xs[1][0] \leftarrow 0$
$xs[1][1] \leftarrow 1$
$xs[1][2] \leftarrow maskB$
para $i = 2$ **hasta** $i = lim$ **hacer**
  // lim puede llegar hasta 16 pues recordemos que estamos trabajando con números de 16 bits
  $xs[i] \leftarrow$ apuntador a entero de tamaño 2^i el tamaño de un entero.
  $base \leftarrow corrimientode(i-1)bitsaladerechademaskB$
  para $j = 0$ **hasta** $j < 2^{i-1}$ **hacer**
    $xs[i][2*j] \leftarrow (corrimientode(17-i)bitsdej)$ **xor** $base$ // De este parte del arreglo salen las x
    $xs[i][2*(j+1)] \leftarrow ((corrimientode(17-i)bitsdej)$ **xor** $base) + 1$ // De este parte del arreglo salen las t
  fin para
fin para

En el caso de las tercias u_1^8, w_1^8, y_1^8, primero se calcularon todas las de 4 bits que satisfacen las ecuaciones (2.3.5) y (2.3.6), aunque en el ataque sólo se utilizan las tercias que cumplen que $\phi_1 - \phi_2 = 1$, $(\theta_1 \oplus \rho_1) + (\theta_2 \oplus \rho_2) = -1$, como puede verse en el algoritmo 7. Una vez calculadas estas tercias de 4 bits es fácil aunque costoso generar u_1^8, w_1^8, y_1^8 de 16 bits, concatenando ordenadamente una cadena 12 bits con los números de 4 bits obtenidos anteriormente.

Ahora para recuperar la llave se ha intentado utilizar el algoritmo que se usó para el miniPES y algunos métodos similares pero no se ha logrado obtener resultados favorables. Esto puede deberse a que el proceso de conteo de las llaves candidatas sea incorrecto o porque el poder de cómputo disponible es insuficiente para obtener dichas llaves. Además la naturaleza estocástica del problema dificulta obtener resultados computacionales en tiempo

razonable. Dado que la probabilidad de éxito es considerablemente baja es de esperarse que el programa necesite una gran cantidad de corridas antes de encontrar su objetivo.

Algoritmo 6 Genera las tercias t, x, v

Entrada: Algoritmo 5
Salida: Tercias t, x, v

```
para j = 0 hasta j < 4 hacer
  para k = 0 hasta k < 4 hacer
    x_1 ← xs[1][j]
    t_1 ← xs[1][k]
    v_1 ← xs[1][j] xor xs[1][k]
  fin para
fin para
para i = 2 hasta i = lim hacer
  para j = 0 hasta j < 4 hacer
    para k = 0 hasta k < 4 hacer
      x_i ← xs[i][j]
      t_i ← xs[i][k]
      v_i ← xs[i][j] xor xs[i][k]
    fin para
  fin para
fin para
```

Con esto se concluye el estudio del sistema PES y en el siguiente capítulo se presentan las modificaciones hechas al sistema IDEA, así como el estudio de la seguridad contra el cripotanálisis diferencial de dichas modificaciones.

Algoritmo 7 Genera las tercias (u, y, w)

Entrada: Matriz M véase (2.3.7) , Estructura Q que contiene 4 elementos enteros digamos theta, rho, phi1 y phi2

Salida:

para $j = 0$ **hasta** $j < 8$ **hacer**
  para $i = 0$ **hasta** $i < 8$ **hacer**
    si $M[j][i] \neq 0$ **entonces**
      $Caja[j][i] \leftarrow$ apuntador a Q de tamaño de $M[i][j] * Q$ // Crea una caja tridimensional con entradas de tipo Q
    si no
      $Caja[j][i] \leftarrow Nulo$
    fin si
  fin para
fin para
$S[8] = [-1, 1, -3, 3, -5, 5, -7, 7]$
para $j = 0$ **hasta** $j < 8$ **hacer**
  para $i = 0$ **hasta** $i < 8$ **hacer**
    $contador \leftarrow 0$
    si $S[i] < 0$ **entonces**
      $inicio \leftarrow 0$
      $fin \leftarrow 16 + S[i]$
    si no
      $inicio \leftarrow 0 + S[i]$
      $fin \leftarrow 16$
    fin si
  fin para
fin para

Algoritmo 8 Continuación algoritmo 7

$t2[16] \leftarrow [17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2]$

$r2[16] \leftarrow [18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3]$

$$p[16][8] \leftarrow \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 0 \\ 15 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 \\ 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 0 & 1 & 2 \\ 13 & 14 & 15 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 0 & 1 & 2 & 3 & 4 \\ 11 & 12 & 13 & 14 & 15 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 9 & 10 & 11 & 12 & 13 & 14 & 15 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{pmatrix}$$

para $t1 = 2$ **hasta** $t1 < 16$ **hacer**
  para $t1 = 3$ **hasta** $t1 < 16$ **hacer**
    para $p1 = Inicio$ **hasta** $pi = fin$ **hacer**
      si ($t1$ **xor** $p1$) + ($t2[t1]$ **xor** $p2[i][p1]$) = 16 **entonces**
        si ($p1$ **xor** $r1$) + (p2[i][p1] **xor** r2[r1]) = 17 **entonces**
          si ($t1$ **xor** $r1$) + ($t2[t1]$ **xor** $r2[r1]$) = $S[j]$ **entonces**
            $Caja[j][i][contador].theta \leftarrow t1$
            $Caja[j][i][contador].rho \leftarrow r1$
            $Caja[j][i][contador].phi1 \leftarrow p1$
            $Caja[j][i][contador].phi2 \leftarrow p2[i][p1]$
            $contador \leftarrow contador + 1$
          fin si
        fin si
      fin si
    fin para
  fin para
fin para

Capítulo 4

Modificaciones a IDEA

El principal objetivo de este trabajo de tesis es presentar algunas modificaciones al sistema IDEA con el objetivo de mejorar su funcionamiento y mantener la seguridad del cifrado. En las modificaciones realizadas no se cambió el esquema general de IDEA, pero los grupos usados en la suma $\boxplus$ y $\odot$ sí fueron sustituidos con el objetivo de reducir la complejidad de dichas operaciones. Estas modificaciones a IDEA se presentan en las siguientes secciones.

4.1. Modificación 1

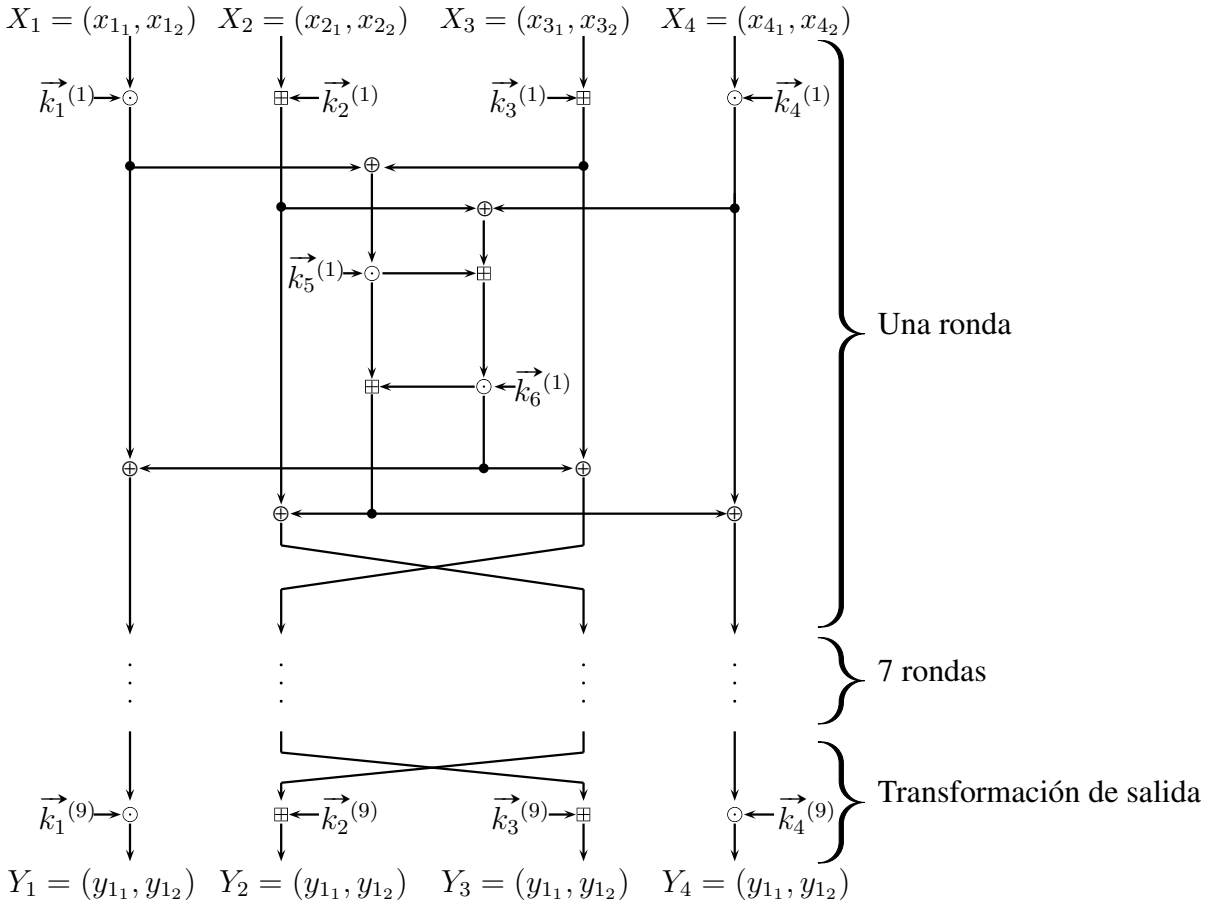
En este sistema los bloques de entrada y salida de 64 bits en lugar de ser divididos en 4 subbloques de 16 bits se dividen en 4 parejas ordenadas de 16 bits con cada entrada de 8 bits, véase la Figura 4.1. Se mantienen las 8 rondas y media del cifrado original y su estructura. Pero se usan las siguientes operaciones:

1. La suma X-OR de dos parejas ordenadas de 16 bits entrada a entrada, denotada $\oplus$
2. La suma de enteros módulo 2^8 de dos parejas ordenadas, la operación se denota como $\boxplus$
3. La multiplicación de enteros módulo $2^8 + 1$ de dos pares ordenados, donde la entrada compuesta sólo de ceros se trata como el entero 2^8. La operación se denota como $\odot$

4.2. Modificación 2

En este sistema los bloques de entrada y salida de 64 bits en lugar de ser divididos en 4 subbloques de 16 bits se dividen en 8 cuartetas ordenas de 16 bits con cada entrada de 4 bits véase la Figura 4.1. Se mantienen las 8 rondas y media del cifrado original y su estructura. Pero se usan las siguientes operaciones:

1. La suma X-OR de cuartetas ordenadas de 16 bits entrada a entrada, denotada $\oplus$
2. La suma de enteros módulo 2^4 de dos cuartetas ordenadas, la operación se denota como $\boxplus$



X_i: Subbloque de 16-bits de texto en claro
Y_i: Subbloque de 16 bits de texto cifrado
x_{i_j}: Entrada de 8 bits de un subbloque de 16-bits de texto en claro
y_{i_j}: Entrada de 8 bits de un subbloque de 16-bits de texto cifrado
$\overrightarrow{k_i}^{(r)}$: Subbloque de llave de 16-bits visto como vector de $\mathbb{Z}_{2^8}^2$
$\oplus$: Suma bit a bit módulo 2 de subbloques de 16 bits
$\boxplus$: Suma módulo 2^8 de dos parejas de 16 bits
$\odot$: Producto módulo $2^8 + 1$ de pares ordenados de 16 bits con el subbloque cero correspondiendo a 2^8

Figura 4.1: Proceso de cifrado de la modificación 1 del sistema IDEA

3. La multiplicación de enteros módulo 2^4+1 de dos cuartetas ordenadas, donde la entrada compuesta sólo de ceros se trata como el entero 2^4. La operación se denota como $\odot$

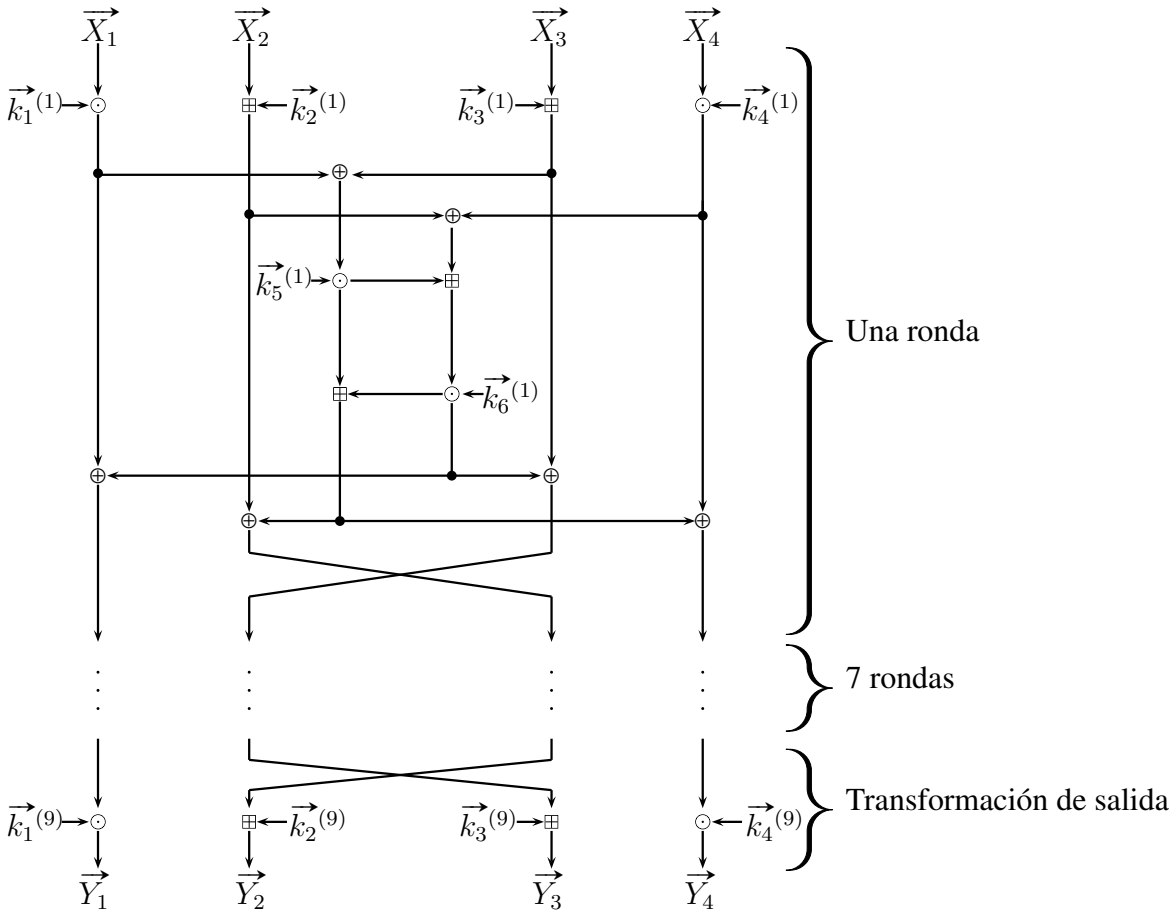
4.3. Resultados

Para probar estas nuevas versiones de IDEA se realizaron programas para los cifrados IDEA, IDEA modificado 1 e IDEA modificado 2 en lenguaje C. Además se comparó la velocidad con la que corren IDEA y sus modificaciones. Se realizaron dos pruebas, en la primera la llave de cada cifrado fue renovada en cada iteración, en la segunda la llave se precalculó y se utilizó la misma llave en cada iteración. Los resultados, en segundos, se muestran a continuación

Prueba 1		
IDEA	Modificación 1	Modificación 2
0.000130	0.000145	0.000158
0.000050	0.000057	0.000073
0.000062	0.000076	0.000087
0.000051	0.000057	0.000076
0.000049	0.000054	0.000082
0.000064	0.000064	0.000071
0.000050	0.000061	0.000076
0.000054	0.000064	0.000069
0.000049	0.000054	0.000068
0.000049	0.000055	0.000066

Prueba 2		
IDEA	Modificación 1	Modificación 2
0.000077	0.000086	0.000084
0.000005	0.000008	0.000014
0.000006	0.000008	0.000013
0.000005	0.000008	0.000013
0.000004	0.000008	0.000013
0.000005	0.000007	0.000011
0.000005	0.000008	0.000013
0.000005	0.000008	0.000013
0.000005	0.000006	0.000011
0.000005	0.000008	0.000012

Como puede apreciarse en las tablas si la llave se renueva en cada sesión es conveniente trabajar con el sistema original de IDEA. En el segundo caso no hay una gran diferencia entre la modificación 1 e IDEA original aunque este último sigue siendo más rápido. Es probable que esto se deba a que al calcular una mayor cantidad de operaciones en la modificaciones de IDEA aunque modularmente sean menos complejas estas hagan a al cifrado más costos en términos computacionales.



$\overrightarrow{X_i}$: Subbloque de 16-bits de texto en claro visto como vector en $\mathbb{Z}_{2^4}^4$

$\overrightarrow{Y_i}$: Subbloque de 16-bits de texto cifrado visto como vector en $\mathbb{Z}_{2^4}^4$

$\overrightarrow{k_i}^{(r)}$: Subbloque de llave de 16-bits visto como vector en $\mathbb{Z}_{2^4}^4$

$\oplus$: Suma bit a bit módulo 2 de subbloques de 16 bits

$\boxplus$: Suma módulo 2^4 de dos parejas de 16 bits

$\odot$: Producto módulo $2^4 + 1$ de pares ordenados de 16 bits con el subbloque cero correspondiendo a 2^4

Figura 4.2: Proceso de cifrado de la modificación 2 del sistema IDEA

4.4. Criptoananálisis diferencial

En esta sección se muestra que las modificaciones de IDEA al mantener la estructura del cifrado de IDEA, también mantienen la seguridad en contra del criptonanálisis diferencial. Estos ataques son similares al que se usó con IDEA.

Empezaremos con la modificación 1, recordando que

$$\bar{a} = (a_1, a_2), \text{tal que } a_1 \text{ y } a_2 \text{ son de 8 bits}$$

y definiendo las diferencias de las distintas operaciones en el cifrado, como:

$$\delta\bar{x} = \bar{x} \odot (\bar{x}^*)^{-1} = (x_1, x_2) \odot (x_1^*, x_2^*)^{-1} = (x_1 \odot (x_1^*)^{-1}, x_2 \odot (x_2^*)^{-1})$$

y

$$\partial\bar{x} = \bar{x} \boxminus \bar{x}^* = (x_1, x_2) \boxminus (x_1^*, x_2^*) = (x_1 \boxminus x_1^*, x_2 \boxminus x_2^*).$$

De este modo las relaciones entre las operaciones del cifrado que se encontraron para la modificación 1 de IDEA quedan de la siguiente manera:

Proposición 4.4.1. *Si la función dada por la estructura MA se escribe como*

$$(\bar{t}, \bar{u}) = MA(\bar{p}, \bar{q}; \bar{k}_5, \bar{k}_6) \tag{4.4.1}$$

entonces para cada elección de llave $(\bar{k}_5, \bar{k}_6)$, las entradas $(\bar{p}, \bar{q})$, $(\bar{p}^*, \bar{q}^*)$ y las salidas $(\bar{t}, \bar{u})$, $(\bar{t}^*, \bar{u}^*)$ de la caja MA para la misma llave $(\bar{k}_5, \bar{k}_6)$ satisfacen la siguiente relación:

$$\delta\bar{p} = \bar{1} = (1,1), \partial\bar{q} = \bar{0} = (0,0) \Leftrightarrow \delta\bar{t} = \bar{1} = (1,1), \partial\bar{u} = \bar{0} = (0,0) \tag{4.4.2}$$

Demostración. Supondremos primero que $\delta\bar{p} = \bar{1}$ y $\partial\bar{q} = \bar{0}$ entonces se tiene que

$$\delta\bar{r} = (\bar{p} \odot \bar{k}_5) \odot (\bar{p}^* \odot \bar{k}_5)^{-1} = \delta\bar{p} = \bar{1}$$

y

$$\partial\bar{s} = (\bar{q} \boxplus \bar{r}) \boxminus (\bar{q}^* \boxplus \bar{r}^*) = \bar{r} \boxminus \bar{r}^*$$

pero como $\delta\bar{r} = \bar{1}$ entonces $\bar{r} = \bar{r}^*$ y así $\partial\bar{s} = \bar{0}$. La siguiente relación en la caja MA nos dice que

$$\delta\bar{t} = (\bar{s} \odot \bar{k}_6) \odot (\bar{s}^* \odot \bar{k}_6)^{-1} = \bar{s} \odot (\bar{s}^*)^{-1}$$

pero $\partial\bar{s} = \bar{s} \boxminus \bar{s}^* = \bar{0}$, luego $\delta\bar{t} = \bar{s} \odot (\bar{s}^*)^{-1} = \bar{1}$, y $\bar{s} = \bar{s}^*$. Finalmente

$$\partial\bar{u} = (\bar{t} \boxplus \bar{r}) \boxminus (\bar{t}^* \boxplus \bar{r}^*) = (\bar{t} \boxminus \bar{t}^*) \boxplus (\bar{r} \boxminus \bar{r}^*) = \bar{0},$$

esta última igualdad se da porque $\delta\bar{t} = \bar{t} \odot (\bar{t}^*)^{-1} = \bar{1}$.

Para la otra implicación se tiene que $\bar{t} = \bar{t}^*$ y $\bar{u} = \bar{u}^*$ pues $\delta\bar{t} = \bar{1}$ y $\partial\bar{u} = \bar{0}$. Luego como

$$\partial\bar{u} = (\bar{t} \boxplus \bar{r}) \boxminus (\bar{t}^* \boxplus \bar{r}^*) = (\bar{t} \boxminus \bar{t}^*) \boxplus (\bar{r} \boxminus \bar{r}^*) = 0$$

y

$$\delta\bar{t} = (\bar{s} \odot \bar{k}_6) \odot (\bar{s}^* \odot \bar{k}_6)^{-1} = \bar{s} \odot (\bar{s}^*)^{-1} = \bar{1}$$

entonces $\bar{r} = \bar{r}^*$ y $\bar{s} = \bar{s}^*$, de este modo $\delta\bar{r} = \bar{1}$ y $\partial\bar{s} = \bar{0}$. Ahora $\partial\bar{q} = \bar{0}$ porque $\partial\bar{s} = (\bar{q} \boxplus \bar{r}) \boxminus (\bar{q}^* \boxplus \bar{r}^*) = \bar{q} \boxminus \bar{q}^* = \partial\bar{q} = \bar{0}$. Por último se tiene que

$$\delta\bar{r} = (\bar{p} \odot \bar{k}_5) \odot (\bar{p}^* \odot \bar{k}_5)^{-1} = \bar{p} \odot (\bar{p}^*)^{-1} = \delta\bar{p} = \bar{1},$$

de este modo $\delta\bar{p} = \bar{1}$ y $\partial\bar{q} = \bar{0}$. □

Ahora para mostrar la resistencia de la primera modificación de IDEA en contra del criptoanálisis diferencial es suficiente con probar que su matriz de transición no es simétrica, de este modo sólo calcularemos la probabilidad de la diferencial más sencilla y en base a ella veremos que la matriz es no simétrica.

Proposición 4.4.2. *Para IDEA modificado 1 se cumple que la diferencia de entrada* $(\bar{1}, \bar{1}, \bar{0}, \bar{0})$ *tiene diferencia de salida* $(\bar{1}, \bar{0}, \bar{1}, \bar{0})$ *con probabilidad*

$$P((\bar{Y}, \bar{Y}^*) \in \Delta(\bar{1}, \bar{0}, \bar{1}, \bar{0}) | (\bar{X}, \bar{X}^*) \in \Delta(\bar{1}, \bar{1}, \bar{0}, \bar{0})) = 2^{-28}.$$

Demostración. Suponga que se tienen un par de entradas $(\bar{a}, \bar{b}, \bar{c}, \bar{d})$ y $(\bar{a}^*, \bar{b}^*, \bar{c}^*, \bar{d}^*)$ con diferencia

$$(\delta\bar{a}, \partial\bar{b}, \partial\bar{c}, \delta\bar{d}) = (\bar{1}, \bar{1}, \bar{0}, \bar{0}).$$

Luego

$$(\delta\bar{e}, \partial\bar{f}, \partial\bar{g}, \delta\bar{h}) = (\bar{1}, \bar{1}, \bar{0}, \bar{0}).$$

Ahora como $\delta\bar{e} = \bar{1}$ y $\partial\bar{g} = \bar{0}$ se tiene que $\bar{e} = \bar{e}^*$ y $\bar{g} = \bar{g}^*$ pues recordemos que $\delta\bar{e} = \bar{e} \odot (\bar{e}^*)^{-1}$ y $\partial\bar{g} = \bar{g} \boxminus \bar{g}^*$. Del siguiente paso del cifrado se tiene que $\bar{p} = \bar{e} \oplus \bar{g}$ y $\bar{p}^* = \bar{e}^* \oplus \bar{g}^* = \bar{e} \oplus \bar{g}$ luego $\bar{p} = \bar{p}^*$ y $\bar{r} = \bar{r}^*$ pues $\delta\bar{r} = \delta\bar{p}$. Por otra parte como $\delta\bar{v} = \bar{1}$ se sigue que $\bar{t} = \bar{t}^*$ porque $\delta\bar{v} = (\bar{e} \oplus \bar{t}) \odot (\bar{e}^* \oplus \bar{t}^*) = \bar{1}$ luego, $(\bar{e} \oplus \bar{t}) = (\bar{e}^* \oplus \bar{t}^*)$ pero $\bar{e} = \bar{e}^*$ por lo tanto $\bar{t} = \bar{t}^*$. Además como $\partial\bar{u} = (\bar{r} \boxplus \bar{t}) \boxminus (\bar{r}^* \boxplus \bar{t}^*)$, $\bar{r} = \bar{r}^*$ y $\bar{t} = \bar{t}^*$ entonces $\partial\bar{u} = \bar{0}$ y $\bar{u} = \bar{u}^*$. Luego se tiene que $\partial\bar{q} = \bar{0}$. Así si la diferencia de la entrada $(\bar{1}, \bar{1}, \bar{0}, \bar{0})$ tiene diferencia de salida $(\bar{1}, \bar{0}, \bar{1}, \bar{0})$ entonces $\partial\bar{q} = \bar{0}$. A continuación se probará que

$$P(\partial\bar{q} = 0 | \partial\bar{f} = \bar{1}, \delta\bar{h} = \bar{0}) = 2^{-14}. \tag{4.4.3}$$

Para demostrar esto se requieren los siguientes resultados

$$\delta\bar{h} = \bar{0} \Leftrightarrow \bar{h} = ((H_1, 10\ldots0, \phi_1), (H_2, 10\ldots0, \phi_2)) \text{ y } \bar{h}^* = ((H_1^c, 10\ldots0, \phi_1^c), (H_1^c, 10\ldots0, \phi_1^c))$$

donde H_i, H_i^c son números de $8-(l+1)$ bits, $l \in \{0, 1, \ldots, 7\}$ y $\phi_i, \phi_i^c \in \{0, 1\}$, de este modo hay $l-1$ ceros antes de ϕ_i, ϕ_i^c

y

$$\partial\bar{f} = \bar{1} \Leftrightarrow \bar{f} = ((F_1, 10\ldots0), (F_2, 10\ldots0)) \text{ y } \bar{f}^* = ((F_1, 01\ldots1), (F_2, 01\ldots1))$$

con F_i, $i \in \{1, 2\}$ de $8-(l+1)$ bits y $l \in \{0, 1, \ldots, 8\}$ para $l < 7$.

Nótese que $\bar{h}\oplus\bar{h}^* = (1\ldots1,0\ldots0,1)$ y que $\bar{f}\oplus\bar{f}^* = (0\ldots0,1\ldots1)$. Además por el esquema del cifrado se tiene que

$$\bar{q} = \bar{q}^* \Leftrightarrow \bar{f}\oplus\bar{h} = \bar{f}^*\oplus\bar{h}^* \Leftrightarrow \bar{f}\oplus\bar{f}^* = \bar{h}\oplus\bar{h}^*,$$

de tal manera que $\bar{h}\oplus\bar{h}^* = \bar{f}\oplus\bar{f}^* = (0\ldots0,1)$. Usando el programa que se encuentra en el apéndice $A.6$ se encontró que hay 16 de tales valores de $\bar{h}$: $\{0,1,2^7,2^7+1\}^2$ y 2^{14} distintas $\bar{f}$ cuyo bit menos significativo es 1. Esto es, de los 2^{32} posibles valores para $\bar{f},\bar{h}$, hay exactamente 2^{18} elecciones que dan como resultado $\partial\bar{q}=\bar{0}$, de este modo se cumple (4.4.3). Ahora si se tiene que $\bar{h}\in\{0,1,2^7,2^7+1\}^2$, $\bar{f}\in\{1,3,5,\ldots,2^7-1\}^2$, $\partial\bar{f}=\bar{1}$, $\delta\bar{h}=\bar{0}$ y $\bar{u}=\bar{u}^*$,

$$\partial\bar{x}=\bar{1} \text{ y } \delta\bar{y}=\bar{0} \Leftrightarrow \bar{u}\in\{0,2^7\}^2.$$

Si se supone cierto que $\partial\bar{x}=\bar{1}$ y $\delta\bar{y}=\bar{0}$ usando de nueva cuenta el programa del apéndice $A.6$ se tiene que $\bar{u}\in\{0,2^7\}^2$. Si $\bar{u}=\bar{0}$ entonces

$$\partial\bar{x} = (\bar{u}\oplus\bar{f})\boxminus(\bar{u}\oplus\bar{f}^*) = \bar{f}\boxminus\bar{f}^* = (f_1\boxminus f_1^*, f_2\boxminus f_2^*) = \bar{1}$$

y

$$\delta\bar{y} = (\bar{u}\oplus\bar{h})\odot(\bar{u}\oplus\bar{h}^*)^{-1} = \bar{h}\odot(\bar{h}^*)^{-1} = (h_1\odot(h_1^*)^{-1}, h_2\odot(h_2^*)^{-1}) = \bar{0}.$$

Ahora el entero $2^7 = 10\ldots0$ tiene 7 ceros después del 1, luego como $f_i = (F,10\ldots0)$ y el vector $f_i^* = (F,01\ldots1)$, $i\in\{1,2\}$, tienen esta forma entonces u_i, $i\in\{1,2\}$ sólo afecta el último dígito de f y f^* pero ambos tienen el mismo valor para este bit entonces u_i no cambia la diferencia entre f_i y f_i^*, de esto se sigue que $\partial\bar{x}=\bar{1}$. Para encontrar el valor de δy, basta con demostrar que $\bar{y}+\bar{y}^*=\bar{1}$ (mód 2^8). Como $\bar{y}+\bar{y}^* = (\bar{u}\oplus\bar{h})+(\bar{u}\oplus\bar{h}^*)$, $h_i = (H,10\ldots0,\phi)$ y $h_i^* = (H^c,10\ldots0,\phi^c)$, $i\in\{1,2\}$, $\bar{u}$ sólo afecta el último bit de H_i y H_i^c, de tal modo que no afecta la suma entre h_i y h_i^*. Pero por hipótesis $\bar{h}+\bar{h}^*=\bar{1}$ pues $\delta\bar{h}=\bar{0}$, de este modo $\delta\bar{y}=\bar{0}$.

Así se ha demostrado que, para $\bar{e},\bar{f},\bar{g},\bar{h}$ y $\bar{k}_5,\bar{k}_6$ elegidas uniformemente e independientes, si la entrada diferencia es $(\bar{1},\bar{1},\bar{0},\bar{0})$ y tiene como salida la diferencia $(\bar{1},\bar{0},\bar{1},\bar{0})$ entonces

$$\bar{h}\in\{0,1,2^7,2^7+1\}^2, \bar{f}\in\{1,3,\ldots,2^7-1\}^2 \text{ y } \bar{u}\in\{0,2^7\}^2.$$

Nótese que en el evento anterior $\bar{u}$ toma valores en $\mathbb{F}_2^{16}$ uniformemente e independiente de $\bar{f}$ y $\bar{h}$, de tal manera que $P((\bar{Y},\bar{Y}^*)\in\Delta(\bar{1},\bar{0},\bar{1},\bar{0})|(\bar{X},\bar{X}^*)\in\Delta(\bar{1},\bar{1},\bar{0},\bar{0}))$ es igual a $\left(\frac{2^{14}}{2^{16}}\right)\left(\frac{2^4}{2^{16}}\right)\left(\frac{2^2}{2^{16}}\right) = 2^{-2}2^{-12}2^{-14} = 2^{-28}$. □

Se debe notar que la probabilidad de estas diferenciales condicionadas a un valor específico de subllave $(\bar{k}_1,\bar{k}_2,\bar{k}_3,\bar{k}_4,\bar{k}_5,\bar{k}_6)$ es invariante a la elección del valor de la subllave.

Finalmente se puede probar que la primera modificación de IDEA es resistente al criptoanálisis diferencial pues su matriz de transición no es simétrica.

Proposición 4.4.3. *La matriz de transición de la primera modificación de IDEA no es simétrica pues*

$$P((Y,Y^*)\in\Delta(1,0,1,0)|(X,X^*)\in\Delta(1,1,0,0)) = 2^{-30}$$

y

$$P((Y,Y^*) \in \Delta(1,1,0,0)|(X,X^*) \in \Delta(1,0,1,0)) = 0$$

Demostración. Ya se demostró que

$$P((\bar{Y},\bar{Y}^*) \in \Delta(\bar{1},\bar{0},\bar{1},\bar{0})|(\bar{X},\bar{X}^*) \in \Delta(\bar{1},\bar{1},\bar{0},\bar{0})) = 2^{-28}.$$

Por lo que falta probar que

$$P((\bar{Y},\bar{Y}^*) \in \Delta(\bar{1},\bar{1},\bar{0},\bar{0})|(\bar{X},\bar{X}^*) \in \Delta(\bar{1},\bar{0},\bar{1},\bar{0})) = 0.$$

Como $\delta\bar{e} = \bar{1} = \delta\bar{v}$ entonces $\delta\bar{v} = (\bar{e} \oplus \bar{t}) \odot (\bar{e}^* \oplus \bar{t}^*)^{-1} = \bar{1}$ luego $\bar{e} \oplus \bar{t} = \bar{e}^* \oplus \bar{t}^*$, de esto se sigue que $\bar{e} \oplus \bar{e}^* = \bar{t} \oplus \bar{t}^*$ y $\bar{t} = \bar{t}^*$, por lo tanto $\delta\bar{t} = \bar{1}$. Por otro lado $\partial\bar{f} = \partial\bar{x} = \bar{0}$ por lo que $\bar{u} = \bar{u}^*$ pues $\delta\bar{x} = (\bar{f} \oplus \bar{u}) \boxminus (\bar{f}^* \oplus \bar{u}^*)$ luego $\bar{f} \oplus \bar{f}^* = \bar{u} \oplus \bar{u}^*$ pero $\bar{f} = \bar{f}^*$ y así $\bar{u} = \bar{u}^*$ y $\partial\bar{u} = \bar{0}$. De estas implicaciones se tiene que $\bar{1} = \delta\bar{t} = (\bar{s} \odot \bar{k}_6) \odot (\bar{s}^* \odot \bar{k}_6)^{-1}) = \bar{s} \odot (\bar{s}^*)^{-1} = \delta\bar{s}$ y que $\partial\bar{u} = (\bar{r} \boxplus \bar{t}) \boxminus (\bar{r}^* \boxplus \bar{t}^*) = \bar{0}$ luego $\bar{r} \boxplus \bar{t} = \bar{r}^* \boxplus \bar{t}^*$, así $\bar{r} = \bar{r}^*$. De este modo se tiene que $\delta\bar{p} = \delta\bar{r} = \bar{1}$ y $\partial\bar{q} = (\bar{r} \boxplus \bar{s}) \boxminus (\bar{r}^* \boxplus \bar{s}^*) = (\bar{r} \boxminus \bar{r}^*) \boxplus (\bar{s} \boxminus \bar{s}^*) = \bar{0}$. Pero por la diferencia de entrada se tiene que $\delta\bar{e} = \bar{1}$ y $\partial\bar{g} = \bar{1}$, lo que implica que $\bar{p} \neq \bar{p}^*$ pues aunque $\bar{e} = \bar{e}^*$, $\bar{g} \neq \bar{g}^*$ que implica que $\bar{p} = \bar{e} \oplus \bar{g} \neq \bar{e} \oplus \bar{g}^* = \bar{p}^*$. Luego se cumple que

$$P((Y,Y^*) \in \Delta(1,1,0,0)|(X,X^*) \in \Delta(1,0,1,0)) = 0.$$

Se tiene así que la matriz de transición no es simétrica. □

En el caso de la modificación 2, se tiene que,

$$\bar{a} = (a_1, a_2, a_3, a_4), \text{tal que } a_i, i \in \{1,2,3,4\} \text{ es de 4 bits}$$

y las diferencias de las operaciones en el cifrado, se definen como $\delta\bar{x} = \bar{x} \odot (\bar{x}^*)^{-1}$ y $\partial\bar{x} = \bar{x} \boxminus \bar{x}^*$

$$\delta\bar{x} = (x_1,x_2,x_3,x_4) \odot (x_1^*,x_2^*,x_3^*,x_4^*)^{-1} = (x_1 \odot (x_1^*)^{-1}, x_2 \odot (x_2^*)^{-1}, x_3 \odot (x_3^*)^{-1}, x_4 \odot (x_4^*)^{-1})$$

y

$$\partial\bar{x} = (x_1,x_2,x_3,x_4) \boxminus (x_1^*,x_2^*,x_3^*,x_4^*) = (x_1 \boxminus x_1^*, x_2 \boxminus x_3^*, x_3 \boxminus x_3^*, x_4 \boxminus x_4^*)$$

Ahora las relaciones entre las operaciones del cifrado para la modificación 2 de IDEA, son muy parecidas a las de la modificación 1, son las siguientes:

Proposición 4.4.4. *Si la función dada por la estructura MA se escribe como*

$$(\bar{t},\bar{u}) = MA(\bar{p},\bar{q};\bar{k}_5,\bar{k}_6) \tag{4.4.4}$$

entonces para cada elección de llave $(\bar{z}_5,\bar{z}_6)$, las entradas $(\bar{p},\bar{q})$, $(\bar{p}^*,\bar{q}^*)$ y las salidas $(\bar{t},\bar{u})$, $(\bar{t}^*,\bar{u}^*)$ de la caja MA para la misma llave $(\bar{k}_5,\bar{k}_6)$ satisfacen la siguiente relación:

$$\delta\bar{p} = \bar{1} = (1,1,1,1), \partial\bar{q} = \bar{0} = (0,0,0,0) \Leftrightarrow \delta\bar{t} = \bar{1} = (1,1,1,1), \partial\bar{u} = \bar{0} = (0,0,0,0) \tag{4.4.5}$$

Así como se mostró la resistencia de la primera modificación de IDEA en contra del criptoanálisis diferencial, en base a probar que su matriz de transición es no simétrica, calculando la probabilidad de la diferencial más sencilla, también se puede usar una demostración similar para la modificación 2. Primero se tiene que:

Proposición 4.4.5. *Para la segunda modificación de IDEA se cumple que la diferencia de entrada* $(\bar{1},\bar{1},\bar{0},\bar{0})$ *tiene diferencia de salida* $(\bar{1},\bar{0},\bar{1},\bar{0})$ *con probabilidad*

$$P((\bar{Y},\bar{Y}^*) \in \Delta(\bar{1},\bar{0},\bar{1},\bar{0})|(\bar{X},\bar{X}^*) \in \Delta(\bar{1},\bar{1},\bar{0},\bar{0})) = 2^{-24}.$$

Finalmente se puede probar que la modificación 2 de IDEA es resistente al criptoanálisis diferencial pues su matriz de transición es no simétrica.

Proposición 4.4.6. *La matriz de transición de IDEA es no simétrica pues*

$$P((Y,Y^*) \in \Delta(1,0,1,0)|(X,X^*) \in \Delta(1,1,0,0)) = 2^{-24}$$

y

$$P((Y,Y^*) \in \Delta(1,1,0,0)|(X,X^*) \in \Delta(1,0,1,0)) = 0$$

Las demostraciones de las proposiciones anteriores son todas similares a las de la modificación 1 por lo que se omiten.

A continuación se muestra una tabla comparando las diferenciales de una ronda con probabilidad más alta de PES, IDEA, IDEA modificado 1, IDEA modificado 2:

PES	$P(\Delta Y = (0,0,0,1)\|\Delta X = (0,0,0,-1)) = .002546$
IDEA	$P(\Delta Y = (1,0,1,0)\|\Delta X = (1,1,0,0)) = 2^{-30}$
Modificación 1	$P(\Delta Y = (\bar{1},\bar{0},\bar{1},\bar{0})\|\Delta X = (\bar{1},\bar{1},\bar{0},\bar{0})) = 2^{-28}$
Modificación 2	$P(\Delta Y = (\bar{1},\bar{0},\bar{1},\bar{0})\|\Delta X = (\bar{1},\bar{1},\bar{0},\bar{0})) = 2^{-24}$

De la tabla anterior se puede concluir que las modificaciones a IDEA no afectan en gran medida la seguridad del cifrado, aunque las probabilidades de éxito sean mayores que las del sistema IDEA original. Pues dichas probabilidades siguen siendo pequeñas y se requeriría una gran cantidad de textos en claro para llevar acabo el ataque. El que la probabilidad de éxito se mantenga se debe a que el esquema de cifrado no se cambia, sobre todo se respetan la estructura de la caja MA y los cruces finales de cada ronda que son las principales componentes en la seguridad del sistema.

Capítulo 5

Conclusiones

Recordemos que el objetivo planteado para este trabajo era estudiar los sistemas criptográficos PES e IDEA junto con los criptoanálisis lineal y diferencial para posteriormente proponer y estudiar algunas modificaciones al cifrado IDEA. Analicemos brevemente los resultados obtenidos sobre este aspecto.

Se debe tomar en cuenta que los cambios propuestos a IDEA fueron con respecto a los grupos utilizados durante el proceso de cifrado, también se debe tomar en cuenta que la estructura del sistema quedó intacta, es decir tanto el orden de las operaciones como el número de rondas no sufrieron cambios. De la investigación realizada durante esta tesis se observó que aunque desde el punto de vista algebraico, las modificaciones propuestas de IDEA reducen la complejidad de las operaciones modulares al cambiar la suma de enteros módulo 2^{16} por la suma módulo 2^8 para la primera modificación y por la suma módulo 2^4 para la segunda, así como el producto de enteros módulo $2^{16}+1$ por la multiplicación módulo 2^8+1 para la primera modificación y el producto módulo 2^4+1 para la segunda. Estos cambios provocaron un aumento en el número de operaciones a realizar en cada ronda pues en el caso de la primera modificación se requieren dos veces más operaciones que en el original y en la segunda se necesitan 4 veces más de estas. Lo cual dio como resultado que el cifrado y descifrado de ambas modificaciones sea ligeramente más lento que en IDEA, en contra a lo que se esperaba cuando fueron propuestas. Por otra parte la seguridad de los sistemas en contra del criptoanálisis diferencial no se ve afectada a pesar de que las probabilidad de éxito sean mayores que las de IDEA, lo cual era de esperarse pues el orden de las operaciones en cada ronda no sufrió cambios.

Al transcurrir el estudio del criptoanálisis diferencial se tomó la decisión de realizar computacionalmente el ataque propuesto por Massey y Lai a PES, con el objetivo de corroborar que el criptoanálisis se pueda llevar acabo a pesar de la cantidad de recursos necesarios para su realización y los cuales eran una limitante cuando se propuso el esquema del ataque. Para verificar que el ataque funcionara primero se programó el esquema para mini-PES, el cual fue exitoso y daba pie a pensar que se podría usar este programa como base para el sistema completo. Dicha aproximación no tuvo los resultados esperados principalmente porque el programa estaba estructurado de tal manera que los cálculos se hicieran de manera directa y los datos se guardaran en arreglos simples, lo cual provocó problemas con la cantidad de memoria requerida y con el tiempo necesario para terminar la ejecución del programa. Por lo

cual se requirió utilizar estructuras que llevaron más tiempo de lo deseado para su creación pero que son útiles porque dan una idea más clara de como se pueden construir las tercias que Massey y Lai proponen en el ataque diferencial del PES. A pesar de que estas estructuras redujeron el problema del almacenamiento el ataque no ha tenido éxito hasta el momento, debido a que muy probablemente se requiere un mayor poder computacional.

Finalmente aunque han pasado más de viente años desde que se diera a conocer el ataque diferencial a PES por Massey y Lai, es muy probable que todavía no sea posible llevarlo acabo, al menos en una computadora de uso comercial. Por lo que se concluye que si bien se puede demostrar teóricamente de manera sencilla que un cifrado no es resistente al criptoanálisis diferencial o algún otro ataque complejo, esto no es garantía de que su vulnerabilidad se pueda mostrar fácilmente en la práctica, principalmente por la cantidad de memoria requerida para llevarlo acabo.

Perspectivas

De la investigación realizada para esta tesis, se desprenden los siguientes temas para desarrollar a futuro. En primer lugar hacer una nueva propuesta de los grupos utilizados en el cifrado IDEA, en busca de que se pueda tener un sistema igual de seguro pero con una mayor velocidad de ejecución. Un segundo objetivo es estudiar otros criptoanálisis y probarlos tanto teóricamente como computacionalmente en el cifrado IDEA. Finalmente con relación al punto anterior se debe profundizar en el estudio de la programación para determinar si es que con nuevas herramientas es posible obtener resultados positivos en el criptoanálisis diferencial del sistema PES y además explotar las nuevas características en hardware para implementar nuevas técnicas de criptoanálisis o mejorar las ya existentes.

Con los conocimientos adquiridos será posible considerar probables ataques a otras clases de cifrados. Estos pueden ser sistemas criptográficos a evaluar u otros que se deseé diseñar.

Con este trabajo se ha mostrado que el uso del álgebra puede producir sistemas de cifrado robustos ante los ataques conocidos en la actualidad. En el futuro puede seguirse esta línea de investigación, tal vez con otras estructuras algebraicas. Por ejemplo el AES es un estándar de cifrado de datos, que es completamente algebraico.

Apéndice A

En esta sección se encuentran las funciones de los programas usados durante este trabajo de tesis, todos fueron elaborados en lenguaje C. Estos códigos se realizaron y corrieron principalmente en las computadoras del laboratorio de criptografía del departamento de matemáticas de la Universidad Autónoma Metropolitana unidad Iztapalapa (UAM-I). El equipo utilizado tiene las siguientes características:

	Maquina 1	Maquina 2	Maquina 3
Sistema operativo	Ubuntu 14.04	OS X 10.8.5	Open Suse 13.2
Procesador	Intel Core 2 Duo a 2.2 Ghz	Intel Core i5 a 2.5 Ghz	Intel i7 4790 a 3.5 Ghz

Además se pueden encontrar los programas en la siguiente dirección electrónica `github.com/PedroSobrevilla/Tesis-MCMAI`.

A.1. IDEA

```
//
//  funciones.c
//  IDEA
//
//  Created by Pedro Jose Sobrevilla Moreno on 5/13/14.
//
//

#include "IDEA.h"

void LLave(int k[8][6],int k9[4])
{
    int i;
    int j,l,m;
    int res;
    int aux, aux1;
    int Potencia[16];
    int sk[7][8];
```

```
Potencia[0]=1;
Potencia[1]=2;
Potencia[2]=4;
Potencia[3]=8;
Potencia[4]=16;
Potencia[5]=32;
Potencia[6]=64;
Potencia[7]=128;
Potencia[8]=256;
Potencia[9]=512;
Potencia[10]=1024;
Potencia[11]=2048;
Potencia[12]=4096;
Potencia[13]=8192;
Potencia[14]=16384;
Potencia[15]=32768;

int llave[128];
srand(time(NULL));

//generacion aleatoria de la llave de 128 bits

for (i=0; i<128; i++)
{
    llave[i]=rand()%2;
    printf("%d", llave[i]);
}

// Subllaves para las primeras 8 rondas (48 subllaves)
for (i=0; i<9; i++)
{
    aux = 25*i;

    for (j=0; j<8; j++)
    {
        res = 0;
        if (aux > 127)
        {
            aux = aux%128;
        }

        aux1 = aux + (16*j);
```

```
            for (l = aux1; l < (aux1+16); l++)
            {
                if (l>127)
                {
                    m=l%128;

                    if (llave[m]==0)
                    {
                        res = res + 0;
                    }
                    else
                    {
                        res = res + Potencia[aux1+15-l];
                    }
                }
                else
                {
                    if (llave[l]==0)
                    {
                        res = res + 0;
                    }
                    else
                    {
                        res = res + Potencia[aux1+15-l];
                    }
                }
            }

            sk[i][j]=res;
        }
    }
    printf("\n---------------------------- \n");
    imprime_matriz2f(sk);

    // Asignacion de llaves
    for (i=0; i<9; i++)
    {
        if (i==0)
        {
            for (j=0; j<6; j++)
            {
                k[i][j]=sk[i][j];
            }
        }
```

```
else if (i==1)
{
    k[i][0]=sk[0][6];
    k[i][1]=sk[0][7];
    for (j=0; j<4; j++)
    {
        k[i][j+2]=sk[i][j];
    }
}
else if (i==2)
{
    for (j=0; j<4; j++)
    {
        k[i][j]=sk[1][j+4];
    }
    k[i][4]=sk[2][0];
    k[i][5]=sk[2][1];
}
else if (i==3)
{
    for (j=0; j<6; j++)
    {
        k[i][j]=sk[2][j+2];
    }
}
else if(i==4)
{
    for (j=0; j<6; j++)
    {
        k[i][j]=sk[3][j];
    }
}
else if (i==5)
{
    k[i][0]=sk[3][6];
    k[i][1]=sk[3][7];
    for (j=0; j<4; j++)
    {
        k[i][j+2]=sk[4][j];
    }
}
else if (i==6)
{
    for (j=0; j<4; j++)
```

```
            {
                k[i][j]=sk[4][j+4];
            }
            k[i][4]=sk[5][0];
            k[i][5]=sk[5][1];
        }
        else if (i==7)
        {
            for (j=0; j<6; j++)
            {
                k[i][j]=sk[5][j+2];
            }
        }
        else if (i==8)
        {
            for (j=0; j<4; j++)
            {
                k9[j]=sk[6][j];
            }
        }
    }
}

void LLaved(int k[8][6], int k9[4],int kd[8][6],int k9d[4])
{
    int i,j;

    for (i=0; i<6; i++)
    {
        if (i==0 || i==3)
        {
            kd[0][i]=AEE(k9[i],MAXM);
        }
        else if (i==1 || i==2)
        {
            kd[0][i]=-k9[i]+MAXN;
        }
        else if(i==4 || i==5)
        {
            kd[0][i]=k[7][i];
        }
    }

    for (i=1; i<8; i++)
```

```
    {
        for (j=0; j<6; j++)
        {
            if (j==0 || j==3)
            {
                kd[i][j]=AEE(k[8-i][j],MAXM);
            }
            else if (j==1)
            {
                kd[i][j]=(-k[8-i][j+1]+MAXN)%MAXN;
            }
            else if (j==2)
            {
                kd[i][j]=(-k[8-i][j-1]+MAXN)%MAXN;
            }
            else
            {
                kd[i][j]=k[7-i][j];
            }
        }
    }

    for (i=0; i<4; i++)
    {
        if (i==0 || i==3)
        {
            k9d[i]=AEE(k[0][i],MAXM);
        }
        else if (i==1 || i==2)
        {
            k9d[i]=-k[0][i]+MAXN;
        }
    }
}

void IDEA(int m[4], int K[8][6],int K9[4],int res[4])
{
    int i;
    int x1, x2, x3, x4;
    int e, f, g, h;
    int p, q;
    int r, s, t, u;

    x1 = m[0];
```

```
    x2 = m[1];
    x3 = m[2];
    x4 = m[3];

    for(i=0;i<8;i++)
    {

        e = prod(x1, K[i][0]);
        f = ((long)x2 + K[i][1]) % MAXN;
        g = ((long)x3 + K[i][2]) % MAXN;
        h = prod(x4, K[i][3]);

        p = e^g;
        q = f^h;


        r = prod(p, K[i][4]);
        s = ((long)r+q)%MAXN;
        t = prod(s, K[i][5]);
        u = ((long)t+r)%MAXN;

        x1 = e ^ t;
        x2 = g ^ t;
        x3 = f ^ u;
        x4 = h ^ u;

    }

    res[0] = prod(x1, K9[0]);
    res[1] = ((long)x3 + K9[1]) % MAXN;
    res[2] = ((long)x2 + K9[2]) % MAXN;
    res[3] = prod(x4, K9[3]);
}



int prod(int a, int b)
{
    int res;


    if (a==0 && b!=0)
    {
        res = ((long)MAXN*b)%MAXM;
```

```
    }
    else if (b==0 && a!=0)
    {
        res = ((long)MAXN*a)%MAXM;
    }
    else if (a==0 && b==0)
    {
        res = ((long)MAXN*MAXN)%MAXM;
    }
    else
    {
        res = ((long)a*b)%MAXM;
    }

    if (res==MAXN)
    {
        res=0;
    }

    return res;
}

long concatenate(long x, long y)
{
    unsigned long pow = 10;
    while(y >= pow)
        pow *= 10;
    return x * pow + y;
}

void imprime_matrizf(int C[N][M])
{
    int i,j;
    for(i=0; i<N;i++)
    {
        for(j=0; j<M;j++)
        {
            printf("[%d]", C[i][j]);
        }
        printf("\n");
    }
}

int AEE(long a, long b)
```

```
{
    long s,t,p,m,q,r,u,v,aux,aux1;

    if (b>a)
    {
        aux1 = a;
        a = b;
        b = aux1;
    }

    aux = a;
    s = 1;
    t = 0;
    p = 0;
    m = 1;

    while (b != 0)
    {
        q = a/b;
        r = a%b;
        u = s - (q*p);
        v = t - (q*m);
        a = b;
        b = r;
        s = p;
        t = m;
        p = u;
        m = v;
    }

    if (t<0)
    {
        t = aux + t;
    }
    return (int)t;
}

void imprime_matriz2f(int C[7][8])
{
    int i,j;
    for(i=0; i<7;i++)
    {
        for(j=0; j<8;j++)
        {
```

```
            printf("[%d]", C[i][j]);
        }
        printf("\n");
    }
}
```

A.2. IDEA modificado 1

```
//
//  funciones1.c
//  IDEAmod1
//
//  Created by Pedro Jose Sobrevilla Moreno on 6/3/14.
//
//

#include "IDEAM.h"

void LLave(int k[8][6][2],int k9[4][2])
{
    int i;
    int j,l,m;
    int res;
    int aux, aux1;
    int Potencia[16];

    Potencia[0]=1;
    Potencia[1]=2;
    Potencia[2]=4;
    Potencia[3]=8;
    Potencia[4]=16;
    Potencia[5]=32;
    Potencia[6]=64;
    Potencia[7]=128;
    Potencia[8]=256;
    Potencia[9]=512;
    Potencia[10]=1024;
    Potencia[11]=2048;
    Potencia[12]=4096;
    Potencia[13]=8192;
    Potencia[14]=16384;
    Potencia[15]=32768;
```

```
int llave[128];

//generacion aleatoria de la llave de 128 bits

for (i=0; i<128; i++)
{
    llave[i]=rand()%2;
    printf("%d", llave[i]);
}

// Subllaves para las primeras 8 rondas (48 subllaves)
for (i=0; i<8; i++)
{
    aux = 25*i;

    for (j=0; j<6; j++)
    {
        res = 0;
        if (aux > 127)
        {
            aux = aux%128;
        }

        aux1 = aux + (16*j);

        for (l = aux1; l < (aux1+8); l++)
        {
            if (l>127)
            {
                m=l%128;

                if (llave[m]==0)
                {
                    res = res + 0;
                }
                else
                {
                    res = res + Potencia[(aux1+7)-l];
                }
            }
            else
            {
                if (llave[l]==0)
                {
```

```
                res = res + 0;
            }
            else
            {
                res = res + Potencia[(aux1+7)-l];
            }
        }
    }

    k[i][j][0]=res;
}

for (j=0; j<6; j++)
{
    res = 0;
    if (aux > 127)
    {
        aux = aux%128;
    }

    aux1 = aux + (16*j);

    for (l = (aux1+8); l < (aux1+16); l++)
    {
        if (l>127)
        {
            m=l%128;

            if (llave[m]==0)
            {
                res = res + 0;
            }
            else
            {
                res = res + Potencia[(aux1+15)-l];
            }
        }
        else
        {
            if (llave[l]==0)
            {
                res = res + 0;
            }
            else
```

```
                {
                    res = res + Potencia[(aux1+15)-l];
                }
            }
        }

        k[i][j][1]=res;
    }
}


// Subllaves de la ultima ronda
aux = (25*8)%128;

for (i=0; i<4; i++)
{
    res = 0;

    for (l = aux; l < (aux+8); l++)
    {
        if (l>127)
        {
            m=l%128;

            if (llave[m]==0)
            {
                res = res + 0;
            }
            else
            {
                res = res + Potencia[aux+7-l];
            }
        }
        else
        {
            if (llave[l]==0)
            {
                res = res + 0;
            }
            else
            {
                res = res + Potencia[aux+7-l];
            }
        }
```

```
        }
        k9[i][0]=res;
        aux = aux + 16;
    }

    for (i=0; i<4; i++)
    {
        res = 0;

        for (l = (aux+8); l < (aux+16); l++)
        {
            if (l>127)
            {
                m=l%128;

                if (llave[m]==0)
                {
                    res = res + 0;
                }
                else
                {
                    res = res + Potencia[aux+15-l];
                }
            }
            else
            {
                if (llave[l]==0)
                {
                    res = res + 0;
                }
                else
                {
                    res = res + Potencia[aux+15-l];
                }
            }
        }
        k9[i][1]=res;
        aux = aux + 16;
    }
}

void LLaved(int k[8][6][2], int k9[4][2],int kd[8][6][2],int k9d[4][2])
{
    int i,j;
```

```
for (i=0; i<6; i++)
{
    if (i==0 || i==3)
    {
        kd[0][i][0]=AEE(k9[i][0],MAXM);
        kd[0][i][1]=AEE(k9[i][1],MAXM);
    }
    else if (i==1 || i==2)
    {
        kd[0][i][0]=-k9[i][0]+MAXN;
        kd[0][i][1]=-k9[i][1]+MAXN;
    }
    else if(i==4 || i==5)
    {
        kd[0][i][0]=k[7][i][0];
        kd[0][i][1]=k[7][i][1];
    }
}

for (i=1; i<8; i++)
{
    for (j=0; j<6; j++)
    {
        if (j==0 || j==3)
        {
            kd[i][j][0]=AEE(k[8-i][j][0],MAXM);
            kd[i][j][1]=AEE(k[8-i][j][1],MAXM);
        }
        else if (j==1)
        {
            kd[i][j][0]=(-k[8-i][j+1][0]+MAXN)%MAXN;
            kd[i][j][1]=(-k[8-i][j+1][1]+MAXN)%MAXN;
        }
        else if (j==2)
        {
            kd[i][j][0]=(-k[8-i][j-1][0]+MAXN)%MAXN;
            kd[i][j][1]=(-k[8-i][j-1][1]+MAXN)%MAXN;
        }
        else
        {
            kd[i][j][0]=k[7-i][j][0];
            kd[i][j][1]=k[7-i][j][1];
        }
```

```
        }
    }

    for (i=0; i<4; i++)
    {
        if (i==0 || i==3)
        {
            k9d[i][0]=AEE(k[0][i][0],MAXM);
            k9d[i][1]=AEE(k[0][i][1],MAXM);
        }
        else if (i==1 || i==2)
        {
            k9d[i][0]=(-k[0][i][0]+MAXN)%MAXN;
            k9d[i][1]=(-k[0][i][1]+MAXN)%MAXN;
        }
    }
}

void IDEA(int m[8], int K[8][6][2],int K9[4][2],int res[8])
{
    int i;
    int x11, x21, x31, x41,x12,x22,x32,x42;
    int e1, f1, g1, h1;
    int e2, f2, g2, h2;
    int p1, q1;
    int r1, s1, t1, u1;
    int p2, q2;
    int r2, s2, t2, u2;

    x11 = m[0];
    x21 = m[1];
    x31 = m[2];
    x41 = m[3];

    x12 = m[4];
    x22 = m[5];
    x32 = m[6];
    x42 = m[7];

    for(i=0;i<8;i++)
    {

        e1 = prod(x11, K[i][0][0]);
        f1 = ((long)x21 + K[i][1][0]) % MAXN;
```

```
        g1 = ((long)x31 + K[i][2][0]) % MAXN;
        h1 = prod(x41, K[i][3][0]);

        e2 = prod(x12, K[i][0][1]);
        f2 = ((long)x22 + K[i][1][1]) % MAXN;
        g2 = ((long)x32 + K[i][2][1]) % MAXN;
        h2 = prod(x42, K[i][3][1]);

        p1 = e1 ^ g1;
        q1 = f1 ^ h1;

        p2 = e2 ^ g2;
        q2 = f2 ^ h2;

        r1 = prod(p1, K[i][4][0]);
        s1 = ((long)r1+q1)%MAXN;
        t1 = prod(s1, K[i][5][0]);
        u1 = ((long)t1+r1)%MAXN;

        r2 = prod(p2, K[i][4][1]);
        s2 = ((long)r2+q2)%MAXN;
        t2 = prod(s2, K[i][5][1]);
        u2 = ((long)t2+r2)%MAXN;

        x11 = e1 ^ t1;
        x21 = g1 ^ t1;
        x31 = f1 ^ u1;
        x41 = h1 ^ u1;

        x12 = e2 ^ t2;
        x22 = g2 ^ t2;
        x32 = f2 ^ u2;
        x42 = h2 ^ u2;
    }

    res[0] = prod(x11, K9[0][0]);
    res[1] = ((long)x31 + K9[1][0]) % MAXN;
    res[2] = ((long)x21 + K9[2][0]) % MAXN;
    res[3] = prod(x41, K9[3][0]);

    res[4] = prod(x12, K9[0][1]);
    res[5] = ((long)x32 + K9[1][1]) % MAXN;
    res[6] = ((long)x22 + K9[2][1]) % MAXN;
    res[7] = prod(x42, K9[3][1]);
```

```
}



int prod(int a, int b)
{
    int res;


    if (a==0 && b!=0)
    {
        res = ((long)MAXN*b)%MAXM;
    }
    else if (b==0 && a!=0)
    {
        res = ((long)MAXN*a)%MAXM;
    }
    else if (a==0 && b==0)
    {
        res = ((long)MAXN*MAXN)%MAXM;
    }
    else
    {
        res = ((long)a*b)%MAXM;
    }

    if (res==MAXN)
    {
        res=0;
    }

    return res;
}

long concatenate(long x, long y)
{
    unsigned long pow = 10;
    while(y >= pow)
        pow *= 10;
    return x * pow + y;
}

void imprime_matrizf(int C[N][M][2],int n)
{
```

```
    int i,j;
    for(i=0; i<N;i++)
    {
        for(j=0; j<M;j++)
        {
            printf("[%d]", C[i][j][n]);
        }
        printf("\n");
    }
}

int AEE(long a, long b)
{
    long s,t,p,m,q,r,u,v,aux,aux1;

    if (b>a)
    {
        aux1 = a;
        a = b;
        b = aux1;
    }

    aux = a;
    s = 1;
    t = 0;
    p = 0;
    m = 1;

    while (b != 0)
    {
        q = a/b;
        r = a%b;
        u = s - (q*p);
        v = t - (q*m);
        a = b;
        b = r;
        s = p;
        t = m;
        p = u;
        m = v;
    }

    if (t<0)
    {
```

```
        t = aux + t;
    }
    return (int)t;
}

void int_a_bin(int k, int bin[NBITS])
{
    int i;

    for (i=NBITS-1; i>=0; i--)
    {
        bin[i]=k%2;
        k = k/2;
    }
}

int bin_a_int(int bin[NBITS])
{
    int Potencia[16];

    Potencia[0]=1;
    Potencia[1]=2;
    Potencia[2]=4;
    Potencia[3]=8;
    Potencia[4]=16;
    Potencia[5]=32;
    Potencia[6]=64;
    Potencia[7]=128;
    Potencia[8]=256;
    Potencia[9]=512;
    Potencia[10]=1024;
    Potencia[11]=2048;
    Potencia[12]=4096;
    Potencia[13]=8192;
    Potencia[14]=16384;
    Potencia[15]=32768;

    int res,i;

    res = 0;

    for (i=0; i<4 ; i++)
    {
        if (bin[i]==0)
```

```
        {
            res = res;
        }
        else
        {
            res = res + Potencia[NBITS-i];
        }
    }

    return res;
}

int bin_a_int1(int bin[NBITS])
{
    int Potencia[16];

    Potencia[0]=1;
    Potencia[1]=2;
    Potencia[2]=4;
    Potencia[3]=8;
    Potencia[4]=16;
    Potencia[5]=32;
    Potencia[6]=64;
    Potencia[7]=128;
    Potencia[8]=256;
    Potencia[9]=512;
    Potencia[10]=1024;
    Potencia[11]=2048;
    Potencia[12]=4096;
    Potencia[13]=8192;
    Potencia[14]=16384;
    Potencia[15]=32768;

    int res,i;

    res = 0;

    for (i=4; i<NBITS ; i++)
    {
        if (bin[i]==0)
        {
            res = res;
        }
        else
```

```
        {
            res = res + Potencia[NBITS-i];
        }
    }

    return res;
}
```

A.3. IDEA modificado 2

```
//
//  funciones.c
//  IDEAmod2
//
//  Created by Pedro Jose Sobrevilla Moreno on 6/10/14.
//
//

#include "ideamod2.h"

void LLave(int k[8][6][4],int k9[4][4])
{
    int i;
    int j,l,m;
    int res;
    int aux, aux1;
    int Potencia[4];

    Potencia[0]=1;
    Potencia[1]=2;
    Potencia[2]=4;
    Potencia[3]=8;

    int llave[128];

    //generacion aleatoria de la llave de 128 bits

    for (i=0; i<128; i++)
    {
        llave[i]=rand()%2;
        printf("%d", llave[i]);
    }
```

```
// Subllaves para las primeras 8 rondas (48 subllaves)
for (i=0; i<8; i++)
{
    aux = 25*i;

    for (j=0; j<6; j++)
    {
        res = 0;
        if (aux > 127)
        {
            aux = aux%128;
        }

        aux1 = aux + (16*j);

        for (l = aux1; l < (aux1+4); l++)
        {
            if (l>127)
            {
                m=l%128;

                if (llave[m]==0)
                {
                    res = res + 0;
                }
                else
                {
                    res = res + Potencia[(aux1+3)-l];
                }
            }
            else
            {
                if (llave[l]==0)
                {
                    res = res + 0;
                }
                else
                {
                    res = res + Potencia[(aux1+3)-l];
                }
            }
        }

        k[i][j][0]=res;
```

```
}

for (j=0; j<6; j++)
{
    res = 0;
    if (aux > 127)
    {
        aux = aux%128;
    }

    aux1 = aux + (16*j);

    for (l = (aux1+4); l < (aux1+8); l++)
    {
        if (l>127)
        {
            m=l%128;

            if (llave[m]==0)
            {
                res = res + 0;
            }
            else
            {
                res = res + Potencia[(aux1+7)-l];
            }
        }
        else
        {
            if (llave[l]==0)
            {
                res = res + 0;
            }
            else
            {
                res = res + Potencia[(aux1+7)-l];
            }
        }
    }

    k[i][j][1]=res;
}

for (j=0; j<6; j++)
```

```
{
    res = 0;
    if (aux > 127)
    {
        aux = aux%128;
    }

    aux1 = aux + (16*j);

    for (l = (aux1+8); l < (aux1+12); l++)
    {
        if (l>127)
        {
            m=l%128;

            if (llave[m]==0)
            {
                res = res + 0;
            }
            else
            {
                res = res + Potencia[(aux1+11)-l];
            }
        }
        else
        {
            if (llave[l]==0)
            {
                res = res + 0;
            }
            else
            {
                res = res + Potencia[(aux1+11)-l];
            }
        }
    }

    k[i][j][2]=res;
}

for (j=0; j<6; j++)
{
    res = 0;
    if (aux > 127)
```

```
            {
                aux = aux%128;
            }

            aux1 = aux + (16*j);

            for (l = (aux1+12); l < (aux1+16); l++)
            {
                if (l>127)
                {
                    m=l%128;

                    if (llave[m]==0)
                    {
                        res = res + 0;
                    }
                    else
                    {
                        res = res + Potencia[(aux1+15)-l];
                    }
                }
                else
                {
                    if (llave[l]==0)
                    {
                        res = res + 0;
                    }
                    else
                    {
                        res = res + Potencia[(aux1+15)-l];
                    }
                }
            }

            k[i][j][3]=res;
        }
    }


    // Subllaves de la ultima ronda
    aux = (25*8)%128;

    for (i=0; i<4; i++)
    {
```

```
        res = 0;

        for (l = aux; l < (aux+4); l++)
        {
            if (l>127)
            {
                m=l%128;

                if (llave[m]==0)
                {
                    res = res + 0;
                }
                else
                {
                    res = res + Potencia[aux+3-l];
                }
            }
            else
            {
                if (llave[l]==0)
                {
                    res = res + 0;
                }
                else
                {
                    res = res + Potencia[aux+3-l];
                }
            }
        }
        k9[i][0]=res;
        aux = aux + 16;
    }

    for (i=0; i<4; i++)
    {
        res = 0;

        for (l = (aux+4); l < (aux+8); l++)
        {
            if (l>127)
            {
                m=l%128;

                if (llave[m]==0)
```

```
                {
                    res = res + 0;
                }
                else
                {
                    res = res + Potencia[aux+7-l];
                }
            }
            else
            {
                if (llave[l]==0)
                {
                    res = res + 0;
                }
                else
                {
                    res = res + Potencia[aux+7-l];
                }
            }
        }
        k9[i][1]=res;
        aux = aux + 16;
    }

    for (i=0; i<4; i++)
    {
        res = 0;

        for (l = (aux+8); l < (aux+12); l++)
        {
            if (l>127)
            {
                m=l%128;

                if (llave[m]==0)
                {
                    res = res + 0;
                }
                else
                {
                    res = res + Potencia[aux+11-l];
                }
            }
            else
```

```
            {
                if (llave[l]==0)
                {
                    res = res + 0;
                }
                else
                {
                    res = res + Potencia[aux+11-l];
                }
            }
        }
        k9[i][2]=res;
        aux = aux + 16;
    }

    for (i=0; i<4; i++)
    {
        res = 0;

        for (l = (aux+12); l < (aux+16); l++)
        {
            if (l>127)
            {
                m=l%128;

                if (llave[m]==0)
                {
                    res = res + 0;
                }
                else
                {
                    res = res + Potencia[aux+15-l];
                }
            }
            else
            {
                if (llave[l]==0)
                {
                    res = res + 0;
                }
                else
                {
                    res = res + Potencia[aux+15-l];
                }
```

```
            }
        }
        k9[i][3]=res;
        aux = aux + 16;
    }
}

void LLaved(int k[8][6][4], int k9[4][4],int kd[8][6][4],int k9d[4][4])
{
    int i,j;

    for (i=0; i<6; i++)
    {
        if (i==0 || i==3)
        {
            kd[0][i][0]=AEE(k9[i][0],MAXM);
            kd[0][i][1]=AEE(k9[i][1],MAXM);
            kd[0][i][2]=AEE(k9[i][2],MAXM);
            kd[0][i][3]=AEE(k9[i][3],MAXM);
        }
        else if (i==1 || i==2)
        {
            kd[0][i][0]=-k9[i][0]+MAXN;
            kd[0][i][1]=-k9[i][1]+MAXN;
            kd[0][i][2]=-k9[i][2]+MAXN;
            kd[0][i][3]=-k9[i][3]+MAXN;
        }
        else if(i==4 || i==5)
        {
            kd[0][i][0]=k[7][i][0];
            kd[0][i][1]=k[7][i][1];
            kd[0][i][2]=k[7][i][2];
            kd[0][i][3]=k[7][i][3];
        }
    }

    for (i=1; i<8; i++)
    {
        for (j=0; j<6; j++)
        {
            if (j==0 || j==3)
            {
                kd[i][j][0]=AEE(k[8-i][j][0],MAXM);
                kd[i][j][1]=AEE(k[8-i][j][1],MAXM);
```

```
            kd[i][j][2]=AEE(k[8-i][j][2],MAXM);
            kd[i][j][3]=AEE(k[8-i][j][3],MAXM);
        }
        else if (j==1)
        {
            kd[i][j][0]=(-k[8-i][j+1][0]+MAXN)%MAXN;
            kd[i][j][1]=(-k[8-i][j+1][1]+MAXN)%MAXN;
            kd[i][j][2]=(-k[8-i][j+1][2]+MAXN)%MAXN;
            kd[i][j][3]=(-k[8-i][j+1][3]+MAXN)%MAXN;
        }
        else if (j==2)
        {
            kd[i][j][0]=(-k[8-i][j-1][0]+MAXN)%MAXN;
            kd[i][j][1]=(-k[8-i][j-1][1]+MAXN)%MAXN;
            kd[i][j][2]=(-k[8-i][j-1][2]+MAXN)%MAXN;
            kd[i][j][3]=(-k[8-i][j-1][3]+MAXN)%MAXN;
        }
        else
        {
            kd[i][j][0]=k[7-i][j][0];
            kd[i][j][1]=k[7-i][j][1];
            kd[i][j][2]=k[7-i][j][2];
            kd[i][j][3]=k[7-i][j][3];
        }
    }
}

for (i=0; i<4; i++)
{
    if (i==0 || i==3)
    {
        k9d[i][0]=AEE(k[0][i][0],MAXM);
        k9d[i][1]=AEE(k[0][i][1],MAXM);
        k9d[i][2]=AEE(k[0][i][2],MAXM);
        k9d[i][3]=AEE(k[0][i][3],MAXM);
    }
    else if (i==1 || i==2)
    {
        k9d[i][0]=(-k[0][i][0]+MAXN)%MAXN;
        k9d[i][1]=(-k[0][i][1]+MAXN)%MAXN;
        k9d[i][2]=(-k[0][i][2]+MAXN)%MAXN;
        k9d[i][3]=(-k[0][i][3]+MAXN)%MAXN;
    }
}
```

```
}

void IDEA(int m[16], int K[8][6][4],int K9[4][4],int res[16])
{
    int i;
    int x11, x21, x31, x41,x12,x22,x32,x42,x13, x23, x33, x43,x14,x24,x34,x44;
    int e1, f1, g1, h1;
    int e2, f2, g2, h2;
    int e3, f3, g3, h3;
    int e4, f4, g4, h4;
    int p1, q1;
    int r1, s1, t1, u1;
    int p2, q2;
    int r2, s2, t2, u2;
    int p3, q3;
    int r3, s3, t3, u3;
    int p4, q4;
    int r4, s4, t4, u4;


    x11 = m[0];
    x21 = m[1];
    x31 = m[2];
    x41 = m[3];

    x12 = m[4];
    x22 = m[5];
    x32 = m[6];
    x42 = m[7];

    x13 = m[8];
    x23 = m[9];
    x33 = m[10];
    x43 = m[11];

    x14 = m[12];
    x24 = m[13];
    x34 = m[14];
    x44 = m[15];

    for(i=0;i<8;i++)
    {

        e1 = prod(x11, K[i][0][0]);
```

```
f1 = ((long)x21 + K[i][1][0]) % MAXN;
g1 = ((long)x31 + K[i][2][0]) % MAXN;
h1 = prod(x41, K[i][3][0]);

e2 = prod(x12, K[i][0][1]);
f2 = ((long)x22 + K[i][1][1]) % MAXN;
g2 = ((long)x32 + K[i][2][1]) % MAXN;
h2 = prod(x42, K[i][3][1]);

e3 = prod(x13, K[i][0][2]);
f3 = ((long)x23 + K[i][1][2]) % MAXN;
g3 = ((long)x33 + K[i][2][2]) % MAXN;
h3 = prod(x43, K[i][3][2]);

e4 = prod(x14, K[i][0][3]);
f4 = ((long)x24 + K[i][1][3]) % MAXN;
g4 = ((long)x34 + K[i][2][3]) % MAXN;
h4 = prod(x44, K[i][3][3]);

p1 = e1 ^ g1;
q1 = f1 ^ h1;

p2 = e2 ^ g2;
q2 = f2 ^ h2;

p3 = e3 ^ g3;
q3 = f3 ^ h3;

p4 = e4 ^ g4;
q4 = f4 ^ h4;

r1 = prod(p1, K[i][4][0]);
s1 = ((long)r1+q1)%MAXN;
t1 = prod(s1, K[i][5][0]);
u1 = ((long)t1+r1)%MAXN;

r2 = prod(p2, K[i][4][1]);
s2 = ((long)r2+q2)%MAXN;
t2 = prod(s2, K[i][5][1]);
u2 = ((long)t2+r2)%MAXN;

r3 = prod(p3, K[i][4][2]);
s3 = ((long)r3+q3)%MAXN;
t3 = prod(s3, K[i][5][2]);
```

```
        u3 = ((long)t3+r3)%MAXN;

        r4 = prod(p4, K[i][4][3]);
        s4 = ((long)r4+q4)%MAXN;
        t4 = prod(s4, K[i][5][3]);
        u4 = ((long)t4+r4)%MAXN;

        x11 = e1 ^ t1;
        x21 = g1 ^ t1;
        x31 = f1 ^ u1;
        x41 = h1 ^ u1;

        x12 = e2 ^ t2;
        x22 = g2 ^ t2;
        x32 = f2 ^ u2;
        x42 = h2 ^ u2;

        x13 = e3 ^ t3;
        x23 = g3 ^ t3;
        x33 = f3 ^ u3;
        x43 = h3 ^ u3;

        x14 = e4 ^ t4;
        x24 = g4 ^ t4;
        x34 = f4 ^ u4;
        x44 = h4 ^ u4;
    }

    res[0] = prod(x11, K9[0][0]);
    res[1] = ((long)x31 + K9[1][0]) % MAXN;
    res[2] = ((long)x21 + K9[2][0]) % MAXN;
    res[3] = prod(x41, K9[3][0]);

    res[4] = prod(x12, K9[0][1]);
    res[5] = ((long)x32 + K9[1][1]) % MAXN;
    res[6] = ((long)x22 + K9[2][1]) % MAXN;
    res[7] = prod(x42, K9[3][1]);

    res[8] = prod(x13, K9[0][2]);
    res[9] = ((long)x33 + K9[1][2]) % MAXN;
    res[10] = ((long)x23 + K9[2][2]) % MAXN;
    res[11] = prod(x43, K9[3][2]);

    res[12] = prod(x14, K9[0][3]);
```

```
    res[13] = ((long)x34 + K9[1][3]) % MAXN;
    res[14] = ((long)x24 + K9[2][3]) % MAXN;
    res[15] = prod(x44, K9[3][3]);
}



int prod(int a, int b)
{
    int res;


    if (a==0 && b!=0)
    {
        res = ((long)MAXN*b)%MAXM;
    }
    else if (b==0 && a!=0)
    {
        res = ((long)MAXN*a)%MAXM;
    }
    else if (a==0 && b==0)
    {
        res = ((long)MAXN*MAXN)%MAXM;
    }
    else
    {
        res = ((long)a*b)%MAXM;
    }

    if (res==MAXN)
    {
        res=0;
    }

    return res;
}

long concatenate(long x, long y)
{
    unsigned long pow = 10;
    while(y >= pow)
        pow *= 10;
    return x * pow + y;
}
```

```
void imprime_matrizf(int C[N][M][4],int n)
{
    int i,j;
    for(i=0; i<N;i++)
    {
        for(j=0; j<M;j++)
        {
            printf("[%d]", C[i][j][n]);
        }
        printf("\n");
    }
}

int AEE(long a, long b)
{
    long s,t,p,m,q,r,u,v,aux,aux1;

    if (b>a)
    {
        aux1 = a;
        a = b;
        b = aux1;
    }

    aux = a;
    s = 1;
    t = 0;
    p = 0;
    m = 1;

    while (b != 0)
    {
        q = a/b;
        r = a%b;
        u = s - (q*p);
        v = t - (q*m);
        a = b;
        b = r;
        s = p;
        t = m;
        p = u;
        m = v;
    }
```

```
    if (t<0)
    {
        t = aux + t;
    }
    return (int)t;
}
```

A.4. Mini IDEA

```
//
//  funciones.c
//  MINIIDEA
//
//  Created by Pedro Jose Sobrevilla Moreno on 5/15/14.
//
//

#include "MINIIDEA.h"

void LLave(int k[3][6],int k9[4])
{
    int i;
    int j,l,m;
    int res;
    int aux, aux1;
    int sk[3][8];

    int llave[32];

    //generacion aleatoria de la llave de 128 bits

    for (i=0; i<32; i++)
    {
        llave[i]=rand()%2;
        printf("%d", llave[i]);
    }

    // Subllaves para las primeras 4 rondas (24 subllaves)
    for (i=0; i<3; i++)
    {
        aux = 6*i;
```

```
        for (j=0; j<8; j++)
        {
            res = 0;
            if (aux > 31)
            {
                aux = aux%32;
            }

            aux1 = aux + (4*j);

            for (l = aux1; l < (aux1+4); l++)
            {
                if (l>31)
                {
                    m=l%32;

                    if (llave[m]==0)
                    {
                        res = res + 0;
                    }
                    else
                    {
                        res = res + (llave[m]*potencia(2, (aux1+3)-l));
                    }
                }
                else
                {
                    if (llave[l]==0)
                    {
                        res = res + 0;
                    }
                    else
                    {
                        res = res + (llave[l]*potencia(2, (aux1+3)-l));
                    }
                }
            }

            sk[i][j]=res;
        }
    }

    printf("\n---------------------------- \n");
    imprime_matriz2f(sk);
```

```
    for (i=0; i<4; i++)
    {
        if (i==0)
        {
            for (j=0; j<6; j++)
            {
                k[i][j]=sk[i][j];
            }
        }
        else if (i==1)
        {
            k[i][0]=sk[0][6];
            k[i][1]=sk[0][7];
            for (j=0; j<4; j++)
            {
                k[i][j+2]=sk[i][j];
            }
        }
        else if (i==2)
        {
            for (j=0; j<4; j++)
            {
                k[i][j]=sk[1][j+4];
            }
            k[i][4]=sk[2][0];
            k[i][5]=sk[2][1];
        }
        else if (i==3)
        {
            for (j=0; j<4; j++)
            {
                k9[j]=sk[2][j+2];
            }
        }
    }
}

void LLaved(int k[3][6], int k9[4],int kd[3][6],int k9d[4])
{
    int i,j;

    for (i=0; i<6; i++)
    {
```

```
        if (i==0 || i==3)
        {
            kd[0][i]=AEE(k9[i],MAXM);
        }
        else if (i==1 || i==2)
        {
            kd[0][i]=-k9[i]+MAXN;
        }
        else if(i==4 || i==5)
        {
            kd[0][i]=k[2][i];
        }
    }

    for (i=1; i<3; i++)
    {
        for (j=0; j<6; j++)
        {
            if (j==0 || j==3)
            {
                kd[i][j]=AEE(k[3-i][j],MAXM);
            }
            else if (j==1)
            {
                kd[i][j]=(-k[3-i][j+1]+MAXN)%MAXN;
            }
            else if (j==2)
            {
                kd[i][j]=(-k[3-i][j-1]+MAXN)%MAXN;
            }
            else
            {
                kd[i][j]=k[2-i][j];
            }
        }
    }

    for (i=0; i<4; i++)
    {
        if (i==0 || i==3)
        {
            k9d[i]=AEE(k[0][i],MAXM);
        }
        else if (i==1 || i==2)
```

```
        {
            k9d[i]=(-k[0][i]+MAXN)%MAXN;
        }
    }
}

void IDEA(int a, int b, int c, int d, int K[3][6],int K9[4],int res[4])
{
    int i;
    int e, f, g, h;
    int p, q;
    int r, s, t, u;

    printf("a0 = %d, b0 = %d, c0 = %d, d0 = %d \n",a,b,c,d);

    for(i=0;i<3;i++)
    {
        e = prod(a, K[i][0]);
        f = ((long)b + K[i][1]) % MAXN;
        g = ((long)c + K[i][2]) % MAXN;
        h = prod(d, K[i][3]);

        printf("e%d = %d, f%d = %d, g%d = %d ,h%d = %d \n",i+1,e,i+1,f,i+1,g,i+1,h);

        p = e^g;
        q = f^h;

        printf("p%d = %d, q%d = %d \n",i+1,p,i+1,q);

        r = prod(p, K[i][4]);
        s = ((long)r+q)%MAXN;
        t = prod(s, K[i][5]);
        u = ((long)t+r)%MAXN;

        printf("r%d = %d, s%d = %d, t%d = %d ,u%d = %d \n",i+1,r,i+1,s,i+1,t,i+1,u);


        a = e ^ t;
        b = g ^ t;
        c = f ^ u;
        d = h ^ u;

        printf("a%d = %d, b%d = %d, c%d = %d ,d%d = %d \n",i+1,a,i+1,b,i+1,c,i+1,d);
    }
```

```
    res[0] = prod(a, K9[0]);
    res[1] = ((long)c + K9[1]) % MAXN;
    res[2] = ((long)b + K9[2]) % MAXN;
    res[3] = prod(d, K9[3]);
}



int prod(int a, int b)
{
    int res;


    if (a==0 && b!=0)
    {
        res = ((long)MAXN*b)%MAXM;
    }
    else if (b==0 && a!=0)
    {
        res = ((long)MAXN*a)%MAXM;
    }
    else if (a==0 && b==0)
    {
        res = ((long)MAXN*MAXN)%MAXM;
    }
    else
    {
        res = ((long)a*b)%MAXM;
    }

    if(res==16)
    {
        res=0;
    }

    return res;
}


int potencia(int numero, int potencia)
{
    int i;
    int res;
```

```
    res=1;

    if (potencia==0)
    {
        return res;
    }
    else
    {
        for (i=0; i<potencia; i++)
        {
            res=res*numero;
        }
        return res;
    }
}

void imprime_matrizf(int C[N][M])
{
    int i,j;
    for(i=0; i<N;i++)
    {
        for(j=0; j<M;j++)
        {
            printf("[%d]", C[i][j]);
        }
        printf("\n");
    }
}

void imprime_matriz2f(int C[3][8])
{
    int i,j;
    for(i=0; i<3;i++)
    {
        for(j=0; j<8;j++)
        {
            printf("[%d]", C[i][j]);
        }
        printf("\n");
    }
}

int AEE(long a, long b)
```

```
{
    long s,t,p,m,q,r,u,v,aux,aux1;

    if(a==0)
    {
        a = MAXN;
    }
    else if (b==0)
    {
        b=MAXN;
    }

    if (b>a)
    {
        aux1 = a;
        a = b;
        b = aux1;
    }

    aux = a;
    s = 1;
    t = 0;
    p = 0;
    m = 1;

    while (b != 0)
    {
        q = a/b;
        r = a%b;
        u = s - (q*p);
        v = t - (q*m);
        a = b;
        b = r;
        s = p;
        t = m;
        p = u;
        m = v;
    }

    if (t<0)
    {
        t = aux + t;
    }
    return (int)t;
```

```
}
```

A.5. Cálculos del criptoanálisis diferencial a IDEA

```
//
//  main.c
//  criptoanalisis
//
//  Created by Pedro Jose Sobrevilla Moreno on 4/9/14.
//
//

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define N 8

int ec1(int n, int m);
int ec2(int n);
void mult_matriz(long A[N][N],long B[N][N],long C[N][N]);
void imprime_matriz(int C[N][N]);
void imprime_matrizf(double C[N][N]);
void imprime_matrizl(long C[N][N]);
void potencia_matriz(long A[N][N],long[N][N], int potencia);
int potencia(int n, int m);

int main(int argc, const char * argv[])
{

    int n[8][8];
    int T[8][8];
    int TB[8][8];
    double TC[8][8];
    long C[N][N];
    long M[N][N];
    int i,j,k,l;

    for (i=0; i<8; i++)
    {
        for (j=0; j<8; j++)
        {
            int paridad1, paridad2;
```

```
            paridad1 = i%2;
            paridad2 = j%2;

            if (paridad1 == 0 && paridad2==0)
            {
                n[i][j]=ec1(-(i+1), -(j+1));
            }

            else if (paridad1 == 1 && paridad2 == 1)
            {
                n[i][j]=ec1(i,j);
            }

            else if (paridad1 == 0 && paridad2==1)
            {
                n[i][j]=ec1(-(i+1), j);
            }

            else
            {
                n[i][j]=ec1(i,-(j+1));
            }
        }
    }

    imprime_matriz(n);

    printf("------------------\n");


    for (i=0; i<8; i++)
    {
        for (j=0; j<8; j++)
        {
            T[i][j]=(potencia(10, 7)*n[i][j])/potencia(2,12);
        }
    }

    printf("------------------\n");

    imprime_matriz(T);

    printf("------------------\n");
    printf("------------------\n");
```

```
for (i=0; i<8; i++)
{
    for (j=0; j<8; j++)
    {
        TB[i][j]=T[i][j]/7;
    }
}

imprime_matriz(TB);

printf("------------------\n");
printf("------------------\n");

for (i=0; i<8; i++)
{
    for (j=0; j<8; j++)
    {
        M[i][j]=(long)n[i][j];
    }
}

potencia_matriz(M, C, 7);

imprime_matrizl(C);

printf("------------------\n");
printf("------------------\n");

for (i=0; i<8; i++)
{
    for (j=0; j<8; j++)
    {
        TC[i][j]=(pow(2, 58)*C[i][j])/(pow(2, 12*7)*pow(7, 7));
    }
}

imprime_matrizf(TC);

printf("------------------\n");



int x,y,w,contador;
```

```
    contador = 0;
    for (i=0; i<16; i++)
    {
        for (j=0; j<16; j++)
        {
            for (k=0; k<16; k++)
            {
                for (l=0; l<16; l++)
                {
                    x = i+j;
                    y = k-l;
                    w = (i^k)+(j^l);
                    if (x==17 && y==-1 && w==16)
                    {
                        contador++;
                    }
                }
            }
        }
    }

    printf("\n contador %d \n",contador);


    return 0;
}

int ec1(int s1, int s2)
{
    int i,j,k,l,m,n;
    int contador;
    int x,y,z,w,v,u;
    contador=0;

    for (i=0; i<16; i++)
    {
        for (j=0; j<16; j++)
        {
            for (k=0; k<16; k++)
            {
                for (l=0; l<16; l++)
                {
                    for (m=0; m<16; m++)
                    {
```

```
                        for (n=0; n<16; n++)
                        {
                            x=i+j;
                            y=k-l;
                            z=m+n;
                            w=(i^k)+(j^l);
                            v=(k^m)+(l^n);
                            u=(i^m)-(j^n);
                            if (x==17 && y==s1 && z==18 && w==16 && v==17 && u==s2)
                            {
                                contador++;
                            }
                        }
                    }
                }
            }
        }
    }
    return contador;
}

int ec2(int s1)
{
    int i,j,k,l,x,y,w;
    int contador;
    contador=0;
    s1=0;
    for (i=0; i<16; i++)
    {
        for (j=0; j<16; j++)
        {
            for (k=0; k<16; k++)
            {
                for (l=0; l<16; l++)
                {
                    x = i+j;
                    y = k-l;
                    w = (i^k)+(j^l);
                    if (x==17 && y==s1 && w==0)
                    {
                        contador++;
                    }
                }
            }
```

```
        }
    }

    return contador;
}

void imprime_matriz(int C[N][N])
{
    int i,j;
    for(i=0; i<N;i++)
    {
        for(j=0; j<N;j++)
        {
            printf(" %d & ", C[i][j]);
        }
        printf("\n");
    }
}

void imprime_matrizf(double C[N][N])
{
    int i,j;
    for(i=0; i<N;i++)
    {
        for(j=0; j<N;j++)
        {
            printf("%F & ", C[i][j]);
        }
        printf("\n");
    }
}

void imprime_matrizl(long C[N][N])
{
    int i,j;
    for(i=0; i<N;i++)
    {
        for(j=0; j<N;j++)
        {
            printf("[%ld]", C[i][j]);
        }
        printf("\n");
    }
}
```

```
int potencia(int numero, int potencia)
{
    int i;
    int res;

    res=1;

    if (potencia==0)
    {
        return res;
    }
    else
    {
        for (i=0; i<potencia; i++)
        {
            res=res*numero;
        }
        return res;
    }
}

void mult_matriz(long A[N][N],long B[N][N],long C[N][N])
{
    int i,j,k;
    long suma;
    suma = 0;
    for(i=0; i<N; i++)
    {
        for(j=0; j<N; j++)
        {
            for (k=0; k<N; k++)
            {
                suma = suma + A[i][k]*B[k][j];
            }
            C[i][j]=suma;
            suma = 0;
        }
    }
}

void potencia_matriz(long A[N][N],long C[N][N], int potencia)
{
    long B[N][N];
```

```
    int i,j,k;
    for (j=0; j<N; j++)
    {
        for (i=0; i<N; i++)
        {
            B[j][i]=A[j][i];
        }
    }
    k=1;
    while(k<potencia)
    {
        k++;
        mult_matriz(A, B, C);
        for (j=0; j<N; j++)
        {
            for (i=0; i<N; i++)
            {
                B[j][i]=C[j][i];
            }
        }
    }
    for (j=0; j<N; j++)
    {
        for (i=0; i<N; i++)
        {
            C[j][i]=B[j][i];
        }
    }
}
```

A.6. Criptoanálisis diferencial a miniPES

```
//
//  main.c
//  criptminipes
//
//  Created by Pedro Jose Sobrevilla Moreno on 5/11/15.
//
//

//
//  main.c
//  MINIPES
```

```
//
//  Created by Pedro Jose Sobrevilla Moreno on 3/10/15.
//
//

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

#define NRONDAS 8
#define MAXN 16
#define MAXM 17
#define NBITS 4
#define numClaro 100000
#define LTEXTO 64
#define N 3
#define M 6


int prod(int a, int b);
void PES(int m[4], int K[3][6],int K9[4], int res[4]);
int potencia(int numero, int potencia);
void LLave(int k[3][6],int k9[4]);
void LLaved(int k[3][6], int k9[4],int kd[3][6],int k9d[4]);
void imprime_matrizf(int C[N][M]);
void imprime_matriz2f(int C[3][8]);
void imprime_matriz3f(int C[10][4]);
int AEE(long a, long b);
void intabin(int x, int y[NBITS]);
void LLaveCrack(int k[3][6],int k9[4], int llave1, int llave2,
                int llave3, int llave4);

int main(int argc, const char * argv[])
{

    int i,j,l;

    int z1,z2,z3,z4;

    int x1,x2,v1,v2,t1,t2,e1,e2;

    int y1,y2,w1,w2,h1,h2,u1,u2,f1,f2;
```

```
int difft,diffe;
int diffF,diffu;
int contador;
int max;
int maxllave1;
int maxllave2;
int maxllave3;
int maxllave4;

maxllave1=-1;
maxllave2=-1;
maxllave3=-1;
maxllave4=-1;

int X1[640];
int V1[640];
int T1[640];

int H[MAXN];
int Y1[1752];
int W1[1752];
int U1[1752];

int kc[8][6];
int kc9[4];

int m[NBITS];
int c[NBITS];
int prueba[NBITS];

for(i=0;i<4;i++)
{
    m[i]=1;
}

for(i=0;i<4;i++)
{
    prueba[i]=-1;
}

contador=0;

for (x1=0; x1<MAXN; x1++)
{
```

```
        for (v1=0; v1<MAXN; v1++)
        {
            for (t1=0; t1<MAXN; t1++)
            {
                x2=(1-x1)%MAXN;
                if (x2<0)
                {
                    x2=(MAXN+x2)%MAXN;
                }
                v2=v1;
                t2=x1^x2^t1;
                e1=v1^t1;
                e2=v2^t2;
                difft=(t1+t2)%MAXN;
                diffe=(e1+e2)%MAXN;
                if (difft==1 && diffe==1)
                {
                    X1[contador]=x2;
                    V1[contador]=v2;
                    T1[contador]=t2;
                    contador++;
                }
            }
        }
    }

    for (h1=0; h1<MAXN; h1++)
    {
        h2=h1-1;
        H[h1]=h2;
    }

    contador=0;

    for (y1=0; y1<MAXN; y1++)
    {
        for (w1=0; w1<MAXN; w1++)
        {
            for (u1=0; u1<MAXN; u1++)
            {
                for (h1=0; h1<MAXN; h1++)
                {
                    y2=(1-y1)%MAXN;
                    w2=(w1+1)%MAXN;
```

```
                    if (y2<0)
                    {
                        y2=(MAXN+y2)%MAXN;
                    }
                    u2=y1^y2^u1^h1^H[h1];
                    f1=u1^w1;
                    f2=u2^w2;
                    diffu=(u1+u2)%MAXN;
                    diffF=(f1+f2)%MAXN;
                    if (diffF==1 && diffu==2)
                    {
                        Y1[contador]=y2;
                        W1[contador]=w2;
                        U1[contador]=u2;
                        contador++;
                    }
                }

            }
        }
    }

    printf("%d\n",numClaro);

    int aee[MAXN]; //Arreglo para guardar los inversos

    for(i=1;i<MAXM;i++) //Se generan inversos
    {
        aee[i]=AEE(i,MAXM);
    }

    int aux1;
    int aux2;
    int aux3;
    int aux4;


    int TP1[numClaro][4];
    int TP2[numClaro][4];
    int TC1[numClaro][4];
    int TC2[numClaro][4];


    int aux[4];
```

```
int auxc[4];

int llave1[MAXN][MAXN];
int llave2[MAXN][MAXN];

srand(time(NULL));

unsigned long fullStartTime = time(NULL);//Prueba inicio reloj

int k[8][6];
int k9[4];

LLave(k, k9); //Se crean la llaves de cifrado y se imprimen

printf("\n--------------------------- \n");
imprime_matrizf(k);
for (i=0; i<4; i++)
{
    printf("[%d]",k9[i]);
}
printf("\n--------------------------- \n");

PES(m,k,k9,c);

contador=0;

while(k9[0]!=maxllave1 || k9[2]!=maxllave2 ||
      k9[1]!=maxllave3 || k9[3]!=maxllave4)
{

    for(i=0;i<MAXN;i++)
    {
        for (j=0; j<MAXN; j++)
        {
            llave1[i][j]=0;
            llave2[i][j]=0;
        }
    }


    for (i=0; i<numClaro; i++) //Arreglo aleatorio de textos en claro
    {
        for (j=0; j<4; j++)
        {
```

```
            TP1[i][j]=rand()%MAXN;
        }
    }

    for (i=0; i<numClaro; i++) //Se cifran los textos en claro aleatorios
    {
        for (j=0; j<4; j++)
        {
            aux[j] = TP1[i][j];
        }

        PES(aux, k, k9, auxc);

        for (l=0; l<4; l++)
        {
            TC1[i][l]=auxc[l];
        }

    }

    for (i=0; i<numClaro; i++) //La diferencia de entrada es (0,0,0,1)
                               //entonces se crean textos en claro con
                               //dicha diferencia
    {
        aux1 = (1-TP1[i][0])%MAXN;
        if (aux1<0)
        {
            aux1=MAXN+aux1;
        }
        aux2 = (1-TP1[i][1])%MAXN;
        if (aux2<0)
        {
            aux2=MAXN+aux2;
        }
        aux3 = TP1[i][2];
        aux4 = (TP1[i][3]+1)%MAXN;


        TP2[i][0]=aux1;
        TP2[i][1]=aux2;
        TP2[i][2]=aux3;
        TP2[i][3]=aux4;
    }
```

```
for (i=0; i<numClaro; i++) // Se cifra el segundo arreglo de textos
                           //en claro que cumplen la diferencia (0,0,0,1)
{
    for (j=0; j<4; j++)
    {
        aux[j] = TP2[i][j];
    }

    PES(aux, k, k9, auxc);

    for (l=0; l<4; l++)
    {
        TC2[i][l]=auxc[l];
    }
}

printf("hola\n");



for (i=0; i<numClaro; i++)
{
    for (l=0; l<MAXN; l++)
    {
        z1=prod(TC1[i][0],aee[X1[l]]);
        z2=(TC1[i][2]-V1[l])%MAXN;
        if(z2<0)
        {
            z2=MAXN+z2;
        }
        llave1[z1][z2]++;
    }
}

max=-1;
maxllave1=0;
maxllave2=0;

for (i=0; i<MAXN; i++)
{
    for (j=0; j<MAXN; j++)
    {
        if (llave1[i][j]>max)
        {
```

```
            max = llave1[i][j];
            maxllave1=i;
            maxllave2=j;
        }
    }
}

printf("%d, %d \n",maxllave1,maxllave2);

printf("llave real %d, contador %d\n",k9[0],maxllave1);
printf("llave real %d, contador %d\n",k9[2],maxllave2);


for (i=0; i<numClaro; i++)
{
    for (l=0; l<MAXN; l++)
    {
        z3=prod(TC1[i][1],aee[Y1[l]]);
        z4=(TC1[i][3]-W1[l])%MAXN;
        if(z4<0)
        {
            z4=MAXN+z4;
        }
        llave2[z3][z4]++;
    }
}

max=-1;
maxllave3=0;
maxllave4=0;

for (i=0; i<MAXN; i++)
{
    for (j=0; j<MAXN; j++)
    {
        if (llave1[i][j]>max)
        {
            max = llave2[i][j];
            maxllave3=i;
            maxllave4=j;
        }
    }
}
```

```
        printf("%d, %d \n",maxllave3,maxllave4);

        printf("llave real %d, contador %d\n",k9[1],maxllave3);
        printf("llave real %d, contador %d\n",k9[3],maxllave4);

        contador++;

        printf("contador=%d\n",contador);
    }

    while(c[0]!=prueba[0] || c[1]!=prueba[1] || c[2]!=prueba[2] || c[3]!=prueba[3])
    {
        LLaveCrack(kc, kc9, maxllave1,maxllave3,maxllave2,maxllave4);

        printf("\n--------------------------- \n");

            imprime_matrizf(kc);

            for (i=0; i<4; i++)
            {
            printf("[%d]",kc9[i]);
            }

            printf("\n--------------------------- \n");

            PES(m,kc,kc9,prueba);

            for (i=0; i<4; i++)
            {
            printf("%d ",prueba[i]);
            }

            printf("\n");
    }

    printf("\n--------------------------- \n");
    imprime_matrizf(k);
    for (i=0; i<4; i++)
    {
        printf("[%d]",k9[i]);
    }
    printf("\n--------------------------- \n");
```

```
    for (i=0; i<4; i++)
    {
        printf("%d ",c[i]);
    }

    printf("\n--------------------------- \n");


    unsigned long fullEndTime = time(NULL); //termina el reloj
        printf("Tiempo total = %ld segundos\n", (fullEndTime - fullStartTime));
    //imprime el tiempo

    return 0;
}

void LLave(int k[3][6],int k9[4])
{
    int i;
    int j,l,m;
    int res;
    int aux, aux1;
    int sk[3][8];

    int llave[32];

    //generacion aleatoria de la llave de 128 bits

    for (i=0; i<32; i++)
    {
        llave[i]=rand()%2;
        printf("%d", llave[i]);
    }

    // Subllaves para las primeras 4 rondas (24 subllaves)
    for (i=0; i<3; i++)
    {
        aux = 6*i;

        for (j=0; j<8; j++)
        {
            res = 0;
            if (aux > 31)
            {
```

```
            aux = aux%32;
        }

        aux1 = aux + (4*j);

        for (l = aux1; l < (aux1+4); l++)
        {
            if (l>31)
            {
                m=l%32;

                if (llave[m]==0)
                {
                    res = res + 0;
                }
                else
                {
                    res = res + (llave[m]*potencia(2, (aux1+3)-l));
                }
            }
            else
            {
                if (llave[l]==0)
                {
                    res = res + 0;
                }
                else
                {
                    res = res + (llave[l]*potencia(2, (aux1+3)-l));
                }
            }
        }

        sk[i][j]=res;
    }
}

printf("\n---------------------------- \n");
imprime_matriz2f(sk);

for (i=0; i<4; i++)
{
    if (i==0)
    {
```

```
            for (j=0; j<6; j++)
            {
                k[i][j]=sk[i][j];
            }
        }
        else if (i==1)
        {
            k[i][0]=sk[0][6];
            k[i][1]=sk[0][7];
            for (j=0; j<4; j++)
            {
                k[i][j+2]=sk[i][j];
            }
        }
        else if (i==2)
        {
            for (j=0; j<4; j++)
            {
                k[i][j]=sk[1][j+4];
            }
            k[i][4]=sk[2][0];
            k[i][5]=sk[2][1];
        }
        else if (i==3)
        {
            for (j=0; j<4; j++)
            {
                k9[j]=sk[2][j+2];
            }
        }
    }
}

void LLaved(int k[3][6], int k9[4],int kd[83][6],int k9d[4])
{
    int i,j;

    for (i=0; i<6; i++)
    {
        if (i==0 || i==1)
        {
            kd[0][i]=AEE(k9[i],MAXM);
        }
        else if (i==2 || i==3)
```

```
        {
            kd[0][i]=-k9[i]+MAXN;
        }
        else if(i==4 || i==5)
        {
            kd[0][i]=k[2][i];
        }
    }

    for (i=1; i<3; i++)
    {
        for (j=0; j<6; j++)
        {
            if (j==0 || j==1)
            {
                kd[i][j]=AEE(k[3-i][j],MAXM);
            }
            else if (j==2 || j==3)
            {
                kd[i][j]=(-k[3-i][j]+MAXN)%MAXN;
            }
            else
            {
                kd[i][j]=k[2-i][j];
            }
        }
    }

    for (i=0; i<4; i++)
    {
        if (i==0 || i==1)
        {
            k9d[i]=AEE(k[0][i],MAXM);
        }
        else if (i==2 || i==3)
        {
            k9d[i]=(-k[0][i]+MAXN)%MAXN;
        }
    }
}

void PES(int m[4], int K[8][6],int K9[4],int res[4])
{
    int i;
```

```
int x1, x2, x3, x4,t0,t1,t2,a,b;
//int e, f, g, h;
//int p, q;
//int r, s, t, u;

x1 = m[0];
x2 = m[1];
x3 = m[2];
x4 = m[3];

for(i=0;i<3;i++)
{
    x1 = prod(x1, K[i][0]);
    x2 = prod(x2, K[i][1]);
    x3 = (x3+K[i][2])%MAXN;
    x4 = (x4+K[i][3])%MAXN;

    t0 = prod(K[i][4], (x1^x3));
    t1 = prod(K[i][5], (t0+(x2^x4))%MAXN);
    t2 = (t0+t1)%MAXN;

    a = x1^t1;
    b = x2^t2;
    x1 = x3^t1;
    x2 = x4^t2;
    x3 = a;
    x4 = b;

    /*
     e = prod(x1, K[i][0]);
     f = prod(x2, K[i][1]);
     g = ((long)x3 + K[i][2]) % MAXN;
     h = ((long)x4 + K[i][3]) % MAXN;

     p = e^g;
     q = f^h;


     r = prod(p, K[i][4]);
     s = ((long)r+q)%MAXN;
     t = prod(s, K[i][5]);
     u = ((long)t+r)%MAXN;

     x1 = g ^ t;
```

```
            x2 = h ^ u;
            x3 = e ^ t;
            x4 = f ^ u;
            */
    }

    res[0] = prod(x1, K9[0]);
    res[1] = prod(x2, K9[1]);
    res[2] = ((long)x3 + K9[2]) % MAXN;
    res[3] = ((long)x4 + K9[3]) % MAXN;
}




int prod(int a, int b)
{
    int res;


    if (a==0 && b!=0)
    {
        res = ((long)MAXN*b)%MAXM;
    }
    else if (b==0 && a!=0)
    {
        res = ((long)MAXN*a)%MAXM;
    }
    else if (a==0 && b==0)
    {
        res = ((long)MAXN*MAXN)%MAXM;
    }
    else
    {
        res = ((long)a*b)%MAXM;
    }

    if(res==16)
    {
        res=0;
    }

    return res;
}
```

```
int potencia(int numero, int potencia)
{
    int i;
    int res;

    res=1;

    if (potencia==0)
    {
        return res;
    }
    else
    {
        for (i=0; i<potencia; i++)
        {
            res=res*numero;
        }
        return res;
    }
}

void imprime_matrizf(int C[N][M])
{
    int i,j;
    for(i=0; i<N;i++)
    {
        for(j=0; j<M;j++)
        {
            printf("[%d]", C[i][j]);
        }
        printf("\n");
    }
}

void imprime_matriz2f(int C[3][8])
{
    int i,j;
    for(i=0; i<3;i++)
    {
        for(j=0; j<8;j++)
        {
            printf("[%d]", C[i][j]);
```

```
        }
        printf("\n");
    }
}

void imprime_matriz3f(int C[10][4])
{
    int i,j;
    for(i=0; i<10;i++)
    {
        for(j=0; j<4;j++)
        {
            printf("[%d]", C[i][j]);
        }
        printf("\n");
    }
}

int AEE(long a, long b)
{
    long s,t,p,m,q,r,u,v,aux,aux1;

    if(a==0)
    {
        a = MAXN;
    }
    else if (b==0)
    {
        b=MAXN;
    }

    if (b>a)
    {
        aux1 = a;
        a = b;
        b = aux1;
    }

    aux = a;
    s = 1;
    t = 0;
    p = 0;
    m = 1;
```

```
    while (b != 0)
    {
        q = a/b;
        r = a%b;
        u = s - (q*p);
        v = t - (q*m);
        a = b;
        b = r;
        s = p;
        t = m;
        p = u;
        m = v;
    }

    if (t<0)
    {
        t = aux + t;
    }
    return (int)t;
}

void LLaveCrack(int k[3][6],int k9[4], int llave1, int llave2, int llave3, int llave4)
{
    int i;
    int j,l,m;
    int res;
    int aux, aux1;
    int sk[3][8];

    int llave[32];

    int z1[NBITS];
    int z2[NBITS];
    int z3[NBITS];
    int z4[NBITS];

    intabin(llave1,z1);
    intabin(llave2,z2);
    intabin(llave3,z3);
    intabin(llave4,z4);

    //generacion aleatoria de la llave de 128 bits

    for (i=0; i<32; i++)
```

```
{
    llave[i]=rand()%2;
}

for(i=0;i<NBITS;i++)
{
    llave[20+i]=z1[i];
}

for(i=0;i<NBITS;i++)
{
    llave[24+i]=z2[i];
}

for(i=0;i<NBITS;i++)
{
    llave[28+i]=z3[i];
}

for(i=0;i<NBITS;i++)
{
    llave[i]=z4[i];
}

for (i=0; i<32; i++)
{
    printf("%d", llave[i]);
}

// Subllaves para las primeras 4 rondas (24 subllaves)
for (i=0; i<3; i++)
{
    aux = 6*i;

    for (j=0; j<8; j++)
    {
        res = 0;
        if (aux > 31)
        {
            aux = aux%32;
        }

        aux1 = aux + (4*j);
```

```
            for (l = aux1; l < (aux1+4); l++)
            {
                if (l>31)
                {
                    m=l%32;

                    if (llave[m]==0)
                    {
                        res = res + 0;
                    }
                    else
                    {
                        res = res + (llave[m]*potencia(2, (aux1+3)-l));
                    }
                }
                else
                {
                    if (llave[l]==0)
                    {
                        res = res + 0;
                    }
                    else
                    {
                        res = res + (llave[l]*potencia(2, (aux1+3)-l));
                    }
                }
            }

            sk[i][j]=res;
        }
    }

    printf("\n--------------------------- \n");
    imprime_matriz2f(sk);

    for (i=0; i<4; i++)
    {
        if (i==0)
        {
            for (j=0; j<6; j++)
            {
                k[i][j]=sk[i][j];
            }
        }
```

```
        else if (i==1)
        {
            k[i][0]=sk[0][6];
            k[i][1]=sk[0][7];
            for (j=0; j<4; j++)
            {
                k[i][j+2]=sk[i][j];
            }
        }
        else if (i==2)
        {
            for (j=0; j<4; j++)
            {
                k[i][j]=sk[1][j+4];
            }
            k[i][4]=sk[2][0];
            k[i][5]=sk[2][1];
        }
        else if (i==3)
        {
            for (j=0; j<4; j++)
            {
                k9[j]=sk[2][j+2];
            }
        }
    }
}

void intabin(int entero, int bin[NBITS])
{
    int i;

    for(i=0;i<NBITS;i++)
    {
        bin[i]=0;
    }

    i = NBITS-1;
    while(entero != 0)
    {
        bin[i]=entero%2;
        entero=entero/2;
        i--;
    }
```

```
}
```

A.7. Criptoanálisis diferencial a PES

```
#ifndef FUNCIONES_H
#define FUNCIONES_H

#ifdef __cplusplus
extern "C" {
#endif

#define maxim 65537                    // 2^16 + 1
#define fuyi  65536                    // 2^16
#define one   65535                    // vector uno (2^16 - 1) 0x0000FFFF
#define round 2

typedef struct Q {
    unsigned short theta;
    unsigned short rho;
    unsigned short phi1;
    unsigned short phi2;
} qpleta;

typedef struct B {
    unsigned short w0;
    unsigned short w1;
    unsigned short w2;
    unsigned short w3;
} block;

void cip( block IN, block *OUT, unsigned Z[7][10], unsigned short n, int opc);
void key(short unsigned uskey[9], unsigned Z[7][10], unsigned short );

unsigned inv(unsigned xin);
unsigned mul(unsigned a, unsigned b);

void impBits( unsigned int , char c, int k );

unsigned diffS( unsigned a, unsigned b );
unsigned diffM( unsigned a, unsigned b );
unsigned potencia( unsigned b, unsigned e);
unsigned long generaPares( block plain[], block plainp[], block cipher[],
                           block cipherp[],  unsigned size, unsigned Z[7][10],
```

```
                          unsigned short n, int opc );
void generaTernas( unsigned short **xs, unsigned lim, int opt );
void generaQpletas( unsigned opc );
void generaDiferencias( void );
void generaInversos( void );

unsigned short diffPP[ fuyi ];
unsigned short invPP[ fuyi ];
short S[ 8 ] = { -1, 1, -3, 3, -5, 5, -7, 7 };
unsigned short t2[ 16 ] = {17, 16, 15, 14, 13, 12, 11, 10,  9, 8, 7, 6, 5, 4, 3, 2};
unsigned short r2[ 16 ] = {18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3};

unsigned short p2[ 8 ][ 16 ] = { {  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12,
                                   13, 14, 15,  0 },
                                 { 15,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10,
                                   11, 12, 13, 14 },
                                 {  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14,
                                   15,  0,  1,  2 },
                                 { 13, 14, 15,  0,  1,  2,  3,  4,  5,  6,  7,  8,
                                   9, 10, 11, 12 },
                                 {  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,  0,
                                   1,  2,  3,  4 },
                                 { 11, 12, 13, 14, 15,  0,  1,  2,  3,  4,  5,  6,
                                   7,  8,  9, 10 },
                                 {  7,  8,  9, 10, 11, 12, 13, 14, 15,  0,  1,  2,
                                   3,  4,  5,  6 },
                                 {  9, 10, 11, 12, 13, 14, 15,  0,  1,  2,  3,  4,
                                   5,  6,  7,  8 } };

unsigned short M[ 8 ][ 8 ] = {  {  0, 73, 36,  0,  0, 27,  2,  0 },
                                { 73,  0,  0, 36, 27,  0,  0,  2 },
                                { 36,  0,  0,  0, 18,  0,  0,  9 },
                                {  0, 36,  0,  0,  0, 18,  9,  0 },
                                {  0, 27, 18,  0,  0,  0,  0,  0 },
                                { 27,  0,  0, 18,  0,  0,  0,  0 },
                                {  2,  0,  0,  9,  0,  0,  0,  0 },
                                {  0,  2,  9,  0,  0,  0,  0,  0 } };

qpleta *box[ 8 ][ 8 ];

/* cifrado */

void cip( block IN, block *OUT, unsigned Z[7][10], unsigned short n, int opc ) {
```

```
unsigned short r, x1, x2, x3, x4, kk, t1, t2, a;

x1 = IN.w0;
x2 = IN.w1;
x3 = IN.w2;
x4 = IN.w3;

if( opc != 0 ) {
    printf("%8d %8d %8d %8d \t", x1, x2, x3, x4 );
    //impBits( x1, ' ', 16 );
    //impBits( x2, ' ', 16 );
    //impBits( x3, ' ', 16 );
    //impBits( x4, '\t', 16 );
    printf("Texto en plano \n");

}

for (r = 1; r <= n; r++) {

    x1 = mul(x1, Z[1][r]);
    x2 = mul(x2, Z[2][r]);

    x3 = (x3 + Z[3][r]) & one;
    x4 = (x4 + Z[4][r]) & one;

    kk = mul(Z[5][r], (x1^x3));
    t1 = mul(Z[6][r], (kk + (x2^x4)) & one);
    t2 = (kk + t1) & one;

    a  = x1^t1;
    x1 = x3^t1;
    x3 = a;
    a  = x2^t2;
    x2 = x4^t2;
    x4 = a;

    if( opc != 0 ) {
        printf("%8d %8d %8d %8d \t", x1, x2, x3, x4 );
        //impBits( x1, ' ', 16 );
        //impBits( x2, ' ', 16 );
        //impBits( x3, ' ', 16 );
        //impBits( x4, '\t', 16 );
        printf("Fin ronda %2d \n", r);
    }
```

```
    }

    OUT -> w0 = (unsigned short) mul(x1, Z[1][n + 1]);
    OUT -> w1 = (unsigned short) mul(x2, Z[2][n + 1]);
    OUT -> w2 = (unsigned short) (x3 + Z[3][n + 1]) & one;
    OUT -> w3 = (unsigned short) (x4 + Z[4][n + 1]) & one;

    if( opc != 0 ) {
        printf("%8d %8d %8d %8d \t", OUT -> w0, OUT -> w1, OUT -> w2, OUT -> w3);
        //impBits( OUT -> w0, ' ', 16 );
        //impBits( OUT -> w1, ' ', 16 );
        //impBits( OUT -> w2, ' ', 16 );
        //impBits( OUT -> w3, '\t', 16 );
        printf("Ultima ronda %u \n", OUT);
    }

}


/* multiplicacion modulo 2^16+1 */
unsigned mul(unsigned a, unsigned b) {

    long int p;
    long unsigned q;

    if (a == 0)
        p = maxim - b;
    else if (b == 0)
        p = maxim - a;
    else {
        q = a*b;
        p = (q & one) - (q >> 16);
        if (p <= 0)
            p = p + maxim;
    }
    return (unsigned) (p & one);
}




/* Algoritmo extendido de Euclides */
unsigned inv(unsigned xin) {
```

```
    long n1, n2, q, r, b1, b2, t;

    if (xin == 0)
        b2 = 0;
    else {
        n1 = maxim;
        n2 = xin;
        b2 = 1;
        b1 = 0;
        do {
            r = (n1 % n2);
            q = (n1 - r) / n2;
            if (r == 0) {
                if (b2 < 0) b2 = maxim + b2;
            } else {
                n1 = n2;
                n2 = r;
                t  = b2;
                b2 = b1 - q*b2;
                b1 = t;
            }
        } while (r != 0);
    }
    return (unsigned) b2;
}


/* Funcion para crear llaves */
void key(short unsigned uskey[8], unsigned Z[7][10], unsigned short n ) {

    short unsigned S[55];
    int i, j, r;


    for ( i = 0; i < 8; i++ ) S[ i ] = uskey[ i ];

    for ( i = 8; i < 55; i++ ) {

        if ((i + 2) % 8 == 0)
            S[i] = (S[i - 7] << 9)^(S[i - 14] >> 7);
        else if ((i + 1) % 8 == 0)
            S[i] = (S[i - 15] << 9)^(S[i - 14] >> 7);
        else
            S[i] = (S[i - 7] << 9)^(S[i - 6] >> 7);
```

```
    }


    for (r = 1; r <= n + 1; r++)
        for (j = 1; j < 7; j++)
            Z[j][r] = S[6 * (r - 1) + j - 1];
}

void impBits( unsigned int n, char c, int k ) {

    int i;
    unsigned int mask = 0x80000000;

    for( i = 0; i < 32 - k; i++, mask >>=1 );

    for( ; i < 32 ; i++, mask >>= 1 )
        printf("%c", ( n & mask )? '1' : '0' );

    printf("%c", c);

    return;

}

unsigned diffS( unsigned a, unsigned b ) { return ( a - b  ); }

unsigned diffM( unsigned a, unsigned b ) { return ( mul( a, inv(b) ) ); }

unsigned potencia( unsigned b, unsigned e) {

    if( e == 0 )
        return 1;
    else if( e == 1 )
        return b;
    else
        return b * potencia( b, e - 1 );

}

/*
 * ESTOS NO SON LOS INVERSOSO MULTIPLICATIVOS !!!!
 */
void generaDiferencias( void ) {
```

```
    unsigned i;

    for (i = 0; i < fuyi; i++)
        diffPP[ i ] = (~i + 2) & one;

}

void generaInversos( void ) {

    unsigned i;

    for( i = 0; i < fuyi; i++ )
        invPP[ i ] = inv( i );

}

/*
 * generaPares, genera las parejas de textos en plano y cifrados

 */
unsigned long generaPares( block plain[], block plainp[], block cipher[],
                           block cipherp[], unsigned size, unsigned Z[7][10],
                           unsigned short n, int opc ) {

    unsigned i;
    unsigned long cont;
    block plano1, plano2;
    block cifrado1, cifrado2;

    for( i = 0, cont = 0; i < size; cont++ ) {

        plano1.w0 = ( unsigned short ) rand();
        plano1.w1 = ( unsigned short ) rand();
        plano1.w2 = ( unsigned short ) rand();
        plano1.w3 = ( unsigned short ) rand();

        cip( plano1, &cifrado1, Z, n, 0 );

        plano2.w0 = diffPP[ plano1.w0 ];
        plano2.w1 = diffPP[ plano1.w1 ];
        plano2.w2 = plano1.w2;
        plano2.w3 = S[ 0 ] + plano1.w3;
```

```
        cip( plano2, &cifrado2, Z, n, 0 );

        if( ( diffM( cifrado1.w0, cifrado2.w0 ) ) == 0 &&
            ( diffM( cifrado1.w1, cifrado2.w1 ) == 0 ) &&
            ( diffS( cifrado1.w2, cifrado2.w2 ) ) == 0 &&
            ( diffS( cifrado1.w3, cifrado2.w3 ) ) == S[ 1 ] ) {

            plain[ i ]   = plano1;
            plainp[ i ]  = plano2;
            cipher[ i ]  = cifrado1;
            cipherp[ i ] = cifrado2;

            i++;

            printf("Parejas encontradas %4d \r", i);

        }

    }

    printf("\n");

    if( opc != 0 )
        for( i = 0; i < size; i ++ ) {

            printf("%04X %04X %04X %04X\t",   plain[ i ].w0,   plain[ i ].w1,
                   plain[ i ].w2,   plain[ i ].w3 );
            printf("%04X %04X %04X %04X\t",  cipher[ i ].w0,  cipher[ i ].w1,
                   cipher[ i ].w2,  cipher[ i ].w3 );
            printf("%04X %04X %04X %04X\t",  plainp[ i ].w0,  plainp[ i ].w1,
                   plainp[ i ].w2,  plainp[ i ].w3 );
            printf("%04X %04X %04X %04X\t", cipherp[ i ].w0, cipherp[ i ].w1,
                   cipherp[ i ].w2, cipherp[ i ].w3 );
            printf("%04X %04X %04X %04X\n",
                   diffM( cipher[ i ].w0, cipherp[ i ].w0 ),
                   diffM( cipher[ i ].w1, cipherp[ i ].w1 ),
                   diffS( cipher[ i ].w2, cipherp[ i ].w2 ),
                   diffS( cipher[ i ].w3, cipherp[ i ].w3 ) );

        }

    return cont;

}
```

```
void generaTernas( unsigned short **xs, unsigned lim, int opt ) {

    unsigned i, j, k, cont, aux, base, maskB = 0x8000;

    xs[ 1 ] = (unsigned short *) malloc(4 * sizeof (unsigned short));
    xs[ 1 ][ 0 ] = 0;
    xs[ 1 ][ 1 ] = 1;
    xs[ 1 ][ 2 ] = maskB;
    xs[ 1 ][ 3 ] = maskB + 1;

    for (i = 2; i <= lim; i++) {

        xs[ i ] = (unsigned short *) malloc(potencia(2, i) * sizeof ( unsigned short));
        base = maskB >> (i - 1);
        cont = potencia(2, i - 1);

        for (j = 0; j < cont; j++) {

            aux = (j << (17 - i)) ^ base;
            xs[ i ][ 2 * j ] = aux;
            xs[ i ][ 2 * j + 1] = aux + 1;

        }

    }

    // se despliegan los resultados de x y t dado un tamaño del bloque, en
    // este caso 8.
    if( opt != 0 ) {
        for (j = 0; j < 4; j++) {
            for (k = 0; k < 4; k++) {

                impBits(xs[ 1 ][ j ], ' ', 16);                 // Dado un x
                impBits(xs[ 1 ][ k ], ' ', 16);                 // exploras los t's
                impBits(xs[ 1 ][ j ] ^ xs[ 1 ][ k ], '\n', 16); // g's

            }
            printf("\n");
        }
        printf("\n");

        for (i = 2; i <= lim; i++) {
```

```
            cont = potencia(2, i);

            for (j = 0; j < cont; j++) {
                for (k = 0; k < cont; k++) {

                    impBits(xs[ i ][ j ], ' ', 16);
                    impBits(xs[ i ][ k ], ' ', 16);
                    impBits(xs[ i ][ j ] ^ xs[ i ][ k ], '\n', 16);

                }
                printf("\n");
            }
            printf("\n");
        }
    }

}

void generaQpletas( unsigned opc ) {

    unsigned i, j, k, cont;
    unsigned t1, r1, p1;
    short inicio, fin;

    for( j = 0; j < 8; j ++ )
        for( i = 0; i < 8; i ++ )
            if( M[ j ][ i ] != 0 )
                box[ j ][ i ] =(qpleta*)malloc(M[ j ][ i ]*sizeof(qpleta));
            else
                box[ j ][ i ] = NULL;

    for (j = 0; j < 8; j++) {
        for (i = 0; i < 8; i++) {

            cont = 0;
            if (S[ i ] < 0) {
                inicio = 0;
                fin = 16 + S[ i ];
            } else {
                inicio = 0 + S[ i ];
                fin = 16;
            }

            for (t1 = 2; t1 < 16; t1++)
```

```
                for (r1 = 3; r1 < 16; r1++)
                    for (p1 = inicio; p1 < fin; p1++) {

                        if (((t1 ^ p1) + (t2[ t1 ] ^ p2[ i ][ p1 ])) == 16)
                            if (((p1 ^ r1) + (p2[ i ][ p1 ] ^ r2[ r1 ])) == 17)
                                if (((t1 ^ r1) - (t2[ t1 ] ^ r2[ r1 ])) == S[ j ] ) {


                                    box[ j ][ i ][ cont ].theta = t1;
                                    box[ j ][ i ][ cont ].rho   = r1;
                                    box[ j ][ i ][ cont ].phi1  = p1;
                                    box[ j ][ i ][ cont ].phi2  = p2[ i ][ p1 ];
                                    cont++;

                                }

                    }

            printf("%2d ", cont);

        }

    printf("\n");

    }

    // Solo está mostrando una pareja de entradas en M
    if( opc != 0 )
        for( i = 0; i < M[ 0 ][ 1 ]; i++ ) {

            impBits( box[ 0 ][ 1 ][ i ].theta, ' ', 4);
            impBits( box[ 0 ][ 1 ][ i ].rho, ' ', 4);
            impBits( box[ 0 ][ 1 ][ i ].phi1, ' ', 4);
            impBits( box[ 0 ][ 1 ][ i ].phi2, '\t', 4);
            impBits( box[ 1 ][ 0 ][ i ].theta, ' ', 4);
            impBits( box[ 1 ][ 0 ][ i ].rho, ' ', 4);
            impBits( box[ 1 ][ 0 ][ i ].phi1, ' ', 4);
            impBits( box[ 1 ][ 0 ][ i ].phi2, '\t', 4);
            impBits( box[ 0 ][ 1 ][ i ].theta - box[ 1 ][ 0 ][ i ].theta, ' ', 4 );
            impBits( box[ 0 ][ 1 ][ i ].rho   - box[ 1 ][ 0 ][ i ].rho, ' ', 4 );
            impBits( box[ 0 ][ 1 ][ i ].phi1  - box[ 1 ][ 0 ][ i ].phi2, ' ', 4 );
            impBits( box[ 0 ][ 1 ][ i ].phi2  - box[ 1 ][ 0 ][ i ].phi1, '\n', 4 );

        }
```

```
    for( i = 0, cont = 0; i < 8; i++ )
        for( j = 0; j < 8; j++ )
            cont += M[ i ][ j ];

    printf("TOTAL qpletas %d \n", cont);

}



#ifdef __cplusplus
}
#endif

#endif /* FUNCIONES_H */
```

Bibliografía

[1] Lai X., Massey J.L. *A proposal for a new block encryption standard.* EUROCRYPT '90, LNCS 473, pp 389-404, 1991.

[2] Lai Xuejia. *On the design and security of block ciphers.* Konstanz, Hartung-Gorre, Diss. Techn. Wiss ETH Zürich, Nr. 9752, 1992.

[3] Menezes, A.J. et al. *Handbook of Applied Cryptography.* CRC Press, 1997.

[4] Paar, C. and Pelzl, J. *Understanding Cryptography: A Textbook for Students and Practitioners.* Springer-Verlag New York Inc, 2010.

[5] Biham Eli, Dunkelman Orr, Keller Nathan. *New Cryptanalytic Results on IDEA.* ASIACRYPT 2006, LNCS 4284, 2006.

[6] Eli Biham, Orr Dunkelman, Nathan Keller, Adi Shamir. *New Attacks on IDEA with at Least 6 Rounds* Journal of Cryptology Volume 28, Issue 2, April 2015

[7] Jorge Nakahara Jr., Bart Preneel, Joos Vandewalle. *The Biryukov-Demirci Attack on IDEA and MESH Ciphers.* Lecture Notes in Computer Science 2908, Springer-Verlag, 2004.

[8] Sheldon M. Ross. *Introduction to Probability Models.* Academic Press, 1997, USA, pág. 30.

[9] Douglas R. Stinson. *Cryptography: Theory and Practice, Third Edition.* Academic Press, 2005, USA,