Criptografía ordenable para bases de datos

Santi Martínez
Dept. d'Enginyeria Informàtica i Matemàtiques
Universitat Rovira i Virgili
UNESCO Chair in Data Privacy
Av. Països Catalans, 26
43007 Tarragona
santiago.martinez@urv.cat

Víctor Mateu Rosana Tomàs Magda Valls
Departament de Matemàtica
Universitat de Lleida

C. Jaume II, 69
25001 Lleida
{vmateu,rosana,magda}@matematica.udl.cat

***Resumen*—Hoy en día, la información almacenada en bases de datos está en constante crecimiento. Dicha información puede contener datos sensibles, por lo que surge la necesidad de impedir el libre acceso a los mismos. Una posible solución es el cifrado de todos o parte de los datos. Sin embargo, a veces se desean permitir ciertas operaciones o consultas que requerirían que la base de datos descifrase todos los campos necesarios para efectuar la operación requerida, e.g. efectuar el sumatorio de un campo para todos los registros, obtener los registros con un campo comprendido entre dos valores. En este artículo se presenta un cifrado homomórfico en el que el orden de los datos se mantiene en los criptogramas.**

I. Introducción

La criptografía permite ocultar información sensible a posibles atacantes, debido a esto, hay un gran número de criptosistemas que enmascaran la información haciéndola ininteligible sin un descifrado. Pero no toda la información que se cifra tiene un mismo uso.

En una base de datos, cada registro tiene varios campos de información, algunos de los cuales pueden contener información sensible. En este artículo, nos centramos en los casos en que se necesite poder comparar (y ordenar) en función de uno de esos campos. La criptografía ordenable (*order-preserving encryption* o OPE) permite que el texto cifrado preserve el orden establecido del texto en claro. Así, aunque algunos campos estén cifrados, se podrán continuar haciendo consultas SQL [4] de forma eficiente, sin que un atacante que acceda a la información almacenada en la base de datos obtenga información sobre los datos en claro.

En esencia, un criptosistema ordenable es una función estrictamente creciente que va del conjunto al que pertenecen los datos al conjunto al que pertenecen los criptogramas. La seguridad del criptosistema recae en que dicha función, aun manteniendo el orden, parezca lo más aleatoria posible [3].

El resto del artículo está dividido en las siguientes secciones: la sección II, en donde veremos la motivación y algún ejemplo práctico de este tipo de criptografía, la sección III, donde se da unas ligeras pinceladas de criptosistemas de este tipo anteriores, la sección IV, donde se exponen los conceptos matemáticos previos usados en este método, la sección V, en donde se explica el método propuesto en este artículo, veremos unos ejemplos prácticos y los datos obtenidos en la práctica, la sección VI, donde se discute la seguridad de este método y, finalmente, una sección de conclusiones VII.

II. Motivación

El esquema propuesto ha sido diseñado para entornos en los que se contempla la posibilidad de que un intruso pueda llegar a acceder a la base de datos cifrada, pero no tenga información sobre la distribución de los valores ni pueda cifrar o descifrar valores arbitrarios. Nótese que si el acceso directo al contenido de la base de datos fuera imposible no sería necesario ningún tipo de cifrado, pues sería el sistema de control de acceso del gestor de la base de datos el que decidiría a que datos puede acceder cada usuario.

Un incidente real se relata en el periódico The Toronto Star [5], donde se explica que un banco vendió por eBay un disco, olvidando borrar los datos almacenados de cientos de sus clientes. El comprador, al darse cuenta, intentó resubastar el disco por eBay.

Imaginemos que tenemos una base de datos médica, como es de esperar cifrada, y queremos saber cuantos de los pacientes están en un rango de edad. Si no existiese la criptografía ordenable, en el momento de hacer la consulta, tendríamos que cifrar cada uno de los elementos que pertenecen a este rango y compararlo con la base de datos cifrada (suponiendo que el cifrado sea determinista, de lo contrario ninguna consulta sería posible sin descifrar la base de datos entera). Pero si esta base de datos está cifrada con criptografía que preserva el orden, al preservar el orden, únicamente tendríamos que cifrar el primer y el último valor del rango y hacer una consulta de cuantos registros están dentro de estos dos valores.

Esto se hace aún más necesario si, en vez de trabajar con campos de valor entero, como el de la edad, trabajásemos ordenando campos de valor real, entonces, sin este tipo de criptografía, sería totalmente impensable realizar cualquier tipo de consulta. Es decir, que en esta base de datos médica, en vez de que nos interesase ordenar por edad de los pacientes, lo que se pretendiese hacer es mirar que porcentaje de pacientes tienen más de un cierto nivel de azúcar en sangre. En este caso, podría haber mucha diversidad de valores, pero usando criptografía con preservación del orden sólo se tendría que cifrar el valor donde iniciar la búsqueda y en la consulta de la

base de datos pedir que se liste todos los registros con valor del campo superior a este valor cifrado. Es más, se podría también hacer una consulta a partir de ésta para ver si hay más índice de diabetes o problemas con el azúcar en hombres o en mujeres u otras consultas sin poner en riesgo la intimidad de los pacientes.

III. Trabajos previos

En [2], se propone un método que permite cifrar un entero p mediante la suma de los p primeros valores de una secuencia pseudo-aleatoria segura. Sin embargo, el cifrado obtenido es bastante predecible. Supongamos que los valores aleatorios pertenecen al intervalo $[0, Max]$, entonces la función $f(x) = \frac{Max}{2}x$ sería una buena aproximación de la función de cifrado (y $g(x) = \frac{2}{Max}x$ aproximaría la de descifrado). Además, el coste de cifrar un valor de n bits es exponencial en n (pues habría que generar y sumar unos 2^n valores de la secuencia).

En [7], se menciona el posible uso de polinomios para cifrar datos enteros. Dichos polinomios no deben tener máximos o mínimos en el intervalo al que pertenezcan los datos a cifrar. Sin embargo, es posible que no se pueda obtener la fórmula que corresponda a la inversa de dicho polinomio, lo que dificultaría el descifrado. Por ello, proponen como alternativa, aplicar iterativamente varios polinomios sencillos (de la forma $f(x) = ax^b + c$), todos ellos invertibles (la inversa sería $f(x) = \sqrt[b]{\frac{x-c}{a}}$), de manera que el descifrado consista en aplicar las inversas de dichos polinomios en orden inverso.

Recordemos que aplicar dos polinomios de primer grado seguidos es equivalente a aplicar sólo uno (si $f(x) = ax + b$, $g(x) = cx + d$, entonces $g(f(x)) = c(ax + b) + d = (ca)x + (cb + d)$, también de primer grado). Respecto a los de mayor grado, es más fácil introducir errores de desbordamiento de enteros (pues el número de bits está limitado). Con el fin de evitarlos, los autores proponen controlar los parámetros que definen los polinomios y el uso de la función logaritmo ($f(x) = \log_2 x + c$), lo que requeriría tratar con valores reales.

En resumen, la elección de la clave es un proceso complejo que requiere considerar detenidamente qué parámetros se pueden escoger para que no haya un desbordamiento de entero. El descifrado puede ser bastante costoso pues se necesita resolver varias raíces de los grados que se hayan escogido para los polinomios.

Un problema añadido es que la función obtenida es excesivamente suave, por lo que un atacante que tenga acceso a unos pocos pares de datos con su correspondiente cifrado podría aproximarla con relativa facilidad.

En 2004, R. Agraval et al. [1] plantean el cifrado de datos pertenecientes a un subconjunto de los enteros $[p_{min}, p_{max})$, aunque se recuerda la posibilidad de tratar valores en coma flotante como si fueran enteros (e.g. interpretar un `float` de 32 bits como si fuera un `int` de ese mismo tamaño), pues los positivos mantienen el orden tal cual, y para los negativos, que tendrían el orden invertido, simplemente habría que restar el entero resultante al mayor negativo posible.

El método que proponen, transforma datos que siguen una cierta distribución en criptogramas que mantienen el orden y además siguen una distribución distinta elegida por el usuario. Para generar la función de cifrado, hacen uso de todos los datos a cifrar (por lo que si la base de datos está inicialmente vacía, el administrador deberá proporcionar muestras de los posibles valores esperados), así como de una lista de muestras de la que será la distribución a simular. A partir de todas estas muestras, se generará la información auxiliar necesaria que permitirá cifrar/descifrar información (i.e. la clave del criptosistema).

Además, para modelar las distribuciones, primero se particionan los datos en cubetas (*buckets*) en los cuales se hará interpolación lineal.

Para cifrar los datos, estos primero se transforman en una distribución uniforme, que a su vez se transforma en la distribución objetivo.

Uno de los problemas del criptosistema que proponen es la generación de la clave. Si bien ésta es relativamente pequeña (su tamaño es el triple del número de cubetas, y aseguran que no se necesita particionar en más de 200), la generación es lineal en el tamaño de la base de datos. Además, si después de generar la clave se añaden gran cantidad de datos (lo que puede suceder fácilmente si la clave se ha creado a partir de una BD vacía), puede ser necesario escoger una nueva clave y repetir el cifrado.

En [6] se propone el esquema COPE (*Chaotic Order Preserving Encryption*). En éste, se aleatoriza el orden de las cubetas en función de la clave, por lo que, en realidad, no se trata de un criptosistema ordenable puro. El hecho de tener que reordenar las cubetas para hacer cualquier consulta repercute negativamente en el coste.

Todos los criptosistemas de este tipo son necesariamente simétricos, puesto que conocer la función de cifrado permite aproximar, tanto como queramos, la función de descifrado.

En este artículo, proponemos un método más eficiente de criptosistema ordenable. Los datos a cifrar pertenecerán al intervalo real $[0, 1]$ y el cifrado consistirá en aplicar varias funciones sencillas, como se verá en la sección V.

IV. Conceptos básicos

La criptografía homomórfica es un tipo de cifrado en el que una operación concreta en el texto llano equivale a otra operación (no necesariamente la misma) en el texto cifrado. En este caso se presenta un cifrado en el que la operación homomórfica es la comparación, por lo que el orden de los textos en claro se corresponde al orden de los criptogramas.

Como se ha comentado en las secciones anteriores, la ventaja de la criptografía ordenable es el hecho de que mantiene el orden de los datos cifrados, es decir, que si se ordenase la base de datos por un campo, el orden de los registros será el mismo tanto si dicho campo está cifrado o no. Para ello, se necesita que los datos de los campos se puedan ordenar de forma creciente. Y además, que el orden obtenido con los datos en claro sea el mismo que el obtenido con los datos cifrados. Para ello, necesitamos que el cifrado sea una función estrictamente creciente.

En el método propuesto, se trabaja con datos de entrada y de salida dentro del intervalo real $[0, 1]$, por este motivo, antes de cifrarlos, los datos se convierten en valores de dicho intervalo. Este proceso de conversión, hace que se pueda perder algo de información (la precisión dependerá del tamaño de la mantisa del tipo real utilizado). Siguiendo con el método propuesto, al pretender mantener el orden, necesitamos utilizar funciones estrictamente crecientes, ya que podremos asegurar que para $x < y$, entonces $f(x) < f(y)$. La composición de funciones estrictamente crecientes es también estrictamente creciente. Así, podemos utilizar la composición de varias funciones estrictamente crecientes como función criptográfica para mantener el orden de los datos.

V. Método de cifrado ordenable

Como ya se ha dicho, para este método queremos una función estrictamente creciente. A su vez, al igual que los otros métodos previos a éste, el criptosistema propuesto también es simétrico. Para construirla, lo haremos a partir de la composición de funciones crecientes sencillas. En realidad, no se necesita que la función sea creciente en todo $\mathbb{R}$, sino sólo en el intervalo $(0, 1)$ porque cualquier intervalo se puede transformar en este intervalo, incluso toda la recta real. Recordemos que para pasar del intervalo de $(-\infty, +\infty)$ al intervalo $(0, 1)$ podemos utilizar la siguiente función:

$$y = \frac{1}{2} + \frac{\arctan(x)}{\pi} \tag{1}$$

Para construir la función de cifrado, lo que haremos es componer diversas funciones estrictamente crecientes muy sencillas, para obtener, a partir de ellas, otra que también será estrictamente creciente, pero más compleja. La función básica estará definida por dos trozos de recta, el primero de los cuales pasará por el origen y un punto $P_k = (x_k, y_k)$, que actuará como clave, y el segundo pasará por el punto P_k y el punto $(1, 1)$. Dicha función se puede expresar como:

$$f_k(x) = \begin{cases} \dfrac{y_k}{x_k} x & \text{si} \quad x \leq x_k \\ 1 - \dfrac{1 - y_k}{1 - x_k}(1 - x) & \text{si} \quad x > x_k \end{cases} \tag{2}$$

Se puede observar una función de este tipo en la figura 1.

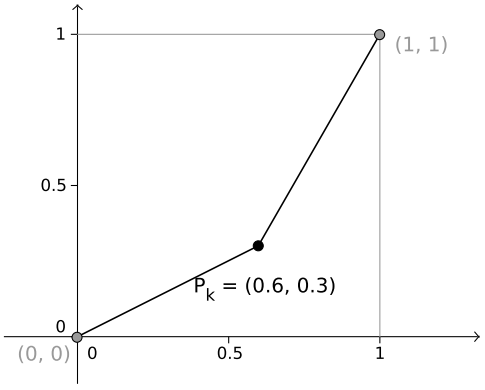

Figura 1. Función básica para el cifrado.

Entonces, para definir la función que se utilizará para cifrar, se deben escoger n puntos $P_{k_i} = (x_{k_i}, y_{k_i})$, para $1 \leq i \leq n$, a partir de los cuales definiremos n funciones f_{k_i} como se ha explicado anteriormente.

El cifrado, por tanto, será la composición de las n funciones de la siguiente forma:

$$f_{k_n}(f_{k_{n-1}}(...(f_{k_2}(f_{k_1}(x)))...)). \tag{3}$$

Por otra parte, para descifrar usando nuestro método, debemos componer las inversas de estas funciones en orden inverso al anterior, es decir:

$$f_{k_1}^{-1}(f_{k_2}^{-1}(...(f_{k_{n-1}}^{-1}(f_{k_n}^{-1}(x)))...)), \tag{4}$$

donde cada una de estas inversas $f_{k_i}^{-1}$ es la función que pasa por el punto $P'_{k_i} = (y_{k_i}, x_{k_i})$, el opuesto respecto la bisectriz, es decir, cambiando el orden de la abscisa y la ordenada del punto P_{k_i}.

Podemos ver como sería una de estas inversas en la figura 2.

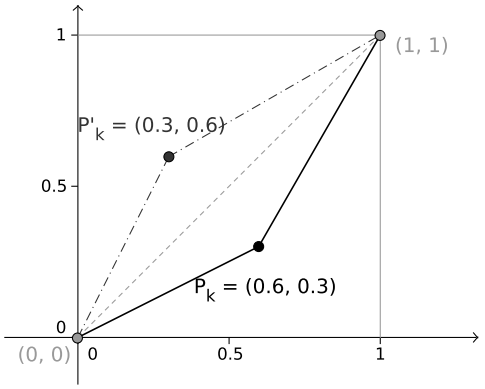

Figura 2. Función básica para el descifrado.

Notemos que al aplicar la función de cifrado y a continuación la de descifrado se puede obtener un valor que difiera en los bits de menor peso respecto al dato original debido a que no se trabaja con precisión infinita. En la sección VI se tratará con mayor detalle este problema.

V-A. Elección de la clave

En esta sección se muestran diversas gráficas correspondientes a posibles funciones de cifrado (en línea continua) y sus correspondientes funciones de descifrado (en línea de puntos y rayas). Para generar estos ejemplos se han usado 20 puntos aleatorios (ver figura 3), aunque la implementación real del criptosistema requeriría escoger una clave mucho más larga.

Sin embargo, si no se aplica ninguna restricción a los puntos aleatorios escogidos como clave, sucede, muy a menudo, que obtenemos lo que denominamos una función degenerada. Esto puede darse de dos formas: o que la función permanezca casi plana durante buena parte de su recorrido acercándose al punto $(1, 0)$, para después crecer abruptamente hacia el punto $(1, 1)$, como podemos ver en la figura 4, o bien, que la función

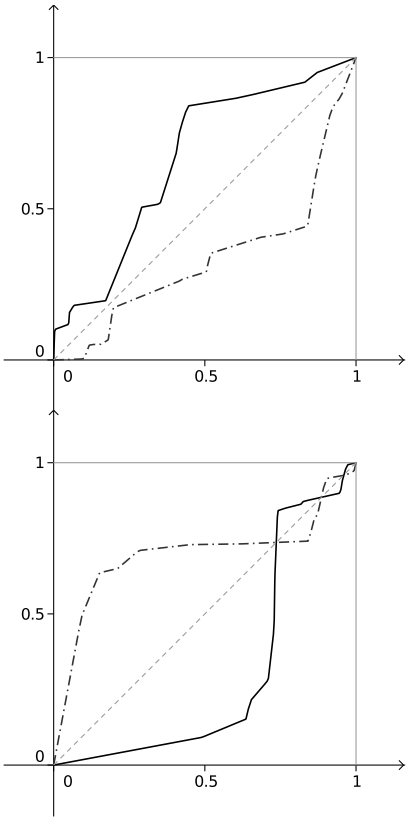

Figura 3. Ejemplos de cifrado y descifrado con clave de 20 puntos.

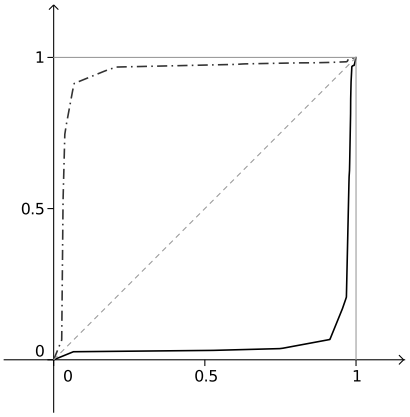

Figura 4. Ejemplo de función degenerada.

crezca rápidamente, acercándose al punto $(0, 1)$, tras lo cual continuará casi plana hasta el punto $(1, 1)$.

Esto puede ser un problema ya que puede conllevar una mayor pérdida de información en los bits de menor peso. Además, una función de este tipo es menos aleatoria, y, por tanto, menos útil.

Una forma de evitar el exceso de estos casos degenerados es acotar la región de donde se escogerán los puntos aleatorios. Una posibilidad sería escoger los puntos dentro de la intersección de las circunferencias de centro $(0, 1)$ y $(1, 0)$, ambas de radio unidad (véase la figura 5), o lo que es lo mismo, se restringe que los puntos aleatorios cumplan las siguientes inecuaciones:

$$\begin{aligned} x_k^2 + (y_k - 1)^2 &< 1 \\ (x_k - 1)^2 + y_k^2 &< 1 \end{aligned} \tag{5}$$

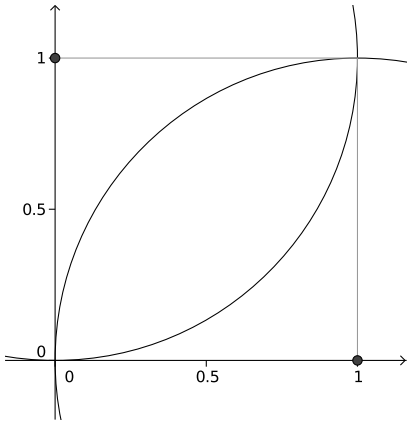

Figura 5. Sección donde elegir los puntos al azar.

Al implementar el criptosistema se ha observado que esta restricción no es suficiente cuando el número de puntos crece, y esto es lo que sucede en los casos reales.

Para ello, hemos dado otras restricciones que deben cumplir los puntos para que no aparezcan casi en la totalidad de las veces los casos degenerados.

Los puntos aceptables para una clave, además de cumplir la restricción anterior, deberán pertenecer al interior de un rectángulo centrado en la diagonal (véase la figura 6). Este rectángulo será más estrecho cuantos más puntos se vayan a generar, de manera que, al acercarse a la diagonal, la función correspondiente a cada punto individual será más próxima a la identidad.

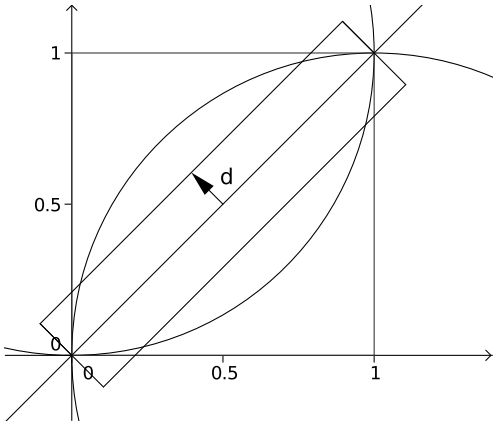

Figura 6. Recinto al que pertenecerán los puntos aleatorios generados.

Esto no impedirá que la función siga teniendo un aspecto aleatorio, pues se deberán componer muchas de estas fun-

ciones. Además, esta variación minimizará la aparición de casos degenerados.

La anchura del rectángulo será $2d$, siendo $d = \frac{\sqrt{2}}{f}$, $f = \sqrt[7]{n^4}$ un factor de escalado, y n el número de puntos a generar.

Para encontrar el valor adecuado del factor f, lo que se ha hecho ha sido, para diversos tamaños de clave, generar varias claves con distintos valores de f, y representar cada una de las funciones obtenidas, escogiendo, para cada tamaño de clave, el valor de f más adecuado, i.e. el que generase una función que no fuera demasiado degenerada ni demasiado cercana a la identidad. Aplicando regresión lineal a los logaritmos de dichos factores en función de los logaritmos de los tamaños de clave, obtuvimos la fórmula mostrada en el párrafo anterior.

V-B. Experimentación

Para probar la eficiencia del criptosistema, se ha implementado y experimentado, con diferentes cantidades de puntos, tanto la generación de la clave como el cifrado/descifrado. Los experimentos se han realizado con cantidades de 2^n puntos, para $4 \leq n \leq 26$, usando un ordenador a $2{,}4$ GHz.

En el cuadro I se muestran los resultados de la experimentación. En él, se indica el número de puntos de la clave, el tamaño de la clave resultante, el tiempo de generación de una clave de dicho tamaño y el tiempo empleado en cifrar un dato.

Para cada valor de n, se han generado diez funciones de cifrado (la clave de cada una de ellas está formada por 2^n puntos). Con cada una de estas funciones se ha cifrado un conjunto de datos cuyo tamaño depende del tamaño de la clave que define la función (por una cuestión de eficiencia a la hora de realizar las pruebas).

Los puntos de una clave se han representado como dos valores en coma flotante de 64 bits (tipo `double` en lenguaje C), por lo que el tamaño de la clave, en bytes, es 16 veces su número de puntos.

Los conjuntos de datos a cifrar son valores aleatorios entre 0 y 1 (también de tipo `double`). Estos conjuntos se han escogido con un tamaño dependiente del número de puntos de la clave, de manera que, mientras para claves pequeñas se han cifrado varios millones de datos, para las claves más grandes se han cifrado sólo varias decenas.

Para obtener el tiempo de generación de las claves se ha hecho la media de los tiempos de generación de las diez claves de cada tamaño (notemos que este tiempo será el de generar los puntos de la clave dentro del área aceptada).

Para medir el tiempo de cifrado de un único valor, se ha medido, para cada tamaño de clave, el tiempo empleado en cifrar los conjuntos de datos con las diez claves de ese tamaño. La columna T. cifrado es la media de los tiempos de cifrado de los diez conjuntos, dividido entre el número de datos del conjunto, por lo que corresponde al tiempo de cifrado de un único dato.

Nótese que, el tiempo de generación de la clave se ha obtenido como la media de diez mediciones, en cambio, el tiempo de cifrado es la media del tiempo empleado en cifrar diez conjuntos de datos (cada uno con una clave) dividido entre el tamaño del conjunto de datos usado para ese tamaño de clave. Debido a ello, los tiempos de cifrado se han obtenido con mayor precisión, a diferencia de los tiempos de generación de las claves que, para cantidades de puntos muy pequeñas, son prácticamente indistinguibles.

Cuadro I
RESULTADOS DE LA EXPERIMENTACIÓN

n	Núm. puntos	Tamaño clave	T. generación	T. cifrado
4	16	256 B	0,29 ms	179,5 ns
5	32	512 B	0,30 ms	344,9 ns
6	64	1 KiB	0,30 ms	0,68 μs
7	128	2 KiB	0,31 ms	1,4 μs
8	256	4 KiB	0,35 ms	3,1 μs
9	512	8 KiB	0,42 ms	6,2 μs
10	1024	16 KiB	0,53 ms	12,2 μs
11	2048	32 KiB	0,76 ms	23,6 μs
12	4096	64 KiB	1,2 ms	51,6 μs
13	8192	128 KiB	2,1 ms	102,4 μs
14	16384	256 KiB	3,9 ms	199,4 μs
15	32768	512 KiB	7,2 ms	396,3 μs
16	65536	1 MiB	13,9 ms	0,8 ms
17	131072	2 MiB	17,8 ms	1,6 ms
18	262144	4 MiB	28,0 ms	3,2 ms
19	524288	8 MiB	53,9 ms	6,4 ms
20	1048576	16 MiB	93,4 ms	12,8 ms
21	2097152	32 MiB	167,6 ms	25,6 ms
22	4194304	64 MiB	332,1 ms	52,9 ms
23	8388608	128 MiB	0,66 s	105,1 ms
24	16777216	256 MiB	1,33 s	211,9 ms
25	33554432	512 MiB	2,57 s	409,1 ms
26	67108864	1 GiB	5,12 s	0,87 s

Como se esperaba, los tiempos son bastante lineales, lo que permite aumentar bastante el tamaño de la clave. Incluso cifrando grandes cantidades de datos a la vez se obtiene tiempos razonables. Por ejemplo, con una clave de 65536 puntos, el tiempo necesario para cifrar un millón de datos serían unos 13 minutos (con AES se tardaría $0{,}1$ s, aproximadamente).

Evidentemente, una vez cifrados los campos necesarios en una base de datos, el cifrado sólo será necesario para añadir nuevos registros o para hacer consultas, por lo que no será habitual que se tengan que cifrar gran cantidad de datos de una sola vez. Esto permitiría escoger claves aún mayores, con el único problema de que el cifrado inicial de la base de datos se verá retardado.

Notemos que, en realidad, la clave no tiene porqué almacenarse en disco, pudiendo almacenar, en su lugar, la semilla aleatoria que se usó para generarla. De esta manera, cada vez que se necesite, podrá ser generada de nuevo (manteniéndola en memoria todo el tiempo posible para minimizar el número de regeneraciones). Obviamente, la semilla (o la clave, si fuera pequeña) deberán almacenarse cifradas con un criptosistema adecuado. En caso contrario, un atacante que accediera al disco, podría usarla para descifrar los datos.

La generación de la clave es mucho más lenta que el cifrado de un dato debido, principalmente, al tiempo requerido para la generación de los valores aleatorios requeridos. En cualquier caso, esto es algo que sólo se realizará ocasionalmente.

VI. Consideraciones de seguridad

En todo criptosistema que conserve el orden, un atacante con acceso a varios pares texto claro-texto cifrado puede aproximar la función de cifrado y la de descifrado, pues cada par corresponde a un punto de la función de cifrado (o descifrado, si se intercambian las coordenadas).

Sin embargo, cuando se haga una consulta a la base de datos, será el gestor de la BD el que cifrará los valores necesarios para realizarla y descifrará los valores a retornar. Por ello, un usuario no debería tener acceso a datos cifrados.

Si finalmente un atacante accede a varios datos cifrados (y sus correspondientes datos en claro), la seguridad recaerá en la aleatoriedad de la función de cifrado [3], de forma que la cantidad de puntos necesarios para obtener una buena aproximación sea lo suficientemente grande.

Así pues, el principal problema al que debería hacer frente una BD, es el posible acceso, por parte de un atacante, a los datos almacenados internamente en disco. En tal contexto, para evitar la exposición de información sensible, lo más adecuado es cifrar todos los campos (pues incluso una combinación inusual de campos no sensibles podría ser usada como identificador), usando un criptosistema u otro en función del tipo de búsquedas que se deban permitir.

Así, si para un campo no se permiten búsquedas, éste debería cifrarse de forma no determinista, de manera que datos iguales produzcan criptogramas distintos. E.g.: una BD de personas podría contener un campo 'color_pelo', y no permitir búsquedas del tipo "color_pelo=castaño".

A los campos en que se permitan búsquedas por igualdad pero no se deseen (o no tengan sentido) las búsquedas por intervalos, se les deberá aplicar un cifrado determinista. De esta manera, al hacer una búsqueda, el gestor de la BD podrá cifrar el parámetro solicitado y comparar con los valores cifrados del campo (sin necesidad de descifrarlos todos previamente). E.g.: 'ciudad_nacimiento' podría estar cifrado de esta manera.

Finalmente, los campos sobre los que se deban permitir búsquedas por intervalos, deberán cifrarse con un criptosistema como el que proponemos. E.g.: 'año_nacimiento' debería cifrarse de esta forma, con el fin de poder hacer búsquedas filtrando por intervalos de edades.

Notemos que tanto en la conversión de los datos a cifrar en un real de coma flotante, como en el cifrado, los bits de menor peso pueden verse modificados, introduciendo pequeños errores. E.g.: si se cifra con este protocolo una cadena de texto solo los 11 primeros caracteres podrán ser codificados correctamente en la mantisa de un `double`. El proceso de cifrado probablemente modifique el último carácter, dejando sólo 10 de ellos intactos.

En la mayoría de los casos esto no será un problema, e.g. para el campo 'año_nacimiento' perteneciente al intervalo entero $[1890, 2145]$, sólo los primeros 8 bits serán relevantes, por lo que pequeños errores no afectarán en absoluto. Sin embargo, con campos que requieran más precisión, se podría perder algo de información al descifrar. Para solucionar esto, basta con almacenar el campo por duplicado, cifrado con el criptosistema propuesto y con otro que permita un descifrado no ambiguo. Una vez realizada una búsqueda por intervalo se deberá descifrar la versión no ambigua del campo antes de retornar los resultados, y, si es el caso, filtrar cualquier resultado no deseado (e.g. si el intervalo de búsqueda fuera de $0{,}5$ a $0{,}7$, podría obtenerse un registro en el que el campo valiera $0{,}70000000000001$). Si el cifrado auxiliar es determinista tendremos la posibilidad de usar el ordenable para hacer búsquedas por intervalo (filtrando después los falsos positivos) y el determinista para hacer búsquedas por coincidencia (y evitar así tener que filtrar los resultados).

Finalmente, si la BD contiene varios campos ordenables, es recomendable cifrar cada uno de ellos con una clave distinta, para dificultar la aproximación de la función de cifrado a un atacante con acceso a pares texto claro-texto cifrado de distintos campos.

VII. Conclusiones

En este artículo se ha expuesto un método para la criptografía que preserva el orden. Este método, no sólo es sencillo de implementar y entender, sino que también es eficiente para ponerlo en práctica. Además, usando unas ligeras restricciones, se eliminan casos degenerados para así tener la entropía deseada y que no sean predecibles los datos cifrados.

En el artículo también podemos ver algunos resultados prácticos obtenidos de varias ejecuciones de la implementación de este método, donde vemos que se comporta con un coste lineal y sería factible llevarlo a la práctica.

Agradecimientos

Este trabajo ha sido financiado parcialmente por el Gobierno de España con los proyectos TIN2009-11689 "RIPUP", CONSOLIDER INGENIO 2010 CSD2007-00004 "ARES" y MTM2010-21580-C02-01, y 2009SGR-442, 2009SGR-1135 de la Generalitat de Catalunya.

S. Martínez es investigador de la Cátedra UNESCO en Privacidad de Datos, pero los puntos de vista expresados en el presente artículo no reflejan necesariamente la posición de la UNESCO ni comprometen a dicha organización.

Referencias

[1] R. Agrawal, J. Kiernan, R. Srikant, Y. Xu, "Order preserving encryption for numeric data", en *Proceedings of the 2004* ACM SIGMOD *international conference on Management of data*, pp. 563–574, 2004.

[2] G. Bebek, "Anti-tamper database research: Inference control techniques", Technical Report EECS 433 Final Report, Case Western Reserve University, 2002.

[3] A. Boldyreva, N. Chenette, Y. Lee, A. O'Neill, "Order-Preserving Symmetric Encryption", en *Proceedings of the 28th Annual International Conference on Advances in Cryptology: the Theory and Applications of Cryptographic Techniques* - EUROCRYPT'09, pp. 224–241, 2009.

[4] J. Groff, P. Weinberg, "SQL The Complete Reference, 3rd Edition", McGraw-Hill, Inc., 2010.

[5] T. Hamilton, "Error sends bank files to eBay", en la edición del 15 de septiembre de 2003 del periódico *The Toronto Star*.

[6] S. Lee, T.J. Park, D. Lee, T. Nam, S. Kim, "Chaotic Order Preserving Encryption for Efficient and Secure Queries on Databases", en *IEICE Transactions on Information and Systems*, Vol. E92-D, Núm.11, pp. 2207–2217, 2009.

[7] G. Ozsoyoglu, D. A. Singer, S. S. Chung, "Anti-Tamper Databases: Querying Encrypted Databases", en *Proc. of the 17th Annual IFIP WG 11.3 Working Conference on Database and Applications Security*, 2003.