UNIVERSIDAD DE CHILE

FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

Evaluación de Dos Implementaciones del Estándar de Encriptación de Datos (DES) ante Ataques de Medición de Tiempos

Alejandro Hevia Angulo

Profesor Guía: Marcos Kiwi
Profesores Comisión: Ricardo Baeza–Yates
Gonzalo Navarro

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

Este trabajo ha sido parcialmente financiado por los proyectos FONDAP en Matemáticas Aplicadas 1997–98 y por FONDECYT No. 1960849.

Santiago de Chile

Noviembre, 1998

A mis padres y a Carolina
por su invaluable ayuda y comprensión.

RESUMEN DE LA MEMORIA
PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN
POR: ALEJANDRO HEVIA ANGULO
FECHA: 16/11/98
PROF. GUÍA: SR. MARCOS KIWI

Evaluación de Dos Implementaciones del Estándar de Encriptación de Datos (DES) ante Ataques de Medición de Tiempos

La criptografía aborda el problema de la comunicación en presencia de adversarios. En particular, busca solucionar un amplio espectro de problemas de seguridad que se presentan en la transmisión y almacenamiento de la información. Ejemplos de ellos son la privacidad (confidencialidad del contenido del mensaje comunicado), la autenticación (verificación que el mensaje fue enviado por el emisor indicado y que no ha sido alterado durante la transmisión) y el intercambio de claves (acuerdo de una clave secreta entre dos personas utilizando un medio de comunicación público).

El ámbito de aplicación de la criptografía se ha extendido vertiginosamente en las últimas dos décadas. Por ejemplo, hoy en día se utilizan técnicas criptográficas en el diseño de sistemas de dinero electrónico, firmas digitales y votación electrónica. Esto alienta el desarrollo de nuevas y mejores aplicaciones, al tiempo que genera múltiples interrogantes y desafíos. Este trabajo aborda una de estas interrogantes.

El problema considerado tiene relación con una nueva técnica de criptoanálisis, denominada "ataque de medición de tiempos" (timing attack) introducida por Paul Kocher en 1996. Este ataque explota las características ingenieriles involucradas en la implementación de criptosistemas y puede ser utilizado para atacar con éxito criptosistemas que han resistido largo tiempo sofisticadas técnicas criptoanalíticas. Esencialmente, un ataque de este tipo obtiene algo de la información privada del usuario de un criptosistema a través de cuidadosas mediciones del tiempo que toma efectuar las operaciones criptográficas. En este trabajo se discuten las fortalezas y debilidades de uno de los criptosistemas más utilizados en la actualidad, el Estándar de Encriptación de Datos (Data Encryption Standard – DES) frente a este tipo de ataques.

Analizamos dos implementaciones de DES. A partir de ellas, se muestra que en ambas un ataque de medición de tiempos puede entregar el peso de Hamming (número de bits no nulos) de la clave usada. Más aún, el ataque es computacionalmente económico. También se muestra que todas las características de diseño del sistema objetivo, necesarias para llevar a cabo el ataque, pueden ser inferidas a partir de mediciones de tiempos. Finalmente, se exhibe una modificación que permite a las implementaciones estudiadas resistir el ataque propuesto. Hasta donde sabemos, este es el primer trabajo que muestra que conjeturas previas acerca de la vulnerabilidad de criptosistemas simétricos ante ataques de medición de tiempos son realistas.

Agradecimientos

Este trabajo no habría sido posible sin la ayuda de un incomparable grupo de personas. Vayan pues mis más sinceros agradecimientos a Shang–Hua Teng, por su oportuno consejo acerca del tema de esta tesis, a Marcos Kiwi, por su invaluable dedicación y paciencia como maestro y amigo, a Luis Mateu por sus muy acertados consejos y valiosa ayuda con el sorprendente proceso de medición de tiempos, a Juan Álvarez por su oportuna sugerencia acerca del título de este trabajo, y a todo el grupo de Sistemas del DCC, comenzando por Williams Contreras y Rodrigo Arenas, por su enorme paciencia y cooperación en el transcurso del lento pero necesario proceso de escribir la tesis de ingeniería.

Finalmente me gustaría agradecer a quienes han contribuido con valiosas sugerencias y observaciones a esta investigación. Entre ellos, quiero manifestar mis agradecimientos a Ricardo Baeza–Yates, Paul Kocher, Gonzalo Navarro, René Peralta, Benny Pinkas y Shang–Hua Teng.

Mis agradecimientos a todos aquellos que de una u otra manera contribuyeron a que este trabajo viera la luz.

Índice General

Índice de Figuras

Capítulo 1

Introducción

La criptografía aborda el problema de la comunicación en presencia de adversarios, es decir, en presencia de personas o entidades tanto curiosas como maliciosas. Informalmente hablando, la criptografía entrega herramientas (esquemas o métodos) para que dos entidades se puedan comunicar secretamente, usualmente compartiendo algún tipo de información privada o *clave*. La labor de un adversario será la de *criptoanalizar* (efectuar un análisis en búsqueda de relaciones) el método de comunicación utilizado a fin de recuperar el mensaje enviado o algo de la información privada. En este sentido, el método utilizado en la comunicación deberá resistir todos los ataques criptoanalíticos que el adversario sea capaz de idear. Por ello, la aparición de nuevas amenazas para dichos métodos ha de significar un cuidadoso análisis de robustez de las herramientas actualmente utilizadas. Este tema, como veremos, será fundamental en este trabajo. Para una introducción al tema, sugerimos ver [Sti95, Sch96, MvOV97].

Un nuevo e ingenioso tipo de ataque criptoanalítico fue introducido por Kocher en [Koc96]. Este nuevo ataque se denomina *ataque de medición de tiempos* (*timing attack*). Explota el hecho de que frecuentemente el tiempo que demora un criptosistema en operar varía ligeramente al procesar distintos datos de entrada. Kocher dio varias explicaciones posibles a este comportamiento, como por ejemplo, instrucciones condicionales y de salto, colisiones en el caché de memoria, instrucciones del procesador que corren en un tiempo no constante, etc. La contribución más importante de Kocher fue mostrar que las diferencias en el tiempo de ejecución pueden ser explotadas para encontrar algo de la información privada de un sistema específico atacado (*sistema objetivo*). De hecho, en [Koc96] se muestra cómo criptoanalizar un exponenciador modular simple. La exponenciación modular es una operación clave en el protocolo de intercambio de claves de Diffie y Hellman [DH76] y en el criptosistema RSA [RSA78]. Un exponenciador modular es un procedimiento que, teniendo como entrada $k, n \in \mathbb{N}$, $n \neq 0$, y $y \in \mathbb{Z}$, calcula $(y^k \bmod n)$. (En los protocolos criptográficos mencionados anteriormente, n es público y k es privado.) Kocher mostró que si un *adversario* o *espía*[1]

[1] En lo que sigue, denominaremos *adversario* o *espía* a un individuo que puede ver los mensajes intercambiados por dos entidades distintas.

puede medir el tiempo que le toma a un sistema específico calcular ($y^k \bmod n$) para varios y's, entonces podrá recuperar el exponente secreto k.

El trabajo de Kocher logró establecer que en teoría, los ataques de medición de tiempos pueden proveer algo de la información privada de un sistema objetivo. Sin embargo, la factibilidad de llevar a cabo un ataque de medición de tiempos en un sistema real está por verse. En efecto, no está claro en cuáles ambientes es posible efectuar las precisas mediciones requeridas por un ataque de medición de tiempos. Más aún, al intentar efectuar un ataque de medición de tiempos en un sistema remoto, es posible que retardos aleatorios en la red obliguen a recolectar una cantidad prohibitiva de mediciones de tiempo, a fin de compensar la mayor imprecisión introducida.[2] A pesar de ello, hay ciertas situaciones en las que consideramos que es realista montar un ataque de medición de tiempos. A continuación describiremos una de ellas.

Los protocolos de *reto–respuesta* son utilizados para determinar si dos entidades participantes de una comunicación son en realidad entidades genuinas[3] y, por lo tanto, se permite que se comuniquen entre sí. En dichos protocolos, una entidad le envía un *reto* a la otra, usualmente un número al azar. La entidad que recibe el reto debe realizar un cálculo criptográfico incluyendo usualmente el uso de una clave secreta. A fin de generar el resultado correcto para dicho cálculo, la última entidad debe poseer la clave secreta correcta y, por lo tanto, podrá suponerse que es auténtica. Muchas tarjetas inteligentes (*smartcards* o *IC cards*), en particular generadores de contraseñas dinámicas[4] y tarjetas usadas como billeteras electrónicas, implementan protocolos de reto–respuesta (por ejemplo, de acuerdo al estándar ANSI X9.26 [MvOV97, pág. 651]). Se espera un uso masivo de tarjetas inteligentes basadas en chips de circuitos integrados programables de propósito general. En ellos la funcionalidad específica de cada tarjeta será configurada a través de programación. Usando técnicas criptográficas y de *resistencia a examen interno* (*tamper–resistance*) se buscará garantizar la seguridad de tales tarjetas.

El escenario arriba descrito da lugar a una situación ideal para efectuar un ataque de medición de tiempos. En efecto, la amplia disponibilidad de un tipo particular de tarjeta hará fácil y económico determinar cualquier información necesaria del sistema en el cual montar el ataque (es decir, sus *características de tiempos*). Después, la obtención de mediciones precisas de tiempo (por ejemplo, monitoreando o alterando un lector de tarjetas, o tomando posesión de una tarjeta) podría permitir recuperar algo de la información secreta almacenada en ella a través de un ataque de medición de tiempos. Por ello, tarjetas que implementan protocolos de reto–respuesta donde es necesario disponer de una *clave maestra* (clave común a varias tarjetas) podrían ser una clara fuente de problemas de seguridad.

---

[2] La necesidad de recolectar un mayor número de mediciones cuando existen retardos aleatorios se basa en ciertos resultados de la estadística y se fundamentará en detalle en el siguiente capítulo.

[3] Informalmente, dos entidades serán genuinas si *son quienes dicen ser*.

[4] Los generadores de contraseñas dinámicas o *dynamic password generators* están siendo ampliamente usados como un medio para permitir que usuarios autorizados utilicen sistemas computacionales en forma remota.

Ahora bien, en el mismo artículo donde Kocher mostró el ataque de medición de tiempos [Koc96] se conjeturó que algunos criptosistemas simétricos podrían ser vulnerables a un ataque similar. En particular, que el ataque aplicado al Estándar de Encriptación de Datos (*Data Encryption Standard* – DES) podría revelar el peso de Hamming[5] de la clave utilizada (bajo la hipótesis que las operaciones de desplazamiento de bits – *shifts* – podrían depender del tiempo). Fue la observación de esta potencial debilidad de DES la motivación inicial de la investigación que culminó con este trabajo. Concretamente, en el presente trabajo evaluamos la resistencia de implementaciones de DES ante un ataque de medición de tiempos y analizamos los factores que hacen posible este ataque.

La elección del Estándar de Encriptación de Datos como criptosistema analizado no es arbitraria. DES es en la actualidad el criptosistema más utilizado en el mundo y, en consecuencia, su nivel de seguridad es crucial en muchos contextos. Además, su notoria resistencia a más de dos décadas de tenaz criptoanálisis lo convierten en un candidato ideal para la evaluación de la mayoría de las nuevas técnicas criptoanalíticas.

La existencia de posibles debilidades en DES plantea diversas e importantes interrogantes adicionales. Por ejemplo, ¿cuán resistentes son los criptosistemas regularmente usados en la actualidad (como DES, IDEA o RC5) ante un ataque de medición de tiempos? ¿Cuánta información se puede extraer usando dicho ataque? Si es posible extraer información privada usando este ataque, ¿cuáles son las causas relevantes en la práctica que permiten esta filtración de información? ¿Son controlables estas causas? ¿Existen entornos realistas donde las mediciones precisas de tiempos son factibles? En este trabajo se intentan responder algunas de las interrogantes anteriores.

Es altamente probable que surjan nuevas y sorpresivas facetas de los ataques de medición de tiempos. Por lo tanto, estos ataques merecen una seria consideración de nuestra parte. Este trabajo contribuye a mejorar nuestro entendimiento de las fortalezas de la recientemente introducida técnica de ataques de medición de tiempos, las debilidades que explota y las maneras de evitarlo.

1.1 Resumen de Resultados

Como fue señalado anteriormente, la motivación inicial de este trabajo fue evaluar la conjetura de Kocher [Koc96]: un ataque de medición de tiempos podría revelar el peso de Hamming de la clave usada en el criptosistema más utilizado en el mundo: DES. Sin embargo, este trabajo no sólo analiza la mencionada conjetura (identificando las causas que posibilitan su veracidad) sino que contribuye con otros resultados nuevos.

En particular, mostramos la existencia de condiciones para la implementación de un ataque de medición de tiempos (en concreto, una dependencia prácticamente lineal del tiempo de encriptación con respecto al peso de Hamming) que, suponiendo conocidas las características de diseño de un sistema objetivo, permite recuperar el peso de Hamming de

[5] Número de bits no nulos de la clave utilizada en el criptosistema.

la clave DES usada. Asimismo, se discuten resultados experimentales que exhiben un efecto concreto de este ataque: la reducción del espacio de claves necesario para encontrar una clave DES. Sobre esta base, evaluamos la magnitud potencial de la efectividad de un ataque de medición de tiempos sobre DES.

Por otra parte, mostramos que los protocolos de autenticación de tarjetas inteligentes constituyen entornos realistas donde llevar a cabo un ataque de medición de tiempos. También discutimos el grado de compromiso que este ataque significa para las claves usadas en estas tarjetas.

Además, identificamos y analizamos las fuentes de variabilidad en los tiempos de ejecución de implementaciones del criptosistema DES. Al respecto, nuestros resultados muestran que el factor primordial de esta variabilidad son las instrucciones condicionales.

Un resultado de interés independiente es el desarrollo de dos técnicas estadísticas de inferencia de parámetros. Como veremos, ellas permiten recuperar *sólo a través de mediciones de tiempo* toda la información requerida para montar un ataque de medición de tiempos.

Junto con poner en evidencia la sorprendentemente clara dependencia del tiempo con respecto al peso de Hamming de la clave, discutimos diversos enfoques orientados a eliminarla. Un esquema basado en una técnica del tipo *enceguecimiento* consigue con éxito esta tarea. Al mismo tiempo, examinamos bajo qué condiciones toda la clave DES, y no sólo el peso de Hamming puede ser recuperado a través de un ataque de medición de tiempos.

1.2 Organización

La exposición de los temas contenidos en este trabajo ha sido organizada como sigue. En el Cap. 2 revisamos los conceptos básicos de la criptografía clásica y moderna, así como sus aplicaciones. También revisamos los principales conceptos del criptoanálisis, la clasificación usual de los ataques clásicos a los criptosistemas y el nuevo enfoque criptoanalítico en el cual se basa el ataque analizado en este trabajo.

El Estándar de Encriptación de Datos (DES) es descrito en el Cap. 3. Allí se detalla su funcionamiento y sus principales ataques conocidos (criptoanálisis diferencial y lineal).

En el Cap. 4 mostramos los resultados del análisis de dos implementaciones del criptosistema DES y el tipo de dependencia evidenciada entre el tiempo de ejecución del proceso de encriptación y la clave utilizada. Exhibimos aquí también resultados experimentales que muestran la reducción del espacio de búsqueda necesario para determinar completamente la clave DES utilizada. En la Sect. 4.1.4, además, identificamos las fuentes de las variaciones de tiempo observadas. En el resto del capítulo discutimos dos métodos estadísticos que reducen la información requerida para efectuar el ataque a sólo mediciones del tiempo de encriptación.

Algunos esquemas básicos de defensa ante este ataque son discutidos en el Cap. 5. En particular, en la Sect. 5.2 se propone la técnica de *enceguecimiento* que permite hacer más resistentes las implementaciones analizadas.

Adicionalmente a lo anterior, en el Cap. 6, se presenta una discusión de las posibilidades y problemas abiertos para recuperar toda la clave DES a través de un ataque de medición de tiempos.

Finalmente, es conveniente señalar que parte del material de esta tesis ha aparecido o aparecerá en actas de conferencias o journals. En particular, el contenido de los capítulos 4 y 6, así como la mayor parte del capítulo 5 es un trabajo conjunto con Marcos Kiwi y una versión preliminar de éste apareció en [HK98].

Capítulo 2

Discusión de la Literatura Relevante

2.1 La Criptografía

> *"Roberto desea escribirle a su amada Alicia. Sin embargo, no le gusta la idea que Eva, su estricta chaperona, pueda llegar a leer el contenido amoroso de su carta, pues ello podrá tener nefastas consecuencias para el sufrido romance. Roberto piensa y piensa, mas sus medievales neuronas no logran solucionar este dilema..."*

En este capítulo revisaremos los conceptos y términos básicos utilizados en el resto del trabajo. Para un tratamiento más detallado recomendamos al lector interesado remitirse a [Kiw98, MvOV97, Sch96, Sti95].

2.1.1 Descripción y Objetivos

La criptografía aborda el problema de la comunicación en presencia de *adversarios*. Un adversario es una entidad que, participante o no en el sistema, trata de obtener más información de la *permitida*, o bien, trata de impedir que se logren alguno de los objetivos de dicho sistema. Así, la criptografía tradicionalmente ha consistido en el estudio de métodos que permitan el envío de mensajes entre dos partes interesadas, a través de un canal inseguro monitoreado por dicho adversario o espía, de forma que sólo aquel al cual los mensajes van dirigidos pueda entender lo que se desea comunicar.

Más concretamente, esta área estudia los diversos problemas que surgen en presencia de dichos adversarios. Entre ellos, la privacidad, la autenticación, el acuerdo de claves y los secretos compartidos.

La *privacidad* se refiere a evitar la extracción de información desde un mensaje (enviado por un canal público) por partes no autorizadas, asegurándole así al enviador que su mensaje será comprensible solamente para el (o los) verdadero(s) destinatario(s). La *autenticación* dice relación con la verificación de que el mensaje ha sido enviado por el emisor indicado y que no ha sido alterado durante la transmisión. En el *acuerdo de claves* dos partes, utilizando un

medio de comunicación público, desean acordar una clave para usar en algún criptosistema de clave privada. Finalmente en los *secretos compartidos* se busca distribuir cierta información privada (como la contraseña para lanzar un misil) entre un grupo de personas, de forma que k individuos trabajando conjuntamente puedan recuperar la información pero que no sea posible para $k-1$ individuos realizarlo.

Actualmente esta área abarca una amplísima variedad de problemas (ver por ejemplo [MvOV97, págs. 3-4]) de los cuales los mencionados son – histórica y actualmente – los de mayor relevancia.

2.1.2 Criptografía Clásica

La criptografía clásica típicamente considera el escenario en que dos partes o entidades (personas, terminales de computadora, etc.), denominadas usualmente Alicia y Roberto, quieren comunicarse a través de un canal inseguro. Este canal inseguro esta monitoreado por un adversario o espía que suele llamarse Eva. Ejemplos típicos de canales inseguros son: una línea telefónica, una red computacional, el correo, ondas radiales, el telégrafo, etc.

A la información o mensaje que Alicia quiere enviarle a Roberto se denomina *texto puro* o *texto plano* (*plaintext*). Para evitar que Eva comprenda lo que Alicia le quiere decir a Roberto, Alicia encripta el texto puro, utilizando una clave, y obtiene el *texto cifrado* (*ciphertext*) el cual envía a Roberto. Roberto, que (supondremos) conoce la clave que se usó en la encriptación, puede descifrar el mensaje, pero Eva que sólo ve el texto cifrado (y no conoce la clave) no debería ser capaz de determinar el texto puro.

En este contexto e informalmente hablando, un sistema criptográfico (o **criptosistema**) es un esquema tal que permite transformar mensajes "legibles" en mensajes "incomprensibles" mediante la aplicación de algún tipo de función matemática, en un proceso denominado **encriptación**. El proceso inverso, volver "legible" un mensaje aparentemente "incomprensible", se denomina **desencriptación** y constituye otra componente de un criptosistema. Por lo general además, un criptosistema incluye una **clave** (o llave), que permite a su poseedor efectuar alguna o ambas operaciones anteriores. Usualmente, es conveniente visualizar un criptosistema como compuesto por una función de encriptación, una función de desencriptación y un conjunto de claves posibles de utilizar. A continuación, precisaremos más formalmente este concepto.

Criptografía de Clave Secreta

Se denominan **criptosistemas simétricos** (o de **clave secreta**) a aquellos esquemas en que se utiliza la misma clave para encriptar y desencriptar.

En este punto requeriremos de una definición más formal.

Definición 1 (Criptosistema Simétrico) *Un criptosistema es una tupla* $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ *tal que:*

- $\mathcal{K}$ *es un conjunto de posibles claves.*
- $\mathcal{E} = (e_K)_{K \in \mathcal{K}}$ *es una lista de funciones de encriptación y* $\mathcal{D} = (d_K)_{K \in \mathcal{K}}$ *es una lista de funciones de desencriptación.*
- *Para toda clave* $K \in \mathcal{K}$, e_K: $\mathcal{P}_K \to \mathcal{C}_K$, d_K: $\mathcal{C}_K \to \mathcal{P}_K$ *y* $d_K(e_K(x)) = x$, *cualquiera sea* $x \in \mathcal{P}_\mathcal{K}$.

MODO DE EMPLEO DE UN CRIPTOSISTEMA

Veamos cómo Alicia y Roberto emplean un criptosistema. Primero eligen una clave $K \in \mathcal{K}$ (cuando no pueden ser observados por Eva[1]). Más tarde, cuando Alicia quiere enviar a Roberto el texto puro $x = x_1 \dots x_n$, $x_i \in \mathcal{P}_K$, $i = 1, \dots, n$, ella calcula $y_i = e_K(x_i)$, $i = 1, \dots, n$ y envía $y = y_1, \dots, y_n$. Una vez que Roberto recibe el texto cifrado y, este obtiene el texto puro x utilizando la función de desencriptación d_K, i.e. determina $x_i = d_K(y_i)$, $i = 1, \dots, n$.

Remark 1 *Por su modo de empleo, el criptosistema definido anteriormente se denomina* **criptosistema de bloque** (block cipher). *El nombre se debe a que cualquier mensaje a enviar debe dividirse en bloques que deben pertenecer al dominio de la función de encriptación que se esté utilizando.*

Los criptosistemas simétricos fueron los únicos tipos de criptosistemas desarrollado hasta antes del año 1976, generalmente en ámbitos militares. Dentro de esta categoría cabe mencionar el *One-Time Pad* y el *Data Encryption Standard* (DES). El *One-Time Pad* fue inventado por Gilbert Vernam en 1917. En este criptosistema la clave secreta es tan larga como el mensaje y cada carácter del mensaje es operado (sumado o restado) con un único y distinto carácter de la clave (para detalles ver [Sch96]). Su importancia se debe a que es posible demostrar su resistencia incondicional ante ataques de un adversario con conocimiento del texto cifrado.[2] Sin embargo, un evidente inconveniente de este criptosistema es la longitud de las claves requeridas. El *Data Encryption Standard* (DES) fue desarrollado en 1975 y, a diferencia del anterior, posibilita utilizar una misma clave para distintos mensajes, aunque no provee las garantías explícitas de seguridad del One-Time Pad. El criptosistema DES será tratado en profundidad más adelante.

[1] Notar que en este tipo de criptosistemas, las partes que desean comunicarse siempre deben acordar, en forma *previa a la comunicación*, el secreto (o clave) K que les permita desencriptar los mensajes posteriormente.

[2] Siempre y cuando la clave no se utilice para encriptar más de un mensaje.

2.1.3 Criptografía Moderna

Criptografía de Clave Pública

El año 1976, W. Diffie y M. Hellman producen una "revolución en la criptografía" [DH76] al proponer el concepto de **criptografía asimétrica** o de **clave pública**. En este tipo de esquemas la clave usada para encriptación es distinta de la usada para desencriptación, y más aún, es computacionalmente infactible[3] deducir la segunda a partir de la primera. Más aún, la clave usada para la encriptación es pública (conocida por todos), mientras que la clave de desencriptación es secreta (sólo conocida por quien la generó, el destinatario del mensaje). Notar que esto es equivalente a decir que Roberto publica la especificación completa de su función de encriptación e_K. Luego de eso, Alicia o cualquier otro individuo puede enviar mensajes encriptados a Roberto usando e_K. Este último será la única persona capaz de desencriptar el texto cifrado usando su regla de desencriptación secreta d_K. Este tipo de criptosistema facilita la comunicación entre partes que no pueden pre-acordar una clave, pues cualquiera de ellas puede encriptar usando la parte *pública* de la clave (e_K), pero sólo quien posea la correspondiente clave de desencriptación (parte *privada*) puede obtener el mensaje original.

Diffie y Hellman no dieron un ejemplo de criptosistema de este tipo. Ellos propusieron que podía construirse tal criptosistema si era posible encontrar una función "fácil" de calcular, pero "difícil" de invertir para cualquiera excepto quien posea cierta información privada denominada *puerta secreta* (para más detalles ver [Sti95, BG96]). Es por ello que la primera realización de un criptosistema público se debe a Rivest, Shamir y Adleman [RSA78] y se conoce como **RSA**. Otros ejemplos de este tipo de criptosistemas son **Rabin**, **ElGamal** y de **Curvas Elípticas**, entre otros. Para más detalles y ejemplos, ver [Sti95, Sch96].

La criptografía de clave pública ha posibilitado que una gran variedad de acciones y conceptos comunes de la vida real se hagan posible en el ámbito virtual, donde su extensión no era fácil o clara. Ejemplo de ello son las firmas digitales (que permiten certificar la procedencia de un mensaje), la distribución de claves (posible ahora entre personas que no necesitan acordar una clave previamente[4]), el dinero electrónico (que permite conservar las propiedades intuitivas del dinero real, como *no-duplicabilidad* y *usabilidad única*) y el voto electrónico (donde se espera obtener la certeza que el voto personal no ha sido *revelado* pero que efectivamente ha sido *contabilizado*), por mencionar los más notables.

El dramático aumento en la capacidad y la interconectividad de los computadores modernos ha llevado a esta redefinición de objetivos y ámbito de la criptografía. Tan decisivo ha sido este cambio hacia nuevas áreas que, además de los ejemplos mencionados, también la criptografía ha ocasionado nuevos desarrollos en aplicaciones tan diversas como:

[3] Informalmente una operación es computacionalmente infactible si, con una cantidad limitada de recursos es imposible realizarla en menos de un período de tiempo *razonable* (por ejemplo, menor que *500 años*).

[4] Pero que sin embargo necesitan estar seguros de la identidad de la otra persona (por ejemplo, usando certificados o un medio de broadcasting seguro). Ver discusión en [MvOV97].

- La protección de la propiedad intelectual: usando técnicas de verificación de integridad (validar que un mensaje no ha sido modificado) o de sellos de agua (*watermarking* o asociación inseparable entre autor y mensaje digital).
- La adopción de responsabilidades compartidas en el uso de armas de destrucción masiva: a través de técnicas de distribución de secretos (clave distribuida entre varias partes, sólo en conjunto es posible usarla).
- El patentar ideas/productos o probar conocimiento de *verdades* sin revelar *detalle alguno*. Esto es posible usando técnicas de divulgación nula (*zero–knowledge* [Gol95]), las cuales permiten convencer a una contraparte que se posee una información sin revelar ningún detalle de ella.

Y hay más. Todo ello muestra la relevancia de la criptografía moderna en áreas previamente impensadas. Su estudio pues, no sólo es académica y profesionalmente necesario sino claramente imprescindible.

2.1.4 Criptoanálisis

El criptoanálisis es el estudio de las técnicas (usualmente matemáticas) para intentar impedir el correcto funcionamiento de un criptosistema. Informalmente, *quebrar* un criptosistema es lograr impedir la obtención de uno de sus objetivos. Por ejemplo, si encriptación es utilizada para garantizar privacidad, un criptoanalista intentará recuperar el texto puro a partir de texto cifrado, o bien, deducir la clave de encriptación. Un *ataque* a un criptosistema es la ejecución de una secuencia de pasos cuya finalidad es quebrarlo. Un supuesto fundamental en criptoanálisis es que el criptoanalista tiene completo conocimiento de los detalles e implementación del algoritmo. Esto es conocido como el *supuesto de Kerckhoffs*,[5] y a lo largo del tiempo, ha demostrado ser una razonable premisa, en cuanto la historia ha demostrado que mantener la especificación y los detalles del algoritmo secretos por largo tiempo es infactible (para una mayor discusión, ver [Kah67, págs. 235–236]).

Clasificación de Ataques

Los ataques criptoanalíticos pueden clasificarse de muchas maneras posibles. De las clasificaciones usuales, la más clásica distingue varios niveles dependiendo de los recursos disponibles por el adversario. En lo que sigue, el objetivo de los ataques es sistemáticamente recuperar el texto puro a partir de (nuevo) texto cifrado, o más drásticamente, deducir la clave de encriptación utilizada.

[5] Enunciado por el militar holandés Auguste Kerckhoffs en [Ker83].

1. **Ataque de texto cifrado solamente:**
   El adversario (o criptoanalista) trata de deducir la clave de encriptación o el texto puro examinando solamente el texto cifrado. Un esquema de encriptación vulnerable ante este tipo de ataque es considerado totalmente inseguro.

2. **Ataque de texto puro conocido:**
   El adversario dispone de una cierta cantidad de pares de texto puro/texto cifrado donde todos los textos cifrados han sido generados con la misma clave. Su objetivo es deducir dicha clave o un algoritmo para desencriptar nuevos mensajes encriptados con la misma clave.

3. **Ataque de texto puro escogido:**
   El adversario elige el texto puro y luego le son dados los textos cifrados correspondientes. A continuación, el adversario usa cualquier información deducida a fin de recuperar textos puros correspondientes a nuevos textos cifrados (o bien la clave).[6]

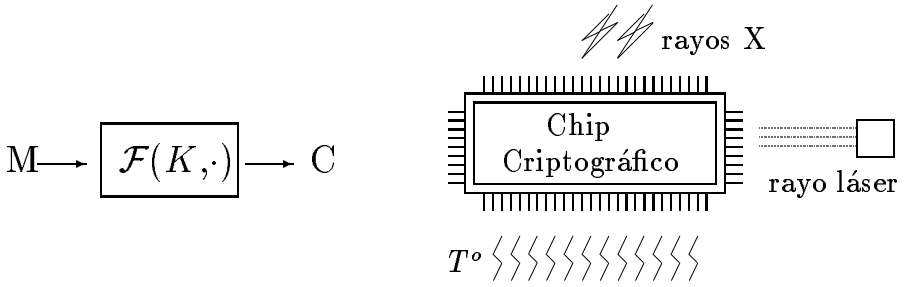
4. **Ataque de texto puro escogido (adaptativamente):**
   Este ataque es una subclase del ataque anterior. Es esencialmente un ataque de texto puro escogido donde la elección del texto puro puede depender del texto cifrado recibido (y analizado) previamente.

5. **Ataque de texto cifrado escogido:**
   El adversario elige el texto cifrado y luego le son dados los textos puros correspondientes. A continuación, y utilizando la información deducida, el adversario intenta recuperar los textos puros de (diferentes) textos cifrados (o bien la clave).

6. **Ataque de texto cifrado escogido (adaptativamente):**
   Este ataque es una subclase del ataque anterior. Es esencialmente un ataque de texto cifrado escogido donde la elección del texto cifrado puede depender del texto puro recibido (y analizado) previamente.

Un ataque se denomina de *búsqueda exhaustiva* o de *fuerza bruta* si implica examinar cada una de las posibilidades disponibles en el criptosistema. Por ejemplo, un ejemplo clásico de ataque de fuerza bruta es tratar de deducir el texto puro a partir de un texto cifrado probando todas las posibles claves hasta encontrar un texto puro significativo. Típicamente el mínimo requerimiento impuesto sobre un criptosistema es *resistir* un ataque de fuerza bruta. En este contexto, *resistir* un ataque significa que este debe ser *infactible*, es decir, imposible de realizar (computacionalmente) en un tiempo razonable.

[6] Tanto este como el ataque de texto puro conocido no son tan infactibles como podría pensarse. En particular, dispositivos criptográficos como tarjetas inteligentes, teléfonos seguros y chips de control de TV–pagada, permiten montar un ataque de este tipo.



(a) Criptoanálisis clásico (b) Nuevo enfoque criptoanalítico

Figura 2.1:

Nuevo Enfoque

A lo largo del tiempo, la criptografía moderna ha defendido el diseño de criptosistemas basados en principios matemáticos sólidos. Por esta razón es que muchos de los criptosistemas diseñados durante las últimas dos décadas han conseguido resistir muchas técnicas criptoanalíticas nuevas y sofisticadas basadas en propiedades matemáticas (siempre que se esté dispuesto a aceptar algunos supuestos razonables). Estas técnicas criptoanalíticas, han explotado ciertas debilidades o fallas en el diseño de los algoritmos de dichos criptosistemas. Estas debilidades usualmente han sido condiciones (matemáticas) especiales en los algoritmos que permiten recuperar parcial o totalmente información privada usada en él.

Sin embargo, a partir de [Koc96] un nuevo tipo de ataques mostró que existía información de alto valor criptoanalítico que no estaba siendo utilizada. Esta información podía venir por medios usualmente considerados "secundarios" o de poca trascendencia: mediciones de tiempo [Koc96] o el comportamiento de criptosistemas ante *pequeños* errores internos (posiblemente inducidos)[BDL97, BS97]. Lo novedoso de este enfoque radica en el nuevo concepto de criptosistema a considerar. Clásicamente (hasta antes de [Koc96]) un criptosistema era considerado como una entidad abstracta, con un funcionamiento específico (un algoritmo) y con un conjunto de posibles datos de entrada y datos de salida a examinar (esto es esquematizado en la Fig. 2.1(a)). En este nuevo enfoque sin embargo, un criptosistema es considerado como un dispositivo al cual podemos hacer funcionar igual que antes pero al cual ahora podemos tener acceso. Esto es, podemos observar su comportamiento *mientras funciona*. Esto significa que ahora podemos medir tiempos [Koc96] o consumo eléctrico [KJJ98], por ejemplo (ver Fig. 2.1(b)). Más aún, podemos hacerlo funcionar en condiciones para las cuales no fue diseñado y, en función de su respuesta, recuperar información privada. Esto permite implementar ataques que explotan los aspectos ingenieriles de la implementación de los criptosistemas. Estos ataques son descritos en más detalle en la siguiente sección.

2.2 Antecedentes del Ataque Evaluado

En esta sección describiremos varios ataques criptoanalíticos propuestos en la literatura.

2.2.1 Ataques de Tolerancia a Fallas

Ataques en Criptosistemas Asimétricos

Recientemente, Boneh, Lipton y DeMillo [BDL97] introdujeron el concepto de *ataque de tolerancia a fallas* (*fault tolerant attacks*). Estos ataques explotan fallas (posiblemente inducidas) del hardware criptográfico. En particular, en [BDL97] se muestra como utilizar estas fallas para obtener información privada de ciertos sistemas de firmas (RSA y Rabin), así como de protocolos de autenticación como Fiat–Shamir y Schnorr [Sti95]. Con ello, Boneh et al. señalaron el peligro que las fallas de hardware significan para varios esquemas basados en criptografía asimétrica y concluyeron que aún sofisticados sistemas criptográficos incorporados dentro de dispositivos resistentes a examen interno (*tamper–resistant devices* o REI) podrían revelar información acerca de las claves secretas.

Remark 2 *Algunos detractores consideraron el trabajo de Boneh et al. en [BDL97] como "poco novedoso", pues ya se conocía que las fallas podían significar una amenaza en la implementación de un criptosistema. Sin embargo, Boneh et al. fueron los primeros en describir ataques de este tipo potencialmente aplicables a cualquier modelo de dispositivo y mostrar la considerable magnitud de la información así obtenida.*

Ataques en Criptosistemas Simétricos

Una nueva veta de los ataques de tolerancia a fallas, el *análisis diferencial de fallas* (*differential fault analysis* o ADF) fue propuesto por Biham y Shamir en [BS97]. Su ataque es aplicable casi a cualquier criptosistema propuesto hasta ahora en la literatura pública. El ADF funciona bajo varios modelos de fallas y usa técnicas criptoanalíticas para recuperar información criptográfica secreta almacenada en dispositivos REI. En particular, Biham y Shamir mostraron que bajo el mismo modelo de fallas de hardware considerado por Boneh et al. una clave de DES de 16 iteraciones puede ser extraída desde un dispositivo REI de encriptación DES mediante el análisis de entre 40 y 200 textos cifrados generados desde textos planos desconocidos pero relacionados. Más aún, en [BS97] se desarrollan técnicas para identificar las claves de criptosistemas simétricos completamente desconocidos incorporados dentro de dispositivos REI.

ENTRADA: $y \in \mathbb{Z}$
CÓDIGO: $z = 1$
sea $k_l \cdots k_0$ **descomposición binaria de** k
para $i = \ell$ **hasta** 0 **hacer**
$z = z^2 \bmod n$
si $k_i = 1$ **entonces** $z = z \cdot y \bmod n$
SALIDA: z.

Figura 2.2: Exponenciador modular.

2.2.2 Ataques de Medición de Tiempos

En [Koc96], Paul Kocher introdujo un nuevo tipo de ataque criptoanalítico denominado *ataque de medición de tiempos*. Éste está basado en la observación que usualmente el tiempo tomado por los criptosistemas en operaciones criptográficas varía ligeramente al procesar distintos datos de entrada. Kocher dio varias explicaciones posibles a este comportamiento, como por ejemplo instrucciones condicionales y de salto, colisiones en el caché de memoria, instrucciones del procesador que corren en un tiempo no constante, etc. La contribución más importante de Kocher fue mostrar que las diferencias en el tiempo de ejecución pueden ser explotadas para encontrar algo de la información privada de un criptosistema examinado específico (*sistema objetivo*). De hecho, [Koc96] muestra claramente que ciertas *decisiones de implementación* de un criptosistema (por ejemplo en software) pueden ser utilizadas para recuperar información privada de él. Más concretamente, en [Koc96] se muestra cómo criptoanalizar un exponenciador modular simple.

La exponenciación modular es una operación crucial tanto en el protocolo de intercambio de claves de Diffie y Hellman [DH76] como en el criptosistema RSA [RSA78]. Un exponenciador modular es un procedimiento que, teniendo como entrada un cierto valor $y \in \mathbb{Z}$ y dos números k y $n \in \mathbb{N}$, con $n \neq 0$, calcula $(y^k \bmod n)$, es decir, el menor residuo no-negativo de y elevado a k. (En los protocolos criptográficos mencionados anteriormente, n es público y k es privado. Asimismo, típicamente k y n son fijos, mientras que y es variable.) En [Koc96] se mostró que si un adversario puede medir el tiempo que le toma a un sistema específico calcular $(y^k \bmod n)$ para varios y's, entonces podrá inferir el valor del exponente secreto k. Más aún, el costo computacional de este ataque es proporcional a la cantidad de trabajo (computacional) efectuada por la víctima. A fin de clarificar y hacer más concreta esta exposición, a continuación describiremos en forma básica la idea del método de [Koc96]. Para ello analizaremos el exponenciador modular mostrado en la Fig. 2.2. (El ataque es similar en otras variaciones del exponenciador modular.)

El ataque permite a alguien que conozca $k_l \cdots k_t$ recuperar k_{t-1}. (Para obtener el exponente entero el atacante comienza con $t = l + 1$ y repite el ataque hasta $t = 1$.) El atacante primero efectúa las $l - t + 1$ iteraciones del ciclo **for**. La siguiente iteración requerirá el primer bit desconocido k_{t-1}. Si este bit es no nulo, la operación $(z = z \cdot y \bmod n)$ es efectuada,

de lo contrario, es saltada. Supongamos ahora que cada medición de tiempo corresponde a una observación de la variable aleatoria $T = e + \sum_{i=0}^{l-1} T_{l-i}$ donde T_{l-i} representa el tiempo requerido por las multiplicaciones y cuadrados correspondientes al bit k_{l-i} y e una variable aleatoria que representa el error de medición, la sobrecarga del ciclo, etc. Un espía que correctamente adivina k_{t-1} puede eliminar el efecto de los $T_l, \ldots, T_{t-1}$ en T y obtener una variable aleatoria reajustada de varianza conocida (siempre que los tiempos necesarios para efectuar multiplicaciones modulares sean independientes entre sí y del error de medición). Adivinanzas incorrectas producirán una variable aleatoria reajustada de varianza mayor a la esperada. Notemos que el cálculo de la varianza será fácil siempre que el adversario pueda obtener suficientes mediciones de tiempo. De esta manera, la adivinanza correcta será identificada exitosamente cuando sus valores reajustados tengan una varianza pequeña. Esta idea es refinada más profundamente en [Koc96].

Este trabajo establece que, en teoría, los ataques de medición de tiempos pueden proveer algo de la información privada de un sistema objetivo. Sin embargo, no está claro que llevar a cabo un ataque de medición de tiempos sea realmente factible. (¿En cuáles ambientes es posible efectuar las muy precisas mediciones requeridas por un ataque de medición de tiempos?) Por ejemplo, al intentar efectuar un ataque de medición de tiempos en un sistema remoto (esto es, accesible sólo por la red), se debe solucionar el obstáculo que significan los retardos aleatorios en la red. Evaluar la magnitud de un inconveniente como éste no es obvio y requiere algunas consideraciones estadísticas [Ros88, pág. 368]. Para ello, notemos que tanto los retardos aleatorios como el proceso repetitivo de medición de un mismo tiempo de ejecución, siguen típicamente una distribución normal. Por ello, su suma también será normal con una varianza igual a la suma de las varianzas de las mediciones de tiempo y de los retardos aleatorios. Por otra parte, el estimador usado al calcular la media de las mediciones de tiempo, tiene una varianza igual a la varianza de los valores observados dividida por el número de valores. Esto obliga a recolectar una mayor cantidad de mediciones de tiempo para compensar la imprecisión obtenida. Lamentablemente, esto no necesariamente es suficiente para garantizar el ataque. Es perfectamente posible que la cantidad de mediciones aumente hasta tornar el ataque totalmente impráctico.

Sin embargo, hay ciertas situaciones en las que sentimos que es realista montar un ataque de medición de tiempos: los protocolos de autenticación implementados usando tarjetas inteligentes. A continuación describiremos las características de este tipo de tarjetas y protocolos, así como su potencial vulnerabilidad ante un ataque de medición de tiempos.

2.3 Escenario del Ataque: Tarjetas Inteligentes

2.3.1 Descripción

Una tarjeta inteligente es una tarjeta plástica, del tamaño y forma de una tarjeta de crédito, con un chip de computador incluido.[7] Es una vieja idea – las primeras patentes fueron registradas hace más de 20 años – pero limitaciones prácticas la hicieron posible sólo hace pocos años. Actualmente muchos países utilizan tarjetas inteligentes para servicios telefónicos (sistemas GSM de telefonía móvil), para el control de TV–pagada, para el acceso a computadores remotos y, más recientemente, como billeteras electrónicas.

Una tarjeta inteligente contiene un pequeño computador (usualmente un microprocesador de 8 bits de no mas de 20 mm^2), memoria RAM (cerca de un cuarto de kilobyte), memoria ROM (cerca de 6 u 8 de kilobytes) y memoria EPROM (*Electrically Programmable Read–Only Memory*), o bien, EEPROM (*Electrically Erasable Programmable Read–Only Memory*) donde esta última puede contener unos pocos kilobytes donde es mantenida la información privada (las claves). La tarjeta tiene su propio sistema operativo, programas y datos. No posee alimentación propia; ésta es obtenida cuando es conectada a un lector de tarjetas.

Las tarjetas inteligentes pueden implementar diferentes protocolos y algoritmos criptográficos programados en ellos. En particular, pueden funcionar como billeteras electrónicas (recibiendo y gastando análogos de dinero), realizar protocolos de autenticación y almacenar sus propias claves secretas.

Sin embargo, las tarjetas inteligentes son susceptibles a una amenaza adicional. Un potencial adversario (un ladrón de tarjetas o, incluso un usuario inescrupuloso) puede lograr un completo y libre (no–supervigilado) acceso a este tipo de tarjetas para tratar de recuperar o alterar la información privada. Por ejemplo, puede aumentar el monto de dinero en una billetera electrónica. Por ello, se requiere que sean resistentes a examen interno, lo cual significa que un intento de ver o modificar su contenido de memoria o circuitos debería ocasionar una inmediata y completa inutilización de la tarjeta. Así, para prevenir acceso directo a los circuitos básicos este tipo de tarjeta puede implementar sensores capacitivos para detectar la presencia continua de la cubierta de protección (*pasivation layer*) o bien sensores ópticos bajo una cubierta opaca, por ejemplo.[8] Se sabe, sin embargo, que estas medidas no siempre son suficientes para protegerlas de adversarios con recursos [AK96]. Pese a ello, se espera que mejoras en la tecnología de estas tarjetas permitan su uso seguro en ámbitos tan diversos como el almacenamiento de información personal (identificación, médica, etc.) y control de acceso a automóviles y edificios, entre otros.

[7] Una extensa descripción e historia de estas tarjetas y su desarrollo puede encontrarse en [Sim92, cap. 12].

[8] Un excelente y detallado artículo acerca de posibles ataques (de nivel físico) y medidas de protección para tarjetas inteligentes puede encontrarse en [AK96].

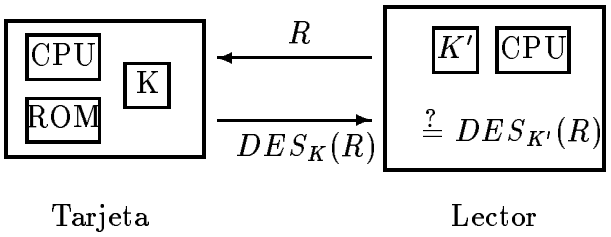


Figura 2.3: Protocolo simétrico tipo reto–respuesta en tarjetas inteligentes.

2.3.2 Funcionamiento

Usualmente una tarjeta inteligente almacena información privada que le permite realizar transacciones con una contraparte. En algunas tarjetas, y a fin de obtener mayor seguridad, la tarjeta intenta primero identificar al poseedor de ella como un usuario válido, usualmente solicitando el ingreso de un número secreto (*personal identification number* o PIN) y comparándolo con un número almacenado internamente. Sin embargo, dado que esto requiere de un pequeño teclado en la tarjeta (y simplemente, por facilidad de uso), no es una opción común en tarjetas orientadas a transacciones de pequeños montos (llamadas de teléfonos, por ejemplo).

Una tarjeta inteligente debe autenticarse hacia el lector con el cual realizará la transacción. Existen varios protocolos estándares para este efecto. En este trabajo nos concentraremos en los protocolos de autenticación que utilizan criptografía simétrica, en particular, el criptosistema DES. Ejemplos de ellos son los protocolos ANSI X9.26 y ISO/IEC 9798–2 (ver [MvOV97, págs. 651–652]).[9] Dichos protocolos, denominados *protocolos de reto–respuesta* operan de la siguiente manera:

1. El lector de tarjetas le envía un *reto* a la tarjeta, usualmente constituido por un número al azar.

2. La tarjeta utiliza este valor para realizar un cálculo predeterminado. Este consiste en encriptar este valor utilizando su clave secreta (posiblemente junto a otro número al azar y un identificador del destinatario, en este caso, el lector de tarjetas). Luego envía este resultado al lector.

3. El lector por su parte calcula la encriptación de los mismos valores utilizando su propia clave secreta. Luego de ello, contrasta el valor recibido con el valor generado y acepta si son idénticos.[10] Claramente, a fin generar el resultado correcto para dicho cálculo, el

[9] El estándar ISO, sin embargo, a pesar de inspirarse en el uso de DES como criptosistema, ya no lo hace obligatorio. El estándar ANSI, sin embargo, requiere explícitamente el uso de DES.

[10] Alternativamente, el lector puede desencriptar el valor recibido y comparar este resultado con el número al azar enviado originalmente.

otro dispositivo deberá poseer la clave secreta correcta y, por lo tanto, podrá suponerse que es auténtico.

4. Opcionalmente, los roles entre la tarjeta inteligente y el lector se invierten para autenticar ahora al lector.

Este proceso es esquematizado en la Fig. 2.3. Para una mayor discusión acerca de los protocolos de reto–respuesta que utilizan criptosistemas simétricos ver [MvOV97, pág. 401].

Este escenario presenta una situación ideal para el ataque propuesto en este trabajo. Debido a la interactividad de los protocolos reto–respuesta, la realización de mediciones precisas del tiempo del encriptación de la tarjeta es claramente factible para un adversario común. Por ejemplo, un esquema simple y altamente preciso para recuperar mediciones de tiempo consistiría en monitorear la comunicación entre una tarjeta y su lector durante el proceso de autenticación. Esto podría lograrse fácilmente mediante mínimas modificaciones en un lector de tarjetas y permitiría registrar los tiempos de encriptación de todas las tarjetas utilizadas en el lector.

Capítulo 3

El Estándar de Encriptación de Datos

3.1 Introducción

En Mayo de 1973 la Oficina Nacional de Estándares (*National Bureau of Standards*, NBS) realizó un llamado público solicitando propuestas para un algoritmo criptográfico estándar. Este llamado finalmente llevó al desarrollo del Estándar de Encriptación de Datos (*Data Encryption Standard*) o **DES**, el cual ha llegado a ser el criptosistema más utilizado en el mundo. DES fue desarrollado en IBM, como una modificación de un criptosistema previo conocido como LUCIFER. DES fue publicado por primera vez en el registro federal [Nb77] del 17 de Marzo de 1975. Después de considerable discusión pública, DES fue adoptado como un estándar (norteamericano) para aplicaciones "no–clasificadas" en Enero de 1977. Asimismo en 1981 fue aprobado como un estándar de la ANSI (*American National Standard Institute*) siendo llamado el Data Encryption Algorithm (DEA). DES ha sido revisado por la NBS aproximadamente cada cinco años desde su adopción. Su más reciente revisión fue en Enero de 1994, siendo renovado como estándar hasta 1998. Pese a que DES ha sobrevivido (casi) incólume a más de veinte años de insistente criptoanálisis, se ha anticipado que no seguirá como estándar con posterioridad a 1998.[1] Esto es principalmente porque ha comenzado a mostrar algunos signos de vejez.[2]

A pesar de ello, DES constituye en la actualidad un paradigma de fortaleza dentro de los criptosistemas simétricos. Tal reputación se debe a la mencionada resistencia a sofisticadas técnicas criptoanalíticas, resistencia basada en un algoritmo de cuidadoso diseño. Al respecto, es interesante notar que los criterios de diseño de DES fueron clasificados durante más de una década. Actualmente es conocido ([Cop92, Cop94]) que dichos criterios se orientaron principalmente a contrarrestar el *criptoanálisis diferencial* (discutido más adelante).

El algoritmo de DES es revisado a continuación. Para una descripción más detallada ver [Sti95, MvOV97].

[1] Sin embargo, la adopción de un nuevo estándar podría tomar varios años en concretarse (ver [Ni97]).

[2] Principalmente su inflexibilidad en el largo de la clave. Se considera que los 56 bits de la clave son actualmente insuficientes ante ataques de fuerza bruta.

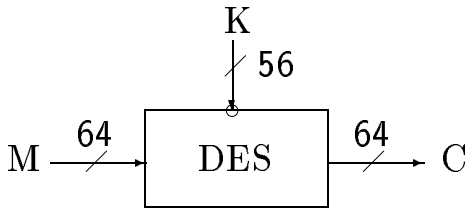


Figura 3.1: Entrada y Salida de DES.

3.2 Descripción de DES

Una completa descripción de DES aparece en [Nb77]. Descripciones más detalladas pueden encontrarse en [Sch96, Sti95].

DES es un criptosistema de bloque; recibe como entrada un bloque M de 64 bits (*texto puro*) y usando una clave K de 56 bits entrega un bloque C de 64 bits (*texto cifrado*). Esto es ilustrado en la Fig. 3.1.

El algoritmo consta de tres etapas:

1. Dado un texto puro x, una secuencia de bits x_0 es construida permutando los bits de x de acuerdo a una permutación inicial fija[3] IP. Escribiremos $x_0 = IP(x) = L_0R_0$, donde L_0 comprende los primeros 32 bits de x_0 y R_0 los últimos 32.

2. A continuación, 16 iteraciones de una cierta función son calculadas. Cada L_iR_i, para $1 \leq i \leq 16$, se calcularán de acuerdo a la siguiente relación:

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

   donde $\oplus$ denota el *OR exclusivo* (XOR) entre dos secuencias de bits. La función f será descrita más adelante. Las *subclaves* K_1, K_2, ..., K_{16} son secuencias de 48 bits cada una calculadas en función de la clave K (de hecho, cada K_i es una selección permutada de los bits de K). K_1, K_2, ..., K_{16} constituyen los elementos del *vector de subclaves* (o *Key Schedule*).

3. Finalmente se aplica la permutación inversa IP^{-1} a la secuencia de bits formada por $R_{16}L_{16}$ obteniendo el texto cifrado y. Esto es, $y = IP^{-1}(R_{16}L_{16})$. Observar la inversión de orden de L_{16} y R_{16}. En la Fig. 3.2 es esquematizado el funcionamiento de DES.

[3] DES utiliza varias tablas de permutaciones y sustituciones. Su detalle no es relevante para el entendimiento del presente trabajo, por lo que recomendamos al lector interesado remitirse a [Sti95].

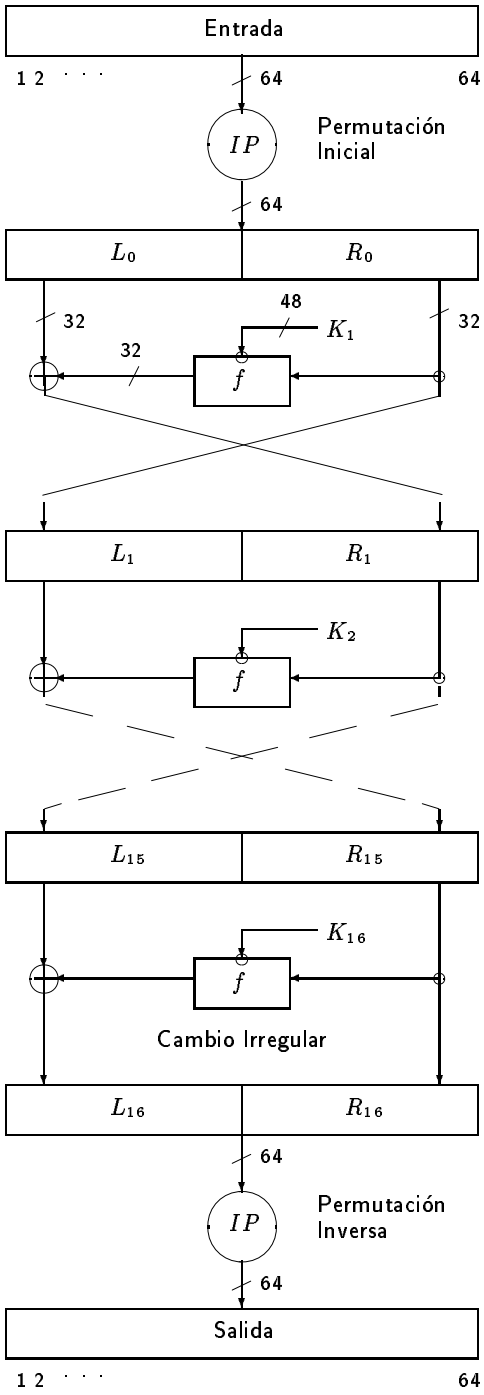


Figura 3.2: Funcionamiento de DES.

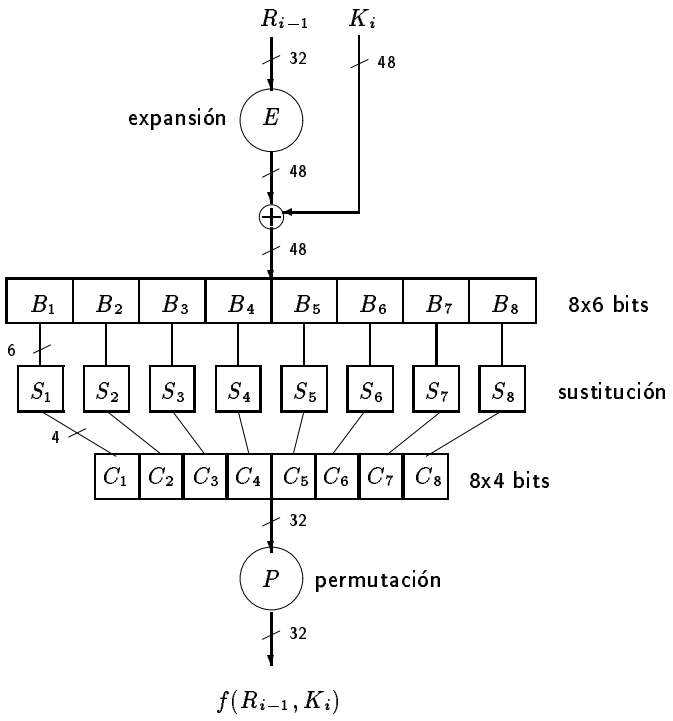


Figura 3.3: Función interna de DES.

La función f toma como entrada un primer argumento R_{i-1}, (una secuencia de bits de largo 32), y un segundo argumento K_i, (una secuencia de largo 48), y produce como salida una secuencia de bits de largo 32. Dentro de esta función se ejecutan los siguientes pasos:

1. El primer argumento R_{i-1} es "expandido" a una secuencia de bits de largo 48 de acuerdo a una *función de expansión* (fija) E. En particular, $E(R_{i-1})$ consiste de los 32 bits de R_{i-1}, permutados de una cierta manera, junto a 16 bits repetidos.

2. Se calcula $E(R_{i-1}) \oplus K_i$. Escribimos el resultado como la concatenación de ocho secuencias de 6 bits cada una. Esto es $B = B_1B_2B_3B_4B_5B_6B_7B_8$.

3. El siguiente paso usa ocho *cajas–S* (denominadas *S–boxes*) $S_1, \ldots, S_8$. Cada S_i es una tabla fija de 4×16 celdas cuyas entrada es un entero entre 0 y 15. Dada una secuencia de largo 6, digamos $B_j = b_1b_2b_3b_4b_5b_6$, calculamos el resultado $S_i(B_j)$ de la caja S_i como sigue. Los bits b_1b_6 determinan la representación binaria de una fila r $(0 \leq r \leq 3)$ en la tabla S_i, y los cuatros bits restantes $b_2b_3b_4b_5$ determinan la representación binaria de una columna c $(0 \leq c \leq 15)$. Entonces, el resultado $S_i(B_j)$ será la celda $S_i(r, c)$, escrita como una secuencia binaria de largo 4. (Por lo tanto, cada S_j puede ser vista como una función que acepta como entrada una secuencia de dos bits y una de cuatro bits, y produce una secuencia de bits de largo cuatro). De esta manera, se calculan $C_j = S_j(B_j)$, $1 \leq j \leq 8$.

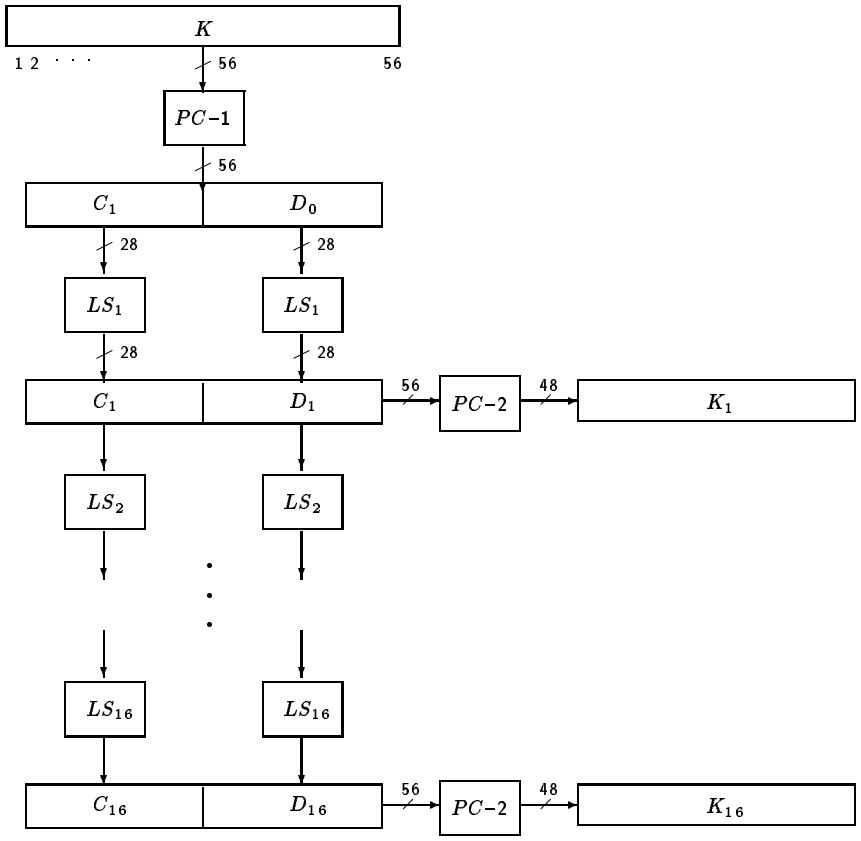


Figura 3.4: Generación del vector de subclaves de DES.

4. La secuencia $C = C_1C_2C_3C_4C_5C_6C_7C_8$ de largo 32 es permutada de acuerdo a una permutación fija P. La secuencia de bits resultante $P(C)$ será $f(R_{i-1}, K_i)$.

La función f es esquematizada en la Figure 3.3. Fundamentalmente consiste de una sustitución (usando las caja–S) seguida de una permutación (fija) P. Las 16 iteraciones de f conforman así un *criptosistema producto*.[4]

Finalmente, describiremos a continuación el cálculo del vector de subclaves a partir de la clave K. En realidad, K es una secuencia de bits de largo 64, de los cuales 56 conforman la clave y el resto, ocho bits, son bits de comprobación de paridad.[5] Estos ocho bits son ignorados en el cálculo del vector de subclaves.

1. Dada una clave K de 64 bits, se descartan los bits de paridad y se permutan los bits

[4] Este es un tipo de criptosistema que combina dos o más transformaciones con el objeto de (intentar) obtener un sistema más seguro que cada uno de los componentes. En el caso de DES son 16 iteraciones de la misma función f.

[5] Esto es, los bits en posiciones $8, 16, \ldots, 64$ son definidos de manera que cada *byte* contenga un número impar de 1's.

restantes de K mediante una permutación (fija) PC–1. Al resultado lo denotamos por C_0D_0, donde C_0 contiene los primeros 28 bits de PC–1(K) y D_0 los últimos 28.

2. Para i variando entre 1 y 16, se calcula

$$\begin{aligned} C_i &= LS_i(C_{i-1}) \ , \\ D_i &= LS_i(D_{i-1}) \ , \end{aligned}$$

donde LS_i representa un desplazamiento cíclico (*shift*) a la izquierda, en una o dos posiciones, dependiendo del valor de i; LS_i desplaza *una* posición si $i = 1, 2, 9, 16$, y *dos* posiciones en los otros casos. La subclave K_i se calcula permutando C_iD_i nuevamente, esto es $K_i = PC$–2(C_iD_i) donde PC–2 es otra permutación fija.

La Fig. 3.4 esquematiza el proceso de generación de subclaves descrito.

La desencriptación es efectuada usando el mismo algoritmo de la encriptación, pero utilizando los elementos del vector de subclaves $K_{16}, \ldots, K_1$ en orden inverso. El resultado será el texto puro x.

3.3 Ataques contra DES

DES ha resistido notablemente bien más de dos décadas de concienzudo e insistente criptoanálisis. Sin embargo, en la actualidad, procesadores más rápidos y baratos permiten construir máquinas específicamente diseñadas, que a un precio razonable, pueden recuperar una clave de DES en unas pocas horas [Sti95, págs. 82–83]. Esto se debe a su reducido tamaño de las posibles claves: 56 bits.

Varios ataques a DES han sido propuestos en la literatura. Entre los más notables podemos destacar el *criptoanálisis diferencial* de Biham y Shamir [BS91, BS93b] y el *criptoanálisis lineal* de Matsui [Mat94a, Mat94b].

3.3.1 Criptoanálisis Diferencial

En 1990, Eli Biham y Adi Shamir introdujeron el criptoanálisis diferencial [BS91, BS93b]. Usando este nuevo método de criptoanálisis, Biham y Shamir encontraron un ataque de texto escogido contra DES más eficiente que el ataque de fuerza bruta.

El criptoanálisis diferencial examina específicamente pares de textos cifrados cuyos textos puros poseen diferencias particulares. Esta técnica analiza la evolución de estas diferencias a través de las iteraciones de DES considerando que dichos textos puros son encriptados usando la misma clave.

Inicialmente se escogen pares de textos puros con una diferencia fija. Más aún, los dos textos puros pueden ser elegidos al azar siempre que satisfagan una diferencia especificada. (Para DES, la "diferencia" se define como el XOR entre los valores.) Entonces, usando las diferencias en los textos cifrados resultantes, se asignan probabilidades distintas a las

distintas claves. La idea es que, a medida que una mayor cantidad de pares de textos cifrados van siendo analizados, una clave emergerá como la más probable. Esa será la clave buscada. En [BS93a] es posible encontrar una explicación detallada de esta técnica.

Usando este método, versiones de DES con un número reducido (menor que 16) de iteraciones han sido atacadas con éxito. Contra DES de 16 iteraciones, el mejor ataque logrado requiere 2^{47} textos puros escogidos (o bien 2^{55} textos puros conocidos [MvOV97, pág. 259] y $2^{55.1}$ operaciones [Sch96, págs. 289–290]).

Notemos que este ataque es principalmente teórico. La enorme cantidad de requerimientos en tiempo y espacio necesitados para montar este ataque lo deja fuera del alcance de la mayoría de los adversarios (para DES de 16 iteraciones requeriría procesar 1.5 megabits por segundo de texto plano elegido durante casi tres años). Por otra parte, este ataque es altamente dependiente de de la estructura de las cajas–S aunque puede potencialmente usarse contra cualquier criptosistema de cajas–S constantes. Es conocido, sin embargo, que en DES las cajas–S fueron optimizadas contra este ataque [Cop92, Cop94]; de allí proviene el hecho que, para DES de 16 iteraciones, este ataque es algo menos eficiente que un ataque de fuerza bruta (que para DES toma 2^{55} operaciones).[6]

3.3.2 Criptoanálisis Lineal

El criptoanálisis lineal es otro tipo de ataque criptoanalítico, desarrollado por Mitsuru Matsui [Mat94a, Mat94b]. Este ataque usa aproximaciones lineales para describir la acción de un criptosistema de bloque (por ejemplo DES).

La exacta aproximación lineal usada en este ataque es altamente dependiente de la estructura de las cajas–S. En particular, si se efectúa el XOR entre algunos de los bits del texto puro, y luego el XOR entre algunos de los bits del texto cifrado, para finalmente realizar el XOR entre ambos resultados, se obtendrá un único bit. Este bit será el XOR de algunos de los bits de la clave con alguna probabilidad p. Si $p \neq \frac{1}{2}$, este sesgo puede ser explotado recolectando textos puros y sus textos cifrados asociados para adivinar el valor de los bits de la clave. El aumentar la cantidad de datos disponibles mejora la aproximación hacia los valores correctos. Además, mientras mayor es el sesgo (i.e. mientras mayor es $| p - \frac{1}{2} |$) mayor es la tasa de éxitos con la misma cantidad de datos conocidos.

Usando varios refinamientos, este ataque contra DES de 16 iteraciones requiere en promedio de 2^{43} textos puros conocidos [Mat94a]. Se sabe que las cajas–S de DES no fueron optimizadas contra este ataque [Cop92, Cop94].

[6] Es posible que un ataque tome una cantidad de operaciones mayor que fuerza bruta. Esto es un claro indicador de la ineficacia del ataque.

Remark 3 *El criptoanálisis diferencial es, fundamentalmente, un ataque de texto escogido mientras que el criptoanálisis lineal es un ataque de texto puro conocido. Sin embargo, aún asumiendo que es factible obtener enormes cantidades de pares de textos escogidos (conocidos), el criptoanálisis diferencial y lineal no son considerados una amenaza para DES en la práctica. De acuerdo a [MvOV97, págs. 258–259] un ataque de fuerza bruta sobre un par de textos cifrado/texto puro es significativamente más factible (utilizando máquinas altamente paralelizadas) con chips DES.*

Capítulo 4

Ataque de Medición de Tiempos sobre DES

En este capítulo consideraremos el problema de recuperar, a través de un ataque de medición de tiempos, el peso de Hamming de una clave DES utilizada en un *sistema objetivo*.[1] Analizaremos también las condiciones en las cuales el peso de Hamming es recuperable y discutiremos su impacto en la seguridad de DES. En particular, nuestro análisis estará orientado a implementaciones en *software* del criptosistema DES.[2]

En la primera sección, mostraremos y analizaremos las características de tiempos encontradas de las implementaciones de DES analizadas y el grado de amenaza que esto significa. En particular, en la Sect. 4.1.2, mostraremos el tipo y grado de dependencia del tiempo de encriptación respecto a la clave usada en el criptosistema y luego describiremos el ataque, bajo la suposición que el adversario conoce el diseño del sistema objetivo. Posteriormente, en la Sect. 4.1.3, evaluaremos experimentalmente la magnitud de la amenaza implicada en este ataque, a través de una estimación del trabajo adicional necesario para descubrir la clave una vez realizado el ataque. Finalmente, en la sección 4.2, desarrollaremos una técnica que nos permitirá deducir, dadas características de tiempos como las aquí mostradas, y usando usando sólo mediciones de tiempo, toda la información requerida para efectuar exitosamente el ataque.

4.1 Características de Tiempos

La ventaja usual de los criptosistemas simétricos como DES sobre los criptosistemas asimétricos (como RSA, por ejemplo) es la velocidad con la que los mensajes son encriptados y des-

[1] Llamaremos *sistema objetivo* a un computador que implementa un criptosistema desde donde queremos extraer información.

[2] Sin embargo, un aspecto interesante es que algunas ideas fundamentales del presente enfoque pueden ser adaptados para el análisis de implementaciones en *hardware*. En particular, dependencias lineales en el peso de Hamming pueden explotarse usando el método de inferencia estadística propuestos en la sección 4.2.

encriptados (por ejemplo, DES es típicamente unas 1000 veces más rápido que RSA [Sti95, pág. 469]). Esto se debe en gran medida a que en los criptosistemas simétricos las operaciones han sido diseñadas y optimizadas para la manipulación directa de bits (lo que en particular es cierto para las implementaciones en software). Por ello, resulta natural suponer que el tiempo de ejecución de un criptosistema a lo más revelará algún indicador agregado de los bits manipulados (o bien sólo ruido). Sin embargo, la suposición hecha en [Koc96] va un paso más allá, descartando el efecto de los bits del mensaje manipulados y estableciendo como factor determinante el estado de los bits de la clave. (Por lo demás, el *Peso de Hamming* o número de bits no nulos constituye un indicador agregado del estado de los bits de la clave suficientemente razonable.) Por ello en los experimentos detallados a continuación se examinó la relación entre el tiempo de ejecución y el peso de Hamming de la clave usada en el criptosistema DES. En términos generales, los resultados observados de estos experimentos configurarán un tipo de relación puntual (lineal, cuadrática, independiente, etc.) que genéricamente denominaremos *característica de tiempos* del criptosistema implementado. Notemos que estas características son dependientes de la implementación, por lo que el análisis de implementaciones prácticas y representativas será necesariamente relevante.

A continuación, describiremos brevemente las implementaciones de DES analizadas y sus principales características.

4.1.1 Implementaciones Analizadas

Consideraciones de la Selección

Para evaluar la resistencia de un criptosistema como DES ante ataques de medición de tiempos fue necesario disponer de implementaciones (en software) de DES que cumplieran con lo siguiente:

1. **Correctitud:** Deben implementar el criptosistema DES correctamente. Más precisamente, cumplir con las especificaciones estipuladas en [Nb77, Nb81] y adicionalmente pasar las pruebas propuestas en [Riv85]. Esto porque, pese a que desde 1994, el NIST (National Institute of Standards and Technology) ha certificado implementaciones en software, sólo algunas implementaciones comerciales han seguido con esta norma.

2. **Factibilidad de usar en la práctica:** Deben ser lo suficientemente eficientes y simples de usar de manera de garantizar su utilización en la práctica.

3. **Representatividad:** Deben representar en alguna medida las implementaciones usuales utilizadas en el diseño de sistemas en la actualidad. Esto implica que serían implementaciones "conocidas" basadas en código público de fuentes conocidas (y por ende, exportables). Notemos que implícitamente esta representatividad busca garantizar la correctitud y facilidad de uso.

Descripción

La cantidad de implementaciones de DES disponibles era amplia. Sin embargo, dadas las restricciones mencionadas, se seleccionaron dos:

RSAEuro DES: Esta versión de DES se incluye dentro del paquete de herramientas criptográficas **RSAEuro** [Kap96]. El objetivo de los diseñadores de este *kit* fue proveer un paquete de funciones compatible con **RSAREF**.[3] Este último fue desarrollado por RSA Data Security por lo que posee una considerable reputación (sin embargo, las restricciones de exportación lo descalificaron para nuestras pruebas). A esta versión la denotaremos por RSA–DES.

Versión de DES de Antti Louko: Esta implementación de DES está disponible en forma gratuita, es eficiente y simple de utilizar. Esta implementación se encuentra lo suficientemente difundida en archivos públicos de software en Internet como para ser una elección típica. Fue desarrollada por **Antti Louko** [Lou92] bajo el nombre de `alodes`. A esta versión la denotaremos por L–DES.

Remark 4 *Un aspecto que será relevante en el análisis de las implementaciones es que el proceso de encriptación en ambas implementaciones está dividido en dos fases separadas: una de generación del vector de subclaves (*Key Schedule*) y otra de encriptación utilizando el vector de subclaves precalculado.*

El código fuente de estas implementaciones está disponible para pública evaluación en [Kap96, Lou92]. Un extracto relevante se incluye en el Apéndice C.

Ambiente de Ejecución

Ambas implementaciones fueron estudiadas en un computador Pentium™ de 120 MHz corriendo MSDOS™. La ventaja de trabajar en MSDOS™ es su característica de sistema operativo monoproceso. Esto facilita la obtención de mediciones de tiempo puesto que no hay otros procesos corriendo que puedan interferir y el número de tareas de mantención que el sistema operativo debe realizar es considerablemente menor.

En cuanto al proceso de medición de tiempos, la unidad de medición fue el microsegundo (μs) y fue posible llegar a medir tiempos de ejecución de hasta una precisión de $0.8381\mu s$. (Para una explicación de cómo medir tiempos con esta precisión en el sistema operativo MSDOS™ ver el Apéndice B.)

[3] Más precisamente, API (Application Programming Interface) compatible.

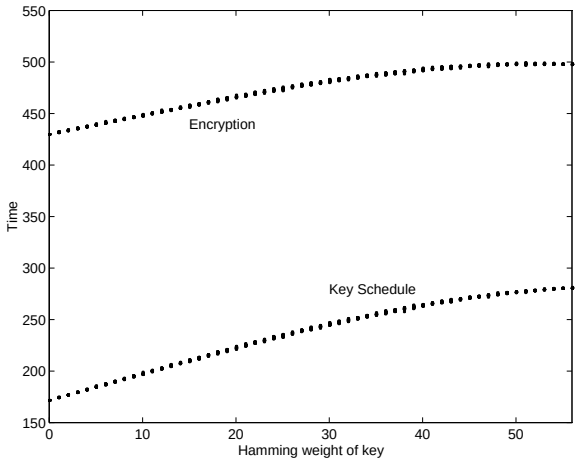


Figura 4.1: RSA-DES.

4.1.2 Dependencia en el Peso de Hamming

En nuestro primer experimento, fijamos la secuencia de 64 bits en cero como el mensaje de entrada. Para cada $i \in \{0, \ldots, 56\}$, elegimos al azar 32 claves de peso de Hamming igual a i. Para cada una de estas claves encriptamos el mensaje un total de 16 veces. Durante cada encriptación medimos el tiempo tomado en generar el vector de subclaves y el tiempo total en encriptar el mensaje (en μs). Los gráficos del promedio (para cada clave) de los tiempos de encriptación y de generación del vector de subclaves se muestran en la Fig. 4.1 y en la Fig. 4.2 (la tabla de valores se puede ver también en el Apéndice A.1.)

Sólo las mediciones fuera de rango[4] obvias fueron eliminadas. De hecho, las únicas mediciones fuera de rango observadas aparecieron a intervalos fijos de 2^{16} *ticks* de reloj. Dichas mediciones fueron causadas por tareas de mantenimiento del sistema (ver Apéndice B).

Notemos que una clave DES elegida al azar tiene un peso de Hamming entre 23 y 33 con una probabilidad de aproximadamente 0.86. Por ello, de los puntos mostrados en la Fig. 4.1 y la Fig. 4.2 los más relevantes son aquellos cerca de la mitad del gráfico.

A continuación, para varias claves elegidas al azar, efectuamos 2^{16} mediciones (para cada clave) del tiempo de encriptación y del tiempo de generación del vector de subclaves. Después de descartar los valores fuera de rango obvios, graficamos las distribuciones de frecuencia empírica de los datos recolectados. Las frecuencias empíricas observadas fueron aproximadamente simétricas y estaban concentradas en tres valores diferentes. Esta concentración en tres valores distintos se debe al hecho que fue sólo posible realizar mediciones de tiempo con una precisión de hasta $0.8381\mu s$. Lo anterior sugiere, tal como podría esperarse, que las variaciones en los tiempos de ejecución observados, cuando el mismo proceso

[4] Aquellas mediciones cuyo valor escapa ostensiblemente del rango de mediciones esperado y cuya desviación se sabe no es función de los factores observados.

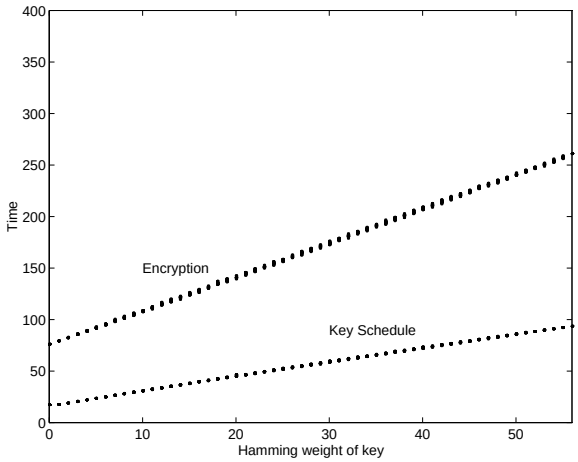


Figura 4.2: L-DES.

es ejecutado varias veces sobre la misma entrada, son debidas al efecto de ruido blanco, es decir, un error normalmente distribuido con media 0.

Seguidamente, para diferentes valores de $i \in \{8, \ldots, 48\}$ elegimos al azar 2^8 claves de peso de Hamming i. Luego de descartar los valores fuera de rango, graficamos la distribución de frecuencia empírica de los datos recolectados. Las frecuencias observadas se asemejaban a distribuciones normales con pequeñas desviaciones (típicamente $1.2\mu s$ para L–DES y $1.8\mu s$ para RSA–DES). Concluimos que las variaciones de los tiempos de encriptación y de generación del vector de subclaves observado entre claves del mismo peso de Hamming son debidas principalmente al número total de bits de la clave que están en uno. Así, el efecto de cuáles bits son no nulos, entre claves del mismo peso de Hamming, es despreciable.

Todos los experimentos descritos en esta sección fueron repetidos, pero ahora, en vez de dejar el mensaje fijo, un nuevo mensaje elegido al azar fue escogido al comienzo de cada proceso de encriptación. Todos los resultados mostrados más arriba permanecieron (esencialmente) inalterados. Solamente se observó un despreciable aumento en las desviaciones de los valores medidos.

Suponiendo que el adversario conoce el diseño del sistema objetivo, es posible para él construir por sí sólo una tabla con los tiempos promedios de encriptación versus el peso de Hamming de la clave. Así, la relación claramente monotónica creciente entre el tiempo de encriptación y el peso de Hamming de la clave – evidenciado por nuestros experimentos – es una significativa deficiencia en la implementación. Esta relación permite al adversario determinar el peso de Hamming de la clave DES. De hecho, el adversario debe sólo obtener unas pocas mediciones de tiempos de encriptación y mirar en la tabla construida para determinar el peso de Hamming del cual las mediciones de tiempo podrían haber venido.

¿Cuánta información entregará este conocimiento a dicho adversario? Para responder a esta pregunta, denotemos primero por $\mathtt{wt}(K)$ al peso de Hamming de la clave K. Como

veremos, si la clave K es elegida al azar, la variable aleatoria $\mathsf{wt}(K)$ seguirá una distribución $Binom\,(56, 1/2)$. Denotemos además por $H(\cdot)$ a la función de entropía binaria [Sti95, pág. 51]) definida por

$$H(X) = -\sum_{i=1}^{n} q_i \log_2 q_i \ .$$

donde $q_1, \ldots, q_n$ es la distribución de probabilidad de la variable aleatoria X. Informalmente, la función de entropía binaria medirá la cantidad información obtenida por la ocurrencia de un evento con una cierta distribución de probabilidad $q_1, \ldots, q_n$ (ver [Ros88, págs. 436–439]). Ahora bien, dado que la variable aleatoria $\mathsf{wt}(K)$ distribuye según una $Binom\,(56, 1/2)$ tendremos que

$$H(\mathsf{wt}(K)) = -\sum_{i=1}^{56} \frac{\binom{56}{i}}{2^{56}} \log_2 \frac{\binom{56}{i}}{2^{56}} \approx 3.95 \ .$$

Con ello, si el adversario puede obtener $\mathsf{wt}(K)$ en promedio estará recuperando $H(\mathsf{wt}(K)) \approx 3.95$ bits de información de la clave.

Remark 5 *El peso de Hamming exacto de la clave DES puede ser recuperado utilizando un ataque de medición de tiempos si dos condiciones se cumplen. Primero, es posible obtener mediciones de tiempo precisas. Segundo, las variaciones en el tiempo de encriptación y generación del vector de subclaves producidas por claves diferentes pero con idéntico peso de Hamming es pequeña comparada con las variaciones de tiempo producidas por claves con un bit no nulo más o con uno menos. Hemos notado que esta última situación aproximadamente se cumple. Es posible lograr una estimación exacta del peso de Hamming de una clave DES si el adversario puede efectuar con alta precisión mediciones de tiempo de varias encriptaciones del mismo mensaje. Sin embargo, esto implica un rol más activo por parte del adversario.*

Más notable que la recién establecida relación monótona creciente entre los tiempos de encriptación y el peso de Hamming de la clave es la evidente dependencia lineal que existe entre estas dos cantidades. Los factores de correlación para los datos mostrados en la Fig. 4.1 y la Fig. 4.2 son 0.9760 y 0.9999 respectivamente. Como veremos, esta clara y precisa dependencia lineal entre los tiempos de encriptación y el peso de Hamming le permitirá a un adversario inferir la información del sistema objetivo necesaria para llevar a cabo el ataque descrito más arriba.

4.1.3 Implementación del Ataque

En esta sección describimos un experimento computacional que muestra la reducción esperada en el tamaño del espacio de claves buscado si se implementa el ataque de medición de tiempos descrito en la sección previa.

Supongamos que para cada $i \in \{0, \ldots, 56\}$ tenemos un valor $T_i \in \mathbb{R}$ con el tiempo esperado que la implementación de DES se demora en encriptar un mensaje (en el sistema

ENTRADA: $M \in \{0,1\}^{64}$, $C \in \{0,1\}^{56}$, $\tau \in \mathbb{R}$.
/* Donde τ tiempo que toma a DES generar texto cifrado C a partir de mensaje M */
CÓDIGO: **Para** $i = 0$ **hasta** 56
Sea l **tal que** $|\{\, j \,:\, |T_j - \tau| < |T_l - \tau| \,\}| = i$
Sea $\mathcal{K}_l = \{\, K \in \{0,1\}^{56} \,:\, \mathsf{wt}(K) = l \,\}$
Elegir al azar m **en** $\{\, 0, \ldots, |\mathcal{K}_l| - 1 \,\}$
Para $j = 0$ **hasta** $|\mathcal{K}_l| - 1$
Sea K **el** $(m + j) \bmod |\mathcal{K}_l|$ **-ésimo elemento (lexicográficamente) de** $\mathcal{K}_l$
Si (encriptación DES de M bajo clave K produce C) **entonces** retornar (K).

Figura 4.3: Procedimiento de recuperación de clave basado en ataque que revela el peso de Hamming de la clave.

objetivo) usando una clave de peso de Hamming igual a i. Más aún, supongamos que $T_i < T_{i+1}$. Consideremos ahora el procedimiento mostrado en la Fig. 4.3 para recuperar la clave DES a través de un ataque de medición de tiempos que explote los hechos reportados en la Sect. 4.1.2.

Notemos que es posible determinar experimentalmente el número esperado de claves que el procedimiento de la Fig. 4.3 probará sin tener que, en realidad, ejecutar dicho procedimiento. De hecho, supongamos que la encriptación del mensaje M bajo la clave K genera el texto cifrado C en τ-μs. Sea l tal que $|T_l - \tau| = \min_{i=0,\ldots,56}\{\, |T_i - \tau| \,\}$. Entonces, el tamaño esperado del espacio de claves buscado por el procedimiento dado es,

$$\left|\{\, K' \in \{0,1\}^{56} \,:\, |\mathsf{wt}(K') - l| < |\mathsf{wt}(K) - l| \,\}\right| + \frac{1}{2}\left|\{\, K' \in \{0,1\}^{56} \,:\, \mathsf{wt}(K') = \mathsf{wt}(K) \,\}\right| .$$

Para ambas implementaciones de DES estudiadas elegimos al azar pares mensaje/clave, medimos el tiempo de encriptación y calculamos el número esperado de claves que el procedimiento de la Fig. 4.3 habría probado hasta antes de encontrar la clave de encriptación correcta. (Ver más abajo para una descripción más detallada de cómo los pares mensaje/clave fueron elegidos.) De nuestra discusión en la Sect. 4.1.2 se deduce que lo mejor que podemos esperar, siempre que no hayan inexactitudes en las mediciones de tiempo, es recorrer el 3.24% de todo el espacio de claves de DES antes de encontrar la clave de encriptación correcta. Nuestros resultados muestran que para RSA–DES es necesario recorrer, en promedio, un 5.30% del espacio de claves antes de encontrar la clave de encriptación correcta. Para L–DES, este porcentaje se reduce a un 3.84%.

La Tabla 4.1 muestra en más detalle los datos recolectados en nuestros experimentos. Las columnas están etiquetadas de acuerdo al peso de la clave DES. El peso de una clave es denotado por k. La segunda fila representa el porcentaje del espacio de claves total corres-

k	12	13	14	15	16	17	18	19	20	21	22
p_k	0.001	0.003	0.008	0.023	0.058	0.136	0.295	0.589	1.090	1.869	2.973
RSA–DES	0.001	0.002	0.004	0.012	0.131	0.108	0.387	0.407	1.155	1.752	2.221
L–DES	0.001	0.002	0.004	0.012	0.029	0.073	0.425	0.431	0.646	1.208	1.654

k	23	24	25	26	27	28	29	30	31	32	33
p_k	4.396	6.044	7.736	9.224	10.249	10.615	10.249	9.224	7.736	6.044	4.396
RSA–DES	3.198	4.274	7.277	6.943	7.110	9.865	7.173	8.744	5.321	6.054	4.217
L–DES	2.745	3.362	4.371	5.366	6.228	6.472	5.690	5.337	4.607	3.768	3.048

k	34	35	36	37	38	39	40	41	42	43	44
p_k	2.973	1.869	1.090	0.589	0.295	0.136	0.058	0.023	0.008	0.003	0.001
RSA–DES	2.088	2.228	1.098	0.771	0.459	0.116	0.074	0.015	0.004	0.033	0.001
L–DES	1.770	1.168	0.571	0.328	0.172	0.086	0.043	0.015	0.004	0.002	0.001

Tabla 4.1: Resultados del experimento computacional.

pondiente a claves DES con peso de Hamming igual a k (con una precisión de 0.0005). Esto es, 100 veces la probabilidad de que la variable aleatoria $\mathsf{wt}(K)$ tome el valor k. Denotaremos este porcentaje por p_k. Para cada clave de DES de peso k estimamos ($16000 \cdot p_k$ veces) el porcentaje del espacio de claves que habría sido buscado hasta antes de encontrar la clave de encriptación. Las dos últimas filas de la Tabla 4.1 muestran, para cada implementación y peso de la clave, el promedio de los valores obtenidos.

Recuperar el peso de Hamming de una clave DES equivale a una recuperación de 3.95 bits de la clave. Esto se traduce en una modesta mejora en el tiempo necesario para recuperar toda la clave. Sin embargo, los datos en la Tabla 4.1 muestran que un ataque que revele el peso de Hamming de la clave es potencialmente más peligroso. En particular, un adversario puede restringir su atención a claves que tengan, ya sea un peso de Hamming significativamente bajo o significativamente alto. El adversario puede realizar esto simplemente efectuando mediciones de tiempo hasta encontrar una clave con peso significativamente bajo o alto. Pese a que dichas claves pueden ser escasas, una vez que el adversario detecta una el tiempo de la consiguiente búsqueda de la clave puede ser mucho menor a lo usual. Así, el adversario puede compensar el tiempo necesario para encontrar dichas escasas claves con el tiempo necesario para recuperar la clave. Notemos que esto puede ser muy grave en algunos sistemas donde la obtención no autorizada de una única clave puede causar un malfuncionamiento general o comprometer la seguridad futura.

4.1.4 Fuentes de Variabilidad en los Tiempos

Como ya fue mencionado, la causa más significativa de los tiempos de ejecución variables en ambas implementaciones de DES examinadas fueron las instrucciones condicionales. A continuación discutimos las decisiones de programación que causan que L–DES revele el peso

de Hamming de la clave cuando es sometida a un ataque como el descrito anteriormente.

En L–DES, uno de los procedimientos de la generación del vector de subclaves, llamado `des_set_key`, contiene un trozo de código de la siguiente forma:

If (*condición*)
then $expr1; \ldots; expr32;$
else $expr;$

El número de veces que *condición* es verdadera resulta ser directamente proporcional al peso de Hamming de la clave DES. Esta estructura es la fuente principal de diferencias en los tiempos de ejecución en L–DES. En RSA–DES existe también un procedimiento de generación del vector de subclaves que contiene dos instrucciones condicionales que causan el mismo efecto agregado que en las instrucciones condicionales vistas de L–DES (ver Apéndice C).

Como fue mencionado, Kocher conjeturó en [Koc96] que en el proceso de generación de claves de DES el tiempo necesario para desplazar (o efectuar un *shift* de) los bits no nulos podrían ser una fuente de tiempos de ejecución variables en la encriptación. En la plataforma utilizada para nuestras pruebas, no encontramos evidencia que respaldara esta conjetura. De hecho, evaluamos el tiempo necesario para desplazar varias palabras[5] de diferentes pesos bajo diferentes valores de desplazamiento y no encontramos diferencias estadísticas en dichas mediciones.[6]

Sin embargo, aunque las razones que dio Kocher para esta dependencia (que el tiempo necesario para efectuar el desplazamiento de los bits no nulos pudiera ser variable) no resultaron ciertas en las implementaciones revisadas, ellas podrían ser pertinentes en arquitecturas más simples, por ejemplo, en un chip de una tarjeta inteligente. Sin embargo, aún en dicha arquitectura esta dependencia no afectará el tiempo de ejecución del proceso de encriptación, por lo menos en las implementaciones analizadas. Esto se debe a que el valor de los desplazamientos efectuados es constante (e independiente de la clave y del mensaje) y en dichos procesadores más simples la dependencia del tiempo del desplazamiento ocurre con respecto al valor del desplazamiento y no al valor desplazado. De ello se concluye que en general la fuente relevante de variabilidad de los tiempos de ejecución son las instrucciones condicionales.

Finalmente, notemos que es claro de la Fig. 4.2 que en L–DES existe una fuente de tiempos de ejecución variables que no depende del proceso de generación de claves. Esto es claramente deducible de la distancia variable entre las dos curvas mostradas en la Fig. 4.2. La fuente de dichas diferencias de tiempos no se debe a instrucciones condicionales sino a una interrelación más compleja de varios factores (donde uno de ellos posiblemente sea la estructura de accesos a memoria). No fue posible explotar esta dependencia de manera de recuperar más información de la clave.

[5] Una *palabra* es la mayor unidad de datos que puede operar la arquitectura de un computador.

[6] Tal como se esperaría en una arquitectura moderna, de acuerdo a [Man79].

4.2 Deducción de las Características

Como discutimos en la Sect. 4.1.2, en ambas implementaciones de DES estudiadas el tiempo de encriptación es aproximadamente igual a una función lineal del peso de Hamming de la clave más ruido blanco. En esta sección explotaremos este hecho para deducir toda la información necesaria para efectuar un ataque de medición de tiempos que revele el peso de Hamming de la clave DES usada en el sistema objetivo.

Primero necesitaremos introducir algo de notación. Supongamos que tenemos m mediciones del tiempo que demora el sistema objetivo en realizar una encriptación usando DES. Estas mediciones de tiempo podrían corresponder a encriptaciones realizadas bajo distintas claves.

Para $i \in \{1, \ldots, k\}$, denotemos por K_i a la i-ésima clave usada en el sistema objetivo durante el período en que las mediciones fueron tomadas. Haremos el supuesto (realista) que $K_1, \ldots, K_k$ son elegidas al azar en $\{0,1\}^{56}$ e independientes las unas de las otras. Sea $X^{(i)}$ el peso de Hamming de la clave K_i. Así, la distribución de $X^{(i)}$ es una $Binom\,(56, 1/2)$.[7] La suposición que los K_i's han sido elegidos independientemente conlleva a que $X^{(1)}, \ldots, X^{(k)}$ son variables aleatorias independientes. Notemos que mediciones de tiempo sucesivas pueden corresponder a encriptaciones del mensaje bajo la misma clave. Para $i \in \{1, \ldots, k\}$, sea $\tau_i \in \{1, \ldots, m\}$ el índice de la última medición correspondiente a una encriptación efectuada usando la clave K_i. Por simplicidad, sea $\tau_0 = 0$. Con ello, $0 = \tau_0 < \tau_1 < \ldots < \tau_{k-1} < \tau_k = m$. Denotemos por I_i al conjunto de índices que corresponden a mediciones de tiempo bajo la clave K_i, i.e. para $i \in \{1, \ldots, k\}$ sea $I_i \stackrel{\text{def}}{=} \{n \in \mathbb{N} : \tau_{i-1} < n \leq \tau_i\}$. Para $i \in \{1, \ldots, k\}$ y $j \in I_i$ sea $T_j^{(i)}$ la variable aleatoria que representa el tiempo que demora el sistema objetivo en efectuar la j-ésima encriptación del mensaje usando la clave K_i. Finalmente, para $j \in I_i$ sea $e_j^{(i)}$ la variable aleatoria que representa el efecto del ruido blanco en la j-ésima encriptación usando la clave K_i. De esta manera, los $e_j^{(i)}$'s representarán las imprecisiones en la medición y las fluctuaciones en el tiempo de ejecución del sistema objetivo.

Ahora disponemos de toda la notación necesaria para enunciar formalmente el problema que queremos resolver.

La dependencia lineal entre el tiempo de encriptación y el peso de Hamming de la clave en ambas implementaciones de DES estudiadas implica que existen α, β, y σ, tales que para todo $i \in \{1, \ldots, k\}$ y $j \in I_i$

$$T_j^{(i)} \;=\; \alpha X^{(i)} + \beta + e_j^{(i)}, \quad X^{(i)} \sim Binom\,(56, 1/2)\,, \quad e_j^{(i)} \sim Norm\,(0, \sigma^2)\,. \qquad (4.1)$$

Nuestro problema es inferir, a partir de mediciones de tiempo, los parámetros α, β, y σ para los cuales (4.1) se cumple.

Veremos dos variaciones de este problema. Primero mostraremos como enfrentar el problema cuando los τ_i's son conocidos, es decir, cuando el adversario conoce cuáles mediciones

[7] Recordemos que la distribución $Binom\,(N, p)$ corresponde a la distribución de la suma de N variables aleatorias $\{0, 1\}$ independientes idénticamente distribuidas con esperanza p.

corresponden a encriptaciones realizadas con una misma clave. (Notemos que esto es equivalente a conocer entre cuáles encriptaciones la clave fue actualizada.) En la Sect. 4.2.2 mostraremos como tratar el caso cuando el adversario no posee esta información, es decir, los τ_i's son desconocidos. Observemos, sin embargo, que el primer caso es el más realista. De hecho, un supuesto criptográfico razonable es asumir que el adversario conoce los procedimientos de administración de claves del sistema objetivo. Claramente ésto es un extensión natural del *supuesto de Kerckhoff* [MvOV97, pág. 225].

4.2.1 Cambio de Claves Conocido

En esta sección propondremos dos métodos estadísticos alternativos para deducir α, β, y σ para los cuales (4.1) se tiene. (Recordemos que los índices – de los τ_i's – entre los cuales se efectúa el cambio de claves son conocidos por el adversario.) Un método está basado en la técnica de los estimadores de máxima verosimilitud y el otro en los estimadores asintóticamente insesgados. Dado que la siguiente discusión se basa fuertemente en conceptos y resultados estándares de probabilidades y estadística, recomendamos al lector poco familiarizado con estos temas remitirse a [Fel66, Ros88, Zac71] para una introducción a los conceptos y términos relevantes.

ESTIMADORES DE MÁXIMA VEROSIMILITUD

Sea $X = (X^{(i)})_{i=1}^{k}$, $T^{(i)} = (T_j^{(i)})_{j\in I_i}$, y $T = (T^{(i)})_{i=1}^{k}$. Por consiguiente, $X, T^{(1)}, \ldots, T^{(k)}$, y T denotan variables aleatorias. Más aún, sean $x = (x_i)_{i=1}^{k}$, $t^{(i)} = (t_j^{(i)})_{j\in I_i}$, y $t = (t^{(i)})_{i=1}^{k}$ los valores reales tomados por X, $T^{(i)}$, y T respectivamente.

Sea $f_T(t;\alpha,\beta,\sigma)$ la distribución marginal de T dados α, β, y σ. Para un conjunto fijo de mediciones de tiempo t, los valores de α, β, y σ que maximizan $f_T(t;\alpha,\beta,\sigma)$ son los estimadores de máxima verosimilitud que buscamos. Los estimadores de máxima verosimilitud pueden ser interpretados como los valores que minimizan la función de pérdida $-\log f_T(t;\alpha,\beta,\sigma)$ y por ello son los valores que más probablemente han producido las mediciones de tiempo observadas. Ello explica por qué los estimadores de máxima verosimilitud se consideran buenos predictores. Así, para determinar buenos estimadores para α, β, y σ primero calcularemos $f_T(t;\alpha,\beta,\sigma)$.

Proposition 1 *La distribución marginal de T dados α, β, y σ es*

$$f_T(\,t\,;\,\alpha,\beta,\sigma\,) \;=\; \left(\frac{1}{2\pi\sigma^2}\right)^{\frac{m}{2}} \prod_{i=1}^{k} \mathbb{E}_{X^{(i)}}\left[\, e^{-\frac{1}{2\sigma^2}\sum_{j\in I_i}(t_j^{(i)}-(\alpha X^{(i)}+\beta))^2}\,\right] \;.$$

Proof: Dado α, β, y σ, denotemos por $f_{X,T}(\,\cdot\,;\alpha,\beta,\sigma)$, $f_{T/X=x}(\,\cdot\,;\alpha,\beta,\sigma)$ y $f_X(\cdot;\alpha,\beta,\sigma)$ a la función de densidad conjunta de X y T, a la función de densidad de T dado $X = x$, y a la distribución de probabilidad de X, respectivamente. Por simplicidad, de aquí en adelante omitiremos α, β, y σ de las expresiones para $f_{X,T}$, $f_{T/X=x}$, y f_X.

Observemos que la independencia de las variables aleatorias $X^{(i)}$'s y $e_j^{(i)}$'s implican que las $T^{(i)}$'s son independientes. Así, la función de densidad conjunta de X y T dados α, β, y σ es

$$f_{X,T}(x,t) = f_X(x) \cdot f_{T/X=x}(t) = \prod_{i=1}^{k} f_{X^{(i)}}(x_i) \cdot f_{T^{(i)}/X^{(i)}=x_i}(t^{(i)}) ,$$

donde la última igualdad se deduce de la independencia de las variables aleatorias $T^{(i)}$'s y $e_j^{(i)}$'s. Ahora, dado que $X^{(i)} \sim Binom(56, 1/2)$,

$$f_{X^{(i)}}(x_i) = \frac{1}{2^{56}} \binom{56}{x_i} .$$

De la relación (4.1) se deduce que $T_j^{(i)}$ dado $X^{(i)} = x_i$ tendrá una distribución $Norm(\alpha x_i + \beta, \sigma^2)$. Más aún, para i fijo, las $T_j^{(i)}$'s son variables aleatorias independientes. Por lo tanto,

$$f_{T^{(i)}/X^{(i)}=x_i}(t^{(i)}) = \prod_{j \in I_i} \left(\frac{1}{2\pi\sigma^2}\right)^{\frac{1}{2}} e^{-\frac{1}{2\sigma^2}(t_j^{(i)} - (\alpha x_i + \beta))^2} = \left(\frac{1}{2\pi\sigma^2}\right)^{\frac{|I_i|}{2}} e^{-\frac{1}{2\sigma^2}\sum_{j \in I_i}(t_j^{(i)} - (\alpha x_i + \beta))^2} .$$

Así,

$$f_{X,T}(x,t) = \left(\frac{1}{2\pi\sigma^2}\right)^{\frac{m}{2}} \prod_{i=1}^{k} \frac{1}{2^{56}} \binom{56}{x_i} e^{-\frac{1}{2\sigma^2}\sum_{j \in I_i}(t_j^{(i)} - (\alpha x_i + \beta))^2} .$$

La distribución marginal de T dado α, β, y σ es igual a la suma de $f_{X,T}(x,t)$ sobre todos los valores posibles tomados por x. Por lo tanto,

$$f_T(t\,;\,\alpha,\beta,\sigma) = \left(\frac{1}{2\pi\sigma^2}\right)^{\frac{m}{2}} \prod_{i=1}^{k} \sum_{x_i=0}^{56} \frac{1}{2^{56}} \binom{56}{x_i} e^{-\frac{1}{2\sigma^2}\sum_{j \in I_i}(t_j^{(i)} - (\alpha x_i + \beta))^2} .$$

La conclusión se obtiene directamente de la igualdad anterior y del hecho que $X^{(i)} \sim Binom(56, 1/2)$. ∎

Para un t dado, los valores de α, β, y σ que maximizan el lado derecho de la expresión en la Proposition 1 son los estimadores de máxima verosimilitud buscados. Como es frecuente cuando se trabaja con estimadores de máxima verosimilitud, es difícil resolver explícitamente estas expresiones para los valores que las maximizan. Se recomienda ver [Zac71, Cap. 5, §2] para una discusión de las algoritmos computacionales que pueden ser usados para calcular los estimadores de máxima verosimilitud.

La ventaja del esquema anterior para determinar los parámetros de importancia a la hora de efectuar un ataque de medición de tiempos es que utiliza todas las mediciones de tiempo disponibles. Sin embargo, este método no permite determinar cuántas mediciones

son necesarias para obtener estimaciones precisas de los parámetros buscados. El método descrito a continuación soluciona este problema.

ESTIMADORES ASINTÓTICOS

Nuestro objetivo aquí es encontrar buenos estimadores $\widehat{\alpha}$, $\widehat{\beta}$, y $\widehat{\sigma}$ para α, β, y σ. Más aún, estamos interesados en determinar el comportamiento asintótico (en el número de mediciones de tiempo) de dichos estimadores. En particular, nos interesará conocer sus distribuciones, sus valores límites y su radio de convergencia. Estos datos nos permitirán determinar la precisión de los estimadores encontrados.

Ahora derivaremos buenos estimadores para α, β, y σ. Comenzaremos con una observación crucial. Dado que conocemos la esperanza y la varianza de una $Binom\,(56, 1/2)$ (28 y 14 respectivamente), podemos tomar la esperanza y la varianza sobre (4.1). Esto nos entrega que para todo $i \in \{\,1, \ldots, k\,\}$ y $j \in I_i$

$$\mu_T \stackrel{\text{def}}{=} \mathbb{E}\left[T_j^{(i)}\right] = 28 \cdot \alpha + \beta \ , \qquad \sigma_T^2 \stackrel{\text{def}}{=} \mathbb{V}\left[T_j^{(i)}\right] = 14 \cdot \alpha^2 + \sigma^2 \ . \tag{4.2}$$

Por lo tanto, si conociéramos μ_T, σ_T^2, y σ^2 podríamos encontrar α y β usando (4.2). Esto último sugiere que si podemos encontrar buenos estimadores para μ_T, σ_T^2, y σ^2, entonces podemos deducir buenos estimadores para α y β.

Ahora podemos proponer candidatos para $\widehat{\mu_T}$, $\widehat{\sigma_T^2}$, y $\widehat{\sigma^2}$, los estimadores de μ_T, σ_T^2, y σ^2 respectivamente. Sin embargo, primero necesitamos introducir algo de notación adicional. Sea

$$\overline{T}^{(i)} \stackrel{\text{def}}{=} (\sum_{j \in I_i} T_j^{(i)})/|I_i| \ , \qquad \overline{T} \stackrel{\text{def}}{=} (\sum_{i=1}^{k} \overline{T}^{(i)})/k \ , \qquad \overline{e}^{(i)} \stackrel{\text{def}}{=} (\sum_{j \in I_i} e_j^{(i)})/|I_i| \ .$$

Definamos

$$\widehat{\mu_T} \stackrel{\text{def}}{=} \overline{T} \ , \qquad \widehat{\sigma_T^2} \stackrel{\text{def}}{=} \frac{1}{k} \sum_{i=1}^{k} \frac{1}{|I_i|} \sum_{j \in I_i} (T_j^{(i)} - \overline{T})^2 \ , \qquad \widehat{\sigma^2} \stackrel{\text{def}}{=} \frac{1}{k} \sum_{i=1}^{k} \frac{1}{|I_i|} \sum_{j \in I_i} (T_j^{(i)} - \overline{T}^{(i)})^2 \ .$$

Resolviendo por α y β en (4.2) encontramos los dos candidatos naturales para $\widehat{\alpha}$ y $\widehat{\beta}$, los estimadores de α y β, los cuales son

$$\widehat{\alpha} \stackrel{\text{def}}{=} \frac{1}{\sqrt{14}} (\widehat{\sigma_T^2} - \widehat{\sigma^2})^{1/2} , \qquad \widehat{\beta} \stackrel{\text{def}}{=} \widehat{\mu_T} - 28 \cdot \widehat{\alpha} \ .$$

Ahora probaremos que $\widehat{\alpha}$ está bien definido.

Proposition 2

$$\widehat{\sigma_T^2} - \widehat{\sigma^2} = \frac{1}{k} \sum_{i=1}^{k} (\overline{T}^{(i)} - \overline{T})^2 \geq 0 \ .$$

Proof: Basta notar que

$$\widehat{\sigma_T^2} = \frac{1}{k}\sum_{i=1}^{k}\frac{1}{|I_i|}\sum_{j\in I_i}(T_j^{(i)} - \overline{T}^{(i)} + \overline{T}^{(i)} - \overline{T})^2 = \widehat{\sigma^2} + \frac{1}{k}\sum_{i=1}^{k}(\overline{T}^{(i)} - \overline{T})^2 \ .$$

■

De aquí en adelante denotaremos por χ^2_ℓ a una variable aleatoria de distribución *chi–cuadrado* con ℓ grados de libertad.

La dos siguientes proposiciones nos serán útiles para determinar la distribución asintótica de $\widehat{\alpha}$.

Proposition 3 *Si $|I_1| = \ldots = |I_k| = n$, entonces la distribución de $\widehat{\sigma_T^2} - \widehat{\sigma^2}$ es (aproximadamente)*

$$\frac{1}{k}(14\cdot\alpha^2 + \frac{1}{n}\,\sigma^2)\chi^2_{k-1} \ .$$

Proof: Dado que $T_j^{(i)} = \alpha X^{(i)} + \beta + e_j^{(i)}$, tenemos que $\overline{T}^{(i)} = \alpha X^{(i)} + \beta + \overline{e}^{(i)}$ donde $X^{(i)} \sim Binom\,(56, 1/2)$. Además, dado que $\overline{e}^{(i)}$ es el promedio de n variables aleatorias independientes de distribución $Norm\,(0,\sigma^2)$, entonces $\overline{e}^{(i)} \sim Norm\,(0,\sigma^2/n)$. Adicionalmente, el Teorema de Moivre–Laplace [Haz88, pág. 397] establece que una distribución $Binom\,(m,p)$ puede ser expresada en términos de una distribución normal. Más aún, si $m \to \infty$, dicha expresión es exacta, y si $mp(1-p) \geq 10$, la expresión proveerá una aproximación bastante buena a la distribución Binomial [Ros88, págs. 170–171]. Así, la distribución de $X^{(i)}$ puede ser bien aproximada por una $Norm\,(28, 14)$. Por lo tanto, dado que $X^{(i)}$ es independiente de $\overline{e}^{(i)}$ y la suma de distribuciones normales independientes es una distribución normal, se sigue que $\overline{T}^{(i)}$ estará aproximadamente distribuida por una $Norm\left(\mu_T, 14\alpha^2 + \frac{\sigma^2}{n}\right)$. La conclusión buscada se deduce de un resultado clásico de la estadística [HT97, Teorema 5.3.4] y de la Proposition 2. ■

Proposition 4 *Si $|I_1| = \ldots = |I_k| = n$ y σ^2/n es despreciable, entonces cuando $k \to \infty$, $\sqrt{k}(\widehat{\alpha}^2 - \alpha^2)$ converge (en distribución)*[8] *a una $Norm\,(0, 3\alpha^4)$ más un término de error pequeño constante.*

Proof: Primero notemos que si despreciamos el término σ^2/n entonces la Proposition 3 y nuestra definición de $\widehat{\alpha}$ implican que la distribución de

$$\widehat{\alpha}^2 = \frac{1}{14}(\widehat{\sigma_T^2} - \widehat{\sigma^2}) \ ,$$

[8] Recordemos que si $X_1, X_2, \ldots, X$ son variables aleatorias en algún espacio de probabilidad $(\Omega, \mathcal{F}, \mathbb{P})$, se dice que X_n converge en distribución a X cuando $n \to \infty$, si $\mathbb{P}\,[X_n \leq x\,] \to \mathbb{P}\,[X \leq x\,]$ cuando $n \to \infty$ para todos los puntos x en donde $F_X(x) = \mathbb{P}\,[X \leq x\,]$ es continua.

es (aproximadamente) una $\frac{\alpha^2}{k}\chi^2_{k-1}$.

Segundo, recordemos que la suma de los cuadrados de l variables aleatorias normales independientes idénticamente distribuidas de media cero y varianza igual a 1 se distribuye de acuerdo a una χ^2_ℓ. De igual manera, la suma de ℓ variables aleatorias χ^2_1 independientes idénticamente distribuidas se distribuye de acuerdo a una χ^2_ℓ. Por lo tanto, dado que la esperanza y la varianza de una variable aleatoria χ^2_1 son 1 y 3 respectivamente, el Teorema Central del Límite implica que

$$\left(\frac{1}{k-1}\chi^2_{k-1} - 1\right) / (1/\sqrt{k-1}) \ ,$$

converge (en distribución) a una $Norm\,(0,3)$.

Juntando estas dos observaciones se muestra que, cuando $k \to \infty$, $\sqrt{k-1}(\widehat{\alpha}^2 - \alpha^2)$ converge (en distribución) a una $Norm\,(0,3\alpha^4)$ más un término constante pequeño. El resultado buscado se deduce inmediatamente. ∎

Theorem 1 *Si $|I_1| = \ldots = |I_k| = n$, σ^2/n es despreciable y k es suficientemente grande, entonces $\widehat{\alpha}$ sigue (aproximadamente) una distribución*

$$Norm\left(\alpha, \frac{3}{4k}\alpha^2\right) \ .$$

Proof: La Ley de los Grandes Números implica que $\widehat{\sigma^2_T}$ y $\widehat{\sigma^2}$ convergen (casi seguramente)[9] a σ^2_T y σ^2 respectivamente. Por lo tanto, por continuidad $\widehat{\alpha} = \frac{1}{\sqrt{14}}\left(\widehat{\sigma^2_T} - \widehat{\sigma^2}\right)^{1/2}$ converge (casi seguramente) a $\alpha = \frac{1}{\sqrt{14}}\left(\sigma^2_T - \sigma^2\right)^{1/2}$ cuando $k \to \infty$. De este último hecho y de la Proposition 4 obtenemos que cuando $k \to \infty$

$$\sqrt{k}(\widehat{\alpha} - \alpha) \ = \ \frac{\sqrt{k}(\widehat{\alpha}^2 - \alpha^2)}{\widehat{\alpha} + \alpha} \ .$$

converge (en distribución) a una $Norm\,(0, \frac{3}{4}\alpha^2)$ más un pequeño término de error. La conclusión deseada se obtiene inmediatamente. ∎

Remark 6 *El Theorem 1 entrega una* aproximación *a la distribución de $\widehat{\alpha}$. El error de aproximación surge de tres fuentes. La primera es el uso del Teorema de Moivre–Laplace para aproximar una $Binom\,(56,1/2)$ en términos de una $Norm\,(28,14)$. La segunda fuente es el uso del Teorema Central del Límite para aproximar la distribución de un estimador por su distribución límite. Y la tercera fuente de error es el uso de la Ley de los Grandes*

[9] Recordemos que si $X_1, X_2, \ldots, X$ son variables aleatorias en algún espacio de probabilidad $(\Omega, \mathcal{F}, \mathbb{P})$, se dice que X_n converge casi seguramente a X cuando $n \to \infty$, si $\{\,\omega \in \Omega : X_n(\omega) \to X(\omega),\ \text{si } n \to \infty\,\}$ es un evento cuya probabilidad es 1.

Números para aproximar un estimador por su valor asintótico. Estas tres fuentes de error de aproximación pueden ser acotadas por el Teorema de Moivre–Laplace, la desigualdad de Berry–Essen [Haz88, pág. 369], y la desigualdad de Chebyshev [Ros88, pág. 337] respectivamente.

El siguiente corolario muestra que el resultado del Theorem 1 es considerablemente preciso.

Corollary 1 *Si* $|I_1| = \ldots = |I_k| = n$, σ^2/n *es despreciable y* k *es suficientemente grande, entonces*

$$\mathbb{P}\left[\,|\widehat{\alpha} - \alpha| \geq \epsilon\,\right] \;\leq\; O\left(\frac{1}{k\epsilon^2}\right) \;, \tag{4.3}$$

$$\mathbb{P}\left[\,|\widehat{\beta} - \beta| \geq \epsilon\,\right] \;\leq\; O\left(\frac{1}{k\epsilon^2}\right) \;. \tag{4.4}$$

Proof:

La desigualdad (4.3) se obtiene del Theorem 1 y de la desigualdad de Chebyshev [Ros88, pág. 337].

Para demostrar (4.4) recordemos que $\widehat{\beta} = \widehat{\mu_T} - 28 \cdot \widehat{\alpha}$ y que $\beta = \mu_T - 28 \cdot \alpha$, por lo que

$$\begin{aligned}
\mathbb{P}\left[\,|\widehat{\beta} - \beta| \geq \epsilon\,\right] &= \mathbb{P}\left[\,|\widehat{\mu_T} - \mu_T - 28(\widehat{\alpha} - \alpha)| \geq \epsilon\,\right] \\
&\leq \mathbb{P}\left[\,|\widehat{\mu_T} - \mu_T| \geq \frac{\epsilon}{2}\,\right] + \mathbb{P}\left[\,28\,|\widehat{\alpha} - \alpha| \geq \frac{\epsilon}{2}\,\right] \\
&\leq \frac{1}{(\epsilon/2)^2}\left(\mathbb{V}\left[\,\widehat{\mu_T}\,\right] + \frac{\mathbb{V}\left[\,\widehat{\alpha}\,\right]}{28^2}\right) \;,
\end{aligned}$$

donde la última desigualdad es consecuencia de aplicar dos veces la desigualdad de Chebyshev.

Notemos que del Theorem 1 tenemos que $\mathbb{V}\left[\,\widehat{\alpha}\,\right] = \frac{3}{4k}\alpha^2$. Además,

$$\mathbb{V}\left[\,\widehat{\mu_T}\,\right] = \frac{1}{k}\mathbb{V}\left[\,\overline{T}^{(i)}\,\right] = \left(\frac{14 \cdot \alpha^2 + \sigma^2/n}{k}\right) \;.$$

Con esto, el resultado es inmediato. ∎

El Corollary 1 nos dice que, con probabilidad a lo menos $1 - \delta$ basta tomar n mediciones de tiempo en cada una de las $\frac{1}{\delta}O(\frac{1}{\epsilon^2})$ claves distintas para aproximar α y β con una precisión ϵ. Con ello, hemos encontrado un método con el cual podemos calcular α y *beta* con una precisión dada sólo a través de mediciones de tiempo.

4.2.2 Cambio de Claves Desconocido

El supuesto hecho en la sección previa, que los τ_i's son conocidos (es decir, que el adversario conoce con cierta precisión los instantes de tiempos en que las claves del sistema objetivo son cambiadas) no es estrictamente necesario. Esto porque un adversario podría alternar entre efectuar mediciones de tiempo en un período de tiempo corto y el descansar durante un período de tiempo apropiadamente largo. Por lo tanto, el problema de deducir las características de diseño del sistema objetivo se reduce al caso en que los τ_i's son conocidos. Esto es, siempre que las claves no sean cambiadas demasiado frecuentemente y el período de descanso del adversario sea mayor que el tiempo de vida (período de uso) de una clave. (De hecho, cambiar demasiado frecuentemente las claves crea un problema de administración para el usuario del criptosistema, por lo que es razonable suponer que la vida útil de una clave no es excesivamente corta.)

Ahora discutiremos otro enfoque para manejar el caso de los τ_i's desconocidos bajo el supuesto que el adversario tiene acceso a varias copias idénticas del sistema objetivo. Por ejemplo, posee varias copias de una tarjeta inteligente que implementa un protocolo de reto–respuesta usando DES. Podemos utilizar el supuesto razonable de que las claves en cada sistema objetivo son generadas de manera independiente. En este caso, un adversario podría efectuar, en un corto período de tiempo, varias mediciones de tiempo para cada copia del sistema objetivo. Si las claves no son cambiadas demasiado frecuentemente, el adversario podrá deducir las características de tiempo relevantes del sistema objetivo usando las técnicas descritas en la sección anterior. De hecho, el adversario podrá suponer que todas las mediciones de tiempo efectuadas sobre la misma copia del sistema provienen de encriptaciones efectuadas con la misma clave. Dado que las claves correspondientes a copias diferentes del sistema objetivo fueron generadas en forma independiente y que las copias del sistema son idénticas, el problema de deducir las características de diseño del sistema objetivo se reduce al caso donde los τ_i's son conocidos.

Asimismo, los test de hipótesis estadísticos pueden entregar una alternativa adicional para solucionar el caso de los τ_i's desconocidos. Por ejemplo, consideremos una situación en la cual el adversario determina m mediciones de tiempo $t_1, \ldots, t_m$ provenientes de variables aleatorias que satisfacen (4.1). Si suponemos que las claves no son cambiadas demasiado frecuentemente (o equivalentemente que a lo menos $n \ll m$ mediciones de tiempo provienen de encriptaciones realizadas con la misma clave) entonces, para cada j tal que $n \leq j \leq m - n$ el adversario puede efectuar un test de contraste de hipótesis para la igualdad de dos distribuciones normales [HT97, págs. 372–385] utilizando las muestras $t_{j-n+1}, \ldots, t_j$ y $t_{j+1}, \ldots, t_{j+n}$. La variación del nivel de significancia de tales tests permitirá al adversario determinar la medición cerca de la cual ocurrió un cambio de clave. Si se descartan las mediciones de tiempos cercanas a la medición donde el adversario sospecha que un cambio de claves ocurrió, es posible obtener una secuencia de mediciones de tiempo que permita deducir las características de diseño del sistema objetivo utilizando la técnica para el caso de los τ_i's conocidos.

Capítulo 5

Posibles Defensas

La discusión en el capítulo anterior muestra que ciertas implementaciones de DES pueden filtrar información privada del criptosistema a través del tiempo tomado por un sistema objetivo en la encriptación de un mensaje. A continuación revisaremos diversas soluciones para este problema y su grado de éxito en la práctica.

En la Sect. 5.1 analizamos tres posibles esquemas de solución. Posteriormente, propondremos una solución basada en una técnica de "enceguecimiento" (*blinding*) que explota la misma dependencia en el peso de Hamming del criptosistema para eliminar las diferencias de tiempo de manera simple y efectiva.

5.1 Medidas Básicas de Defensa

Como se vio en el Cap. 4 las diferencias en los tiempos de encriptación observadas se deben esencialmente a instrucciones condicionales. Esto motiva el buscar maneras de generar encriptaciones que tomen tiempo constante o posean diferencias no dependientes del peso de Hamming de la clave (más aún, no dependan de ninguna información parcial[1] de la clave).

AUMENTAR IMPRECISIÓN[2]

Este primer esquema busca diluir el efecto del peso de Hamming de la clave a través de retrasos aleatorios. Concretamente, en esta solución la rutina de encriptación es modificada de manera que, luego de realizar una encriptación, espere un tiempo aleatorio. Este tipo de retrasos puede implementarse por software a través de un generador de números pseudo-aleatorios simple [Knu98] y una rutina cuyo tiempo de proceso es proporcional al número pseudo-aleatorio generado.

Lamentablemente, esta solución no evita el ataque. Esto es claro si consideramos que el efecto del tiempo aleatorio agregado (modelado como variables aleatorias uniformes independientes idénticamente distribuidas) es un aumento de la varianza en las mediciones

[1] Cualquier información posible de deducir de la clave.

[2] Esta solución y la siguiente son mencionadas en forma similar en [Koc96].

de tiempo. Con ello, si $X_1, \ldots, X_n$ representan ahora variables aleatorias independientes idénticamente distribuidas (que en nuestro caso serían sumas de variables aleatorias normales y uniformes) de media μ y varianza σ^2 (finitas) entonces por la versión débil de la ley de los grandes números [Ros88, pág. 398]

$$P\left\{\left|\frac{X_1 + \cdots + X_n}{n} - \mu\right| \geq \epsilon\right\} \leq \frac{\sigma^2}{n\epsilon^2}$$

Luego, un adversario para compensar la existencia de una mayor varianza sólo deberá recolectar un mayor número de mediciones de tiempo para tener éxito.

FORZAR TIEMPO CONSTANTE

En esta solución se impone la restricción de tomar tiempo constante a través de medir el tiempo durante la encriptación usando un *cronómetro interno*. Usando este dispositivo, el sistema efectuará la encriptación de un mensaje M usando una clave K en un tiempo fijo (independiente de la entrada) preestablecido T tal que,

$$T = \max_{M \in \{0,1\}^{64}, K \in \{0,1\}^{56}} \{\, t \in \mathbb{R} : M \text{ es encriptado con K en Tiempo t} \,\} \quad .$$

En la práctica, esto significa que el sistema objetivo, antes de entregar el resultado de la encriptación, deberá verificar si el valor en un cronómetro local ha superado T y, en caso contrario, esperar en forma inactiva hasta que ello ocurra. Claramente, la desventaja de esta solución es la necesidad de un reloj (es decir, aumentar el *hardware*). En algunos sistemas, en particular en tarjetas inteligentes, donde las restricciones de costos y espacio son importantes, esto puede ser una solución no deseable o, más aún, infactible. Incluso utilizando un dispositivo como éste, de acuerdo a [Koc96, pág. 111] la tarea podría ser más difícil de lo pensado por influencias en el tiempo de respuesta del procesador.

ELIMINAR INSTRUCCIONES DE TIEMPO VARIABLE

Una tercera posible solución constituye el eliminar las causas directas de estas diferencias, es decir, las instrucciones de tiempo variable. Esto significa principalmente reemplazar las instrucciones condicionales (**if**'s) por estructuras de tipo *asignación* e *iteración* de tiempo constante, independiente de la entrada, que en conjunto realicen un proceso equivalente dentro del algoritmo de encriptación. Pese a ser una solución que eliminaría la mencionada dependencia[3] sin agregar dispositivos o rutinas adicionales, tiene la clara desventaja de requerir un enfoque individual y *especializado* para cada implementación. En particular, notemos que las anteriores soluciones sólo requieren modificar el *proceso final* de entrega (output) del resultado de la encriptación, mientras que esta solución requiere alterar las rutinas internas del criptosistema DES usado. Esto significa la re-escritura de varias secciones esenciales de la implementación (con influencias potenciales en todo el código).

[3] Siempre y cuando no se introduzcan nuevas variaciones debido a colisiones de acceso a RAM o al caché, o a optimizaciones del compilador.

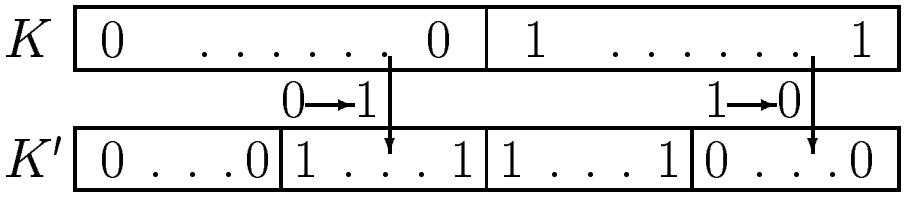


Figura 5.1: Generación de la clave K'.

Una solución que aprovecha las ventajas del segundo esquema mencionado más arriba (independencia de la implementación), sin necesitar *hardware* adicional o modificaciones específicas a las rutinas internas de DES, es la que se propone en la siguiente sección.

5.2 Técnica de Enceguecimiento

A fin de prevenir un ataque de medición de tiempos contra un exponenciador modular, en [Koc96] es propuesta un especie de técnica de "enceguecimiento"(o *blinding*), similar a las usadas para los esquemas de *firmas ciegas* (*blind signatures* [Cha83]). En ambas implementaciones estudiadas, este tipo de técnicas de enceguecimiento pueden ser adaptadas a fin de producir tiempos de ejecución (prácticamente) fijos en el proceso de generación del vector de subclaves.[4] A continuación describiremos estas adaptaciones.

Sea K una clave DES de peso de Hamming igual a $\mathsf{wt}(K)$ cuyo vector de subclaves queremos generar. Sea K' una secuencia de bits de largo 56 generada de la siguiente manera: seleccionamos al azar $\lfloor \frac{\mathsf{wt}(K)}{2} \rfloor$ (resp. $\lfloor \frac{56-\mathsf{wt}(K)}{2} \rfloor$) bits desde el conjunto de bits en uno (resp. en 0) de K y fijamos los bits correspondientes de K' a 0 (resp. a 1). El proceso de construcción de esta clave K' se ilustra en la Figure 5.2. Notemos que el peso de Hamming de K' es igual a 28, independiente del peso de K, y el peso de $K \oplus K'$ (el *OR* exclusivo entre K y K') es 28 cuando $\mathsf{wt}(K)$ es par y 27 cuando es impar. Modificamos ahora el proceso de generación del vector de subclaves de manera que primero se generan los vectores para las claves K' y $K \oplus K'$. Notemos que el trabajo necesario para generar ambas claves (K' y $K \oplus K'$) es independiente del peso de Hamming de K, por lo que ninguna fuente de tiempos variables es introducida en la construcción de dichas claves. Sean $K'_1, \ldots, K'_{16}$ y $K_1, \ldots, K_{16}$ los elementos de los vectores de subclaves así obtenidos. Dado que cada K'_i (resp. K_i) es una selección permutada de los bits de la clave K' (resp. $K \oplus K'$), el vector de subclaves de K es $K_1 \oplus K'_1, \ldots, K_{16} \oplus K'_{16}$. La Figure 5.2 muestra los tiempos de encriptación de RSA–DES y de la modificación previamente explicada. Notemos la muy clara reducción de las diferencias de tiempo. Esta reducción es lograda a expensas de un incremento del tiempo de encriptación en un factor de aproximadamente 1.6.

[4] Recordemos que el proceso de generación del vector de subclaves es el responsable de todas las diferencias de tiempo observadas de RSA–DES y la mayoría de las diferencias de L–DES.

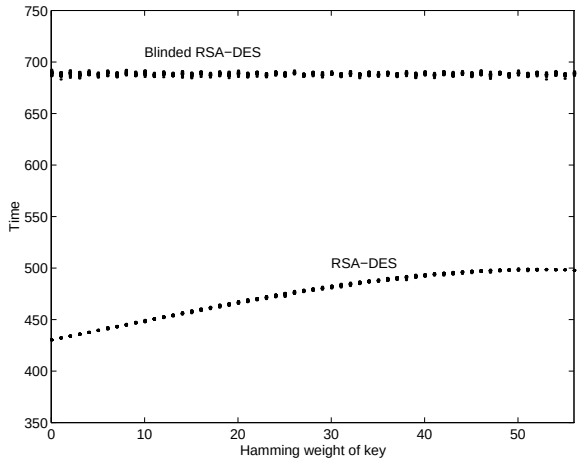


Figura 5.2: Tiempos de encriptación de RSA–DES y RSA–DES modificado.

Desafortunadamente, esta técnica de enceguecimiento aún filtra la paridad del peso de Hamming de la clave DES original, es decir, 1 bit de información. (Un examen cuidadoso de la Fig. 5.2 confirma este hecho). Lo anterior ocurre debido a la variación del peso de Hamming de $K \oplus K'$, esto es, el peso de Hamming de esta clave será 28 o 27 dependiendo si el peso de Hamming de la clave original K es par o impar. Sin embargo, este último problema puede ser arreglado extendiendo un poco más la misma idea. Así, para una clave K dada, construiremos una tercera clave K'', además de K' y $K \oplus K'$. Esta clave K'' se generará en forma análoga a K' sólo que tomando ahora $\lceil \frac{\mathsf{wt}(K)}{2} \rceil$ bits al azar (resp. $\lceil \frac{56-\mathsf{wt}(K)}{2} \rceil$) desde el conjunto de bits en uno de K (resp. en 0) y fijando los bits correspondientes de K'' a 0 (resp. a 1). Adicionalmente, si $\mathsf{wt}(K)$ es impar, un bit al azar de los fijados a 1 en K'' es cambiado a 0. Así el peso de Hamming de K'' será igual a 28 cuando $\mathsf{wt}(K)$ sea impar y 27 cuando sea par. A diferencia de las otras, esta última clave será *descartable*, lo que significa que su vector de subclaves debe ser generado pero los resultados correspondientes a ella desechados. Con ello, el vector de subclaves de la clave original es calculado de la misma forma que en el esquema anterior, es decir efectuando el XOR entre los vectores de subclaves de $K\prime$ y de $K \oplus K'$.

Lo anterior resultará en un tiempo de generación del vector de subclaves dependiente del proceso de las tres claves siguientes

1. K' de peso igual a 28,
2. $K \oplus K'$ de peso igual a 28 y
3. K'' de peso igual a 27.

siempre que el peso de Hamming de la clave original sea par.

Cuando el peso $\mathsf{wt}(K)$ sea impar, las claves generadas serán

1. K' de peso igual a 28,
2. $K \oplus K'$ de peso igual a 27 y
3. K'' de peso igual a 28.

El procedimiento de la Fig. 5.3 muestra en detalle la generación de claves descrita.

ENTRADA: $\mathcal{K} \in \{0,1\}^{56}$ /* clave DES */
CÓDIGO: **Sea** $w = \mathsf{wt}(\mathcal{K})$
Sea $\Lambda_0 = \{\, i \in \{\,1,\ldots,56\,\} : \mathcal{K}_i = 0 \,\}$ /* $\mathcal{K}_i$ es el i-ésimo bit de $\mathcal{K}$ */
Sea $\Lambda_1 = \{\, i \in \{\,1,\ldots,56\,\} : \mathcal{K}_i = 1 \,\}$
Sea $\pi : \{\,1,\ldots,56\,\} \mapsto \{\,1,\ldots,56\,\}$ **permutación al azar, tal que**
$\pi(i) \in \Lambda_0$ **si** $i \in \{\,1,\ldots,w\,\}$ **y**
$\pi(i) \in \Lambda_1$ **si** $i \in \{\,w+1,\ldots,56\,\}$
Sean $\mathcal{K}'$ **y** $\mathcal{K}'' \in \{0,1\}^{56}$ **ambos el vector nulo.**
Para $i = 1$ **hasta** $\lfloor \frac{w}{2} \rfloor$ **hacer**
$\mathcal{K}'_{\pi(i)} = 0$
Si $w = 56$ **y** $i = \lfloor \frac{w}{2} \rfloor$ **entonces** $\mathcal{K}''_{\pi(i)} = 0$
si no $\mathcal{K}''_{\pi(i)} = 1$
Para $i = \lfloor \frac{w}{2} \rfloor + 1$ **hasta** w **hacer**
$\mathcal{K}'_{\pi(i)} = 1$
$\mathcal{K}''_{\pi(i)} = 0$
Para $i = w+1$ **hasta** $w + \lfloor \frac{56-w}{2} \rfloor$ **hacer**
$\mathcal{K}'_{\pi(i)} = 1$
$\mathcal{K}''_{\pi(i)} = 0$
Para $i = w + \lfloor \frac{56-w}{2} \rfloor + 1$ **hasta** 56 **hacer**
$\mathcal{K}'_{\pi(i)} = 0$
Si w **es impar y** $i = 56$ **entonces** $\mathcal{K}''_{\pi(i)} = 1$
si no $\mathcal{K}''_{\pi(i)} = 0$
Sean $KS(\mathcal{K}')$ **y** $KS(\mathcal{K} \oplus \mathcal{K}'') \in \{0,1\}^{48 \cdot 16}$
/* $KS(\mathcal{K}')$ vector de subclaves de $\mathcal{K}'$ */
/* $KS(\mathcal{K} \oplus \mathcal{K}'')$ vector de subclaves de $\mathcal{K} \oplus \mathcal{K}''$ */
Calcular $KS(\mathcal{K}) = KS(\mathcal{K}') \oplus KS(\mathcal{K} \oplus \mathcal{K}'')$.
SALIDA: $KS(\mathcal{K})$ /* vector de subclaves de $\mathcal{K}$ */

Figura 5.3: Generación modificada del vector de subclaves.

Notemos que este procedimiento no introduce diferencias de tiempo adicionales, puesto que el número de operaciones en él realizadas es proporcional al largo (fijo) de la clave DES. De esta manera, se tendrá que el tiempo de generación será el mismo para ambos casos, eliminando la filtración de la paridad de la clave original.

Remark 7 *Hemos visto que las principales fuentes de tiempos de ejecución variables fueron causadas por el proceso de generación del vector de subclaves (*Key Schedule*). En muchas implementaciones en software eficientes el proceso de inicialización de la clave es una operación separada del proceso de encriptación. Esto frustraría un ataque de medición de tiempos si el tiempo de encriptación es constante. Sin embargo, en varios sistemas es impráctico precalcular el vector de subclaves. Por ejemplo, en las tarjetas inteligentes, almacenar valores precalculados no es deseable debido a restricciones de memoria. En particular, no es posible almacenarlos en memoria RAM (debido a que dichos valores deben mantenerse durante largos períodos sin transacciones) y la memoria no volátil (EEPROM) usualmente es degradada en cada ciclo escritura–borrado, haciendo poco conveniente su uso intensivo. (Más detalles en Sect. 2.3 y* [Sim92]*).*

Capítulo 6

Conclusiones

6.1 Resumen y Conclusiones

Estudiamos la vulnerabilidad de uno de los criptosistemas más ampliamente usados en el mundo, DES, contra un ataque de medición de tiempos. Nuestro punto de partida fue la observación de Kocher [Koc96] que en el proceso de generación del vector de subclaves de una clave DES, el tiempo necesario para desplazar cíclicamente los bits no nulos podría ser una fuente de tiempos variables en la encriptación. Por lo tanto, él conjeturó que un ataque de medición de tiempos contra DES podría revelar el peso de Hamming de la clave. Mostramos que, pese a que la observación de Kocher es incorrecta (al menos para las implementaciones analizadas), su conjetura es cierta.

Presentamos un ataque contra DES que supone que el adversario conoce las características de diseño del sistema objetivo. Este ataque es del tipo de *medición de tiempos* y su costo computacional es despreciable. Además, discutimos varios resultados experimentales que muestran que un ataque de medición de tiempos sobre dos implementaciones de DES permite recuperar el peso de Hamming de la clave usada. Por lo tanto, suponiendo que las claves en DES son elegidas al azar, un adversario puede recuperar aproximadamente 3.95 bits de información de la clave (en general, esto puede ser algo optimista, ver Remark 5 en Cap. 4). Según nuestro conocimiento, este trabajo constituye la primera implementación de un ataque de medición de tiempos contra el criptosistema DES. Además, mostramos un experimento computacional que muestra la reducción esperada en el tamaño del espacio de claves buscado usando una implementación real de un ataque de medición de tiempos que recupera el peso de Hamming de la clave en DES.

Vimos también que deducir 3.95 bits de una clave DES es una mejora relativamente modesta con respecto a una búsqueda de fuerza bruta. Sin embargo, señalamos que la obtención del peso de Hamming de la clave es, potencialmente, un hecho más preocupante. En particular, discutimos el caso en que un adversario decide enfocar su atención sobre claves con un peso de Hamming significativamente bajo o significativamente alto. Allí, el adversario puede balancear el tiempo que le toma encontrar una de estas claves (más bien

escasas) con el tiempo necesario para recuperar la clave restringida al peso en cuestión. Esto nos indica que ataques como el aquí examinado deben considerarse con detenimiento en sistemas donde el compromiso de una sola clave puede ser vital para la seguridad futura del sistema.

Mostramos además que en ambas implementaciones de DES analizadas, el tiempo de encriptación T era muy cercano a una función lineal del peso de Hamming de la clave X más algo de ruido blanco e. Como la clave de DES es una secuencia de 56 bits, y las claves son elegidas uniformemente, sabemos que $X \sim Binom\,(56, 1/2)$. Luego, para ciertos α, β, y σ, la dependencia del tiempo en el peso de Hamming de la clave esta dada por

$$T \;=\; \alpha X + \beta + e \;\;, \qquad X \sim Binom\,(56, 1/2) \;\;, \qquad e \sim Norm\,(0, \sigma^2) \;\;.$$

Otra conclusión importante de este trabajo es que no es necesario, a fin de realizar un ataque de medición de tiempos sobre DES, suponer que las características de diseño del sistema objetivo son conocidas. De hecho, propusimos dos métodos estadísticos que permiten a un espía pasivo inferir dichas características sólo a través de mediciones de tiempo, y con ellos montar exitosamente un ataque de medición de tiempos sobre DES. Según nuestro conocimiento, esta es la primera demostración que es posible deducir las características de diseño de un sistema objetivo a través de mediciones de tiempo.

Finalmente, y para evitar el ataque en discusión, propusimos una "técnica de enceguecimiento" que puede ser usada para eliminar casi todas las variaciones de tiempos en las implementaciones analizadas. Esta técnica de enceguecimiento hace que ambas implementaciones analizadas sean más resistentes al tipo de ataque descrito en este trabajo.

6.2 Problemas Abiertos

Es interesante señalar que la filtración de información debido a las características de tiempos no es un problema exclusivo de DES. Por ejemplo, el criptosistema IDEA [LM90, Lai92b] exhibe una dependencia similar. (La Fig. 6.1 muestra los tiempos en una implementación propuesta por los autores en [Lai92a].) Esto entrega indicios potencialmente preocupantes de que generalizaciones y extensiones del ataque aquí propuesto pudieran ser aplicables a otros criptosistemas.

Por otra parte, pese a que en las dos implementaciones examinadas, la de A. Louko y la de RSAEuro, existe una filtración de información de la clave, en general son altamente resistentes a un ataque de medición de tiempos. Esto nos lleva a la interrogante acerca de si un ataque de medición de tiempos pudiera recuperar toda la clave y no sólo el peso de Hamming. Ahora bien, pese a que no fue posible afinar la técnica de los ataques de medición de tiempos para deducir todos los bits de la clave DES, logramos detectar una fuente de tiempos variables de ejecución que no es debida al proceso de generación del vector de subclaves. De hecho, las diferencias en las pendientes de las curvas graficadas en Fig. 4.2 muestran que el tiempo de encriptación, sin contar el tiempo del proceso de generación del vector de subclaves, depende de la clave usada. Esta dependencia es una

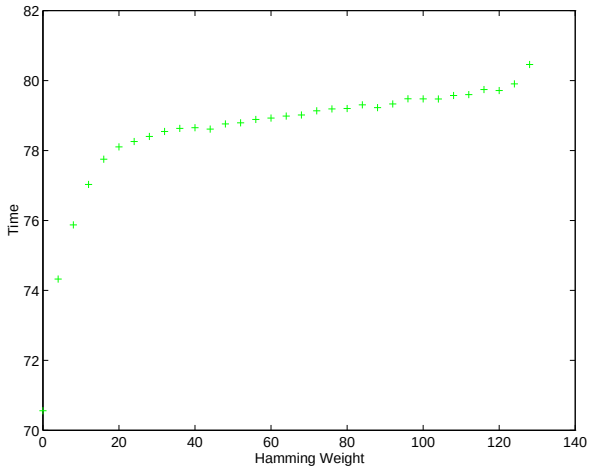


Figura 6.1: Características de tiempos en IDEA

debilidad que podría ser (potencialmente) utilizada para recuperar toda la clave en DES. En particular, abre la posibilidad de que el tiempo demorado en encriptar un mensaje M con una clave K sea una función no–lineal de (ambos) M y K, como por ejemplo, una función monótona creciente en el peso de Hamming de $M \oplus K$. Esto último permitiría a un ataque de medición de tiempos recuperar una clave de DES mediante una selección precisa de los mensajes a encriptar. Sin embargo, para las implementaciones examinadas no nos fue posible identificar fuentes claras de dependencias no–lineales entre las diferencias de tiempo y las entradas del proceso de encriptación (mensaje). Pese a ello, creemos que la información parcial filtrada por ambas implementaciones analizadas de DES dan una clara indicación acerca del mucho cuidado que debe tomarse en la implementación de criptosistemas como DES. De otra manera, no sólo el peso sino toda la clave podría ser comprometida a través de un ataque de medición de tiempos.

Bibliografía

[AK96] R. Anderson y M. Kuhn. Tamper Resistance, a Cautionary Note. En *2nd. Workshop On Electronic Commerce*. USENIX Association, Nov 1996.

[BDL97] D. Boneh, R. A. Demillo, y R. J. Lipton. On the importance of checking cryptographic protocols for faults. En *Advances in Cryptology — EUROCRYPT'97*, Lecture Notes in Computer Science, págs. 37–51. Springer–Verlag, 1997.

[BG96] Mihir Bellare y Shafi Goldwasser. Lecture Notes on Cryptography. Documento sin publicar, Disponible desde `http://www-cse.ucsd.edu/users/mihir/papers/gb.ps.gz`, 1996.

[BS91] E. Biham y A. Shamir. Differential cryptanalysis of DES-like cryptosystems. *Journal of Cryptology*, 4:3–72, 1991.

[BS93a] E. Biham y A. Shamir. *A Differential Cryptanalysis of the Data Encryption Standard*. Springer-Verlag, 1993.

[BS93b] E. Biham y A. Shamir. Differential cryptanalysis of the full 16–round DES. En E. F. Brickell, editor, *Advances in Cryptology — CRYPTO'92*, número 740 en Lecture Notes in Computer Science, págs. 494–502, Santa Barbara, California, 1993. Springer–Verlag.

[BS97] E. Biham y A. Shamir. Differential fault analysis of secret key cryptosystems. Reporte Técnico CS0910, Technion, Computer Science Department, 1997.

[Cha83] D. Chaum. Blind signatures for untraceable payments. En D. Chaum, R. L. Rivest, y A. T. Sherman, editores, *Advances in Cryptology — CRYPTO'82*, págs. 199–203, Santa Barbara, California, 1983. Plenum Press.

[Cop92] D. Coppersmith. The Data Encryption Standard (DES) and Its Strength against Attacks. Reporte Técnico RC 18613, IBM T.J. Watson Center, 1992.

[Cop94] D. Coppersmith. The Data Encryption Standard (DES) and Its Strength against Attacks. *Journal of Cryptology*, 38(3):243–250, Mayo 1994.

[DH76] W. Diffie y M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT–22(6):644–654, Nov. 1976.

[Fel66] W. Feller. *An introduction to probability theory and its applications*, volume I & II. John Wiley & Sons, Inc., 1966. segunda impresión.

[Gol95] Oded Goldreich. Foundations of Cryptography (Fragments of a Book). Documento sin publicar, Disponible desde `http://theory.lcs.mit.edu/~oded/frag.html`, 1995.

[Haz88] M. Hazewinkel, editor. *Encyclopaedia of Mathematics*, volume 1. Kluwer Academic Press, 1988. Una traducción actualizada y anotada de la 'Mathematical Encyclopaedia' soviética.

[Hei95] Kris Heidenstrom. FAQ / Application notes: Timing on the PC family under DOS. Versión 19951220, release 3, Documento sin publicar, Disponible desde `ftp://garbo.uwasa.fi/pc/programming/pctim003.zip`, 1995.

[HK98] A. Hevia y M. Kiwi. Strength of Two Data Encryption Standard Implementations under Timing Attacks. En Cláudio L. Lucchesi y Arnaldo V. Moura, editores, *Latin American Theoretical Informatics, LATIN'98*, Lecture Notes in Computer Science 1380, págs. 192–205. Springer–Verlag, 1998.

[HT97] R. Hogg y E. Tanis. *Probability and statistical inference.* Prentice Hall, quinta edición, 1997.

[Kah67] David Kahn. *The Code-Breakers.* Simon & Schuster, 1967.

[Kap96] J. S. A. Kapp. RSAEuro: A cryptographic toolkit, 1996. Versión 1.04 Internet Release Distribution. Disponible desde `http://www.reapertech.com/RSAEuro`.

[Ker83] Auguste Kerckhoffs. *La Cryptographie Militaire.* Librarie Militaire de L. Baudoin & Cie., Paris, 1883.

[Kiw98] Marcos Kiwi. Introducción a la Criptografía, 1998. Apuntes del Curso, Disponible desde `http://www.dim.uchile.cl/~mkiwi/cripto_98.html`.

[KJJ98] Paul Kocher, Joshua Jaffe, y Benjamin Jun. Introduction to Differential Power Analysis and Related Attacks. Manuscrito sin publicar. Disponible desde `http://www.cryptography.com/dpa/technical/index.html`, 1998.

[Knu98] D. E. Knuth. *The Art of Computer Programming II: Seminumerical Algorithms.* Addison–Wesley, Reading, Massachusetts, tercera edición, 1998.

[Koc96] P. Kocher. Timing attacks on implementations of Diffie–Hellman, RSA, DSS, and other systems. En N. Koblitz, editor, *Advances in Cryptology — CRYPTO'96*, número 1109 en Lecture Notes in Computer Science, págs. 104–113, Santa Barbara, California, 1996. Springer–Verlag.

[Lai92a] Xuejia Lai. Detailed Description and a Software Implementation of the IPES Cipher. Institute for Signal and Information Processing, 1992. ETH-Zentrum, Zurich, Suiza.

[Lai92b] Xuejia Lai. *On the design and security of block ciphers*, volumen 1. Hartung–Gorre Verlag Konstanz, Technische Hochschule (Zurich), 1992. ETH Series in Information Processing, J.L. Massey (editor).

[LM90] Xuejia Lai and James L. Massey. A proposal for a new block encryption standard. En I. B. Damgård, editor, *Advances in Cryptology—EUROCRYPT 90*, volumen 473 de *Lecture Notes in Computer Science*, págs. 389–404. Springer-Verlag, 1991, Mayo 1990.

[Lou92] A. Louko. DES package. Helsinki University of Technology Computing Centre, 1992. Version 2.1, (Disponible via FTP desde `kampi.hut.fi`).

[Man79] M. Morris Mano. *Digital Logic and Computer Design.* Prentice–Hall, primera edición, 1979.

[Mat94a] M. Matsui. The first experimental cryptanalysis of the data encryption standard. En Y. G. Desmedt, editor, *Advances in Cryptology — CRYPTO'94*, número 839 en Lecture Notes in Computer Science, págs. 1–11, Santa Barbara, California, 1994. Springer–Verlag.

[Mat94b] M. Matsui. Linear cryptanalysis method for DES cipher. En T. Helleseth, editor, *Advances in Cryptology — EUROCRYPT'93*, número 765 en Lecture Notes in Computer Science, págs. 386–897, Lofthus, Norway, 1994. Springer–Verlag.

[MvOV97] A. J. Menezes, P. C. van Oorschot, y S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, primera edición, 1997.

[Nb77] National Bureau of Standards. Data encryption standard (DES), 1977. Publicación FIPS 46.

[Nb81] National Bureau of Standards. Guidelines for implementing and Using the NBS Data Encryption Standard, 1981. Publicación FIPS 74.

[Ni97] National Institute of Standards and Technology. Announcing Request for Candidate Algorithm Nominations for the Advanced Encryption Standard (Aes). Registro Federal: 12 de Septiembre 1997 (Volumen 62, Número 177).

[Riv85] Ronald L. Rivest. Testing Implementations of DES, 1985. Manuscrito sin publicar, Disponible desde `http://theory.lcs.mit.edu/~rivest/destest.txt`.

[Ros88] S. Ross. *A first course in probability.* Macmillan Publishing Company, tercera edición, 1988.

[RSA78] R. L. Rivest, A. Shamir, y L.M. Adleman. A method for obtaining digital signatures and public–key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[Sch96] B. Schneier. *Applied Cryptography: protocols, algorithms and source code in C.* John Wiley & Sons, Inc., segunda edición, 1996.

[Sim92] G.J. Simmons (Ed.). *Contemporary Cryptography, The Science of Information Integrity.* IEEE Press, primera edición, 1992.

[Sti95] D. R. Stinson. *Cryptography, Theory and Practice.* CRC Press, primera edición, 1995.

[Zac71] S. Zacks. *The theory of statistical inference.* John Wiley & Sons, Inc., 1971.

Apéndice A

Tiempos vs. Peso de Hamming

	RSA–DES				L–DES			
	Gen. subclaves		Encriptación		Gen. subclaves		Encriptación	
i	Prom.	Dev.	Prom.	Dev.	Prom.	Dev.	Prom.	Dev.
0	171.6	0.51	430.1	0.36	17.4	0.50	76.1	0.56
4	181.8	0.87	437.6	1.87	22.0	0.38	89.0	0.63
8	192.2	0.90	445.0	1.79	28.0	0.48	102.0	0.78
12	202.4	0.84	452.3	1.86	33.7	0.41	115.2	0.85
16	212.8	1.23	459.5	1.42	39.6	0.59	128.3	1.10
20	223.0	1.27	466.6	1.52	45.3	0.76	141.4	1.02
24	232.0	0.86	473.0	2.05	50.7	0.58	154.6	1.25
28	240.9	1.38	479.0	1.84	56.1	0.56	167.6	1.16
32	249.3	1.48	484.2	1.51	61.8	0.62	181.3	1.07
36	257.0	0.76	489.0	1.61	67.1	0.66	194.6	1.09
40	263.6	0.99	492.8	1.97	72.6	0.69	207.6	0.93
44	269.6	0.98	495.7	1.25	77.8	0.65	221.1	0.94
48	274.7	1.58	497.7	1.62	83.2	0.59	234.0	0.86
52	278.3	0.79	498.5	1.44	88.4	0.59	247.5	0.96
56	280.7	1.51	498.0	1.84	93.8	0.57	261.0	0.54

Tabla A.1: Tiempos de generación de las subclaves y de encriptación

Apéndice B

Medición de Tiempo

Técnicas Básicas en MSDOS™

A continuación describiremos ciertas técnicas de medición de tiempo en MSDOS™ utilizadas en este trabajo. Recordemos que nuestra plataforma de trabajo fue un procesador Pentium™ de 120 MHz corriendo MSDOS™.

En los equipos XT y posteriores existe un oscilador de cristal que genera una señal de reloj de 14.31818 MHz. Esta frecuencia es dividida por 12 dando la frecuencia interna de 1.1931816667 MHz (o período de 83809534452μs) con la cual se alimenta a los tres canales del chip contador/cronómetro (*counter/timer chip*, CTC). La precisión de esta frecuencia de reloj depende principalmente de la calidad del oscilador de cristal, el cual tiene típicamente un error de entre 0.0005% y 0.002% del período. En estos equipos el CTC es un chip intel 8253, mientras que en los equipos AT y más modernos es un chip 8254.

El CTC tiene tres *canales* totalmente independientes, numerados *cero*, *uno* y *dos*. Cada canal puede ser operado en uno de los seis modos distintos de operación. Este chip divide la frecuencia de 1.1931866667 MHz en frecuencias más bajas usando divisores programables, al tiempo que produce tres señales de salida.

La salida del canal cero del CTC está directamente conectada a la interrupción IRQ0 y genera un pulso de interrupción de cronómetro (*timer tick interrupt*) `int` 8 unas 18.2065 veces por segundo (cada 54.9254 milisegundos). Este pulso de cronómetro es una *interrupción* regular que permite que ciertas acciones sean ejecutadas periódicamente (como actualizar la hora del sistema y apagar los motores de la unidad de discos flexibles dos segundos después del último acceso).

El contador de pulsos del BIOS es un contador de 32 bits (ubicado en una posición baja de la memoria 0040:006C) el cual contiene el número de pulsaciones (*ticks*) del cronómetro (en unidades de 54.9254 milisegundos) desde medianoche y es usado por MSDOS™ para calcular la hora del día.

El chip CTC subdivide su señal de reloj de 1.1931866667 MHz hasta 18.2065 MHz usando un contador de 16 bits. Este contador es un número entre 1 y 65536 (65536 re-

presentado como 0) el cual parte desde cierto valor programado (llamado el valor de registro de división o *divisor register value*) y se decrementa continuamente en 1. Cuando se debe decrementar a cero, en vez de ello este registro carga nuevamente el valor de registro de división original. Por ejemplo, si el divisor es cero (es decir 65536) la secuencia será $0, 65535, 65534, \ldots, 2, 1, 0, 65535, \ldots$ etc.

Usando una instrucción especial (el comando *Latch*) en el modo dos del canal cero, es posible leer el valor actual en este registro del CTC. En combinación con la variable contador del BIOS, esto puede dar un valor absoluto de tiempo en unidades de $0.8381\mu s$.

El procedimiento de medición de tiempos en nuestras pruebas está basado en estas técnicas y en consecuencia permitió obtener registros absolutos de tiempo en unidades de 0.83809534452 microsegundos. En particular, se usó el contador de pulsos del BIOS y el contador dinámico del canal cero del CTC, asumiendo que el canal cero opera en modo 2 con valor de registro de división de 0 (o divisor de 65536).

Al lector interesado en más detalles y referencias se le recomienda ver [Hei95].

Apéndice C

Código Fuente Implementaciones

A continuación se muestra el código fuente de cada una de las implementaciones analizadas. El código es el original excepto por ciertas secciones irrelevantes para el funcionamiento y comprensión del resto que fueron removidas.

C.1 RSADES

En la implementación de DES de **RSAEuro** se utilizaron las funciones `DES_CBCInit` y `DES_CBCUpdate`.

C.1.1 Archivo header

```
/*  DES.H - header file for DESC.C

    Copyright (c) J.S.A.Kapp 1994 - 1996.

        RSAEURO - RSA Library compatible with RSAREF 2.0.

        All functions prototypes are the Same as for RSAREF.
        To aid compatiblity the source and the files follow the
        same naming comventions that RSAREF uses.  This should aid
        direct importing to your applications.

        This library is legal everywhere outside the US.  And should
        NOT be imported to the US and used there.

        DES Code header file.
        Revision 1.00 - JSAK.
*/
```

```
#ifndef _DES_H_
#define _DES_H_

typedef struct {
  UINT4 subkeys[32];                                    /* subkeys */
  UINT4 iv[2];                                  /* initializing vector */
  UINT4 originalIV[2];                      /* for restarting the context */
  int encrypt;                                          /* encrypt flag */
} DES_CBC_CTX;
#endif /* _DES_H_ */
```

C.1.2 Funciones Principales

```
/*  DESC.C - Data Encryption Standard routines for RSAEURO

    Copyright (c) J.S.A.Kapp 1994 - 1996.

        RSAEURO - RSA Library compatible with RSAREF(tm) 2.0.

        All functions prototypes are the Same as for RSAREF(tm).
        To aid compatiblity the source and the files follow the
        same naming comventions that RSAREF(tm) uses.  This should aid
        direct importing to your applications.

        This library is legal everywhere outside the US.  And should
        NOT be imported to the US and used there.

        Based on Outerbridge's D3DES (V5.09) 1992 Vintage.

        DESX(tm) - RSA Data Security.

        All Trademarks Acknowledged.
*/

#include "rsaeuro.h"
#include "des.h"

static UINT2 bytebit[8] = {
        0200, 0100, 040, 020, 010, 04, 02, 01
};
static UINT4 bigbyte[24] = {
```

```
        0x800000L, 0x400000L, 0x200000L, 0x100000L,
        0x80000L,  0x40000L,  0x20000L,  0x10000L,
        0x8000L,   0x4000L,   0x2000L,   0x1000L,
        0x800L,    0x400L,    0x200L,    0x100L,
        0x80L,     0x40L,     0x20L,     0x10L,
        0x8L,      0x4L,      0x2L,      0x1L
};
static unsigned char totrot[16] = {
        1, 2, 4, 6, 8, 10, 12, 14, 15, 17, 19, 21, 23, 25, 27, 28
};
static unsigned char pc1[56] = {
        56, 48, 40, 32, 24, 16,  8,      0, 57, 49, 41, 33, 25, 17,
         9,  1, 58, 50, 42, 34, 26,     18, 10,  2, 59, 51, 43, 35,
        62, 54, 46, 38, 30, 22, 14,      6, 61, 53, 45, 37, 29, 21,
        13,  5, 60, 52, 44, 36, 28,     20, 12,  4, 27, 19, 11,  3
};
static unsigned char pc2[48] = {
        13, 16, 10, 23,  0,  4,  2, 27, 14,  5, 20,  9,
        22, 18, 11,  3, 25,  7, 15,  6, 26, 19, 12,  1,
        40, 51, 30, 36, 46, 54, 29, 39, 50, 44, 32, 47,
        43, 48, 38, 55, 33, 52, 45, 41, 49, 35, 28, 31
};
UINT4 Spbox[8][64] = {
        0x01010400L, 0x00000000L, 0x00010000L, 0x01010404L,
        0x01010004L, 0x00010404L, 0x00000004L, 0x00010000L,
        0x00000400L, 0x01010400L, 0x01010404L, 0x00000400L,
        .
        .
        .
        0x10040000L, 0x10001000L, 0x10001040L, 0x00000000L,
        0x10041040L, 0x00041000L, 0x00041000L, 0x00001040L,
        0x00001040L, 0x00040040L, 0x10000000L, 0x10041000L
};

/* Initialize context.  Caller must zeroize the context when finished. */

void DES_CBCInit(context, key, iv, encrypt)
DES_CBC_CTX *context;           /* context */
unsigned char *key;             /* key */
unsigned char *iv;              /* initializing vector */
int encrypt;         /* encrypt flag (1 = encrypt, 0 = decrypt) */
```

```
{
        /* Save encrypt flag to context. */
        context->encrypt = encrypt;

        /* Pack initializing vector into context. */

        scrunch(context->iv, iv);
        scrunch(context->originalIV, iv);

        /* Precompute key schedule */

        deskey(context->subkeys, key, encrypt);
}

/* DES-CBC block update operation. Continues a DES-CBC encryption
         operation, processing eight-byte message blocks, and updating
         the context.
         This requires len to be a multiple of 8.
*/
int DES_CBCUpdate(context, output, input, len)
DES_CBC_CTX *context;           /* context */
unsigned char *output;          /* output block */
unsigned char *input;           /* input block */
unsigned int len;               /* length of input and output blocks */
{
        UINT4 inputBlock[2], work[2];
        unsigned int i;

        if(len % 8)  /* block size check */
                return(RE_LEN);

        for(i = 0; i < len/8; i++) {
                scrunch(inputBlock, &input[8*i]);

                /* Chain if encrypting. */

                if(context->encrypt == 0) {
                        *work = *inputBlock;
                        *(work+1) = *(inputBlock+1);
                }else{
                        *work = *inputBlock ^ *context->iv;
```

```
                *(work+1) = *(inputBlock+1) ^ *(context->iv+1);
        }

        desfunc(work, context->subkeys);

        /* Chain if decrypting, then update IV. */

        if(context->encrypt == 0) {
                *work ^= *context->iv;
                *(work+1) ^= *(context->iv+1);
                *context->iv = *inputBlock;
                *(context->iv+1) = *(inputBlock+1);
        }else{
                *context->iv = *work;
                *(context->iv+1) = *(work+1);
        }
        unscrunch (&output[8*i], work);
    }

    /* Clear sensitive information. */

    R_memset((POINTER)inputBlock, 0, sizeof(inputBlock));
    R_memset((POINTER)work, 0, sizeof(work));

    return(ID_OK);
}

void scrunch (into, outof)
UINT4 *into;
unsigned char *outof;
{
    *into    = (*outof++ & 0xffL) << 24;
    *into   |= (*outof++ & 0xffL) << 16;
    *into   |= (*outof++ & 0xffL) << 8;
    *into++ |= (*outof++ & 0xffL);
    *into    = (*outof++ & 0xffL) << 24;
    *into   |= (*outof++ & 0xffL) << 16;
    *into   |= (*outof++ & 0xffL) << 8;
    *into   |= (*outof   & 0xffL);
}
```

```
void unscrunch(into, outof)
unsigned char *into;
UINT4 *outof;
{
        *into++ = (unsigned char)((*outof >> 24) & 0xffL);
        *into++ = (unsigned char)((*outof >> 16) & 0xffL);
        *into++ = (unsigned char)((*outof >>  8) & 0xffL);
        *into++ = (unsigned char)( *outof++      & 0xffL);
        *into++ = (unsigned char)((*outof >> 24) & 0xffL);
        *into++ = (unsigned char)((*outof >> 16) & 0xffL);
        *into++ = (unsigned char)((*outof >>  8) & 0xffL);
        *into   = (unsigned char)( *outof        & 0xffL);
}

/* Compute DES Subkeys */

void deskey(subkeys, key, encrypt)
UINT4 subkeys[32];
unsigned char key[8];
int encrypt;
{
        UINT4 kn[32];
        int i, j, l, m, n;
        unsigned char pc1m[56], pcr[56];

        for(j = 0; j < 56; j++) {
                l = pc1[j];
                m = l & 07;
                pc1m[j] = (unsigned char)((key[l >> 3] & bytebit[m]) ? 1 : 0);
        }
        for(i = 0; i < 16; i++) {
                m = i << 1;
                n = m + 1;
                kn[m] = kn[n] = 0L;
                for(j = 0; j < 28; j++) {
                        l = j + totrot[i];
                        if(l < 28) pcr[j] = pc1m[l];
                        else pcr[j] = pc1m[l - 28];
                }
                for(j = 28; j < 56; j++) {
                        l = j + totrot[i];
```

```
                        if(l < 56) pcr[j] = pc1m[l];
                        else pcr[j] = pc1m[l - 28];
                }
                for(j = 0; j < 24; j++) {
                        if(pcr[pc2[j]])
                                kn[m] |= bigbyte[j];
                        if(pcr[pc2[j+24]])
                                kn[n] |= bigbyte[j];
                }
        }
        cookey(subkeys, kn, encrypt);

        R_memset((POINTER)pc1m, 0, sizeof(pc1m));
        R_memset((POINTER)pcr, 0, sizeof(pcr));
        R_memset((POINTER)kn, 0, sizeof(kn));
}

static void cookey(subkeys, kn, encrypt)
UINT4 *subkeys;
UINT4 *kn;
int encrypt;
{
        UINT4 *cooked, *raw0, *raw1;
        int increment;
        unsigned int i;

        raw1 = kn;
        cooked = encrypt ? subkeys : &subkeys[30];
        increment = encrypt ? 1 : -3;

        for (i = 0; i < 16; i++, raw1++) {
                raw0 = raw1++;
                *cooked    = (*raw0 & 0x00fc0000L) << 6;
                *cooked   |= (*raw0 & 0x00000fc0L) << 10;
                *cooked   |= (*raw1 & 0x00fc0000L) >> 10;
                *cooked++ |= (*raw1 & 0x00000fc0L) >> 6;
                *cooked    = (*raw0 & 0x0003f000L) << 12;
                *cooked   |= (*raw0 & 0x0000003fL) << 16;
                *cooked   |= (*raw1 & 0x0003f000L) >> 4;
                *cooked   |= (*raw1 & 0x0000003fL);
                cooked += increment;
```

```
        }
}

#define         F(l,r,key){\
        work = ((r >> 4) | (r << 28)) ^ *key;\
        l ^= Spbox[6][work & 0x3f];\
        l ^= Spbox[4][(work >> 8) & 0x3f];\
        l ^= Spbox[2][(work >> 16) & 0x3f];\
        l ^= Spbox[0][(work >> 24) & 0x3f];\
        work = r ^ *(key+1);\
        l ^= Spbox[7][work & 0x3f];\
        l ^= Spbox[5][(work >> 8) & 0x3f];\
        l ^= Spbox[3][(work >> 16) & 0x3f];\
        l ^= Spbox[1][(work >> 24) & 0x3f];\
}

void desfunc(block,ks)
UINT4 *block;                   /* Data block */
UINT4 *ks;          /* Key schedule */
{
        unsigned long left,right,work;

        left = block[0];
        right = block[1];

        work = ((left >> 4) ^ right) & 0x0f0f0f0f;
        right ^= work;
        left ^= work << 4;
        work = ((left >> 16) ^ right) & 0xffff;
        right ^= work;
        left ^= work << 16;
        work = ((right >> 2) ^ left) & 0x33333333;
        left ^= work;
        right ^= (work << 2);
        work = ((right >> 8) ^ left) & 0xff00ff;
        left ^= work;
        right ^= (work << 8);
        right = (right << 1) | (right >> 31);
        work = (left ^ right) & 0xaaaaaaaa;
        left ^= work;
        right ^= work;
```

```
        left = (left << 1) | (left >> 31);

        /* Now do the 16 rounds */
        F(left,right,&ks[0]);
        F(right,left,&ks[2]);
        F(left,right,&ks[4]);
        F(right,left,&ks[6]);
        F(left,right,&ks[8]);
        F(right,left,&ks[10]);
        F(left,right,&ks[12]);
        F(right,left,&ks[14]);
        F(left,right,&ks[16]);
        F(right,left,&ks[18]);
        F(left,right,&ks[20]);
        F(right,left,&ks[22]);
        F(left,right,&ks[24]);
        F(right,left,&ks[26]);
        F(left,right,&ks[28]);
        F(right,left,&ks[30]);

        right = (right << 31) | (right >> 1);
        work = (left ^ right) & 0xaaaaaaaa;
        left ^= work;
        right ^= work;
        left = (left >> 1) | (left  << 31);
        work = ((left >> 8) ^ right) & 0xff00ff;
        right ^= work;
        left ^= work << 8;
        work = ((left >> 2) ^ right) & 0x33333333;
        right ^= work;
        left ^= work << 2;
        work = ((right >> 16) ^ left) & 0xffff;
        left ^= work;
        right ^= work << 16;
        work = ((right >> 4) ^ left) & 0x0f0f0f0f;
        left ^= work;
        right ^= work << 4;

        *block++ = right;
        *block = left;
}
```

C.2 L–DES

En la implementación de DES de **A. Louko** se utilizaron las funciones `des_set_key` y `des_ecb_encrypt`.

C.2.1 Archivo header

```
typedef unsigned char    des_u_char; /* This should be an 8-bit unsigned type */
typedef unsigned long    des_u_long; /* This should be a 32-bit unsigned type */

typedef struct {
  des_u_char        data[8];
} C_Block;

typedef struct {
  des_u_long        data[32];
} Key_schedule;

#define DES_DECRYPT        0x0001
#define DES_NOIPERM        0x0100
#define DES_NOFPERM        0x0200
#define DES_REVBITS        0x0400        /* For SUN compatibility */
```

C.2.2 Funciones Principales

```
#define P_IND(x) (x)

int
des_ecb_encrypt(input,output,schedule,mode)
C_Block                *input;
C_Block                *output;
Key_schedule        *schedule;
int                mode;
{
  C_Block        ibuf;
  des_u_long        L[2],R[2],R0,R1;
  des_u_long        Lnext[2];
  int        i;
  int        encrypt;
  des_u_long        *des_spe_table0 = des_spe_table;
#define des_spe_table des_spe_table0
```

```
if (!(mode & DES_NOIPERM)) {
  if (mode & DES_REVBITS) {
    des_do_iperm_rev(input,&ibuf);
  } else {
    des_do_iperm(input,&ibuf);
  }
} else {
  if (mode & DES_REVBITS)
    des_bitrev(input,&ibuf);
  else
    copy8(*input,ibuf);
}

encrypt = !(mode & DES_DECRYPT);
des_expand(&ibuf.data[0],&L[0]);
des_expand(&ibuf.data[4],&R[0]);
R0 = R[0]; R1 = R[1];

for(i = 0; i < 16; i++) {
  Lnext[0] = R0; Lnext[1] = R1;
  {
    int                ki = encrypt ? i : 15 - i;
    des_u_long        *keyptr = &schedule->data[ki*2];
    des_u_char        *p;
    des_u_long        F[2];
    int                i;

    F[0] = R0; F[1] = R1;
    F[0] ^= keyptr[0];
    F[1] ^= keyptr[1];
    p = (des_u_char*)F;
    R0 = R1 = 0;
    i = 0;

    R0 ^= des_spe_table[i++*64 + p[P_IND(0)]];
    R1 ^= des_spe_table[i++*64 + p[P_IND(0)]];
    R0 ^= des_spe_table[i++*64 + p[P_IND(1)]];
    R1 ^= des_spe_table[i++*64 + p[P_IND(1)]];
    R0 ^= des_spe_table[i++*64 + p[P_IND(2)]];
    R1 ^= des_spe_table[i++*64 + p[P_IND(2)]];
    R0 ^= des_spe_table[i++*64 + p[P_IND(3)]];
```

```
      R1 ^= des_spe_table[i++*64 + p[P_IND(3)]];
      R0 ^= des_spe_table[i++*64 + p[P_IND(4)]];
      R1 ^= des_spe_table[i++*64 + p[P_IND(4)]];
      R0 ^= des_spe_table[i++*64 + p[P_IND(5)]];
      R1 ^= des_spe_table[i++*64 + p[P_IND(5)]];
      R0 ^= des_spe_table[i++*64 + p[P_IND(6)]];
      R1 ^= des_spe_table[i++*64 + p[P_IND(6)]];
      R0 ^= des_spe_table[i++*64 + p[P_IND(7)]];
      R1 ^= des_spe_table[i++*64 + p[P_IND(7)]];
    }
    R0 ^= L[0];
    R1 ^= L[1];
    copy8(*Lnext,*L);
  }

  R[0] = R0; R[1] = R1;
  val4(ibuf.data[0]) = des_unexpand(R);
  val4(ibuf.data[4]) = des_unexpand(L);

  if (!(mode & DES_NOFPERM)) {
    if (mode & DES_REVBITS)
      des_do_fperm_rev(&ibuf,output);
    else
      des_do_fperm(&ibuf,output);
  } else {
    if (mode & DES_REVBITS)
      des_bitrev(&ibuf,output);
    else
      copy8(ibuf,*output);
  }
}

const static des_u_long ksched_arr[] = {
#include "ksched.h"
};

des_set_key(key,schedule)
C_Block        *key;
Key_schedule        *schedule;
{
  const des_u_long        *kp;
```

```
  des_u_long        *kp2;
  int                i;
  int                j;

  for(i = 0; i < 32; i++) {
    schedule->data[i] = 0;
  }
  kp = ksched_arr;
  for(i = 0; i < 8; i++) {
    for(j = 0; j < 7; j++) {
      if (key->data[i] & (1 << j)) {
        kp2 = schedule->data;
        *kp2++ |= *kp++; *kp2++ |= *kp++; *kp2++ |= *kp++; *kp2++ |= *kp++;
        *kp2++ |= *kp++; *kp2++ |= *kp++; *kp2++ |= *kp++; *kp2++ |= *kp++;
        *kp2++ |= *kp++; *kp2++ |= *kp++; *kp2++ |= *kp++; *kp2++ |= *kp++;
        *kp2++ |= *kp++; *kp2++ |= *kp++; *kp2++ |= *kp++; *kp2++ |= *kp++;
        *kp2++ |= *kp++; *kp2++ |= *kp++; *kp2++ |= *kp++; *kp2++ |= *kp++;
        *kp2++ |= *kp++; *kp2++ |= *kp++; *kp2++ |= *kp++; *kp2++ |= *kp++;
        *kp2++ |= *kp++; *kp2++ |= *kp++; *kp2++ |= *kp++; *kp2++ |= *kp++;
        *kp2++ |= *kp++; *kp2++ |= *kp++; *kp2++ |= *kp++; *kp2++ |= *kp++;
      } else {
        kp += 32;
      }
    }
  }
}
```