Se plantea la necesidad de disponer de un mecanismo de almacenamiento de información que proporcione cifrado. La idea es disponer de una clave de cifrado almacenada en una tarjeta inteligente que se utilice para acceder al equipo (dicha tarjeta dispondrá de una firma electrónica y las claves de cifrado necesarias). Las claves de dicha tarjeta se utilizarán en todas las operaciones de lectura/escritura de información. Se quiere dotar de esta capacidad no sólo a los datos de un fichero, sino a los contenidos de directorios (nombres de ficheros) y demás estructuras de un sistema de ficheros (bitmaps, i-nodos, superbloque). El objetivo de esta medida es hacer que los empleados con dicha tarjeta sean los únicos en poder usar el equipo y que en el caso de que los discos sean robados nadie pueda extraer los datos.

Una característica del mecanismo de cifrado es que dos fragmentos de datos (de cualquier tamaño) que se diferencien en un byte generan un cifrado que difiere en, al menos, un 85%. Esto impide que el cifrado de un fragmento de datos sea equivalente al cifrado de dos fragmentos de la mitad de tamaño. Siendo de esta forma necesario definir una unidad mínima de cifrado.

Las alternativas que se barajan al respecto son las siguientes:

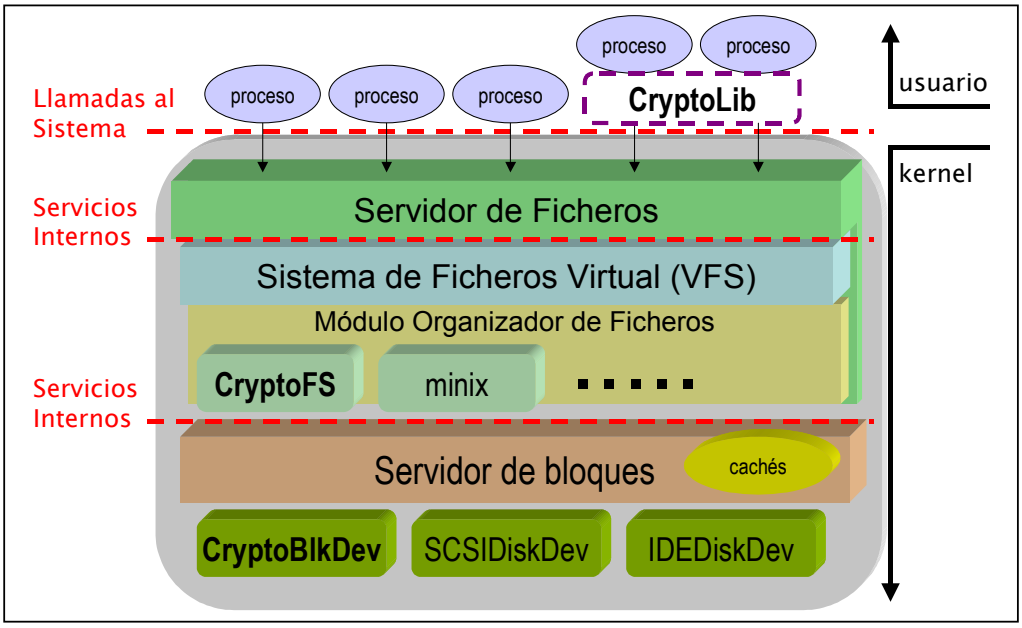
(i).- **CryptoLib**: Implementarlo como una **biblioteca** que sobreescriba todas las funcionalidades de entrada/salida (*read*, *write*, *open*, *mkdir*, ...) por funciones que cifren la información y que posteriormente invocen a las verdaderas llamadas de lectura y escritura cifrada.

(ii).- **CryptoFS**: Implementarlo como un **sistema de ficheros**, partiendo del código de un sistema de ficheros ya existente, modificarlo para que cada operación de escritura cifre los datos a escribir y cada operación de lectura lo descifre, y de forma análoga lo haga sobre las operaciones que afectan a directorios.

(iii).- **CryptoBlkDev**: Implementarlo como un **dispositivo de bloques** (similar a un disco) cuyas llamadas a bajo nivel estén implementadas para que cifren y descifren los datos. Evidentemente estas llamadas deben invocar despues a llamadas de bajo nivel de un dispositivo de bloques real (disco, CD-Rom, ...)

a). Dibuje la estructura de todo el sistema de almacenamiento de un sistema operativo que implemente VFS (*Virtual File System*), desde los procesos hasta los dispositivos. Indique sobre esta figura:

(1) la parte correspondiente a espacio de usuario y espacio de *kernel*,
(2) cuál es el interfase (llamadas al sistema o servicios internos) entre cada uno de los elementos y
(3) dónde se colocarían las tres alternativas antes citadas.



La arquitectura dibujada arriba representa la de un kernel monolítico (sistema de ficheros dentro del kernel). De las tres alternativas sólo **CryptoLib** es a nivel de usuario, el resto son implementaciones dentro del kernel (**CryptoFS** es un sistema de ficheros dentro de VFS y **CryptoBlkDev** es un nuevo manejador de dispositivos). La interfaz de llamadas entre el espacio de usuario y el kernel es por medio de llamadas al sistema. Internamente los interfaces son servicios internos o APIs específicas de cada uno de los módulos.

b). ¿Qué datos/metadatos del sistema de ficheros estarían cifrados en cada método?

Según el enunciado, se propone cifrar la mayor información posible. De todas formas hay que considertar que esto está restringido a la información visible por cada una de las alternativas:

CryptoLib: Sólo entiende de los elementos tratados a nivel de librería de entrada/salida o llamada al sistema. De esta forma pueden cifrarse los datos de los ficheros regulares y de forma individual las entradas de los directorios (no todo el contenido del directorio, únicamente los nombres de los ficheros. Metadatos tales como los bitmaps de i-nodos/bloques, los propios i-nodos o el superbloque no se podrán ciufrar puesto que a este nivel no son visibles por el mecanismo de cifrado.

CryptoFS: Este es un nivel más profundo dentro de las capas del sistema operativo. A este nivel toda la información y metainformación es visible (en realidad el SF es el verdadero encargado de manejar dicha metainformación). Aquí, el contenido tanto de ficheros y directorios estaría completamente codificado, asimismo los bitmaps e i-nodos. La única excepción (y eso dependería mucho de cómo sea diseñada la capa VFS) sería el superbloque.

CryptoBlkDev: Aquí absolutamente todos los datos y metadatos estarían cifrados. Es más Un manejador de dispositivo de bloques como éste no sabe nada de la diferencia entre datos y metadatos, todos son bloques de disco y le da igual lo que haya en ellos.

c). ¿Estaría el contenido de la cache de bloque cifrado? Indíquelo para cada caso.

Aquí el análisis es similar. El contenido estaría cifrado si dicho elemento es accesible de forma directa por cada una de las diferentes alternativas. La caché de bloques es un espacio de almacenamiento que es rellena a partir de los bloques de disco por el manejador de dispositivo, de esta forma:

CryptoLib: Una librería de usuario no puede acceder nunca a la caché de bloques de forma directa. Por lo tanto como se cifra y se descifra a nivel de usuario la caché tendrá lo que la librería mande hacia el dispositivo y lo que recibe del mismo, es decir datos cifrados. Una matización importante es que determinados bloques (como los contenidos de directorios estarán cifrados de una forma diferente, es decir no se cifraría el bloque entero, sino que se cifra únicamente los nombres de las entradas de los mismos. Por otro lado los bloques de i-nodos no estarían cifrados en absoluto.

CryptoFS: Aunque este elemento está más cerca de la caché de bloques, ésta como ya hemos dicho la lee el manejador con los datos tal y como están en el dispositivo. Se puede plantear que **CryptoFS** cifra/descifra datos sobre la propia caché de bloques pero eso no sería muy limpio a nivel de diseño, lo apropiado sería que el sistema de ficheros deje ahí los datos como quiera que sean escritos en el dispositivo y que al leerlos los lea de la misma forma. En resumen, en este caso también el contenido estaría cifrado, aunque ahora todos los bloques estarían cifrados completamente.

CryptoBlkDev: Este nivel es el único que se encarga de recuperar los datos del disco y meterlos en la caché, por lo tanto es el único que (de una forma razonable) podría hacer el paso de descifrar los datos al leerlos del dispositivo. Esta opción es la que parece más razonable, así los elementos que lean datos de la caché los leerán ya en claro, y en el momento de escribir la información de nuevo a disco es cuando se cifrarían de nuevo. La alternativa más purista sería rellenar los bloques de la caché tal y como están en el dispositivo (sin alterarlos) y sólo en el momento de que alguien por encima los solicite se descifren, sin embargo la otra opción es la que proporciona un tratamiento más limpio y cómodo.

d). Para una operación de lectura y para cada una de las tres alternativas indique el camino seguido por la operación resaltando cláramente cuándo los datos están en claro y cuándo están cifrados. Considere dos casos, que el bloque de datos afectado por la operación haya sido leido recientemente y que el bloque no se haya leído aún.

Con lo que hemos contestado en las preguntas anteriores podemos responder este apartado de forma sencilla:

CryptoLib: se descifran sólo al final

(1) El cliente solicita leer "*n*" bytes.
(2) La petición es cursada por la librería al sistema de ficheros.
(3) El sistema de ficheros puede solicitar que el bloque sea traído a la caché por el manejador de dispositivos (si no se ha leído recientemente) o en el caso de haberlo accedido hace poco tomarlos e la caché directamente. En cualquiera de los casos los datos están cifrados.
(4) Los datos son servidos por el sistema de ficheros a la librería, la cual los descifra y se los pasa al usuario.

CryptoSF: se descifran dentro del sistema de ficheros

(1) El cliente solicita leer "*n*" bytes.
(2) La petición es cursada por directamente al sistema de ficheros.
(3) El sistema de ficheros puede solicitar que el bloque sea traído a la caché por el manejador de dispositivos (si no se ha leído recientemente) o en el caso de haberlo accedido hace poco tomarlos e la caché directamente. En cualquiera de los casos los datos están cifrados.
(4) Antes de enviarlos a espacio de usuario al sistema de ficheros descifra los datos.
(5) Los datos son servidos por el sistema de ficheros al usuario ya en claro.

CryptoBlkDev: se descifran la primera vez que se meten en la caché

(1) El cliente solicita leer "*n*" bytes.
(2) La petición es cursada por hacia el sistema de ficheros directamente.
(3) El sistema de ficheros puede solicitar que el bloque sea traído a la caché por el manejador de dispositivos (si no se ha leído recientemente) o en el caso de haberlo accedido hace poco tomarlos e la caché directamente.
(4) Sólo en el caso de tener que leer el bloque del dispositivo se hará la operación de descifrado. Los datos de la caché se guardan en claro.
(5) Los datos son tomados por el sistema de ficheros que se los pasa al usuario (en claro).

e). Ídem para el caso de escritura, pero considere ahora otros dos casos, que la operación de escritura afecte a parte de los datos de un bloque o que se escriba completamente los datos del bloque.

Un elemento muy importante en este apartado es una característica que se ha comentado del método de cifrado. Por lo que se comenta el proceso de cifrado no se puede hacer en plan divide-y-vencerás, si cifras unos datos, cifrando mitad y mitad por separado no se consigue lo mismo que todo junto. Esto es evidentemente un problema a la hora de descifrar puesto que habría que saber cuál es el tamaño que se ha usado para cifrar los datos. De esto se extrae que es necesario definir una unidad de cifrado, es decir, se cifrarán y se descifrarán los datos siempre de "*x*" en "*x*" bytes. Lo más cómodo es hacer coincidir este tamaño x con el tamaño de un bloque (hacemos esa suposición para resolver este apartado):

CryptoLib: se cifran justo al principio

(1) El cliente desea escribir "*n*" bytes.
(2) La librería lee previamente el bloque de datos afectados por la escritura, inserta los datos a escribir y lo cifra entero. Si la operación afecta a un bloque entero entonces no hace falta leerlo previamente.
(3) El sistema de ficheros recibe el bloque entero de datos a escribir y lo pasa al servidor de bloques, todo cifrado.
(4) Los datos cifrados se meten en la caché de bloques (marcado como no escrito aun en disco) y cuando corresponda se sincronizará la copia del bloque que está en caché con el disco.

CryptoFS: se cifran dentro del sistema de dicheros

(1) El cliente desea escribir "*n*" bytes.
(2) La librería pasa esos bytes al sistema de ficheros en claro.
(3) El sistema de ficheros recibe los datos a escribir y los escribe en el bloque. Si se trata de un bloque entero no hay que leerlo y descifrarlo previamente, en el resto de casos si. Después cifra el bloque entero y se lo pasa al servidor de bloques.
(4) Los datos cifrados se meten en la caché de bloques (marcado como no escrito aun en disco) y cuando corresponda se sincronizará la copia del bloque que está en caché con el disco.

CryptoBlkDev: se cifran justo antes de escribirlos en el disco

(1) El cliente desea escribir "*n*" bytes.
(2) La librería pasa esos bytes al sistema de ficheros en claro.
(3) El sistema de ficheros recibe los datos a escribir y los escribe en el bloque. Si se trata de un tamaño de menos de un bloque hay que leerlo previamente (de ello se encarga el el servidor de bloques y el manejador de dispositivo). En ningún caso se cifran los datos.
(4) En cualquiera de los dos casos (bloque entero o sólo parte) los datos que correspondan se escribirán en la caché. Sólo en el momento de escribir un bloque modificado de la caché al disco se haría el cifrado. Si la escritura es parcial, como se ha dicho esto implica una lectura previa que como se vio en el apartado anterior sólo en el caso de ser un bloque que no se haya leído recientemente será leída de disco, en otros casos ya estará en caché y ya se habrá descifrado.

f). Suponiendo que el cifrado/descifrado de un fragmento de datos (equivalente en tamaño a un bloque de disco) emplea un tiempo igual a *0,30xT*, siendo *T* el tiempo de lectura/escritura sobre disco. Calcule (en función de *T*) el tiempo empleado en las siguientes operaciones:

1. Lectura de 29 bytes de datos leídos recientemente (usando **CryptoLib**).
   No se lee de disco, sólo de la caché. Pero hay que descifrar el bloque entero: *0,3xT*
2. Lectura de 29 bytes de datos leídos recientemente (usando **CryptoFS**).
   No se lee de disco, sólo de la caché. Pero hay que descifrar el bloque entero: *0,3xT*
3. Escritura de 100 bytes de un bloque previamente leido (usando **CryptoLib**).
   No se lee de disco, sólo de la caché.
   Pero hay que descifrar el bloque entero, modificar los datos y volver a cifrarlo.
   Total: *(0,3+0,3)xT*
   La escritura puede ser diferida.
4. Escritura de 100 bytes de un bloque previamente leido (usando **CryptoBlkDev**).
   No se lee de disco, sólo de la caché.
   Ni se cifra ni se descifra, simplemente se modifican los datos en la caché.
   Total: aprox. *0*
   La escritura puede ser diferida, entonces es cuando se cifrarán.
5. Borrar un fichero del directorio actual (usando **CryptoFS**).
   Borrar un fichero implica: modificar el contenido del directorio (suponemos un bloque), modificar un i-nodo (marcarlo como borrado) y el *bitmap*.
   La operación implicaría escribir (parcialmente) tres bloques de disco. Leer, descifrar, modificar y cifrar.
   Total (suponiendo que los 3 residen en caché/memoria): *3x(0,3+0,3)xT*
   Dependiendo del diseño del sistema de ficheros alguno de estos bloques al escribirse se actualizan automáticamente en disco [tal es el caso de los i-nodos]). No hemos considerado esto y suponemos que las escrituras son diferidas.
6. Borrar un fichero del directorio actual (usando **CryptoBlkDev**).
   Si al igual que antes suponemos que los bloques afectados están leídos en caché, todas las modificaciones se guardarán en claro en dicha caché y ya se cifrarán en la escritura real sobre el disco.
   Total: aprox. *0*.
7. Crear un enlace simbólico (*symbolic link*) sobre un fichero dentro del directorio actual (usando **CryptoFS**).
   Crear un enlace simbólico implica modificar el directorio (crear una nueva entrada) y crear un nuevo fichero (escribir un nuevo i-nodo y un bloque de datos).

Los i-nodos y la entrada de directorio siempre son modificaciones parciales de un bloque de disco, el bloque de datos de contenido del nuevo fichero se escribe entero.
Total: *2x(0,3+0,3)T+0,3xT* (dos cifrado/descifrado y uno con cifrado únicamente).

8. Crear un enlace físico (*hard link*) sobre un fichero dentro del directorio actual (usando **CryptoFS**).

Crear un *hard link* implica crear una nueva entrada de directorio y alterar el contador de enlaces del fichero en cuestión.
Total: *2x(0,3+0,3)T+0,3xT* (dos cifrado/descifrado)

g). Este mecanismo de cifrado se quiere aplicar tanto a discos SCSI, discos IDE como unidades extraibles (e.g.: floppies y CDs). Asimismo también se quiere aplicar el cifrado a dos diferentes formatos de sistema de ficheros (uno usado en los volúmenes pequeños y otro que se usa para grándes particiones) ¿Cómo afectaría esto a cada una de las alternativas? ¿En qué casos habría que codificar el mecanismo de cifrado varias veces?

Para el caso de **CryptoLib** cualquiera de las modificaciones no afectan en absoluto.
A **CryptoFS** le afecta principalmente el ajuste a dos formatos de sistema de fichero, lo cual implicaría tener que recodificar todos los mecanismos de cifrado para los dos tipos de sistema de fichero.
El cambio de dispositivo de almacenamiento es lo que más afecta a **CryptoBlkDev**, pero en este caso se puede usar un mecanismo similar al de los dispositivos virtuales (como RAID). Esto consiste en tener un dispositivo de bloque como una capa por encima de otro. **CryptoBlkDev** podría cifrar y descifrar bloques y usar otro manejador de dispositivo para leer y escribir bloques directamente sobre el dispositivo. Esto además de simplificar el desarrollo del mismo permite ajustarlo a cualquier dispositivo de bloques que se quiera sin mucho problema.

h). Suponga que esta parte del proyecto se la asignan al grupo del cual usted es jefe de proyecto. Haga una evaluación de cuál sería la alternativa más conveniente considerando tanto las ventajas e incovenientes que plantean como el esfuerzo para desarrollarlas. ¿Cuál sería su recomendación a su superior inmediato?

La opción de librería es sencilla de abordar (no hay que tocar el núcleo), pero implicaría interceptar todas las llamadas que afectan al sistema de ficheros. Además es la menos segura (vimos que no se codificadan todos los elementos de sistema de ficheros y además plantea serias pegas a nivel de seguridad e implantación puesto que toda se basa en que todas las aplicaciones del sistema usan la librería en cuestión. Un usuario malicioso puede enlazar con otra librería que intente decodificar los datos usando claves que no están en la tarjeta inteligente. Además las aplicaciones existentes hay que re-enlazarlas y adaptarlas a dicha librería....a lo mejor no es tan sencillo.

La opción a nivel de sistema de ficheros hace que los cambios sean trasparentes a nivel de usuario pero con diferencia es la que es más compleja de codificar. Además hemos visto que implica reimplementarla para cada tipo de sistema de ficheros del que se quiera tener una versión cifrada.

La alternativa de manejador de dispositivos es más elegante (trasparente para usuarios y sistemas de ficheros), sobre todo si se aborda como un manejador de dispositivo virtual sobre otros dispositivos reales. El desarrollo es abordable (aunque sea dentro del núcleo). Un factor muy importante es que como se gana el acceso a la caché de bloques se ha visto que muchas de las operaciones son más eficientes en esta alternativa que en las anteriores.