Searchable Encryption Through Dispersion

C. Aiello, L. Vidal and T. Schwarz

***Abstract*— Cryptography is the universal tool to protect the privacy of data. Today's cryptography still requires encrypted data to be decrypted before it can be searched. We propose here an alternative way of protecting the privacy of data through dispersion of a compressed version of the original data that can be searched without recovering the original data. Our scheme compresses the original data and then generates several chunks that are stored at different nodes. The chunks are stored in the form of an index. To search for a string, we convert the string into chunks with the same scheme and then have each site consult its index in order to obtain a list of all possible positions where the search string might be found. These local results are then sent to the user who performs a logical intersection to find all likely positions in the original, where the search string might be located in the text. The user can then decrypt only those parts or records to obtain all parts or records where the search strings. Our scheme has no false negatives (all occurrences of the search string will be found). We show that the precision becomes close to 100% for longer strings using a corpus consisting of texts in the English language. We also show that the chunks are somewhat, but not quite similar to random bit streams, and that each individual stream has less information content than a typical English language stream of the same length.**

***Keywords*— Searchable Encryption.**

I. INTRODUCCIÓN

LA CRIPTOGRAFÍA es una herramienta universal para proteger la confidencialidad de los datos. Desafortunadamente, con los esquemas criptográficos actuales, para procesar datos cifrados es necesario descifrar antes. La encriptación homomórfica ofrece la capacidad de procesar los datos cifrados directamente, incluyendo búsquedas. Pero a pesar del gran progreso en los últimos años, especialmente a través del trabajo de Gentry, es demasiado pesado para aplicaciones practicas [8], [9], [10], [25].

El estado del arte actual usa encriptación simétrica y asimétrica para lograr búsquedas por palabras clave y proteger datos proporcionando una seguridad probada y alta (Sección VI). En este artículo proponemos una alternativa radical, donde la seguridad se logra limitando la cantidad de la información almacenada en un sitio. Un adversario que logra subvertir un sitio simplemente no tendrá datos suficientes para reconstruir la información original. La verdad de esta afirmación por supuesto depende de la cantidad de información restante y por lo tanto, es esquema es menos seguro. Compramos más fácil procesamiento y mejor capacidad de búsqueda para esta leve pérdida de seguridad.

Nuestra alternativa está basado en la dispersión y permite buscar en los datos protegidos por cualquier subcadena, no solamente por palabras clave como en el estado del arte. Analizamos dos planteamientos diferentes. El primero es buscar en los datos protegidos directamente. Usando dispersión, el texto original es comprimido y dividido en una cadena por sitio, cada cadena es procesada en cada sitio y los resultados en cada sitio se intersectan con el resto para identificar con gran probabilidad las posiciones de la cadena buscada en el texto original. Para mejorar el desempeño, proponemos una estructura de búsqueda dispersada – un índice.

Ahora presentamos primero nuestro esquema, evaluamos sus costos y seguridad, y discutimos sobre el trabajo relacionado y futuro.

C. Aiello, Universidad Católica del Uruguay, c@it.com.uy
L.Vidal, Universidad Católica del Uruguay, luisvidalintroini@gmail.com
T. Schwarz, SJ, SSRC, University of California at Santa Cruz, tschwarz@calprovorg

II. ARQUITECTURA

Nuestro esquema crea un *índice* distribuido a partir de datos cifrados. Los datos originales son cifrados y almacenados en otro lugar. El objetivo de nuestra aplicación es de proveer al cliente rápidamente con una lista de las probables posiciones del elemento buscado. La probabilidad de falsos positivos es baja y no hay falsos negativos ya que cada posición del elemento buscado es retornada en la lista. El costo de un falso negativo sería solamente el de descifrar el bloque que contiene la posición errónea. Nuestra propuesta permite al usuario buscar por cualquier subelemento, no solamente por palabras claves.

A pesar de que la idea fundamental es muy general, su detalle depende del tipo de dato. Las búsquedas dentro de una imagen dependerán fuertemente del formato de la imagen y lo mismo ocurre con mapas, películas, etc. A continuación presentamos una versión optimizada para registros en lenguaje natural.

Nuestro esquema usa compresión con pérdida de información. Esto disminuye el tamaño del texto a la mitad, pero duplica la carga de información de cada carácter. Cada letra del alfabeto original es puesta en un balde (bin /conjunto de caracteres), luego esa letra es codificada con el número del balde en su versión comprimida. El siguiente paso toma cada carácter del texto comprimido y distribuye el número del balde al que pertenece el carácter en varias secuencias de bits. El número de secuencias es k, siendo k=3 y cada bit de cada secuencia corresponde a un carácter en el texto original. El número de baldes es 2^k. Si usamos un alfabeto diferente o los datos están en un dominio diferente, por ejemplo audio, el número de secuencias podría y debería ser diferente. El resultado de estos dos pasos es la dispersión de la información contenida en un texto original en tres cadenas de bits diferentes, Fig. 1. arriba.

Buscar un texto usando estas cadenas de bits es algo pesado debido a que la arquitectura de microprocesadores actual está optimizada para el procesamiento por bytes. Una implementación puede almacenar cada bit como un carácter 0

o 1 (un array de char en C) o utilizar las operaciones *shift* y *and* lógico con una máscara para buscar subcadenas en las tres cadenas de bits. Como ambas alternativas no son atractivas, decidimos utilizar una conocida técnica de recuperación de información, construir un índice de búsqueda completo para *trozos* (shingles) que son subcadenas de largo *l*. El índice de búsqueda usa todas las 2^l combinaciones posibles de trozos con largo *l* como vocabulario e indiza todas las ocurrencias de estos trozos en una lista conocida como la lista de *postings*. Las listas de postings pueden ser comprimidas efectivamente, primero almacenanando la diferencia con el índice anterior en lugar de las ocurrencias directamente, y después utilizando compresión gama o un código variable de bytes. Fig. 1 muestra el texto original, su dispersión en tres sitios. y después la lista de postings para cada sitio.

A. Binning

La cantidad de información que contiene un único símbolo en el idioma Inglés, ha sido de mucho interés [24]. Las mejores técnicas de comprensión clásicas han obtenido una compresión de 1.78 bits por carácter, mientras los actuales llegan a solamente 1.269 bits por carácter [15], pero no logran este rendimiento en todas las circunstancias. En nuestra propuesta, una secuencia contiene un único bit por carácter, y por ende, nuestro método se justifica si la cantidad de información de una letra es mayor a un bit. En caso de que un adversario obtenga acceso solo a una secuencia, no sería capaz de reconstruir el texto original por falta de información. (Pero claramente con información adicional, seriá capaz de sacar conclusiones sobre el texto original.)

La selección del método de bining tiene como objetivo obtener una distribución de ceros y unos en la secuencia de bits que sea parecida a la de una secuencia aleatoria, para que no sea posible distinguir entre una y otra. La precisión de las búsquedas dependerá en la especificidad de la cadena que queremos buscar. Por lo tanto, tratamos de igualar lo máximo posible la cantidad de símbolos que se asignan a cada bin .

Cada cuerpo tiene su propia distribución de frecuencia de sus símbolos, que depende principalmente pero no en su totalidad del idioma en que está escrito. Dado que elegimos evaluar textos en inglés, usamos una tabla de frecuencias de letras en textos en ingles tomada del trabajo de Lee [13] la cual tiene 26 letras, un espacio en blanco y otra categoría. Desarrollamos manualmente el binning dado en la Tabla I.

En otros idiomas se podría encontrar ventajas para el preprocesamiento. Por ejemplo, una palabra en español podría considerar “ñ”, “ch”, “ll” y “rr” como letras diferentes, ubicándolas posiblemente en baldes diferentes a donde se encuentran sus versiones con acento o con una letra sola.

B. Creación de las secuencias

Cada carácter pertenece a un balde. Codificamos el carácter como el número del balde (en binario). En el ejemplo de la Fig. 1, el texto “Cryptography…” se particiona primero en secuencias. La primera letra “C” está en el balde 4 (100 en binario), por lo tanto, el primer bit de la secuencia para el sitio 0 es 1, para el Sitio 1 y Sitio 2 es 0. La segunda letra “R” está

Texto Original Cryptography is a universal tool to

```
Sitio 0   01100101001100100010011110100111001
Sitio 1   00001010001001100011110010101001010
Sitio 2   11110111111101001011100101000110001
```

Patrón	Sitio 0	Sitio 1	Sitio 2
0000	{}	{0}	{}
0001	{15}	{1,7,15}	{26,31}
0010	{3,12,16}	{2,8,22,29}	{14,21}
0011	{8,19,27}	{11,16}	{27}
0100	{6,13,17,25}	{5,9,27}	{12,24}
0101	{4}	{3,23,25,30}	{15,22}
0110	{0,9}	{12}	{28}
0111	{20,28}	{17}	{4,17}
1000	{14}	{6,14}	{25,30}
1001	{2,7,11,18,26,31}	{10,21,28}	{13,20}
1010	{5,24}	{4,24,26,31}	{11,23}
1011	{}	{}	{3,16}
1100	{1,10,30}	{13,20}	{19,29}
1101	{23}	{}	{2,10}
1110	{22,29}	{19}	{1,9,18}
1111	{21}	{18}	{0,5,6,7,8}

Figura 1. Ejemplo de procesamiento de un texto original en tres cadenas de bits y en seguida en un índice con trozos de largo 3 y una lista de postings (occurrencias).

TABLA I
TABLA DE FRECUENCIAS DE LETRAS EN EL IDIOMA INGLÉS Y UNA POSIBLE SELECCIÓN DE BINNING CON 8 BALDES

Balde	Símbolos	Frecuencia de símbolos	Fr. Balde
0	espacio, K	(12.17+0.41)%	12.58%
1	E,B	(11.36+1.05)%	12.41%
2	T, M, F	(8.03+2.76+1.79)%	12.58%
3	S, D, L, V	(5.68+2.92+2.92+0.97)%	12.49%
4	A, C, P, W, J	(6.09+2.84+1.95+1.38+0.24)%	12.50%
5	O, R, Y, Q	(6.00+4.95+1.3+0.24)%	12.49%
6	N, I, G, X	(5.44+5.44+1.38+0.24)%	12.50%
7	Otros, H, U, Z	(6.57+3.41+2.43+0.03)%	12.44%

en el Balde 5, entonces el segundo bit es 1 en el Sitio 0, y para el Sitio 1 y Sitio 2 es 1.

Simplemente permutando los números de los baldes entre los baldes, podemos crear diferentes codificaciones. La que semuestra en la Tabla 1 intenta distribuir equitativamente la cantidad de unos y ceros dentro de las secuencias.

C. Creación del Índice

Con el fin de facilitar las búsquedas, usamos en cada sitio índices con una estructura que usa cadenas de bits de largo l = 8 (o l=10) asociadas a una lista de postings para tener una estructura clásica usada en recuperación de información [16]. Damos un ejemplo de la estructura en la Fig. 1, pero como no queremos representar índices con 2^8 o 2^{10} entradas, entonces usamos cadenas de bits de largo 4 que generan el vocabulario de la Fig. 1. La cadena de bits 0110 aparece en la primera y quinta posición del Sitio 0. Por lo tanto, la lista de postings para la entrada 0110 es {0,5}.

Cada letra en el texto original corresponde a un posting a menos que la letra esté a una distancia menor de *l* del final del texto. Asumimos que una compresión de la lista de los postings usa un promedio de *x* bytes por posting. La comunidad de investigadores en recuperación de información ha desarrollado esquemas de compresión buenos que usan desde menos de 4 a 5 bits por posting, dándonos un valor de x

alrededor de 0.5 [16]. Dado que la compresión se beneficia de no tener grandes diferencias entre dos ocurrencias siguientes en la lista de postings y dado que nuestras cadenas de bits son similares a cadenas aleatorias, podemos orientar más hacia el extremo inferior del valor de *x*. Los tres sitios almacenan entonces *x* bits para cada byte del texto original y por ende utilizan 1.5 veces el espacio de la cadena original. Como el conjunto de trozos no se cambia, no lo necesitamos aguardar.

El hecho de haber elegido un índice implica que hemos mejorado el tiempo de búsqueda al costo de almacenar solamente las cadenas de bits para cada sitio. Estos tienen un bit para cada byte del texto original. El tamaño del almacén para los tres cadenas de bits hubiera sido 3/8 del texto original.

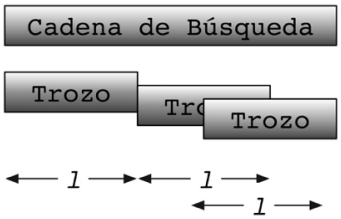

Figura 2. División de una cadena de búsqueda en trozos (grupos de *l* símbolos).

D. Realizando búsquedas

Para realizar una búsqueda de una subcadena, primero usamos el binning para codificar la búsqueda exactamente de la misma manera que obtenemos los trozos de secuencia binaria. Luego dividimos cada cadena de búsqueda en trozos - grupos de *l* bits continuos - y usamos los índices para encontrar las posiciones donde se encuentran las cadenas en cada sitio. Con mas detalle, si b_0, b_1, b_2,.....,b_{n-2}, b_{n-1} es una cadena de búsqueda binaria, entonces el primer trozo es b_0, b_1, …., b_{l-1}, el segundo b_l, b_{l+1}…..b_{2l-1} etc. Si *n* no es divisible entre *n*, tendremos un último corte b_{n-l-1}, b_{n-l},....,b_{n-1} , este se superpone parcialmente con el corte anterior como se muestra en la Fig. 2. Luego usamos la lista de postings en un sitio para obtener las posiciones de la cadena binaria dentro del trozo de secuencia. Si el primer corte de la cadena binaria empieza en la posición *x*, entonces el segundo trozo (a menos que esté solapado con el primero) empieza en la posición *x*+l, el tercer corte (a menos que sea el último y esté solapado con el segundo) empieza en la posición *x*+2*l* etc. Para el último corte, es necesario ajustar el desplazamiento a *x*. Después se pasa por el índice para determinar las ubicaciones posibles de la cadena de búsqueda (según la información disponible en el sitio), buscando situaciones donde una ocurrencia del segundo trozo se encuentra *l* posiciones después de una ocurrencia del primer trozo, etc. Después de determinar las posibilidades locales, todos los sitios envían su lista de posiciones a un punto central. En una posición donde todos los sitios coinciden, la compresión de la cadena de búsqueda está en la cadena comprimida del texto original. El último paso consiste en descifrar los bloques donde se encuentran las posibles ubicaciones de la cadena de búsqueda en el texto original para verificar que no se trate de un falso positivo.

E. Ejemplo

Asumimos que queremos buscar la subcadena “a univers” en el ejemplo de la Fig. 1. La cadena de búsqueda es comprimida con pérdida de información por los números de los baldes, dando “407663153”. La cadena de búsqueda binaria en el sitio 0 será “001001111”, la cual es cortada como “0010”, “0111” y “1111”. En nuestro pequeño ejemplo, el primer corte es encontrado en las posiciones 3,12 y 16. Si una de las posiciones es una posición verdadera de la cadena de búsqueda binaria, entonces el segundo corte debería ser encontrado en la posición 7,16 y 20, respectivamente. Dado que el segundo corte solo se encuentra en la posición 20, solo queda 16 como posible posición de la cadena binaria de búsqueda. Dado que el tercer corte empieza en un carácter eliminado del segundo corte, “1111” tiene que estar en la posición 21, lo cual es así. Por lo tanto, de acuerdo al Sitio 0, la cadena de búsqueda probablemente esté en la posición 16. Los resultados de los otros dos sitios también contienen este valor, esta es la única posición posible que puede contener la cadena de búsqueda.

Asumimos ahora que queremos buscar por la subcadena “too” en el texto de la Fig. 1. La compresión con pérdida de información genera la cadena de búsqueda “255”. Buscamos en el sitio 0 por la cadena binaria “011”, la cual completamos y queda como “011*”. Luego buscamos por las posiciones “0110” y “0111”. Esto nos retorna la lista {20,28}. Buscamos en el Sitio 1 por la cadena “100”. Combinamos los postings de “1000” y “1001”, obteniendo {6,10,14,21,28}. Buscamos en el sitio 2 por “011”, obteniendo {4,17,28}. Luego se intersectan las listas en el cliente, obteniendo 28 como la única posible posición, lo cual es correcto.

F. Búsquedas con comodines.

Existen dos tipos principales de búsquedas con comodines, una donde el comodín represente un único y desconocido carácter (el cual es trivial de implementar en nuestro sistema), y otro donde el comodín representa un conocido o desconocido número de caracteres. Incluso es posible de utilizar este tipo de búsquedas si las partes determinadas de la cadena de búsqueda conforman casi o completamente el corte. La idea es similar a la usada por Litwin *et al.* [14].

III. VERSIÓN CON SEGURIDAD MEJORADA

Como cada trozo posee un bit por símbolo en el texto original, la cantidad de información es menor, aunque cercana a la del texto en inglés. Podemos incrementar la seguridad solo almacenando la mitad de bits para cada símbolo en un trozo. Simplemente debemos dividir cada trozo en dos, almacenando los bits impares en un lado y los pares en otro. La mejora resultante por este cambio, tiene como costo el incremento en la complejidad de las búsquedas.

A. Dividiendo trozos

Si tenemos una cadena de bits b_0, b_1, b_2, …, definimos dos cadenas de bits más pequeños, la cadena de pares b_0, b_2, b_4, …, y la cadena de impares b_1, b_3, b_5, … . Los trozos pares e impares son almacenados en dos sitios diferentes. Para

Original 10010000111011000101010001011100011101010
Sitio A 1 0 0 0 1 1 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0
Sitio B 0 1 0 0 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1

Sitio A	Sitio B
000 [1, 7, 8, 9, 10, 11, 18]	000 []
001 [2, 12, 15]	001 [2]
010 [13, 16]	010 [0, 3, 5]
011 [3]	011 [7, 11, 15]
100 [0, 6, 14, 17]	100 [1]
101 []	101 [4, 6, 10, 14]
110 [5]	110 [9, 13]
111 [4]	111 [8, 12, 16, 17]

Figura 3. Dividiendo cadenas de bits en partes pares (sitio A) e impares.

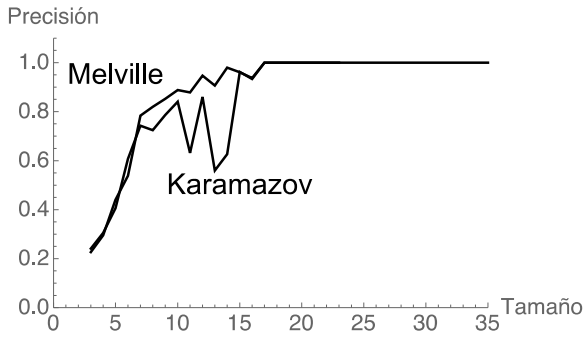

Figura 4. Precisión de 1000 búsquedas aleatorias en dos archivos dependiente del largo de la cadena de búsqueda.

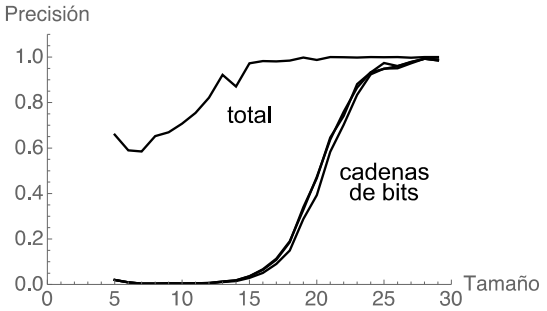

Figura 5. Precisión buscando 1000 subcadenas existentes en "Moby Dick".

permitir un procesamiento más veloz, como antes, creamos un índice de postings que toma cadenas de bits de largo *l* como vocabulario, y asocia las ocurrencias para mantener la misma precisión en la búsquedas que la variante básica.

B. Definiendo búsquedas

Después de dividir las cadenas de bits en componentes pares e impares, realizamos la búsqueda utilizando bloques de largo 2*l*. Un bloque de búsqueda, es en primera instancia convertido en tres cadenas de bits como anteriormente por poner las letras en baldes y asignar un bit del número del balde a una de las cadenas. Luego es dividido en partes pares e impares. Los índices en los sitios pares e impares son consultados por los postings, los cuales luego traducimos en posiciones en los datos iniciales. Si la cadena de búsqueda se encuentra en una posición par 2*p* en el texto, entonces la parte par de la cadena de búsqueda ocurre en la posición *p* en la cadena de los pares, y la parte impar en la misma posición *p* en la cadena de los impares. Si la subcadena debe ser encontrada en la posición impar 2*p*+1 en el texto, entonces la parte impar de la cadena de búsqueda comienza en la posición *p* en la cadena de bits pares, y la parte par de la cadena de búsqueda aparecerá en la posición *p*+1 en la cadena de los impares.

C. Ejemplo

Asumiendo que tenemos que encontrar "100010" en la cadena de bits de la Fig. 3. Separamos la cadena en partes pares e impares para obtener "101" y "000" respectivamente. Según el índice, "101" no aparece en el sitio A y tampoco "000" en el sitio B. Al revés, el "000" aparece en la subcadena par (sitio A) en las posiciones 1, 7, 8, 9, 10, 11, y 18 y el "101" en las posiciones 4, 6, 10, y 14. Entonces solamente las posiciones 7 en el sitio A y 6 en el sitio B y 11 en A y 10 en B cumplen la condición. Por lo tanto, la cadena de búsqueda se encuentra en las posiciones 13 y 21, como se puede verificar en Fig. 3.

IV. EVALUACIÓN

Para nuestras evaluaciones elegimos dos novelas, Moby Dick de Herman Melville y una traducción en Inglés de Los Hermanos Karamazov de Fyodor Dostoyevsky.

A. Precisión

Primero medimos la precisión de las búsquedas. La precisión se define como la proporción de las posiciones actuales de una cadena de búsqueda sobre el número de posiciones reportadas. Una precisión de 1 es por lo tanto ideal: cada ocurrencia reportada es una ocurrencia verdadera. Tratamos las líneas de las novelas como un registro. Generamos una lista de 1000 palabras aleatorias en la novela y comparamos el número de ocurrencias verdaderas con el número de ocurrencias en el texto comprimido para obtener la proporción de veces que el registro que supuestamente tiene un acierto, realmente tiene uno. El resultado es dado por la Fig. 4. Básicamente, si el tamaño de la palabra es mayor a 10, entonces la precisión es razonablemente alta, y se convierte en 1 si la palabra tiene más de 20 letras.

Investigamos entonces la precisión en una cadena de bits, resultante de la dispersión. Esta vez, seleccionamos al azar 1000 subcadenas del texto. Calculamos entonces cuatro valores de precisión, uno para cada una de las cadenas de bits y otra para el texto original comprimido. La Fig. 5 muestra los resultados. Primero, observamos que la precisión de búsqueda en las cadenas de bits se comporta como se esperaba, resultando en curvas de forma S. Para tamaños pequeños de las cadenas de búsqueda, una coincidencia aleatoria es bastante rara. Segundo, podemos observar que el comportamiento no es exactamente idéntico, la cadena en el último sitio tiene casi siempre menos precisión. Esto indica que el flujo de trozos es diferente a una cadena de bits aleatorios a un nivel estadísticamente significativa.

B. Aleatoriedad aparente

Una primer prueba de aleatoriedad son las distribuciones de ceros y unos en los flujos de trozos. Como muestra la Tabla II, los números de unos para "Moby Dick" es menor a la esperada. Esto es una consecuencia de que la frecuencia de letras en los dos corpus no es la misma que la dada en Lee [13]. Desarrollamos una alternativa para almacenar el corpus de Melville, pero la distribución no se pudo realizar por los espacios en blanco que correspondían a más de un octavo de

todas las letras. En sí mismo, que la distribución de ceros y unos se aleje un poco de la distribución ideal 50-50 en una cadena no es un argumento en contra para considerarla como esencialmente aleatoria. Mucho más importante es la posibilidad de predecir el próximo bit dado una secuencia de varios bits. Si medimos la frecuencia de bloques de 2 bits (Tabla III), podemos observar que la distribución es significativamente diferente de la prevista asumiendo la independencia y observando la proporción de ceros y unos. Para el texto de Melville, los valores χ^2 son 858 para el sitio 0, 3518 para el sitio1, y 551 para el sitio 2, para el texto de Dostoyevsky, los valores χ^2 son 4775, 7331, y 852, y por ende, las diferencias entre los valores teóricos y observados son estadísticamente significativas.

Si observamos las cadenas de bits pares/impares, podemos ver que la aleatoriedad aparente no mejora al hacer la división. La Tabla IV claramente muestra esto usando bloques de dos bits. Los valores de χ^2 aumentan a 4578.31, 2710.58, 10746.2 para las cadenas de bits pares y 4147.98, 2029.08, 11063.8 para las cadenas de bits impares.

Nuestras observaciones confirman que todavía queda información sobre el texto original en los sitios, aún después

TABLA II
FRECUENCIA DE UNOS EN LAS CADENAS DE BITS

Sitio	Frecuencia Melville	Frecuencia Dostoyevsky
0	48.19%	49.80%
1	49.10%	50.28%
2	48.68%	50.48%

TABLA III
FRECUENCIA DE BLOQUES DE 2 BITS EN LAS CADENAS DE BITS

Dostoyevsky

Bloque	Sitio 0		Sitio 1		Sitio 2	
	obser.	teórico	obser.	teórico	obser.	teórico
00	26.89%	25.20%	23.91%	24.72%	24.93%	24.52%
01	24.24%	25.00%	26.73%	25.00%	25.50%	25.00%
10	23.47%	25.00%	26.18%	25.00%	24.90%	25.00%
11	25.41%	24.80%	23.18%	25.28%	24.66%	25.48%

Melville

Bloque	Sitio 0		Sitio 1		Sitio 2	
	obser.	teórico	obser.	teórico	obser.	teórico
00	27.79%	26.84%	25.29%	25.91%	27.10%	26.34%
01	25.04%	24.97%	26.61%	24.99%	25.23%	24.98%
10	24.00%	24.97%	25.85%	24.99%	24.52%	24.98%
11	23.17%	23.22%	22.24%	24.11%	23.15%	23.70%

TABLA IV
FRECUENCIA DE BLOQUES DE DOS BITS EN LAS CADENAS DE BITS PARES / IMPARES EN EL CORPUS DE MELVILLE

Par

Bloque	Sitio 0		Sitio 1		Sitio 2	
	obser.	teórico	obser.	teórico	obser.	teórico
00	26.06%	26.92%	25.95%	25.87%	24.48%	26.51%
01	27.64%	24.96%	26.68%	24.99%	28.60%	24.97%
10	25.79%	24.96%	25.41%	24.99%	27.62%	24.97%
11	20.51%	23.25%	21.97%	24.14%	19.29%	23.53%

Impar

Bloque	Sitio 0		Sitio 1		Sitio 2	
	obser.	teórico	obser.	teórico	obser.	teórico
00	25.89%	26.76%	25.96%	25.94%	23.85%	26.15%
01	27.38%	24.97%	26.47%	24.99%	28.57%	24.99%
10	26.16%	24.97%	25.32%	24.99%	28.05%	24.99%
11	20.58%	23.30%	22.25%	24.08%	19.52%	23.87%

de la división de las cadenas en partes pares e impares. La reconstrucción de un texto desconocido por un adversario que haya logrado obtener las cadenas de bits de los registros de un solo sitio es imposible porque las cadenas robadas no contienen suficiente información, pero aún existe suficiente información para que pueda sacar conclusiones si el texto original es parcialmente conocido.

V. TRABAJO RELACIONADO

Rabin propuso en 1989 distribuir un archivo entre varios nodos de un sistema distribuido (Information Dispersal Algorithm – IDA) [18] para mejorar su disponibilidad, pero de hecho, IDA es una versión de un código de corrección de errores [17]. Krawczyk fue el primero en considerar el problema de validar los trozos en que un archivo es dividido [12]. El problema esta relacionado con la compartición de secretos [4], [23]. Después aparecieron numerosos métodos para verificar datos remotos [1], [7], [20], [22]. Como IDA está basado en una transformación lineal, es posible buscar en trozos de IDA [19]. Desafortunadamente, IDA y otros esquemas similares cifran los trozos o no proveen ninguna seguridad.

La búsqueda en datos cifrados es reconocida como un problema importante. A pesar de los importantes avances recientes sobre la criptografía completamente homomórfica [8], [9], [25], estamos lejos de poseer un esquema útil en la práctica.

Song *et al.* propusieron cifrar cada palabra de un texto independientemente para permitir búsquedas por palabras claves [26] y un número de autores han propuesto mejoras para búsquedas por palabras clave. Goh utiliza un filtrado Bloom [11] y Chang y Mitzenmacher un índice similar que es seguro según una definición más fuerte. Innovaciones más recientes son las de Boneh *et al.* utilizando criptografía asimétrica [5], Bellare *et al.* [3], y v. Liesdonk *et al*. [27] donde el mayor esfuerzo está en la prevención del análisis de tráfico. Aparentemente, solamente Schwarz *et al.* han tratado de permitir búsquedas por cualquier cadena [21].

La mayor diferencia entre estos logros y el método de dispersión es el grado de seguridad que brindan. La dispersión gana seguridad por la imposibilidad estadística de reconstruir datos completos de los trozos almacenados en un solo sitio y ese grado de seguridad sencillamente no es suficiente para muchas aplicaciones. Pero los costos de utilizar criptografía son altos si solamente pensamos en los problemas de manejo de claves durante largos períodos de tiempo y hay muchas casos donde solamente se necesita un nivel de seguridad “confidencial”. Para estos casos, nuestra propuesta sirve.

VI. TRABAJO FUTURO

El mayor inconveniente que tiene este trabajo es la falta de evaluación sobre otros dominios de datos tales como el audio. Por ejemplo, la vigilancia anti-mafia de la policía puede generar cientos de horas de datos de llamadas telefónicas, la

cual debe ser procesada apenas aparecen nuevas pistas. Una búsqueda típica podría ser por todas las menciones del apodo de un mafioso en las llamadas intervenidas durante un cierto período de tiempo. Este ejemplo muestra la necesidad de tener capacidades adicionales de búsqueda, dado que representar la llamada telefónica como una cadena de fonemas no sería muy preciso.

Una fácil adaptación de nuestro método a otro dominio de datos requiere la generación automática del esquema de compresión con pérdida y la asignación de números de baldes que minimiza las diferencias entre un trozo de secuencia y una cada aleatoria de bits. El esquema necesita generar trozos de secuencia que no puedan ser usados para reconstruir las secuencias originales de símbolos (los fonemas en nuestro ejemplo).

VII. CONCLUSIONES

Presentamos un método para dispersar texto en varios trozos de secuencias de una manera tal que permite realizar búsquedas arbitrarias en el texto dispersado. Nuestro método es sencillo y permite búsquedas arbitrarias. Sus principales inconvenientes son una seguridad baja y la presencia de falsos positivos que casi desaparece cuando las cadenas buscadas tienen más de 20 caracteres. El esquema puede extenderse a otros dominios de datos. Cuando se aplica nuestro método a texto escrito, está listo para el uso en situaciones donde se necesita solamente un nivel básico de la confidencialidad para datos archivados. Un contexto posible sería el almacenamiento de correos electrónicos archivados para cumplimiento de normas donde en el futuro una acción legal podría requerir encontrar ciertos mensajes archivados.

REFERENCIAS

[1] G. Ateniese, R. Burns, R. Curtmola, J. Herring, O. Khan, L. Kissner, Z. Peterson, and D. Song, “Remote data checking using provable data possession,” ACM Transactions on Information and System Security (TISSEC), vol. 14, no. 1, p. 12, 2011.

[2] T. Bell, I. H. Witten, and J. G. Cleary, “Modeling for text compression,” ACM Computing Surveys (CSUR), vol. 21, no. 4, pp. 557–591, 1989.

[3] M. Bellare, A. Boldyreva, and A. ONeill, “Deterministic and efficiently searchable encryption,” in Advances in Cryptology-CRYPTO 2007. Springer, 2007, pp. 535–552.

[4] J. Benaloh and J. Leichter, “Generalized secret sharing and monotone functions,” in Proceedings on Advances in cryptology. Springer-Verlag New York, Inc., 1990, pp. 27–35.

[5] D. Boneh, E. Kushilevitz, R. Ostrovsky, and W. E. Skeith III, “Public key encryption that allows PIR queries,” in Advances in Cryptology-CRYPTO 2007. Springer, 2007, pp. 50–67.

[6] Y.-C. Chang and M. Mitzenmacher, “Privacy preserving keyword searches on remote encrypted data,” in Applied Cryptography and Network Security. Springer, 2005, pp. 442–455.

[7] R. Curtmola, O. Khan, R. Burns, and G. Ateniese, “MR-PDP: Multiple replica provable data possession,” in Distributed Computing Systems, 2008. ICDCS’08. The 28th International Conference on. IEEE, 2008, pp. 411–420.

[8] C. Fontaine and F. Galand, “A survey of homomorphic encryption for nonspecialists,” EURASIP J. Inf. Secur., vol. 2007, pp. 15:1–15:15, January 2007.

[9] C. Gentry, “Fully homomorphic encryption using ideal lattices,” in Proceedings of the 41st annual ACM symposium on Theory of computing, ser. STOC ’09, 2009, pp. 169–178.

[10] C. Gentry and S. Halevi, “Implementing gentry’s fully-homomorphic encryption scheme,” in Proceedings of the 30th Annual international conference on Theory and applications of cryptographic techniques: advances in cryptology, ser. EUROCRYPT’11, 2011, pp. 129–148.

[11] E.-J. Goh et al., “Secure indexes.” IACR Cryptology ePrint Archive, vol. 2003, p. 216, 2003.

[12] H. Krawczyk, “Distributed fingerprints and secure information dispersal,” in Proceedings of the Twelfth Annual ACM Symposium on Principles of Distributed Computing, ser. PODC ’93, 1993, pp. 207–218.

[13] E. S. Lee. (1999) Essays about computer security. Centre for Communications Systems Research Cambridge, Cambridge. [Online]. Available: http://www.proselex.net/Documents/Essaay about Computer Security.pdf

[14] W. Litwin, R. Mokadem, P. Rigaux, and T. Schwarz, “Fast ngram based string search over data encoded using algebraic signatures,” in Proceedings of the 33rd international conference on Very large data bases. VLDB Endowment, 2007, pp. 207–218.

[15] M. Mahoney. Large text compression benchmark. Accessed: 2014-06-19. [Online]. Available: http://mattmahoney.net/dc/text.html

[16] C. D. Manning, P. Raghavan, and H. Schutze, ¨ Introduction to Information Retrieval. Cambridge University Press, 2008.

[17] F. Preparata, “Holographic dispersal and recovery of information,” Information Theory, IEEE Transactions on, vol. 35, no. 5, pp. 1123–1124, 1989.

[18] M. O. Rabin, “Efficient dispersal of information for security, load balancing, and fault tolerance,” Journal of the ACM (JACM), vol. 36, no. 2, pp. 335–348, 1989.

[19] ——, “The information dispersal algorithm and its applications,” in Sequences. Springer, 1990, pp. 406–419.

[20] T. Schwarz and E. L. Miller, “Store, forget, and check: Using algebraic signatures to check remotely administered storage,” in Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on. IEEE, 2006, pp. 12–12.

[21] T. Schwarz, P. Tsui, and W. Litwin, “An encrypted, content searchable scalable distributed data structure,” in Proceedings of the 22nd International Conference on Data Engineering Workshops (ICDEW ’06), 2006, p. 18.

[22] M. A. Shah, M. Baker, J. C. Mogul, R. Swaminathan et al., “Auditing to keep online storage services honest.” in Proceedings, Eight workshop on Hot Topics in Operating Systems (HotOS), 2007.

[23] A. Shamir, “How to share a secret,” Communications of the ACM, vol. 22, no. 11, pp. 612–613, 1979.

[24] C. E. Shannon, “Prediction and entropy of printed english,” Bell system technical journal, vol. 30, no. 1, pp. 50–64, 1951.

[25] D. X. Song, D. Wagner, and A. Perrig. "Practical techniques for searches on encrypted data." In Security and Privacy, 2000.. Proceedings. 2000 IEEE Symposium on, pp. 44-55.

[26] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, “Fully homomorphic encryption over the integers,” in Proceedings of the 29st Annual international conference on Theory and applications of cryptographic techniques: advances in cryptology, 2010.

[27] P. van Liesdonk, S. Sedghi, J. Doumen, P. Hartel, and W. Jonker,“Computationally efficient searchable symmetric encryption,” in Secure Data Management. Springer, 2010, pp. 87–100.

Carlos Aiello Montandon es graduado en Ingeniería Informática por la Universidad Católica del Uruguay. Actualmente trabaja como Ingeniero de Software en Montevideo, Uruguay.

Luis Vidal Introini es graduado en Ingeniería Informática por la Universidad Católica del Uruguay. Actualmente trabaja como Ingeniero de Software en Montevideo, Uruguay.

Thomas Schwarz, SJ es doctor de matemáticas (FernUniversität Hagen) y de informática (UC San Diego). Es miembro de Storage Systems Research Center de la Universidad de California, Santa Cruz.