# Computing Range Queries on Obfuscated Data

**E. Damiani**[1] **S. De Capitani di Vimercati**[1] **S. Paraboschi**[2] **P. Samarati**[1]

(1) Dip. di Tecnologie dell'Infomazione
Università di Milano
26013 Crema - Italy
{damiani,decapita,samarati}@dti.unimi.it

(2) Dip. di Ing. Gestionale e dell'Informazione
Università di Bergamo
24044 Dalmine - Italy
parabosc@unibg.it

## Abstract

Data obfuscation techniques transforms data into other data that are harder to understand. These techniques are receiving an increasing amount of attention, largely due to their applications in different areas. We illustrate an approach for obfuscating data that guarantees protection of data while allowing the execution of both equality and range queries on the obfuscated data.

**Keywords:** Data obfuscation, range query, shift register

## 1 Introduction

Data obfuscation techniques are receiving an increasing amount of attention, largely due to their applications to e-business [7]. While conventional encryption refers to modifications where data cannot be easily decrypted without the proper key, obfuscation corresponds to a looser notion of making data unusable to some extent. Therefore, while all encrypted data is also obfuscated, the converse is not true. Of course, one might wonder why merely obfuscate data when strong encryption algorithms are available. An example could be a report of a government employment agency. It is not unusual that such a report is stored in the form of a database or a worksheet on an independent Web server open to researchers looking for employment data patterns. In this case, the employment agency policy may require that names, Social Security Numbers and other personal information are not disclosed to the site, for example because they are only made available to the researchers on a need-to-know basis.[1] On the other hand, researchers must be able to inspect, display and, to some extent, query the data held by the Web site to carry out their work [1]. A common solution to this problem is obfuscating the data before releasing them to the Web site by applying a suitable *obfuscation function* like, for example, a substitution cipher such as RC4 or DES. While RC4 or DES are not considered strong encryption nowadays, they may well suffice for this kind of application. Substitution-based obfuscation has the additional advantage that it keeps the obfuscated data bytes inside the UTF character set, keeping the encrypted data displayable on the client machines. Alternatively, sensitive data can be obfuscated by partial masking, which is useful in situations where it is only necessary to display/disclose a portion of a data field.

### 1.1 Order-preserving function

One of the main problems in data obfuscation is the widespread use of order-preserving obfuscation functions on ordered data domains. More precisely, a function $f : D \rightarrow C$ is said to be *order-preserving* whenever $\forall a, b \in D : a < b \Rightarrow f(a) < f(b)$ [8]. The main justifica-

[1]In other words, when a researcher really needs access to the real name and SSN of a party involved (e.g., because she suspects fraud or violation of the employment regulations) she will have to contact an authorized representative of the data owner.

tion for order-preserving data obfuscation is that it allows easy execution of *range queries*. Here, by range queries we mean queries selecting obfuscated data whose plaintext values lie on a given range defined by endpoints belonging to the data domain, but not necessarily present among the obfuscated data.

Order-preserving obfuscation supports efficient execution of range queries on the server side, but also makes very simple for the site holding the obfuscated data and/or for an eavesdropper to reconstruct the plaintext values. On the other hand, it is not easy to execute range queries on obfuscated data if a non-order preserving obfuscation function is used, especially if all the servers holding the data are based on relational databases [3, 4]. In this paper, we shall introduce a technique for executing range queries on obfuscated data. Our approach is aimed at making range queries completely undistinguishable from standard equality-based ones on the part of the server.

## 2 A computational model for range queries on obfuscated data

In this section we give the basic assumptions on which we base our proposal and introduce some notations.

### 2.1 Basic concepts and scenario

Let $D$ be a data domain equipped with a total order denoted $<$. Let $P \subseteq D$ be a subset of the data domain $D$. For instance, if $D$ coincides with the set of letters, $P$ can be any subset of letters (e.g., $P$ can be equal to {a,b,c,d}). In the following, we use $a$ to denote the minimum element in $P$, that is, $a \in P$ and $\forall x \in P, a < x$.

We are interested in a non-order-preserving bijective function $g : P \rightarrow E$ such that $\exists f_g : E \rightarrow 2^E : f_g(e) = \{y \mid g^{-1}(y) < g^{-1}(e)\}$. We will incorporate the values of $g$ into obfuscated data, and later use them to execute range queries. Since $g$ will be used as a part of an obfuscation function, we would like $f_g$ to be reasonably hard to compute without knowing $g$, even if $E$ is known. Our solution relies on defining a computational model capable of computing $g$ so that pseudo-random values are associated with consecutive domain values, while allowing easy computation of $f_g$.

Figure 1 summarizes the basic scenario we consider [4]. At the client-side, a user can submit a query which is mapped onto a query on the obfuscated data stored at the server-side. The transformed query is executed at the server-side and the result is returned to the client. At the client-side the result is decrypted and the data are returned to the user. The goal of our approach is to obfuscate data in order to efficiently compute equality and range queries at the server-side.

### 2.2 Our proposal

To fix our ideas, let us consider a specific computational model: a simple shift register of length $n$, where $n \geq \mid P \mid$.[2] Our claim is that we shall use this model to produce obfuscation bits to be added to the obfuscated data values. In its simplest form, our obfuscation function $g : D \rightarrow E$ is defined by a pair $(s, d)$ such that $s \in E$ and:

$$g(x) = \begin{cases} s & \text{if } x = a; \\ s >>_{|x-a| \cdot d} & \text{if } x > a \end{cases}$$

where $>>$ is the bitwise shift operator on the shift register's content. Once the data containing the obfuscation bits have been published, all clients knowing $s$ and $d$ will be able to execute range queries, while the server will see a sequence of unrelated equality queries. In order to clarify our definition, suppose that this time our plaintext dataset is $P = \{a, b, d, e\}$ (whose values are totally ordered but not consecutive). Let the start value of the register be $s = 00101100 = 0x2e$ and, for the sake of simplicity, $d = 1$. We get the following obfuscation bits:

$$g(a) = 2e; g(b) = 16; g(d) = 0b; g(e) = 85$$

Note that obfuscation bits can be hidden in any non-order preserving obfuscation function

[2]For all domain X, $\mid X \mid$ denotes the cardinality of X.

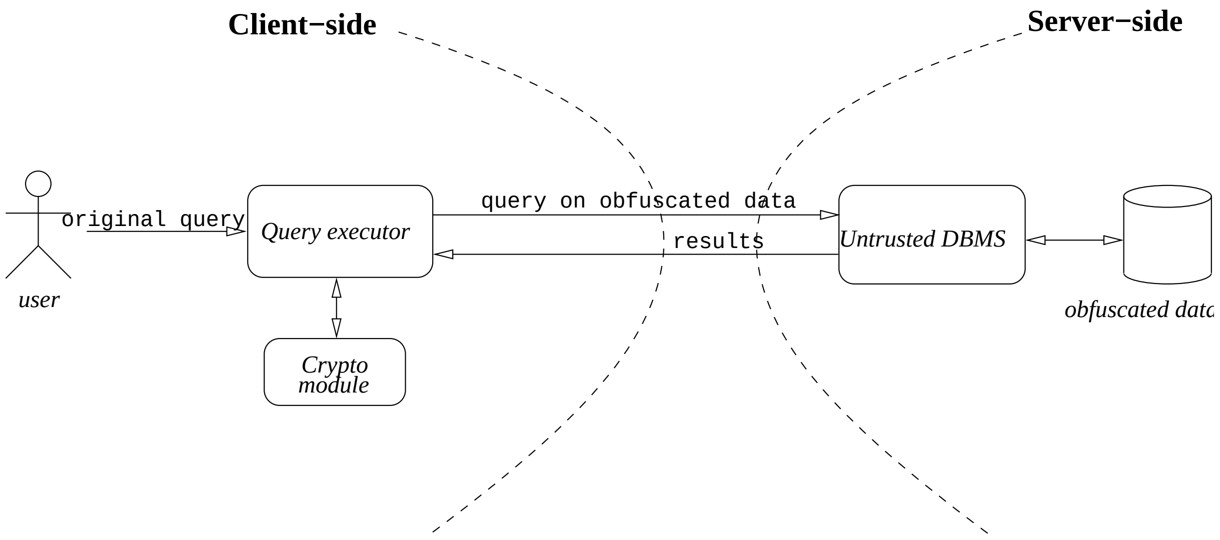

Figure 1: Scenario

like the ones mentioned in Section 1. This would simply require an additional secret to be shared between the original data owner and the client about where the obfuscation bits are located within the obfuscated data.

## 3 Range Query Algorithm

We are now ready to describe the method that can be used to evaluate range queries on obfuscated data. Suppose therefore that a client needs to select all data $x$ such as $x > a$ where $a \in D$ is a value of the data domain, not necessarily stored in the remote database. The remote database holds a set of values whose obfuscation bits are $\{g(y_1)...g(y_k)\}$. The query execution can be summarized as follows.

**Step 1.** Find a value $t$ such that $t \in P$ and $t$ is the element of $P$ closest to $a$ from above (according to the natural order relation in $D$).

**Step 2.** Use $g$ to compute $g(x)$ for all elements $x \in P$ and such that $x > t$.

**Step 3.** Send an equality query selecting each $g(x)$ to the server holding the data.

While Steps 2 and 3 are rather straightforward, an additional word of explanation may be needed for Step 1. The idea is simply to

| Plaintext data P | Obfuscation bits |
|---|---|
| a | 1101 |
| b | 1110 |
| d | 0111 |
| e | 1011 |

Figure 3: An example of plaintext data and the corresponding obfuscation bits

exploit the shift register and the definition of $g$ to generate a sequence of equality queries whose cumulative effect is executing a binary search among the obfuscated data. Figure 2 illustrates the binary search algorithm.

### 3.1 A Worked-out example

As an example, consider the plaintext dataset in Figure 3 together with the corresponding obfuscation bits. Here, we assume that $n = 4$, $s = 1101$, and $d = 1$. Suppose now that we want to find all plaintext values $x$ such that $x > c$. According to the algorithm described in the previous section, we first need to find a value $t \in P$ such that $t \geq c$ and $\nexists y \in P : t \geq y \geq c$.

**Step 1.**

Shift $s$ of $\lfloor \frac{n}{2} \rfloor = 2$ positions rightwards: $u = s >>_2 = 0111$

Decrypt $u$ and obtain $g^{-1}(u) = d$

Shift $u$ leftward of $\lfloor \frac{n}{4} \rfloor = 1$position:

**Algorithm 1** *Binary Search*

```
/* Input:   A value $a \in D$ */
/* Output: A value $y \in P : y \geq a$ and $\nexists y' \in P : y \geq y' \geq a$ */

1. i :=2; u:=‘ ’;
2. Shift $s$ of $\lfloor \frac{n}{2} \rfloor$ positions rightwards to obtain the obfuscation bits $u =>>_{\lfloor \frac{n}{2} \rfloor} s$
3. Repeat
     3.1 Send to the server an equality query to get the obfuscated data item corresponding to $u$
     3.2. Decrypt the obfuscated dat. Let $y$ be the result
     3.3 If($y > a$)
           then shift $u$ leftwards of $n = \lfloor \frac{n}{2^i} \rfloor$ bits;
           else If($y < a$)
                  then shift $u$ rightwards of $n = \lfloor \frac{n}{2^i} \rfloor$ bits;
     3.4 i:= i + 1;
   until($\lfloor n \rfloor \leq 0$ or $y = a$)
```

Figure 2: Binary search algorithm

$u = u <<_1 = 1110$
Decrypt $u$ and obtain $g^{-1}(u) = b$

At this point we know that $d$ is the lowest element of $P$ greater than $c$.

**Step 2&3.**
Send an equality query to the server in order to retrieve all dataset whose obfuscation bits are equal to 0111 or 1011.

## 4 Conclusions

In this paper we have proposed a simple and effective technique for obfuscating data while allowing the efficient execution of range queries. Each range query is translated into a sequence of equality queries composed of two part: the first, which is logarithmic in the number of database entries, locates the database entry which is closest to the range query limit; the second, which is linear in the number of the database entries actually lying in the desired range, collects one by one all the elements of the query answer. The server cannot distinguish the queries belonging to the sequence from other equality queries it may receive, making statistical reconstruction of the plaitext value more difficult. We plan to develop this topic in a future paper evaluating the degree of strength of our technique with respect to unauthorized inferences, comparing it to server side techniques like [4].

**Acknowledgements**

The work reported in this paper has been partially supported by the Italian MURST within the KIWI and MAPS projects.

## References

[1] R. Agrawal. Privacy cognizant information systems, October 2003. http://www.acm.org/sigs/sigsac/ccs/ /CCS2003/keynote.html.

[2] R. Agrawal, A. Evfimievski, and R. Srikant. Information sharing across private databases. In *Proc. of the International Conference on Management of Data*, San Diego, California, June 2003.

[3] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. Implementation of a storage mechanism for untrusted dbmss. In *Proc. of the Second International IEEE Security in Storage Workshop*, Washington, DC, USA, October 2003.

[4] E. Damiani, S. De Capitani di Vimercati, S. Jajodia, S. Paraboschi, and P. Sama-

rati. Balancing confidentiality and efficiency in untrusted relational dbmss. In *Proc. of the 10th ACM Conference on Computer and Communications Security*, Washington, DC, USA, October 2003.

[5] H. Hacigümüs, B. Iyer, C. Li, and S. Mehrotra. Executing SQL over encrypted data in the database-service-provider model. In *Proc. of the ACM SIGMOD'2002*, Madison, Wisconsin, USA, June 2002.

[6] H. Hacigümüs, B. Iyer, C. Li, and S. Mehrotra. Providing database as a service. In *Proc. of the 18th International Conference on Data Engineering*, San Jose, California, USA, February 2002.

[7] R. Khosla, E. Damiani, and W.I. Grosky. *Human-Centered E-Business.* Kluwer Academic Publishers, 2003.

[8] B. Schneier. *Applied Cryptography.* Wiley & Sons, 1996.

[9] E.Y. Yang, J. Xu, and K.H. Bennett. Private information retrieval in the presence of malicious failures. In *Proc. of the 26th Annual International Computer Software and Applications Conference*, Oxford, England, August 2002.