JITEL 2017
XIII Jornadas de Ingeniería Telemática
27-29 Septiembre 2017, Valencia (España)

Actas de las XIII Jornadas de Ingeniería Telemática (JITEL 2017), Valencia (España), 27-29 de Septiembre de 2017

ISBN: 978-84-9048-595-8
DOI: http://dx.doi.org/10.4995/JITEL2017.2017.6576

Sistema de cifrado basado en contexto aplicado a prevención de fuga de datos

Alberto García[1], Pilar Holgado[1], Jose Javier Garcia[2], Jorge Roncero[2], Víctor A. Villagrá[1], Helena Jalain[1].

[1]Departamento de Ingeniería y Sistemas Telemáticos, Universidad Politécnica de Madrid, Avenida Complutense, 30, 28040, Madrid, España.

[2]Nokia, Departamento de Innovación, Calle de María Tubau, 9, 28050, Madrid, España

garciamoro@dit.upm.es, pilarholgado@dit.upm.es, jose_javier.garcia_aranda@nokia.com, jorge.roncero_mayoral@nokia.com, villagra@dit.upm.es, hjalain@dit.upm.es.

***Resumen*- Las herramientas DLP (Data Leak Prevention) están adquiriendo un valor elevado en los últimos años debido a la importancia de proteger los datos sensibles de una organización. Muchas de las herramientas DLP se basan principalmente en la analítica de datos, ya sea un análisis de archivos almacenados o estando en tránsito por la red. La solución DLP propuesta usa el cifrado basado en contexto para evitar fugas de información. La clave de cifrado y descifrado se obtiene a partir de la ejecución de un conjunto de retos basados en el contexto de entorno y en las políticas de la empresa. En este artículo se explica la arquitectura y el diseño de la solución DLP y de los retos propuestos.**

***Palabras clave*- Data Leakage Prevention, cifrado basado en contexto.**

I. Introducción

Hoy en día, muchas empresas se ocupan de datos sensibles, incluyendo la propiedad intelectual, información financiera o información personal de los usuarios. La distribución accidental o no intencionada de datos privados a una entidad no autorizada es un problema grave para las empresas. El daño potencial de la fuga de datos puede influir en la reputación de la empresa, en la exposición de la propiedad intelectual a los competidores, o en la pérdida de ventas futuras.

En el contexto de la fuga de datos, el atacante puede ser un empleado interno o un atacante externo que intenta obtener información sensible. Incluso, no siempre es causada con mala intención, sino también por un error inadvertido. Además, un usuario autorizado no es el mismo que un usuario de confianza. En muchos casos, las organizaciones son víctimas de sus propios empleados que comparten intencionadamente datos confidenciales con personas externas con fines personales [1]. En este caso, el usuario está autorizado a acceder a información sensible y no es detectado a partir de medidas externas clásicas como cortafuegos.

La Prevención de Fugas de Datos o Data Leak Prevention (DLP) [2] se ha propuesto como una solución a estos problemas. Distintas soluciones DLP han sido estudiadas tanto en áreas de investigación académica como en aplicaciones prácticas. Sin embargo, la fuga de datos y el uso indebido de información se sigue considerando una amenaza emergente para las organizaciones, especialmente cuando son llevadas a cabo por sus propios empleados. En muchos casos, es muy difícil detectar a los usuarios internos porque hacen mal uso de sus credenciales para realizar un ataque.

En este artículo, proponemos una solución DLP que aplica el concepto de cifrado basado en el contexto. Esta propuesta se basa en un proceso de cifrado/descifrado de documentos confidenciales, donde la clave de cifrado se obtiene a través de la ejecución de un conjunto de retos. Estos retos utilizan el contexto del entorno y las políticas de la empresa en el momento de cifrado/descifrado. De esta manera, los archivos sensibles están cifrados en todo momento y sólo se pueden leer dentro de nuestro sistema DLP.

El resto del trabajo se organiza como sigue. Los antecedentes sobre soluciones DLP se describen en la Sección 2. La Sección 3 describe el estado actual de distintas técnicas de cifrado basado en contexto. La Sección 4 explica el sistema DLP usando cifrado basado en contexto. Los retos propuestos se explican en la

EDITORIAL UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Sección 5. La Sección 6 describe cómo se genera la clave de cifrado en base a los resultados de los retos. Finalmente, las conclusiones finales obtenidas durante este estudio se incluyen en la Sección 7.

II. Herramientas DLP

Una herramienta DLP [3] es una utilidad que ayuda a mantener segura la información confidencial, evitando posibles filtraciones o difusiones no autorizadas de información.

A. *Qué se protege*

Dentro de esta categoría podemos encontrar: Datos permanentes almacenados en el disco duro protegidos mediante cifrado o control de acceso, datos en uso utilizando medidas de limitación de acciones como copiar-pegar y realizar capturas de pantalla, y datos en tránsito por la red, ya sea entre equipos de la red interna o con un host externo utilizando controles sobre la red, como son la detección y la inspección de los datos transmitidos.

B. *Dónde se protege*

El despliegue se puede llevar a cabo en dispositivos finales para monitorizar y controlar el acceso a los datos desde los dispositivos finales. En este caso es necesario un servidor de supervisión remoto que se haga cargo de las tareas administrativas, la distribución de políticas y la generación de eventos de registro. O despliegue en red mediante análisis del tráfico de la red y sujeto a una política predefinida, incluso activar eventos y bloquear transmisiones sospechosas.

C. *Cómo se protege*

La protección se puede llevar a cabo de distintas maneras. Una de ellas es la inspección basada en contexto que consiste en inspeccionar el contexto de un fichero, como el tamaño o el tipo de archivo. Otra opción es la inspección basada en contenido que detecta la fuga de información mediante el análisis de contenido, utilizando distintas técnicas como análisis de lenguaje natural o estadísticas.También existen métodos basados en el establecimiento de una política para el cifrado de los ficheros confidenciales y el acceso a dicha información. Además también son utilizados métodos como el etiquetado o el control de acceso.

Nuestra propuesta de solución DLP añade una política de seguridad adicional, haciendo que las claves de cifrado de los ficheros sean obtenidas de una función dentro de un contexto específico, el cual puede ser personalizado y parametrizado por un administrador.

III. Cifrado basado en contexto

El cifrado basado en contexto es una forma de autorizar a los usuarios para acceder a información si cumplen una serie de requisitos de entorno. Este tipo de cifrado puede ser usado en diferentes entornos, como por ejemplo en entornos de Internet of Things (IoT) [4], utilizando diferentes metodologías o con servidores exteriores que comprueben el contexto.

El cifrado basado en contexto para entornos de IoT se puede definir de dos formas [4]: pueden establecerse los datos de contexto en el usuario o pueden establecerse en la propia información a proteger. A continuación se detallan ambos modelos.

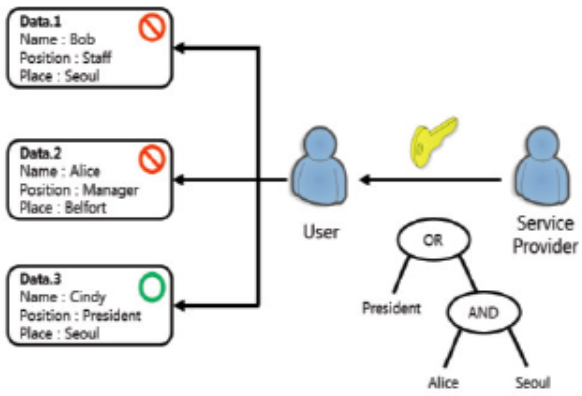

Fig. 1. Ejemplo de un árbol de autorización

En la Figura1 se puede ver un árbol de autorización basado en funciones lógicas. En este caso, lo que hace el proveedor de servicios es generar un árbol de operaciones lógicas que tiene que resultar en TRUE para poder acceder a los datos y que son suministrados al usuario junto con la clave. De esta forma, los datos tienen una serie de atributos y si cumplen el árbol de autorización del usuario, este dato estará accesible para ese usuario específico.

Por otro lado, se puede realizar lo opuesto: los usuarios tienen una serie de parámetros asociados y a los datos se les asigna un árbol de autorización, como se puede ver en la figura 2. Estos datos están cifrados, aunque la clave de cifrado se guarda en el mismo lugar que el dato en sí. En este caso, el usuario que tenga los parámetros necesarios que validan el árbol están autorizados a obtener el fichero, pero dichos parámetros no forman parte de la clave. Este sistema puede ser menos seguro debido a que la contraseña de descifrado está en el propio fichero y que su valor no depende de los parámetros.

Por otro lado, el cifrado basado en contexto también puede ser implantado mediante un servidor externo que haga las comprobaciones de contexto [5]. De esta manera, las comprobaciones del contexto se resuelven de forma externa, sin necesidad de que el árbol esté con el usuario o con el dato. Este servidor devuelve la respuesta en forma booleana para indicar si se puede acceder o no a los datos.

El cifrado basado en contexto se puede utilizar para distintas aplicaciones [6]. Un caso de ejemplo es cuando un proveedor desea compartir o habilitar el acceso a datos basándose en las credenciales del usuario receptor. El proveedor de datos proporciona una función f (*) donde describe cómo quiere compartir o habilitar el acceso a los datos y asigna al usuario una clave secreta

con credenciales X. Si f (X) = 1 (u otro resultado fijado), el usuario puede descifrar los datos.

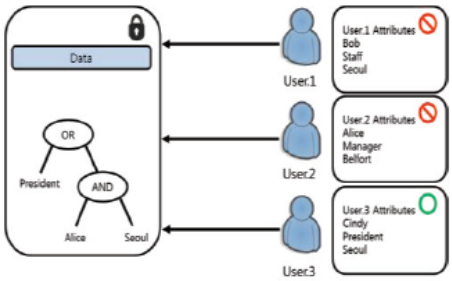

Fig. 2. Ejemplo en árbol de autorización

Las credenciales de este usuario están representadas por un conjunto de datos de tipo *string* y se conoce como cifrado basado en atributos o Attribute-Based Encryption (ABE). La función es representada por una fórmula de tipo matemático sobre estos atributos, que en este caso se corresponden con cadenas de texto pero pueden ser números u otros parámetros. Esta fórmula variará su estructura y tipo dependiendo de cómo se quieran manejar los parámetros y de su tipo.

Podemos definir dos formas de cifrado basado en atributos. En la primera de ellas, llamada Key-Policy ABE, los atributos se anotan en los datos cifrados y la fórmula f (*) se da al usuario, como en la figura 1. Por otra parte, existe la otra modalidad, llamada Ciphertext-policy ABE, donde los atributos se utilizan para las credenciales del usuario y la fórmula f (*) se encuentra junto con los datos, como en la Figura 2.

En otra metodología [7] se propone un control de acceso que se expresa con una matriz Linear Secret Sharing Scheme (LSSS) sobre los atributos en lugar de las clásicas estructuras de árbol equivalentes. Sin embargo, estas matrices LSSS son mucho menos intuitivas de usar que las fórmulas booleanas o árboles de acceso [8].

La diferencia principal entre las metodologías radica en la situación de los datos o atributos de autorización, que pueden encontrarse en la información en sí o pertenecer al propio usuario. Además, también hemos visto comprobaciones de contexto con servidor exterior, que da más seguridad y flexibilidad a la hora de configurar y comprobar contextos de usuarios.

En nuestra propuesta aplicamos este concepto de contexto de los usuarios, como su nombre, su identificador de usuario o incluso su localización para hacer que estos parámetros formen parte de la clave que dará acceso a un fichero cifrado. Así, si no se cumple ese contexto específico, la clave generada será incorrecta.

IV. Cifrado basado en contexto aplicado a DLP

Nuestra propuesta se basa en construir una herramienta de DLP utilizando el contexto de los usuarios para la autenticación y el cifrado/descifrado de los datos sensibles de una organización. Se propone utilizar, no sólo los atributos tradicionales de usuarios como son el nombre de usuario y su rol en la empresa, sino que vamos un paso más allá, incluyendo el contexto de entorno de los equipos, como la hora, la geolocalización, etc.

Aplicar el cifrado basado en contexto en una herramienta DLP proporciona mayor seguridad en el acceso a los ficheros por parte de los usuarios autorizados, ya que estos usuarios acceden a los datos confidenciales o sensibles en condiciones controladas y seguras. En nuestra propuesta, los usuarios solo podrán escribir y leer datos en el contexto que ha sido configurado por el administrador, y por tanto, únicamente cumpliendo este contexto se puede acceder a la información sensible. De esta forma se puede asegurar que el contexto de entorno del usuario es seguro tanto en el momento de escritura como en el de lectura. Cuando el contexto no sea correcto, ya sea por parte de un usuario no autorizado o un empleado autorizado que no se encuentra en el contexto apropiado, los datos no podrán ser descifrados correctamente con lo que se evita la fuga de información.

Para el correcto funcionamiento de esta propuesta, suponemos que el contexto siempre debe ser el mismo tanto en el momento de escritura de un nuevo documento como en el de lectura del mismo, es decir, los usuarios utilizan los equipos en el mismo intervalo de horas, en el mismo lugar, etc. De esta manera se pueden generar las claves de cifrado/descifrado de los distintos ficheros, debido a que siempre se calcularán las mismas si el contexto es el adecuado y serán diferentes si el contexto es inadecuado.

Por tanto, la clave para el cifrado/descifrado de un documento se obtiene a partir de la ejecución de una serie *retos*. Cada uno de estos *retos* procesan la información de contexto recibida, para calcular una subclave. Todas estas subclaves darán lugar a la clave final para el cifrado y descifrado de dicho fichero.

Esta propuesta de cifrado basado en contexto aplicado a DLP se integra en dos proyectos de investigación CiberNoid y DroneFS [9].

En el proyecto DroneFS se propone una arquitectura para el cifrado basado en contexto de la información que recopilan los drones (figura 3).

Esta arquitectura es la más adecuada para su uso en drones, por ejemplo para uso militar, donde los *retos* solo pueden ser ejecutados de forma local. Sin embargo, para integrar esta herramienta DLP en una organización, en la que el acceso a los datos se puede realizar a través de distintos tipos de dispositivos, como móviles o portátiles, añadir un servidor exterior puede facilitar la administración de nuevas políticas basadas en el contexto, aportar mayor seguridad dificultando la manipulación de los datos de contexto y permitir la ejecución de los retos fuera del dispositivo del cliente aliviando por ejemplo la carga de procesamiento y el gasto de batería de los dispositivos móviles. Este es el caso de la arquitectura del proyecto CiberNoid (Figura

4) que incluye un servidor externo para comprobar la autorización del usuario y ejecutar *retos remotos*.

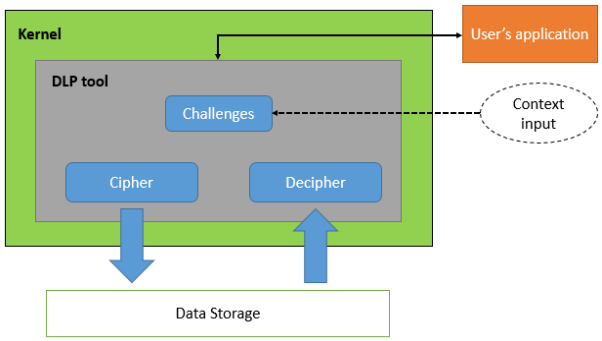

Fig. 3. Arquitectura DLP

En la figura 4 se puede ver que la arquitectura propuesta para la protección de ficheros sensibles está compuesta principalmente por la *Herramienta DLP* instalada en cada dispositivo cliente y el *Servidor externo* de la organización.

La *Herramienta DLP* propuesta se encarga de capturar a nivel de *Kernel* las llamadas que las aplicaciones realizan para el manejo de ficheros. En concreto, cuando el usuario crea un nuevo fichero y lo quiere guardar en el disco duro se cifran los datos. De igual forma, cuando un usuario abre un fichero, la herramienta DLP se encarga de descifrarlo antes de que sea visualizado en la aplicación final del usuario. Todas estas operaciones se realizan de manera totalmente transparente al usuario, sin que sea consciente de esta protección aplicada a la información, ni de los *retos* y parámetros necesarios. La herramienta propuesta no necesita estar instalada en todo el disco duro destinado a datos de la máquina, es decir, que se puede tener distintas particiones en disco y que sólo sobre una de ellas se aplique el cifrado de los ficheros. Por tanto, sólo las peticiones sobre estos ficheros, realizadas desde distintas aplicaciones, serán capturadas por nuestra herramienta a bajo nivel. Las peticiones de aplicaciones que no utilicen datos almacenados en dicha partición de disco no serán capturadas por nuestra herramienta, siguiendo su funcionamiento habitual.

El *Servidor externo* de la organización es un servidor HTTP con una API REST, el cual se utiliza para atender distintas peticiones de ejecución de los *retos remotos* recibiendo el contexto como parámetro y devolviendo un JSON con las subclaves calculadas. Además, tiene una base de datos donde el administrador almacena la información de los usuarios autorizados y cada uno de los parámetros necesarios para ejecutar los distintos *retos* configurados siguiendo la política de la empresa.

Cuando se quiere abrir o guardar un fichero, la llamada de la aplicación de usuario se captura a nivel de *Kernel* para que la herramienta DLP pueda realizar el cifrado/descifrado de la información. El primer paso es obtener los *retos locales* y *los retos remotos* asociados al equipo a partir de un fichero de configuración incluido por el administrador. La herramienta DLP ejecuta los *retos locales* directamente en el dispositivo y realiza una petición POST al servidor externo con los parámetros necesarios para la ejecución de los *retos remotos*.

En la Figura 5 se muestra el proceso que se lleva a cabo en la ejecución de *retos remotos*. Cuando un dispositivo necesita cifrar o descifrar un fichero, la herramienta DLP recoge los datos de contexto y los envía al servidor externo. Este servidor ejecuta todos los *retos remotos* utilizando tanto el contexto recibido para cada uno de ellos como las políticas de la empresa almacenadas por el administrador en la base de datos. Cada uno de los *retos remotos* calcula una subclave. Estas subclaves puede ser correctas o no dependiendo de los datos de contexto enviados al servidor.

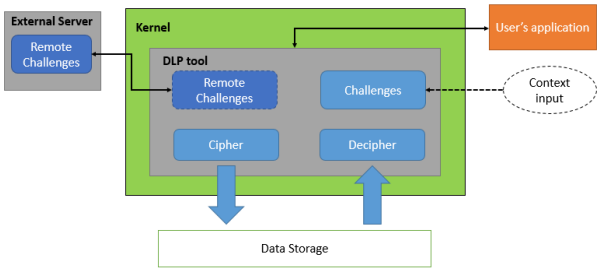

Fig. 4. Arquitectura DLP con servidor exterior

Una vez obtenidas todas las subclaves, tanto de los *retos locales* como de los *retos remotos*, la herramienta DLP calcula la clave de cifrado, como se detalla en la sección VI-C, y procede al cifrado/descifrado de la información.

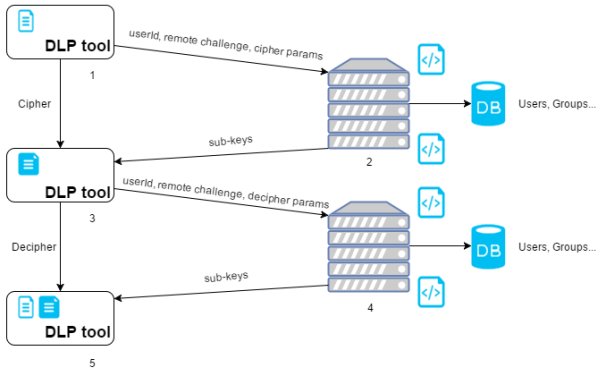

Fig. 5. Flujo del servidor con el FileSystem

V. Retos

Los *retos* son el mecanismo que permite calcular la clave de cifrado a partir de un conjunto de datos de contexto. En este sentido, se han diseñado una serie de *retos* capaces de generar una subclave a partir de unos datos de entrada. Cabe destacar que *los retos* que ejecuta el servidor tanto en el proceso de cifrado como en el descifrado de documentos son las mismas, por lo que no hace falta hacer ningún tipo de diferenciación entre las peticiones de cifrado y descifrado.

Los *retos remotos* propuestos se basan en geolocalización, la hora, la fecha, el operador de

telefonía y las redes wifi al alcance. En concreto, se proponen tres *retos* para determinar dónde se encuentra el usuario, basadas en la geolocalización, el operador de telefonía y las redes wifi al alcance; y el cuándo se determina mediante la ejecución de los *retos* de fecha y hora. En los siguientes subapartados se explica en mayor detalle cada uno de ellos.

A. Reto de localización GPS

Gracias al GPS se puede localizar a un usuario y comparar su situación geográfica con la que debería tener, por ejemplo la situación de la oficina. En este caso, el reto consiste en limitar el acceso a los ficheros confidenciales en todos los lugares que no estén acotados dentro de una zona delimitada por el administrador del sistema. Esta zona queda delimitada por dos parámetros incluidos por el administrador en la base de datos del servidor:

- Centro: coordenadas geográficas del centro del área permitida. Por ejemplo, en un edificio de oficinas, se almacena el centro del edificio.
- Área a cubrir: por defecto, el área a implementar será circular, por lo que el administrador solo deberá dar como dato la medida del radio del círculo a cubrir por sistema.

Con estos dos parámetros, la zona queda perfectamente delimitada. Siempre que se acceda desde dentro de este área se obtiene la misma subclave para un fichero concreto, mientras que fuera de ella se retorna un valor de subclave aleatorio y no válido.

Cuando un equipo necesita cifrar o descifrar un fichero realiza una petición POST al servidor con las coordenadas geográficas en las que se encuentra. En primer lugar, el *reto* utiliza el área y el centro almacenado en la base de datos por el administrador, para calcular los 4 puntos del círculo cuyas latitudes sean mayores y menores (es decir, los valores mínimos y máximos de latitud que se puedan dar en todos los puntos del interior del círculo) y comprueba cuál es la parte común entre los puntos. Por ejemplo, los puntos de latitud 3.4567º y 3.4589º tienen en común la parte 3.45º. O dicho de otra manera, tienen en común los 3 primeros dígitos de la coordenada. Este proceso se hace tanto para la latitud como para la longitud, obteniendo el número de dígitos invariantes en cada uno de ellos, generando 2 variables que contienen el número de dígitos invariantes en cada una de las dimensiones del área. Una vez hecho esto, el servidor sabe con cuántos dígitos de cada coordenada enviada como parámetro en la petición se tiene que quedar , obteniendo 2 valores para generar la clave.

Una posible complejidad podría venir dada por los lugares cuya coordenada cambie completamente. Por ejemplo, desde la latitud 3.9986 hasta la latitud 4.0056. En este caso, el sistema redondea las coordenadas al número x,y más cercano, siendo “x” la parte entera e “y” el primer decimal. Si la longitud presentara el mismo problema, se haría de la misma forma para obtiener los 2 valores necesarios para generar la clave.

B. Reto de fecha

Un reto básico es la comprobación de la fecha actual en el momento de intentar descifrar un fichero. Así, se puede limitar su acceso según la fecha, como por ejemplo poder descifrar los archivos pertenecientes a un proyecto en las fechas en las que se está trabajando en él. Este *reto* podría resolverse de manera local, pero, dado que el cambio de fecha en un dispositivo suele ser muy fácil de llevar a cabo y se podría engañar al sistema, es mejor ejecutarlo desde el servidor.

En este caso, se comprueba si un fichero se puede abrir en una franja de fechas predefinidas en el equipo. Nuestra propuesta se basa en utilizar una máscara que determine la duración del rango de fechas válido, como en direccionamiento IP. La idea es que variando la máscara y haciendo una operación del tipo *Fecha AND Mascara*, se pueda calcular siempre la misma subclave si se está en el rango correcto.

Para llevarlo a cabo, se van a codificar los meses en binario según su orden, de forma que los meses cercanos entre sí compartan el mayor número de bits posibles para poder hacer uso de la máscara y que se puedan determinar distintos rangos de fecha. Además, los meses se van a dividir en quincenas, de forma que la primera quincena de un mes tendrá una codificación distinta a la segunda. Por tanto, para codificar las 24 quincenas que hay en un año, se tienen que utilizar 5 bits (como mínimo). Una posible codificación sería: enero (00000, 00001), febrero (00010, 00011).... Con esta forma de codificar los meses, se consigue configurar distintos períodos de tiempo según la longitud de la máscara. Por ejemplo:

- Máscara 11111: Se corresponde con un periodo de una quincena, ya que al hacer *Time AND Mascara* (entendiendo *Time* como la fecha actual según nuestra codificación) nos quedamos con los 5 bits de la codificación. Estos bits solo pertenecen a una quincena concreta, es decir, si se hace la misma operación en quincenas distintas, el resultado sería distinto y la subclave no coincidiría.
- Máscara 11110: Sigue el mismo principio que el caso anterior, solo que ahora la validez es de un mes entero, ya que las 2 quincenas del mismo mes comparten los primeros 4 bits, y por tanto la subclave seguiría siendo la misma.

Como la máscara sólo tiene en cuenta los meses, podría darse el caso de que un fichero se pueda abrir cada enero (o el periodo que sea) de cada año. Para evitar eso, la clave generada también tendrá en cuenta el año actual en el momento de cifrado, haciendo que la función que calcula la clave reciba como parámetro el año.

Otra consideración a tener en cuenta es el día del mes en que comienza el tramo de fechas, como por ejemplo cifrar un fichero para un mes a finales de ese mes. Como el tramo es cerrado, solo será válido ese mes concreto, por tanto cuando empiece el mes nuevo, el cifrado dejará de tener validez. Para evitar los

problemas de cifrar un fichero en los últimos días de un periodo y que luego no sea válido, se introduce un offset que se corresponde con el día de creación del fichero, de forma que la codificación de los meses es dinámica. Por ejemplo, si un fichero se cifra el día 4 de enero, una codificación de los meses dinámica debe contemplar que las dos quincenas incluyan del 4 de enero al 4 de febrero. De esta forma, para cada fichero, se haría una codificación distinta dependiendo del día de creación del mismo.

El único inconveniente de aplicar el dinamismo en los tramos de meses es que el procesamiento será mayor al tener que hacer una codificación dinámica cada vez que llega una petición al servidor. Además, se necesita que la petición POST realizada desde el dispositivo incluya como parámetro la fecha de creación del fichero.

C. *Reto de hora*

Otro posible *reto* es la comprobación de la hora en el momento de intentar cifrar/descifrar un fichero. Así, se puede limitar el tramo de horas en las que se permite manejar la información sensible, como por ejemplo abrir solo una serie de archivos en horario de oficina. Este *reto* podría resolverse de manera local, pero dado que el cambio de hora en un dispositivo suele ser muy fácil de llevar a cabo y se podría engañar al sistema, es mejor ejecutarlo en el servidor.

Este *reto* tiene que comprobar que un fichero se puede abrir en una franja de horas predefinidas en el equipo. Esto se podría hacer de diferentes formas, pero al igual que en el *reto* de fecha, se utiliza una máscara que determina la duración del rango de horas válido. La idea es que variando la máscara y haciendo una operación del tipo *Tiempo AND Mascara*, se pueda obtener siempre la misma subclave si se está en el rango de horas correcto. Es decir, cualquier petición a este *reto* siempre dará el mismo resultado si se hace durante el mismo rango de horas.

Para implementarlo, se van a codificar las horas del día en binario según su orden, de forma que las horas cercanas entre sí compartan el mayor número de bits posibles para poder hacer uso de la máscara y así determinar distintos rangos de hora. Es decir, cada hora tendrá una representación en binario, por lo que en total serán 24 horas y se necesitarán 5 bits para poder codificar todas las horas, con la posibilidad de añadir más bits de relleno. Codificando así las horas, se consigue configurar distintos períodos de hora según la longitud de la máscara. Por ejemplo:

- Máscara 11111: Se corresponde con un periodo de una hora, que es la unidad más pequeña codificada. Al realizar la operación lógica *Time AND Mascara* (entendiendo *Time* como la hora actual según nuestra codificación) nos quedamos con los 5 bits de la codificación. Estos bits solo pertenecen a una hora concreta, por lo que si se hace la misma operación en horas diferentes, el resultado sería distinto y la subclave no coincidiría.
- Máscara 11110: Sigue el mismo principio que el anterior caso, solo que ahora la validez es de 2 horas, ya que dos horas consecutivas comparten los primeros 4 bits, y por tanto la subclave seguiría siendo la misma.

En principio, contar con un período mayor de 8 horas de validez no tendría mucho sentido, ya que suele ser la jornada laboral y, además, el siguiente tramo se correspondería con 16 horas, un tramo que no es nada práctico.

Como pasaba en el reto de fecha, los tramos vuelven a ser estáticos, lo que ocasiona problemas similares. Para resolverlo, vamos a seguir la metodología del reto de fecha pero aplicado a variar la hora de inicio, para encontrar un intervalo de tiempo adaptable. Para consiguir una codificación dinámica de las horas es necesario configurar en el equipo la "hora de inicio" y así poder determinar los rangos concretos de horas de cada usuario.

D. *Reto de wifi*

Las redes Wifi que están al alcance del equipo se pueden utilizar para determinar la localización del usuario. Para determinar en qué lugar se puede acceder a los ficheros confidenciales el administrador almacena en la base de datos el SSID, el canal y la potencia mínima de las redes Wifi configuradas para resolver el *reto*. El valor de potencia mínima se utiliza para constatar que se está en el lugar especificado, como por ejemplo el edificio de la empresa y no en la calle a una distancia próxima.

Cuando un equipo necesita cifrar/descifrar un fichero realiza una petición al servidor con el SSID, el canal y la potencia de recepción de las redes Wifi que están a su alcance. Cabe destacar que el equipo no sabe cuáles son las redes configuradas para pasar correctamente el *reto*, por lo que tiene que mandar todas las redes Wifi que están a su alcance. Con estos datos el *reto* genera un trozo de la subclave utilizando cada red Wifi enviada en la petición que concuerde con algún valor de SSID y canal configurado por el administrador, siempre que sea alcanzada a la potencia mínima especificada. De esta forma, si se encuentran todas las redes Wifi necesarias, se obtiene la subclave completa y correcta, mientras que si falta alguna, la subclave generada estará incompleta y, por tanto, no será válida para descifrar el fichero del equipo.

E. *Reto de operador*

Si se tiene una lista de operadores telefónicos por país, se puede comprobar el operador de los dispositivos que tengan conexión telefónica para saber en qué país se encuentra y así tener otro parámetro de localización.

Normalmente, las empresas tienen su servicio de red y móvil con la misma compañía, por lo que el operador siempre es el mismo y puede ser una condición para poder descifrar el fichero.

Por tanto, se configura el reto para que genere una subclave, haciendo una serie de operaciones con el nombre del operador. Es decir, con cada operador se obtendrá una clave distinta y por tanto proporciona un impedimento más para el acceso a los datos sensibles.

F. Robustez

Todos los *retos* planteados tienen como parte común la generación de una subclave y se debe asegurar que a partir de ellas no sea posible conocer el contexto con las que fueron calculadas. De esta forma se evita que, aunque un atacante pueda interceptar la comunicación entre cliente y servidor, sea imposible conseguir el contexto de entorno válido. Esto se consigue con la aplicación de la función hash SHA-256. Por otro lado, cada uno de los *retos* utiliza un número y tamaño de parámetros de entrada distintos sobre los cuales se aplica una serie de operaciones dependiendo del *reto,* a los que se incluirá el valor de identificador del fichero y el identificador de usuario o departamento, asegurando así la aleatoriedad de los valores de subclave obtenidos para cada uno de los ficheros con independencia de que compartan el mismo contexto y que solo puedan ser visualizados por el grupo de usuarios permitido.

Cada uno de los *retos* calculan una subclave y todas ellas son necesarias para que el equipo pueda obtener la clave de cifrado/descifrado. En este sentido, aunque un atacante supiera el valor de una subclave o de todas ellas no sabría cómo obtener la clave final necesaria para el descifrado de la información robada.

Otra opción es que el atacante, ya sea externo o interno de la organización, tenga acceso a un dispositivo autorizado. Por tanto, este dispositivo tiene la herrammienta DLP (Figura 4), con el que es posible calcular la clave final. La principal dificultad radica en que para calcular la clave correcta de un fichero se necesitan cumplir todos y cada uno de los *retos*, por tanto, también sería necesario que el atacante tenga conocimiento del contexto válido y tener la posibilidad de cumplirlo.

Por ejemplo, en el caso de un empleado despedido, cuya empresa utilice este sistema, además de tener que seguir registrado en el servidor como usuario autorizado y de tener a su disposición un dispositvo cliente válido, tendría que generar un escenario que cumpla el contexto correcto para descifrar los ficheros. Esto es, generar la posición geográfica correcta, a la hora y la fecha correctas, con un operador determinado y teniendo a su alcance unas redes wifi determinadas, en el canal adecuado y la potencia necesaria. Y esto teniendo en cuenta solo los retos explicados anteriormente.

VI. Módulo de cifrado

El módulo de cifrado es el encargado de proteger los ficheros confidenciales y de generar la clave de cifrado para cada fichero a partir de los *retos*. Este módulo se encuentra en el dispositivo del cliente y será ejecutado cada vez que un usuario quiera realizar una operación sobre un documento confidencial.

A. Proceso de cifrado

El proceso necesario para el cifrado de un fichero se lleva a cabo mediante una serie de pasos. Primero se genera la clave asociada a los retos a partir de las subclaves obtenidas tras la ejecución de los *retos*. Tras ello, se obtiene el algoritmo de cifrado, su modo de ejecución y el algoritmo de autenticación a partir del fichero de configuración almacenado en el dispositivo cliente. Además, es necesario realizar el cálculo del vector de inicialización (valor aleatorio diferente para cada fichero) y almacenamiento de dicho valor en la cabecera del fichero. Por último, se realiza el cifrado del contenido del documento utilizando la clave final generada y el vector de inicialización.

B. Proceso de descifrado

El proceso de descifrado se realiza siguiendo una serie de etapas. Se inicia con la generación de la clave de cifrado a partir de las subclaves obtenidas tras la ejecución de los *retos*. En segundo lugar se obtiene el algoritmo de cifrado, su modo de ejecución y el algoritmo de autenticación a partir del fichero de configuración almacenado en el dispositivo cliente. Tras ello se procede a la autenticación de la cabecera del fichero si fuera necesario y la obtención del vector de inicialización. Y finliza con el descifrado del contenido del documento.

C. Obtención de clave

La clave final con la que se cifra o descifra el fichero debe calcularse a partir de las subclaves obtenidas de la ejecución de los retos. El objetivo es añadir complejidad al algoritmo que genera la clave final para que sea difícil reproducir su comportamiento.

El algoritmo debe poseer dos propiedades importantes, que sea computacionalmente eficiente y resistente a colisiones. Una colisión ocurre cuando dos entradas distintas producen el mismo resultado. En nuestro caso podría ocurrir si distintas subclaves produgeran la misma clave final. Esto invalidaría en gran parte la funcionalidad de los *retos*. Para evirlarlo, vamos a utilizar una función Hash [10] teniendo en cuenta sus características principales: Aceptan cadenas de cualquier tamaño como entrada, producen una salida con un tamaño fijo, son computacionalmente eficientes, son unidireccionales (difíciles de invertir), resistentes a colisiones (propiedad inyectiva) para un tamaño suficientemente grande de la cadena, deterministas, es decir que para una misma entrada producen un mismo resultado.

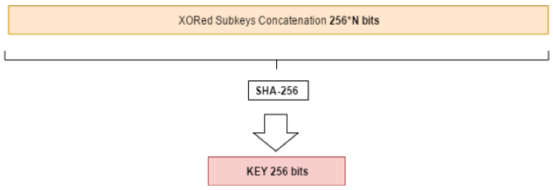

Fig. 6. Obtención de la clave final

Este módulo recibe como entradas las subclaves de los distintos *retos*, todos ellos con un tamaño de 256 bits (suficiente para evitar colisiones). Todas ellas se concatenan obteniendo una única cadena de 256*N bits. Finalmente, se obtiene la clave final de 256 bits haciendo el hash SHA-256 [11] de la cadena, como se puede ver en la figura 6.

D. *Elección de algoritmo de cifrado*

Se ha decidido implementar el algoritmo de cifrado simétrico AES utilizando el modo CTR ya que es el modo de cifrado que tiene mejor rendimiento, independientemente del tamaño del fichero [12][13]. Se utilizarán bloques de 128 bits y claves con tamaño de 256 bits que actualmente se consideran seguras [14]. Con este algoritmo, con un valor aleatorio y con la clave calculada anteriormente se cifran y descifran los ficheros confidenciales.

VII. CONCLUSIONES

Hoy en día la fuga de datos y el uso indebido de información se consideran una amenaza emergente para las organizaciones, especialmente cuando son llevadas a cabo por sus propios empleados. Además, las herramientas de DLP del mercado suelen enfocarse en evitar fugas de información desde atacantes externos y tratan a sus usuarios como si fueran absolutamente de confianza. La aplicación de cifrado basado en contexto en una herramienta DLP es un gran paso adelante para resolver este problema.

La arquitectura de DLP propuesta utiliza una serie de *retos* para obtener la clave de cifrado/descifrado a partir del contexto de entorno del dispositivo cliente que quiere utilizar datos sensibles. De esta forma se asegura que cada usuario solo pueda acceder a los datos críticos a los que tenga autorización dentro del contexto válido configurado por el administrador basado en las políticas de la empresa.

Estos *retos* pueden ser ejecutados de manera local en el dispositivo o de forma remota utilizando un servidor externo. La herramienta DLP integrada en el *Kernel* del dispositivo cliente se encarga de cifrar y descifrar los ficheros con claves generadas a partir de las subclaves obtenidas de la ejecución de los *retos*. Todo este proceso se lleva a cabo de manera totalmente transparente al usuario, el cual no tiene conocimiento del contexto de entorno válido. Además, el rendimiento de las aplicaciones de usuario se ve afectado mínimamente; por ejemplo la lectura de un fichero pasa de 20ms sin la herramienta a 150ms con la herramienta DLP, de los cuales 120ms se deben a la ejecución de los retos, que es independiente del tamaño del fichero. De esta forma, es posible desarrollar una herramienta DLP que realmente proteja a las organizaciones, no sólo de los atacantes externos, sino también de sus propios empleados.

Como trabajo futuro, quedan por realizar y definir más *retos* diferentes de los propuestos, como podrían ser *retos* cuya entrada de contexto sea multimedia, además de hacer estudios sobre la fortaleza de las contraseñas finales generadas por este método. Por último, se estudiará la posibilidad de controlar la seguridad en entornos donde el contexto de lectura y escritura sean distintos, mediante la realización de un conjunto de *retos* que compartan distintas propiedades matemáticas para obtener el mismo valor de subclave.

AGRADECIMIENTOS

Este trabajo ha sido parcialmente financiado con el apoyo del MINECO (proyecto DroneFS), con el código RTC-2015-4064-8 y del MINETUR (proyecto CiberNoid) con el código TSI-100200-2015-035.

REFERENCIAS

[1] Imad M. Abbadi, Muntaha Alawneh, “Preventing insider Information Leakage for Enterprises”, International Conference on Emerging Security Information, Systems and Technologies, pp. 27-31, 2011.

[2] Preeti Raman, Hilmi Güneş Kayacık, Anil Somayaji, “Understanding Data Leak Prevention”, Annual symposium on information assurance (ASIA), pp. 27-31, 2011.

[3] Asaf Shabtai, Yuval Elovici, Lior Rokach ”A Survey of Data Leakeage Detection and Prevention Solutions”, SpringerBriefs in Computer Science, 2012.

[4] Jungyub Lee, Sungmin Oh, Ju Wook Jang, “A Work in Progess: Context based Ecryption Scheme for Internet of Things”, Procedia Computer Science, v. 56, pp. 271-275, 2015.

[5] J. Al-Muhtadi, R. Hill, R. Campbell, M. D. Mickunas, “Context and location-aware encryption for pervasive computing environments”, Pervasive Computing and Communications Workshops, 2006.

[6] Vipul Goyal, Omkant Pandey, Amit Sahai, Brent Waters, “Attribute-based encryption for fine-grained access control of encrypted data”, Proceedings of the 13th acm conference on computer and communications security, pp. 89-98, 2006.

[7] Brent Waters, “Ciphertext-Policy Attribute-Bsed Encryption: An Expressive, Efficient, and Provably Secure Realization”, International Workshop on Public Key Cryptography, pp. 53-70, 2011.

[8] Z. Liu, Z. Cao, and D. S. Wong, “Efficient generation of linear secret sharing scheme matrices from threshold access trees”, IACR Cryptology ePrint Archive, 2010:374, 2010.

[9] Marina González, José J. García, Alberto García, Jorge Roncero, Víctor A Villagrá, “Propuesta de Sistema de Protección de la Información para Vehículos Aéreos no Tripulados”, II Jornadas Nacionales de Investigacion´ en Ciberseguridad (JNIC 2016), 2016, pp. 125–129.

[10] Phillip Rogaway, Thomas Shrimpton, “Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance
”, International Workshop on Fast Software Encryption, 2004, pp. 371–388.

[11] Harris E. Michail, George S. Athanasiou, George Theodoridis, Andreas Gregoriades, Costas E. Goutis, “Design and implementation of totally-self checking SHA-1 and SHA-256 hash functions’ architectures”, Microprocessors and MicroSystems, v. 45, Part B, pp. 227-240, 2016.

[12] Masram, Ranjeet, et al. "Analysis and comparison of symmetric key cryptographic algorithms based on various file features." *International Journal of Network Security & Its Applications* 6.4 (2014): 43.

[13] Altigani, Abdelrahman, Muawia Abdelmagid, and Bazara Barry. "Evaluating AES Performance Using NIST Recommended Block Cipher Modes of Operation." (2015).

[14] Paola Ceminari, Ariel Arelovich, Martín Di Federico, “Diseño de tres arquitecturas para un módulo criptográfico AES”, Biennial Congress of Argentina (ARGENCON), 2016 IEEE.