CAPITULO 18

PROTECCION Y SEGURIDAD

18.1. PROTECCION

Hasta ahora hemos visto elementos de protección que iban surgiendo según las necesidades de los mecanismos o administraciones que hemos ido estudiando. Estos elementos fueron: instrucciones privilegiadas, protecciones de memoria, operaciones de E/S realizadas por el Sistema Operativo, permisos de acceso a archivos, etc. Ahora trataremos de formalizar este concepto.

Los diversos procesos en un SO deben estar protegidos de las actividades de los otros. Con ese propósito se crearon diversos mecanismos, que pueden usarse para asegurar que los archivos, segmentos de memoria, CPU y otros recursos pueden ser usados solo por los procesos que tienen autorización del SO.

Por ejemplo, la facilidad de control de acceso en un sistema de archivos permite a los usuarios dictar cómo y quién puede acceder a sus archivos. El hardware de direccionamiento de memoria asegura que cada proceso solo ejecutará dentro de su espacio. El timer asegura que ningún proceso obtenga la CPU y no la libere. Finalmente, los usuarios no tienen permiso de hacer su propia E/S para proteger la integridad de los periféricos.

Examinemos en más detalle este problema, y construyamos un modelo único para implementar protección.

18.1.1. Objetivos de la protección

A medida que los sistemas se hacen más sofisticados, se hace necesario proteger su integridad. La protección, originalmente era un agregado a los sistemas operativos con multiprogramación, para que los usuarios pudieran compartir de forma segura espacio en común, como ser un directorio, o memoria. Los conceptos modernos de protección están relacionados con la posibilidad de aumentar la rentabilidad de los sistemas complejos que comparten recursos.

La Protección se refiere a un mecanismo para controlar el acceso de programas, procesos o usuarios a los recursos definidos por un sistema. Este mecanismo debe proveer los medios para especificar los controles a ser impuestos, junto con medios de refuerzo. Distinguimos entre protección y seguridad, que es una medida de la confianza de que la integridad de un sistema y sus datos serán preservados.

Hay varios motivos para la protección. El más obvio es la necesidad de prevenir la violación de una restricción de acceso por un usuario del sistema. Sin embargo es de mayor importancia la necesidad de asegurar que cada componente de programa activo en un sistema usa recursos de forma consistente con los permisos establecidos para el uso de esos recursos.

La protección puede mejorar la rentabilidad detectando errores latentes en interfases entre componentes de subsistemas. La detección temprana de ellos pueden prevenir la contaminación de un subsistema saludable por un sistema que no funciona bien. Un recurso no protegido no puede defenderse del uso de un usuario incompetente o desautorizado. Un sistema orientado a protección provee medios para distinguir entre el uso autorizado y el que no.

18.1.2. MECANISMOS Y POLITICAS

Un sistema puede verse como un conjunto de procesos y recursos. Para asegurar la operación eficiente y ordenada del sistema, los procesos están sujetos a políticas que gobiernan el uso de esos recursos. El rol de la protección es proveer un mecanismo para reforzar las políticas que gobiernan el uso de recursos. Estas políticas se pueden establecer en diversas formas. Algunas se fijan en el diseño del sistema, mientras que otras se formulan por la construcción de un sistema. Además hay otras definidas por los usuarios individuales para proteger sus propios archivos y programas. Un sistema de protección tiene que tener flexibilidad para permitir una variedad de políticas.

Las políticas para uso de recursos pueden variar, dependiendo de la aplicación, y están sujetas a cambio a través del tiempo. Por estas causas, la protección no se considera solo un problema del diseñador del sistema operativo. Tiene que ser una herramienta para el programador, de tal forma que los recursos creados y soportados por subsistemas de aplicación puedan cuidarse del mal uso. En este capítulo describimos los mecanismos que tiene que proveer el sistema operativo, de tal forma que los diseñadores de aplicaciones los puedan usar para diseñar su software de protección propio.

Es fundamental separar la Política del Mecanismo. El mecanismo determina como hacer algo, las políticas determinan que es lo que se hará, pero no como. Esta separación es muy importante por cuestiones de flexibilidad. Las políticas pueden cambiar de tiempo en tiempo, y en el peor de los casos, un cambio en la política requerirá un cambio en el mecanismo. Los mecanismos generales podrían ser más deseables, porque un cambio en la política solo requerirá la modificación de una serie de tablas o parámetros.

18.1.3. Dominios de protección

Un sistema de computación es una colección de procesos y objetos (de hardware - CPU, impresoras, etc. - y software - archivos, programas, semáforos, etc -). Cada objeto tiene un nombre único que lo diferencia de los demás objetos del sistema, y puede ser accedido solo a través de operaciones bien definidas y significativas. Los objetos son, esencialmente, Tipos Abstractos de Datos.

Las operaciones que son posibles pueden depender del objeto. Por ejemplo, una CPU solo puede ejecutar. Los segmentos de memoria pueden ser leídos o escritos, una lectora de tarjetas sólo puede ser leída. Lo mismo respecto de cintas, discos, archivos y otros dispositivos.

Obviamente, un proceso debería tener permiso de acceder solo a los recursos a los que tiene permiso. Más aún, en cualquier momento solo debe poder acceder a los recursos que necesita para su tarea. Por ejemplo, cuando el proceso p invoca al procedimiento A, el procedimiento solo debe poder acceder a sus variables y los parámetros formales que se le pasaron; no debe poder acceder a todas las variables de p. Similarmente consideremos cuando p invoca un compilador para compilar un archivo en particular. El compilador no debe poder acceder a cualquier archivo arbitrario, sino solo a un subconjunto bien definido de archivos (fuentes, listados, objetos, etc.) relacionados con el archivo a ser compilado. Inversamente, el compilador puede tener archivos privados usados para propósitos de optimización o estadísticas, que el proceso no debe acceder.

Para facilitar este esquema, introducimos el concepto de *Dominio de Protección*. Un proceso opera con un dominio de protección, que especifica los recursos a los que puede acceder el proceso. Cada dominio define un conjunto de objetos y los tipos de operaciones que se pueden invocar en cada objeto. La posibilidad de ejecutar una operación en un objeto es un derecho de acceso. Un dominio es una colección de derechos de acceso, cada uno de los cuales es un par ordenado { nombre de objeto, conjunto de derechos }. Si, por ejemplo, el dominio D tiene el derecho de acceso { archivo F, {lectura, escritura} }, entonces un proceso que ejecuta en el dominio D puede leer y escribir el archivo F, y no puede hacer ninguna otra operación en ese objeto.

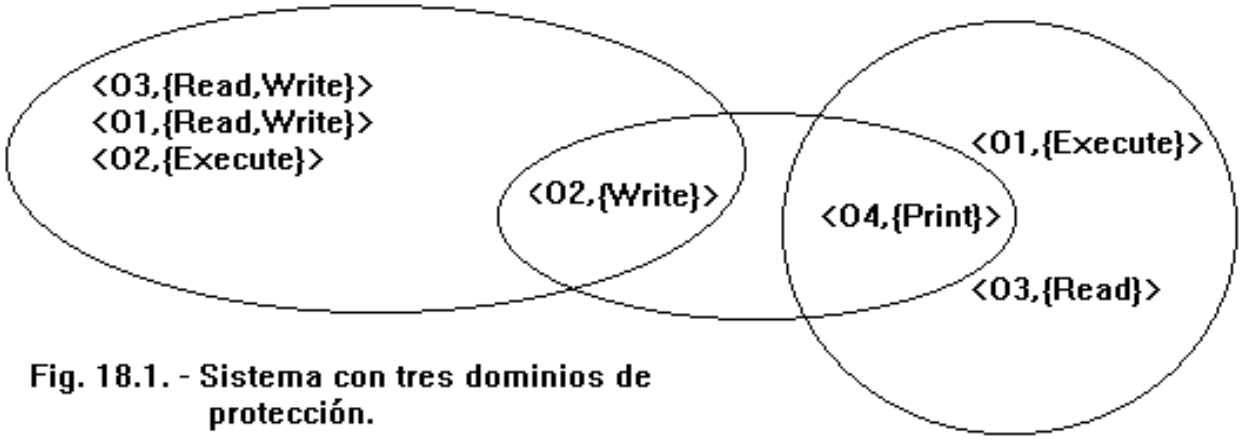

Fig. 18.1. - Sistema con tres dominios de protección.

Los dominios no necesitan ser disjuntos: pueden compartir los derechos de acceso. Por ejemplo, en la figura 18.1. tenemos 3 dominios: D1, D2 y D3. El derecho de acceso { O4, {print} } está compartido por D2 y D3, significando que un proceso ejecutando en cualquiera de estos dos dominios puede imprimir el objeto O4. Notar que un proceso debe estar ejecutando en el dominio D1 para leer y escribir en el objeto O1. Por otro lado, solo los procesos en el dominio D3 pueden ejecutar el objeto O1.

Consideremos el modo standard usuario/supervisor. Cuando un proceso ejecuta en modo supervisor, puede ejecutar instrucciones privilegiadas, y tomar control de todo el sistema. Por otro lado, si el proceso ejecuta en modo usuario, solo puede invocar instrucciones no privilegiadas. Estos dos modos protegen al sistema operativo (que ejecuta en modo supervisor) de los procesos de usuarios (que ejecutan en modo usuario). En un sistema operativo multiprogramado, dos dominios de protección son insuficientes, porque los usuarios quieren estar protegidos uno del otro. Es necesario un esquema más elaborado.

18.1.4. LA MATRIZ DE ACCESOS

Nuestro modelo de protección puede verse de forma abstracta como una matriz. Las filas de la misma representa los dominios, y las columnas, objetos. Cada entrada en la matriz consiste de un conjunto de derechos de acceso. Como los objetos están definidos explícitamente por la columna, podemos omitir el nombre del objeto del derecho de acceso. La entrada acceso(i,j) define el conjunto de operaciones que un proceso, ejecutando en el dominio Di, puede invocar sobre el objeto Oj.

Para ilustrar estos conceptos, la matriz de 18.2. tiene 4 dominios y 5 objetos: 3 archivos, una lectora y una impresora. Cuando un proceso ejecuta en el dominio D1, puede leer F1 y F3.

Objeto / Dominio	F1	F2	F3	Lector tarjetas	Impresora
D1	Read		Read		
D2				Read	Print
D3		Read	Execut		
D4	Read Write		Read Write		

Fig. 18.2. - Matriz de accesos.

18.1.5. Implementación de la Matriz de Accesos

Esta matriz generalmente es una matriz esparza, o sea que casi todas sus entradas están vacías. Existen varias implementaciones posibles:

18.1.5.1. TABLA GLOBAL

La implementación más simple de la matriz de accesos es una tabla global, consistiendo de un conjunto de ternas { dominio, objeto, conjunto de derechos }. Cuando se ejecuta una operación M en un objeto Oj dentro del dominio Di, se busca en la tabla { Di, Oj, Rk }, donde M pertenece a Rk. Si se encuentra la terna, se permite la operación, sino ocurre una excepción. Esta implementación tiene varias desventajas. La tabla suele ser muy grande, y no puede mantenerse en memoria por lo tanto existen más E/S. Además es difícil agrupar objetos o dominios con características similares. Por ejemplo., si un objeto puede ser leído por cualquiera, tiene que tener una entrada por separado en cada dominio.

18.1.5.2. LISTAS DE ACCESO

Cada columna en la matriz de accesos se puede implementar como una lista de accesos para un objeto. Las entradas vacías se obvian. La lista resultante para cada objeto consta de pares ordenados { dominio, conjunto de derechos }, que define todos los dominios con un conjunto no vacío de derechos de acceso para ese objeto.

Esta aproximación puede extenderse para definir una lista más un conjunto Default de derechos de acceso. Cuando la operación M en un objeto Oj es intentada en el dominio Di, buscamos la lista de accesos para el objeto Oj, y miramos una entrada { Dj, Rk } con M perteneciente a Rk. Si se encuentra, permitimos la operación; si no miramos el conjunto default. Si M esta, también permitimos el acceso, sino, se niega el acceso.

18.1.5.3. LISTAS DE CAPACIDADES

Aquí el orden está dado por las filas de la matriz, donde cada fila es para un dominio.

Una **lista de capacidades** para un dominio es una lista de objetos, y las operaciones permitidas sobre él.

Un objeto, en general, está representado por su nombre o dirección (llamado capacidad).

Para ejecutar una operación r sobre un objeto O(j), el proceso invoca r especificando la "capacidad" (pointer) para el objeto O(j) como un parámetro. La posesión de la capacidad hace posible su acceso.

Este sistema es muy seguro si no se permite el acceso directo de la "capacidad" en el espacio de dirección del usuario (o sea, no tiene acceso a ella, por lo tanto no la puede modificar en forma directa).

Si todas las "capacidades" están protegidas, el objeto al cual protegen está seguro.

Las capacidades se distinguen de otros datos (tipos) por una de estas dos razones:

- Cada objeto tiene un rótulo (tag) para denotar su tipo como capacidad o como dato accesible. Los rótulos no deben ser accesibles por los programas de aplicación. Se usa al Hardware o al Firmware para hacer esto más fuerte. El hardware debe distinguir los tipos de datos (enteros, punteros, instrucciones, puntos flotantes, etc.).
- El espacio de direcciones de un programa debe ser dividido en dos partes: la de código y datos (accesible por el mismo), y la parte de lista de capacidades, accesible solo por el Sistema Operativo.

Ejemplos:
Un ejemplo serían los semáforos, a los que solo es posible acceder por medido de operadores P y V.
Una forma de implementación sería la de memoria segmentada.
Otro ejemplo es una L.C.U. del Administrador de Información.

18.1.5.4. IMPLEMENTACION DEL MECANISMO DE LOCK/KEY

El mecanismo de Lock/key es un compromiso entre listas de acceso y listas de capacidad.
Cada objeto tiene una lista de bits de candado (locks), y cada dominio tiene otra lista de bits de llave (keys).

Un proceso solo puede ejecutar (acceder) al objeto si el dominio al cual pertenece tiene llaves que coincidan con los candados del objeto.

Las llaves del dominio son manejadas solo por el sistema operativo.

Un ejemplo de esto es el Storage Control Key (deben coincidir los bits de la PSW o del canal con los bits de la página cuando no se usa el DAT).

18.1.6. Comparación de las implementaciones

Las Listas de Acceso corresponden a las necesidades de usuario. Cuando se crea un objeto especifica a qué dominio pertenece. En sistemas grandes, la búsqueda puede llegar a ser muy grande.

Las Listas de Capacidades no son sencillas de implementar por el usuario. Es eficiente una vez implementado, pues solo es necesario verificar que la capacidad es válida. La revocación es muy complicada, porque las capacidades están distribuidas por todo el sistema (recordemos L.C.U. de Administración de Información).

El mecanismo de Llave/Candado es una solución de compromiso. Es efectivo y flexible. Si bien cualquier cambio solo requiere cambiar la configuración de unos pocos bits, si el sistema posee una gran volatilidad (muchos dominios u objetos nuevos o que desaparecen, muchos permisos que cambian o se agregan constantemente) puede ser necesario replantear la función de mapeo lo cual, en estos casos, torna el mecanismo muy complejo.

18.1.7. Estructuras de protección dinámicas

La asociación entre un proceso y un dominio puede ser estática si los recursos disponibles para ese proceso lo serán para toda su ejecución, o dinámica si existen cambios de accesos o dominios. Si se permiten estos cambios, posiblemente sean violadas las protecciones.

Un mecanismo que nos permite implementarlo es incluir a los mismos dominios como objetos de la matriz de accesos, y cuando sean necesarios cambios en los accesos incluimos a la propia matriz como objeto. Como lo que se quiere es que cada entrada pueda ser modificada en forma individual, consideraremos cada entrada como un objeto.

18.1.8. Cambio de Dominio

SWITCH.

Un proceso puede cambiar del dominio D(i) al dominio D(j) sii el derecho de acceso SWITCH pertenece a Acceso(i,j).

Ejemplo:

Objeto / Dominio	F1	F2	F3	Lector tarjetas	Impresora	D1	D2	D3	D4
D1	Read		Read				Swit.		
D2				Read	Print			Swit.	Switch
D3		Read	Execut						
D4	Read Write		Read Write			Swit.			

Fig. 18.3. - Matriz de la figura 18.2 con Dominios como Objetos.

Luego: D(2) puede cambiar a D(4) o D(3)
D(4) puede cambiar a D(1) pero no a D(2)
D(1) puede cambiar a D(2) pero no a D(3).

18.1.7. Cambio de contenido de la matriz de accesos

El permitir cambios controlados en la matriz de acceso requiere de tres operaciones adicionales: copy, owner y control

COPY

La capacidad de copiar un derecho de acceso existente de un dominio a otro (fila) de la matriz de accesos se indica agregando un asterisco al derecho de acceso.

El derecho COPY solo permite copiar el derecho de acceso dentro de la misma columna (es decir para el mismo objeto) en la cual está definido tal derecho.

Objeto / Dominio	F(1)	F(2)	F(3)
D1	Read		Write *
D2		Read *	Ejecutar
D3	Read		

Fig. 18.4.

El ejemplo anterior se transforma en :

Objeto / Dominio	F(1)	F(2)	F(3)
D1	Read		Write *
D2		Read *	Ejecutar
D3	Read	Read	

Fig. 18.5.

(*) Habilita el copiado del "acceso" en la misma columna (objeto)

Un proceso ejecutando en D(2) puede "copiar" el acceso Read en cualquier entrada asociada a F(2).

Algunas variantes de esto podrían ser:

- Transfer: se pasa de acceso(i,j) al acceso(k,j) pero se quita el acceso(i,j) (Removed).
- Copia limitada: se pasa solo el acceso R y no la capacidad R*.

OWNER

Necesitamos además del copiado algún mecanismo para agregar nuevos derechos y eliminar otros, el derecho de acceso OWNER controla estas operaciones.

Si en acceso(i,j) tenemos Owner, significa que un proceso ejecutando en D(i) puede agregar o quitar accesos en toda la columna j.

Ejemplo:

Objeto / Dominio	F(1)	F(2)	F(3)
D1	Read Owner		Write
D2		Read * Owner *	Read * Owner *
D3	Read		

Fig. 18.6.

Se transforma en :

Objeto / Dominio	F(1)	F(2)	F(3)
D1	Read Owner		
D2		Read/Write Owner *	Read Owner *
D3		Write	Write

Fig. 18.7.

CONTROL

El Copy y el Owner permiten que un proceso altere entradas en una columna. Un mecanismo para alterar entradas en las filas es asimismo necesario. Tal mecanismo es el derecho CONTROL que habilita cambios (remover derechos) en otras filas (dominios)

Objeto / Dominio	F(1)	D(1)	D(2)	D(3)
D(1)	Read		Switch	
D(2)	Read			Switch Control
D(3)	Read/Write	Switch		

Fig. 18.8.

El ejemplo anterior se transforma en :

Objeto / Dominio	F(1)	D(1)	D(2)	D(3)
D(1)	Read		Switch	
D(2)	Read			Switch Control
D(3)	Read	Switch		

Fig. 18.9.

O sea que D(2) puede cambiar el modo de acceso a F(1) de D(3) de R/W a R.

Lo importante de esto es que con estos esquemas y los conceptos de objetos y capacidad, es posible crear un nuevo tipo de monitor, llamado **manager**, que es usado para cada recurso, el cual planifica y controla el acceso a ese recurso. Cuando un proceso necesita un recurso, llama al manager, el cual le devuelve la capacidad para ese recurso. El proceso debe presentar la capacidad cuando usa el recurso. Cuando el proceso finaliza el uso del recurso devuelve la capacidad al manager, quien lo asignará a otro proceso, de acuerdo a su planificador (scheduler).

18.1.8. Revocación

En Protección Dinámica, cuando se revocan los accesos podemos tener varias opciones:

- Inmediata / Postergada.
- Selectiva / General (para algunos usuarios, o para todos).

- Parcial / Total (todos los accesos a un objeto, o solo algunos).
- Temporario / Permanente (se podrá obtener nuevamente o no).

En la lista de accesos la revocación es fácil. Se busca la lista de accesos y se hacen los cambios, luego cualquier combinación anterior es posible.

En el esquema de Listas de Capacidad, la revocación es más dificultosa, pues las capacidades están distribuidas por todo el sistema. Veamos los distintas sistemas de revocación en Listas de Capacidad:

- *Readquisición:* las capacidades son borradas periódicamente. Si un proceso intenta readquirirla, y la capacidad se revocó, NO puede hacerlo.
- *Back-pointers:* una lista de punteros asocia a cada objeto con todas las capacidades asociadas. Siguiendo los punteros es posible cambiarlos. La implementación es general pero muy costosa. Es utilizado en Multics.
- *Indirección:* las capacidades no apuntan al objeto en forma directa, sino a una única entrada en una tabla global, la cual, a su vez, apunta al objeto. La revocación se implementa buscando en la tabla global la entrada y borrándola. Puede ser reutilizada para otra capacidad. No se puede hacer revocación selectiva.
- *Llaves:* la revocación se hace cambiando la llave. Si un objeto tiene asociadas varias llaves es posible hacer revocación selectiva.

18.1.9. SISTEMAS EXISTENTES

UNIX

Cada archivo tiene asociado 3 campos (dueño, grupo, universo). Cada campo tiene 3 bits: R (read), W (write), X (execution).

Un dominio está asociado con el usuario.

Hacer switch de dominio corresponde a cambiar la identificación del usuario temporariamente. Cuando un usuario A comienza ejecutando un archivo de dueño B, el usuario es "seteado" a B, y cuando termina, es "reseteado" a A.

MULTICS

Su sistema de protección está sobre su sistema de archivos (todo es archivo), y cada archivo tiene asociada una lista de acceso, como el acceso del dueño y del universo.

Además tiene una estructura de anillos que representan los dominios, tal que dados los dominios D(i) y D(j), con i menor j, D(i) tiene mayores privilegios que D(j). Dados solo dos dominios, nos encontraríamos frente a un sistema maestro/esclavo. D(0) tiene el mayor privilegio de todos.

El espacio de direcciones es segmentado, y cada segmento tiene asociado su número de anillo y sus bits de acceso (R,W,X). Cuando un proceso ejecuta en D(i) no puede acceder a un segmento de D(j) si j menor i pero sí a una de D(k) si i menor = k, respetando, desde ya, los bits de acceso.

Para llamar procesos de anillos inferiores, se produce un trap (SVC), entonces es más controlado.

Protecciones de la forma "capacidad" los tenemos en sistemas como el Hydra y el Cambridge CAP System.

18.2. SEGURIDAD

La **seguridad de datos** estudia cómo proteger los datos de un sistema de computación contra accesos no autorizados, permitiendo, obviamente, los autorizados.

Existen diversos contextos en los cuales es necesaria la seguridad de datos:

- Sistemas de información y de archivos: controles de acceso (lo visto en Protección).
- Telecomunicaciones y redes de computadoras: controles criptográficos.
- Bases de Datos: controles sobre la información, búsquedas estadísticas y controles de inferencia.

En este apunte se tratan los dos últimos temas. La seguridad en Sistemas de Información ya fue tratada en el capítulo de Protección.

Un sistema de seguridad depende de que se pueda identificar correctamente al usuario, lo que se hace por medio de un protocolo de conexión (Login). El usuario se identifica, y el sistema le pide una palabra clave (password). Para ser almacenada, la palabra clave debe ser encriptada, porque sino sería necesario proteger el archivo que las contiene. La palabra clave que escribe el usuario se protege de su visualización eliminando el eco de lo escrito (shadow); luego se encripta y se la compara con la almacenada en el archivo. Si ambas coinciden, se permite el acceso; sino se le niega.

18.2.1. Principios de diseño para Sistemas de Seguridad

* Privilegio mínimo: sólo se deben otorgar los derechos necesarios para las funciones necesarias, y no más.

* Economía de mecanismos: los mecanismos de seguridad deben ser implementados en un solo módulo pequeño, que debe ser incluido en el diseño inicial del Sistema (sino se hiciera así, sería posible evitarlo).
* Mediación completa: debe controlar todos los accesos y debe ser imposible de obviar.
* Diseño abierto: la seguridad no debe depender de que los mecanismos sean secretos: solo deben serlo las claves.
* Aceptación psicológica: deben ser de uso sencillo, porque sino se corre el riesgo de que los usuarios no protejan sus datos.

18.2.2. Seguridad en Telecomunicaciones o Redes de Computadoras

"Si no hay comunicación, en principio hay privacidad; luego, el problema no existe". Cuáles son algunos de estos problemas ? :

- PRIVACIDAD: Es necesario mantener los datos en secreto de un punto al otro de la comunicación.
- AUTENTICIDAD: Es necesario conocer si los datos son auténticos o si fueron modificados antes de llegar.

Por ejemplo, en la Figura 18.10, M podría ser un espía industrial o financiero. También podría hacer cosas como generar o repetir mensajes de depósitos en cuentas corrientes (esto se podría evitar teniendo en cuenta el número de mensaje, la hora, la identificación de la terminal, etc.). Una solución para este tipo de problemas es la CRIPTOGRAFIA.

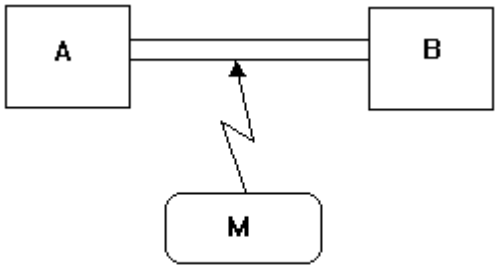

Fig. 18.10. - Escucha, genera o modifica datos.

La base de la criptografía es la siguiente:

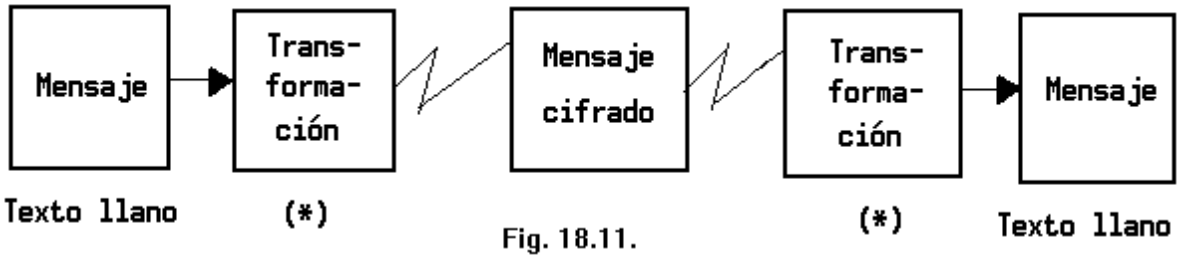

Fig. 18.11.

(*) Se aplica un método y una clave (generalmente depende de que la clave sea desconocida).

Para realizar esto existen métodos clásicos de sustitución:

18.2.2.1. ENCIFRAMIENTO DE CESAR (SUSTITUCION)

Se suma un número entero fijo a las letras del alfabeto, por ejemplo + 3. Entonces, un texto llano, por ejemplo CESAR se convierte en FHVDU. Luego, el lugar de recepción debe restar 3 a lo recibido.

18.2.2.2. SUSTITUCION CON PALABRA CLAVE

Se escriben las letras del alfabeto mezcladas con una palabra clave, por ejemplo:
SIMON BOLIVAR. Sin letras repetidas: SIMON BLVAR.

Alfabeto	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Alf. Transformado	S I M O N B L V A R C D E F G H J K P Q T U W X Y Z

Aquí, CESAR = MNPSK.

Estos dos primeros tipos de encriptamiento que pueden ser resueltos fácilmente. Son de fácil deducción, porque hay mucha información que facilita la misma. En muchos idiomas se conoce la frecuencia aproximada de todas las letras en todas las palabras del idioma, y de esta forma el mensaje puede ser desencriptado fácilmente. Si se conoce parte del mensaje (por ejemplo: Logon), la tarea es más fácil aún.

Para mayor información acerca de como deducir claves de este tipo, y para regocijo del lector, léase "El escarabajo de oro", cuento de Edgar Allan Poe, que contiene suficiente información sobre el tema.

18.2.2.3. TRANSPOSICION (DES)

Consiste en permutar las letras de los mensajes, e inclusive hacerlo a nivel de bits.

Un método muy sencillo es usar el O exclusivo. La sutileza radica en la clave: se conoce el método pero no la clave:

XOR	0	1
0	0	1
1	1	0

Se aplica según el siguiente ejemplo:

TEXTO	CLAVE	CRIPTOGRAMA	CLAVE	TEXTO
1	1	0	1	1
1	0	1	0	1
0	1	1	1	0
1	0	1	0	1
0	1	1	1	0

En 1976 se estableció el DES (Data Encryption Standard) como norma para mensajes no relacionados con la seguridad nacional de los EEUU, por la NBS (National Bureau of Standards), y fue desarrollado por IBM. Consiste en dividir los datos en bloques de 64 bits, utilizando 56 para datos, y los otros 8 como paridad. Luego, existen 2^{56} claves posibles (2^{56} es aproximadamente igual a 7.2 E+16).

Supongamos tener una computadora capaz de analizar 10^6 claves por segundo:

1E+ 6 claves 1 seg.
7.2E+16 claves 7.2E+10 seg.
1 año = 31536000 seg = 31536 * 10^3 seg. Luego, tardaríamos
7.2E+10 / 31536 * 10^3 = **2283 años**

para obtener todas las claves posibles. Digamos que, en promedio, la hallaríamos en 1000 años.

Además, este es un mecanismo muy fácil de implementar por Hardware.

18.2.2.4. ONE-TIME PAD (BLOQUE DE USO UNICO)

Es un método totalmente seguro. Trabaja como el método de Cesar, pero el entero a sumar varía para cada caracter del texto en forma aleatoria.

Es seguro, porque el criptograma "SECRETO" podría provenir de:

MENTIRA con la clave	6	0	15	2	22	23	12, o de:
REALEZA con la clave	1	0	2	6	0	6	12.

El problema es que la clave es tan larga como el mensaje, y nunca debe ser reutilizada.

18.2.2.5. DISTRIBUCION DE CLAVES

Los métodos criptográficos (DES y One-time Pad) tienen el problema de la distribución de las claves. Todos los que participan deben conocerlas.

El problema reside en que las claves deben ser cambiadas con cierta frecuencia; en consecuencia se necesita un canal de distribución de claves. No debería usarse la misma red, por problemas de seguridad; hay que buscar otra equivalente. Esto puede parecer un problema insoluble. En general se termina utilizando un mensajero de confianza, pues se llega a la conclusión de que el cambio será esporádico.

18.2.2.6. CLAVES PUBLICAS

Los métodos tradicionales se denominan simétricos, pues usan la misma clave para encriptar y desencriptar. Los denominados asimétricos tienen dos claves: una para encriptar y otra distinta para desencriptar.

Supongamos dos interlocutores A y B. A usa una clave E para encriptar, y B usa una clave D para desencriptar, de tal manera que aunque se conozcan E y el método, no se pueda deducir D en un tiempo razonable (por ejemplo, que para deducir D se tarden miles de años).

Esto permite que A y B publiquen sus claves de encriptamiento Ea y Eb, y mantengan en secreto sus claves de desencriptamiento Da y Db. Con esto, el problema de distribución de claves desaparece.

Obviamente, el problema radica en generar claves E y D.

Uno de los métodos más importantes es el RSA (por sus autores Rivest, Shamir, Adleman).

La seguridad de este método depende de ciertas propiedades de los números primos, y de la dificultad de encontrar divisores de números muy grandes (cientos de dígitos).

RSA:

Se buscan dos primos grandes p y q, y se obtiene n = p.q .

Se buscan dos números e y d que contemplen la siguiente propiedad:

$e.d \bmod [(p-1).(q-1)] = 1$

Se representa el texto llano por medio de un M tal que

$0 \leq M \leq n-1$

de forma tal que el encriptamiento resulta:

$C = M^e \bmod(n)$

y el desencriptamiento es:

$M = C^d \bmod(n)$

Veamos un ejemplo con números pequeños:

p=3; q=5; n=15

Se elige e = 3 y d = 11 tal que 3.11 mod [2.4]=33 mod(8) = 1

Si suponemos un mensaje M = 8, resulta:

$c = 8^3 \bmod (15) = 512 \bmod(15) = 2$

siendo c = 2, el mensaje cifrado que viaja por el medio de comunicación. En el lugar de destino, se hace:

$M = 2^{11} \bmod(15) = 2048 \bmod(15) = 8$

donde M=8 es el mensaje original desencriptado.

La clave pública es el par (e,n) y la privada el par (d,n). Conocer e y n no da suficiente información como para hallar d (p y q son secretos).

Un espía lo que puede hacer es factorizar n, que es público, para obtener p y q, y luego podrá calcular d, ya que e es también público.

Pero el problema es que no se conocen algoritmos eficientes de factorización. Si p y q tienen 100 dígitos c/u, el mejor algoritmo de factoreo tardaría miles de millones de años (existen del orden de 10^{97} números primos de 100 dígitos).

El mejor algoritmo de factoreo, en la franja de 10^{200}, es el de Schroeppel (no publicado), del cual se han calculado algunos tiempos:

dígitos n = p * q	Tiempo (1 ms por operación aritmética)	
50	3.9	horas
75	104	días
100	74	años
200	3.8 E+9	años
300	4.9 E+15	años
500	4.2 E+25	años

Justificando el RSA

Elegidos p y q primos, y n = p.q, se eligen e y d tal que

$e \cdot d \bmod [(p-1)(q-1)] = 1$

Dado un mensaje M / $0 \leq M \leq n$, debe ocurrir que:

$P(M) = M^e \bmod n = C$ y

$S(C) = C^d \bmod n = M$

o sea que esto funciona si S o P = Identidad, o sea,

$S(P(M)) = M$ con $0 \leq M \leq n$.

Si escribimos

$M = S(P(M)) = (M^e \bmod n)^d \bmod n = M^{ed} \bmod n$, que es lo que habría que probar.

Para esto usamos el Teorema de Fermat, que dice:

Si p es primo, y M un entero no múltiplo de p, entonces para todo natural r,

$M^r \bmod p = M^{r \bmod(p-1)} \bmod p$

En nuestro caso es:

$e * d \bmod [(p-1)(q-1)] = 1$ (por construcción)

que se puede escribir:

$e * d = 1 + K\,(p-1).(q-1)$ para algún K

(esto sale de $e * d / [(p-1)(q-1)] = K$ con resto 1).

Si le aplicamos mod (p-1) en ambos miembros, resulta que:

$e * d \bmod (p-1) = 1 + 0$ [1]

pues K (p-1) (q-1) es múltiplo de (p-1).

Si M no es múltiplo de q, y por el teorema, tenemos:

$M^{ed} \bmod p = M^{ed \bmod(p-1)} \bmod p = M^1 \bmod p$, por [1].

Si M es múltiplo de p, M^{ed} también lo es, por lo tanto:

$M^{ed} \bmod p = M \bmod p$ (pues obviamente es 0)

Entonces, $M^{ed} = M + s.p$ para algunos enteros s.

Análogamente

$M^{ed} = M + t.q$ para algunos enteros t; luego

$M^{ed} - M = s.p = t.q.$

Como p y q son primos distintos, y como p divide a s.p entonces p también divide a t.q; luego, divide a t entonces t = w.p. O sea:

$M^{ed} - M = w.p.q$ para algún w.

Luego $M^{ed} = M + w.n$ (p.q = n).

Entonces, $M^{ed} \bmod n = M \bmod n = M$ (pues M menor = n), o sea, S(P(M)) = M

En forma inmediata obtenemos:

$P(S(M)) = (M^d \bmod n)^e \bmod n = M^{ed} \bmod n = S(P(M))$.

O sea, que RSA es un sistema criptográfico conmutativo.

Si bien el RSA es un método laborioso, resuelve el problema de la distribución de claves, y permite la utilización de otros métodos. Por ejemplo:

A quiere conversar con B y genera una clave S de DES. A calcula S' = RSA(S), usando la clave pública de B, y envía S'. B desencripta S' con su clave privada y la transforma en S y responde a A con la clave S en encriptado DES.

18.2.2.7. AUTENTICIDAD (FIRMA)

Si pensamos el RSA al revés, por ejemplo, si encriptamos con una clave D secreta, se puede desencriptar con una clave E pública, y nadie podría falsificar el mensaje, ni siquiera el receptor. Esto podría usarse como "firma" de documentación electrónica.

18.2.2.8. REDES LOCALES

Estas redes tienen problemas de seguridad, pues son pensadas como "broadcast" (todos escuchan). Inclusive existe la posibilidad de daño físico, pues si alguien conecta el cable de transmisión al toma de corriente eléctrica, es posible dañar la interfase de nodos y hacerla inoperante.

18.2.3. Seguridad de datos en Bases de Datos

Si suponemos un acceso público a ciertas Bases de Datos, habría que mantener ciertos controles sobre su actualización. Por ejemplo:

Dependiente de los datos: solo es posible borrar registros que no hayan sido actualizados en por lo menos más de tres meses.

Dependiente del tiempo: el campo de sueldo solo es posible de ser cambiado en horarios determinados.

Dependiente del contexto: solo se pueden listar nombres o sueldos, pero no ambos a la vez.

Generalmente las bases de datos comerciales chequean con una Lista de Control de Acceso asociada a una relación o conjunto de campos o registros.

Además es necesario un control de consistencia de datos constante y antes de permitir un acceso o modificación.

En los casos que se permite el acceso a Bases de Datos con fines estadísticos no debería permitirse el acceso a datos individuales. O sea, no permitir preguntas del tipo: ¿Cuánto gana el Sr. X?. Pero si se conocen algunas características del sujeto, esto solo no basta para proteger la información individual. Supongamos que se sabe que el Sr. X es una Sra., y que es la única integrante femenina de un área determinada de una firma, es válido preguntar cual es el promedio de sueldo del personal femenino de esa área.

Por lo tanto habría que bloquear el acceso a la información cuando los datos recuperados son pocos; pero aún así se podría deducir algo si se pregunta: ¿Cuál es el sueldo promedio?, y luego, ¿Cuál es el sueldo promedio de los hombres?. Estos casos son muy difíciles de detectar.

18.2.4. Seguridad de Datos en general

Los casos más dañinos son los que se producen por descuido o malicia del personal de la misma instalación. Por ejemplo:

- Bomba de tiempo: cuando el programador es despedido, deja códigos que se destruyen o destruyen información a partir de una fecha.
- Redondeo de centavos: embolsar en una cuenta particular los centavos sobrantes por redondeo.
- Caballo de Troya: hacer copias falsas de pantallas de Login para capturar palabras claves.
- Basureros: buscar en los cestos de basura listados de directorios con palabras claves.
- Descuido: aprovechar que la terminal quedó conectada mientras el operador se fue, o utilizar palabras claves sencillas de adivinar.