DOI: 10.1109/ICDEW.2013.6547434

On Information Leakage by Indexes over Data Fragments

Sabrina De Capitani di Vimercati [1], Sara Foresti [1], Sushil Jajodia [2], Stefano Paraboschi [3], Pierangela Samarati [1]

[1] *DI - Università degli Studi di Milano, 26013 Crema - Italy*
firstname.lastname@unimi.it

[2] *CSIS - George Mason University, Fairfax, VA 22030-4444 - USA*
jajodia@gmu.edu

[3] *Università degli Studi di Bergamo, 24044 Dalmine - Italy*
parabosc@unibg.it

***Abstract*—Data fragmentation has recently emerged as a complementary approach to encryption for protecting confidentiality of sensitive associations when storing data at external parties. In this paper, we discuss how the use of indexes, typically associated with the encrypted portion of the data, while desirable for providing effectiveness and efficiency in query execution, can - combined with fragmentation - cause potential leakage of confidential (encrypted or fragmented) information. We illustrate how the exposure to leakage varies depending on the kind of indexes. Such observations can result useful for the design of approaches assessing information exposure and for the definition of safe (free from inferences) indexes in fragmented data.**

I. Introduction

Cloud computing represents today a successful paradigm for delegating data storage and management to external services. To protect data confidentiality from the storing/processing servers themselves, which can be *honest but curious*, data are typically encrypted – so to make them non-intelligible to the storing servers – and queries are executed on associated indexes [1], [2]. Recent approaches have proposed the use of data fragmentation, possibly combined with encryption, to protect confidentiality when what is sensitive is the data association, in contrast to the specific data values [3], [4], [5].

Fragmentation allows the storage of plaintext values at the server side thus providing more convenience in terms of data accessibility and query performance, since the server can evaluate selection conditions on plaintext attributes in a precise way. However, when operating on a fragment, the server is not able to evaluate conditions on attributes not appearing in the clear. These conditions would then need to be evaluated by the client, who can access the whole dataset, with potentially high cost in terms of communication and client-side computation. To avoid such a limitation and effectively exploit fragmentation, it is important to couple fragments with indexes on the encrypted portion of the data enabling some server-side evaluation of conditions on attributes not appearing in the clear [6]. In this paper, we discuss the combined use of fragmentation and indexes and show how it may cause leakage of confidential information, otherwise protected by encryption or fragmentation. The contribution of our paper is on pointing out such information leakage issues, together with observations on strengths and weaknesses of different indexing strategies. Our paper represents a first step for the definition of safe indexes (i.e., free from inferences) for data fragments, and of metrics assessing the risk of information leakage to which indexed fragments are exposed. The analysis we present makes the assumption that the adversary is not monitoring the queries and only has access to the static representation of the data. This assumption is consistent with a scenario where users are worried by adversaries accessing the data stored on the server (e.g., due to authentication errors or the use of side channels that access the data at the physical level), rather than by adversaries with long-term control over the server processing the queries.

The remainder of this paper is organized as follows. Section II presents the basic concepts on fragmentation and indexing techniques. Section III describes inference exposure caused by the introduction of indexes in fragments. Section IV discusses the properties an index should have to limit exposure to inference. Section V illustrates related work. Finally, Section VI presents our conclusions.

II. Fragments and Indexes

Consistently with existing proposals, we consider the outsourced data to be represented as a single relational table and *confidentiality constraints* to express attributes, or combinations thereof, that are considered sensitive. Fragmentation avoids encrypting attributes when their values are not sensitive, allowing instead the protection of sensitive associations separating the involved attributes by vertically splitting the relation into different fragments that cannot be joined [3], [4], [5]. To illustrate, consider the relation and the confidentiality constraints in Figure 1(a) and in Figure 1(b), respectively. While `SSN` values are sensitive and need to be encrypted for external storage, for all the other attributes what is sensitive is their association. Fragmentation $\mathcal{F}$={{`Name`,`State`}, {`Job`}, {`Disease`}} protects these sensitive associations allowing the plaintext representation of the four attributes in three different fragments. At the physical level [4], fragments are composed of tuples, each reporting a salt (used in the encryption of the tuple), the encrypted tuple (including all the

PATIENTS

SSN	Name	State	Job	Disease
123456789	Adams	VA	Nurse	Flu
234567891	Brown	MN	Nurse	Flu
345678912	Cooper	CA	Teacher	Flu
456789123	Davis	VA	Lawyer	Diabetes
567891234	Eden	NY	Clerk	Diabetes
678912345	Falk	CA	Teacher	Gastritis
789123456	Green	NY	Doctor	Arthritis
891234567	Hack	NY	Nurse	Arthritis

(a)

c_0 = {SSN}
c_1 = {Name, Disease}
c_2 = {Job, Disease}
c_3 = {Name, Job}

(b)

F_1^e

salt	et	Name	State
s_{11}	et_{11}	Adams	VA
s_{12}	et_{12}	Brown	MN
s_{13}	et_{13}	Cooper	CA
s_{14}	et_{14}	Davis	VA
s_{15}	et_{15}	Eden	NY
s_{16}	et_{16}	Falk	CA
s_{17}	et_{17}	Green	NY
s_{18}	et_{18}	Hack	NY

(c)

F_2^e

salt	et	Job
s_{21}	et_{21}	Nurse
s_{22}	et_{22}	Nurse
s_{23}	et_{23}	Teacher
s_{24}	et_{24}	Lawyer
s_{25}	et_{25}	Clerk
s_{26}	et_{26}	Teacher
s_{27}	et_{27}	Doctor
s_{28}	et_{28}	Nurse

(d)

F_3^e

salt	et	Disease
s_{31}	et_{31}	Flu
s_{32}	et_{32}	Flu
s_{33}	et_{33}	Flu
s_{34}	et_{34}	Diabetes
s_{35}	et_{35}	Diabetes
s_{36}	et_{36}	Gastritis
s_{37}	et_{37}	Arthritis
s_{38}	et_{38}	Arthritis

(e)

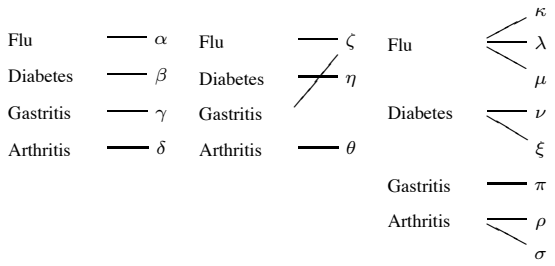
Fig. 1. Plaintext relation (a), confidentiality constraints (b), and fragments (c-e)

attributes in the original relation that are not represented in the clear), and the plaintext values of the attributes in the fragment (Figures 1(c-e)). For clarity, in our examples we always list tuples in the order in which they appear in the original relation (in practice, a different order will be used in each fragment to break the correspondence among their tuples).

Fragmentation enables the precise execution of selection conditions over plaintext attributes by the server, but results limited when conditions involve also attributes that are encrypted in the fragment because they are sensitive (by themselves or in association with the ones already present in the clear). In fact, fragments cannot be joined and each query operates on one fragment only, as each fragment stores (either in encrypted or plaintext form) all the attributes of the original relation. For instance, the server can return all the tuples of patients living in Virginia by querying fragment F_1^e, or the tuples of patients suffering from Diabetes by querying fragment F_3^e. However, since attributes `State` and `Disease` do not appear in the clear in the same fragment and the server can neither decrypt nor join F_1^e and F_3^e, it cannot compute the tuples of patients suffering from Diabetes and living in Virginia. At least one of these conditions would need to be evaluated by the client. To avoid such a burden it is necessary to complement fragments with indexes for those attributes not appearing in plaintext, so to allow the server-side evaluation (even if not precise) of conditions over them, without disclosing sensitive values or associations. Indexes should however not permit the server to (loosely) join fragments as this would violate confidentiality constraints.

Intuitively, an index is a function ι that maps plaintext values to obfuscated values. In the data outsourcing scenario, different kinds of indexes have been proposed, which can be summarized in the following three classes.

- **Direct index (di)**: ι maps each plaintext value to one index value and viceversa. The index can be obtained by applying an encryption (unsalted) function on the plaintext values of the attribute (e.g., [7]).
- **Bucket index (bi)**: ι maps each plaintext value to one index value, with collisions (i.e., different plaintext values can be mapped to a same index value). The index can be obtained via a hash function with collisions (e.g., [7]) or by partitioning plaintext values and mapping all values in a partition to a same index value (e.g., [1]).
- **Flattened index (fi)**: ι maps each plaintext value to one or more index values in such a way that all index values have the same number of occurrences (flattening), but each index value represents one plaintext value. The index can be obtained by applying an encryption function over plaintext values that ensures such properties (e.g., [8]).



(a) **di** (b) **bi** (c) **fi**

Fig. 2. Index functions for attribute `Disease`

Figure 2 illustrates an example of index functions on attribute `Disease` in Figure 1(a), considering the different types of indexing.

III. INFORMATION LEAKAGE

We now discuss the possible information leakage caused by the introduction of indexes in fragments. For simplicity, in the discussion we consider a single indexed attribute to appear in a fragment, adding an index for `Disease` in fragment F_1^e (Figures 3(a-c)), and we investigate how the combination of the index with the plaintext information exposes sensitive association {`Name`, `Disease`}. This consideration is just for simplicity of exposition and is not to be considered a limitation, since our goal is to investigate the information leakage caused by the combined presence of plaintext values (due to fragmentation) and indexes. The observations naturally extend to the consideration of multiple indexes, while the exposure due to the combination of indexes by themselves has been investigated in outsourcing scenarios [7].

To examine the information leakage from indexed fragments, we consider two kinds of knowledge that a curious observer (e.g., the storing server or adversaries gaining access to the stored data) can exploit: *vertical knowledge* and *horizontal knowledge*, characterized as follows.

- **Vertical knowledge** (**vk**) is due to values appearing in the clear in one fragment and indexed in other fragments.

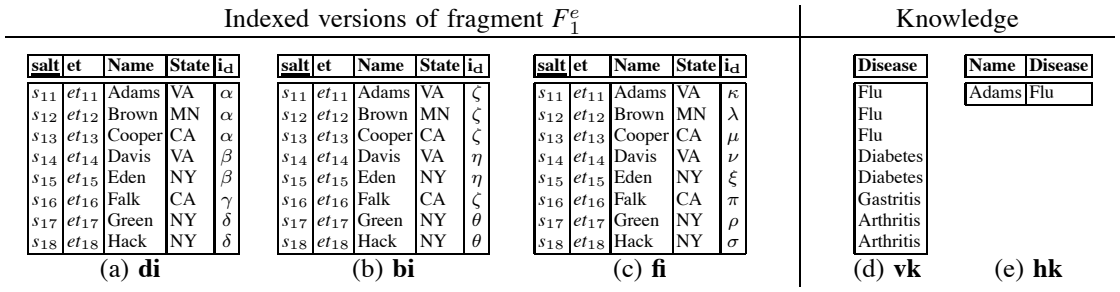


Fig. 3. Choices for indexed fragments (a-c) and additional knowledge (d,e)

For instance, fragment F_3^e in Figure 1(e) makes visible the plaintext values (and their number of occurrences) of attribute `Disease` (Figure 3(d)).

- **Horizontal knowledge** (**hk**) is due to possible external knowledge that an observer has with respect to the presence of specific tuples (corresponding to sensitive associations) in the table. In its simplest form, horizontal knowledge is represented by the knowledge of a single tuple. For instance, the observer can know that Adams suffers from Flu (Figure 3(e)). Note that the effectiveness of an attack based on horizontal knowledge is lower if the indexed fragment F_1^e includes multiple occurrences of the known value of the plaintext attribute (Adams, in our example) that are associated with different index values. Here, we consider the worst case scenario, assuming one occurrence of the known plaintext value.

We now examine the information leakage caused by the inclusion of index i_d for `Disease` in F_1^e, considering the different indexing approaches (Figures 3(a-c)) and additional knowledge (Figures 3(d-e)).

Direct index, vertical knowledge (di-vk). Sensitive associations are exposed depending on their distinguishability with respect to the number of occurrences of the index values. In the example (Figure 3(a)), the index values corresponding to Gastritis and Flu are recognizable being the only one with 1 and 3 occurrences, respectively. An observer can then easily infer that Falk has Gastritis and Adams, Brown, Cooper have Flu. As for the other 4 patients, the observer can estimate that they have one of the two other diseases with equal probability.

Direct index, horizontal knowledge (di-hk). By joining the knowledge of a tuple with the plaintext attribute (`Name`) appearing in the indexed fragment, the adversary can retrieve the index value corresponding to the specific plaintext value of the indexed attribute (Flu). This exposes all the associations having the same index value as the one the observer knows. In the example (Figure 3(a)), knowledge of the association (Adams, Flu) allows the observer to infer that ι(Flu)=α and then that also Brown and Cooper have Flu. Had the observer known instead association (Davis, Diabetes), she could have inferred that ι(Diabetes)=β and then that Eden has Diabetes.

Bucket index, vertical knowledge (bi-vk). Bucket indexes reduce the exposure of associations since different plaintext values may be represented by the same index value. However, values with a high number of occurrences (outliers) remain more exposed. In fact, the buckets to which they can correspond are only those whose size is at least the number of occurrences of the plaintext values. In the example (Figure 3(b)), the observer can easily infer that ι(Flu)=ζ since ζ is the only index value with at least 3 occurrences. She can then infer that 3 out of the 4 patients with i_d=ζ have Flu and 1 patient has Gastritis, which is the only plaintext value with 1 occurrence (i.e., each one has Flu with 0.75 probability and Gastritis with 0.25 probability).

Bucket index, horizontal knowledge (bi-hk). Like in the direct index case, the observer can recognize the index value representing the known plaintext value, with the only difference that the index value may correspond also to other plaintext values. The observer can then infer that some associations are not present in the database (tuples with a different index value will certainly not be associated with the known plaintext value). Together with vertical knowledge (that the storing server always has), it permits to infer the probability that some sensitive associations with the known plaintext value belong to the database. In the example (Figure 3(b)), knowledge of the association (Adams, Flu) allows the observer to infer that ι(Flu)=ζ. Since there are 3 occurrences of Flu and 4 occurrences of ζ, by removing the known tuple, the observer can infer that Brown, Cooper, and Falk have 0.66 probability of suffering from Flu. Had the observer known instead association (Davis, Diabetes), she could have inferred that ι(Diabetes)=η and, since there are 2 occurrences of Diabetes and 2 occurrences of η, Eden certainly has Diabetes.

Flattened index, vertical knowledge (fi-vk). Flattened indexes are not vulnerable to attacks exploiting vertical knowledge. Indeed, flattening the occurrences of the index values makes it impossible to establish correspondences between plaintext and index values based on the number of occurrences.

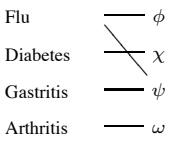
Flattened index, horizontal knowledge (fi-hk). Flattened indexes reduce the risk of exposure of associations since different occurrences of the same plaintext value are possibly mapped to different index values. The observer can then recognize one of the index values representing the known plaintext value (Flu) by joining the tuple she knows with the indexed fragment on the plaintext attribute (`Name`). This exposes all the associations having the same index value as the one the observer knows. Note that since a plaintext value can

be mapped to several index values, the exposed tuples could be a subset of the tuples with the same plaintext value as the one in the known tuple (Flu). In the example (Figure 3(c)), knowledge of the association (Adams, Flu) allows the observer to infer that ι(Flu)=κ but since each index value has only one occurrence, no other association is exposed.

IV. Discussion

As noted in the previous section, vertical knowledge enables the observer to exploit the frequency distribution of the index values to infer the corresponding plaintext values, retrieving them in a precise way or decreasing the uncertainty over them. Such an attack is clearly not possible when index values are equally distributed (flattened index). While blocking inferences from vertical knowledge, flattened indexes remain vulnerable to inferences from horizontal knowledge. Such inferences can however be mitigated when the index function generates collisions (bucket index). Hence, to provide protection against both vertical and horizontal knowledge, the *index function should both flatten the frequency distribution of index values and generate collisions*. To illustrate, Figure 4(a) reports an example of index function providing both flattening and collisions. The indexed fragment in Figure 4(b), which uses such an index, is protected against inferences from the vertical and horizontal knowledge previously discussed. In fact, vertical knowledge cannot be exploited since all index values look alike (each have 2 occurrences). The horizontal knowledge (Adams, Flu), while disclosing to an observer that ϕ is one of the index values representing Flu, does not allow her to determine what is the plaintext value corresponding to the other occurrence of ϕ.

As usual, protection comes at some cost (in this case, lower query performance and higher communication costs). The presence of collisions (confusing different plaintext values together under the same index value) necessarily implies less precision in the server's response to queries based on the index (with reference to Figure 4, a client search with condition `Disease`="Gastritis" would be transmitted to the server as a condition i_d=ψ, thus returning also a tuple referring to Flu). The client needs then to perform a query post-processing to filter out spurious tuples from the server's response. The split of a same plaintext value over different index values increases the index values that are to be retrieved. For instance, with reference to Figure 4, a client search with condition `Disease`="Flu" will imply requesting the server to retrieve all tuples with index value equal to ϕ or ψ. The collisions in the indexing will contribute to increase the size of the result. Splitting and collisions introduce an increase of the query execution cost which should be evaluated with care. The expectation is that a flattened index with collisions is in most cases a convenient solution, with the advantages in terms of leakage protection dominating the increased query execution costs. We speculate that the definition of a flattened index function with collisions can be based on an adaptation of perfect hash functions [9]. These functions enrich hashing with additional information and produce a flat distribution of



(a)

F_1^e

salt	et	Name	State	i_d
s_{11}	et_{11}	Adams	VA	ϕ
s_{12}	et_{12}	Brown	MN	ϕ
s_{13}	et_{13}	Cooper	CA	ψ
s_{14}	et_{14}	Davis	VA	χ
s_{15}	et_{15}	Eden	NY	χ
s_{16}	et_{16}	Falk	CA	ψ
s_{17}	et_{17}	Green	NY	ω
s_{18}	et_{18}	Hack	NY	ω

(b)

Fig. 4. Flattened index function with collisions (a) and indexed fragment F_1^e (b)

the index values on singleton buckets. They can be extended to equal-size non-singleton buckets, obtaining at the same time flattening and collisions (the precise adaptation of the function is an interesting research direction).

It is interesting to note the special case of *indexes over key attributes* (i.e., values having a single occurrence). In this case even *direct indexing provides protection*. In fact, though implying a one to one correspondence between plaintext and index values, it does not result vulnerable: being all equal to one, frequencies cannot be exploited, and since no value appears more than once, knowledge of the fact that a plaintext value corresponds to a given index value cannot be exploited.

We close this section with a note on dynamic observations and on the presence of multiple indexes. Dynamic observations refer to the knowledge that an observer can acquire from queries. For instance, the server can acquire information on the splitting by observing that certain index values are queried together. As an example, with respect to fragment F_1^e in Figure 4(b), a query looking for all patients suffering from Flu is translated into a query asking the tuples where i_d=ϕ or i_d=ψ, thus revealing that ι(Flu)=$\{\phi,\psi\}$. Protection against dynamic observations requires changing the index function at every query, with an approach similar to those employed to protect access confidentiality based on shuffling and dynamic memory allocation [10]. As for multiple indexes, we note that if several index attributes appear in the same fragment, the observations above on flattening and collisions directly apply, with the only difference that they should be referred to the combination of index values in a tuple (in contrast to individual index values). Other two cases of multiple indexing need to be considered, namely: two attributes appear one in plaintext and the other indexed in one fragment and reversed in another fragment, and the indexing of a same attribute in different fragments. Both cases do not introduce further vulnerabilities with respect to the single index case, assuming that the index provides flattening and collisions for non-key attributes (otherwise, the observer's linking capabilities would be enhanced by the presence of multiple indexes). With reference to the first case, the release of fragments F_2^e and F_3^e in Figure 5, each complemented with a flattened index with collision over the attribute appearing in the clear in the other fragment, does not cause additional information leakage than the release of F_2^e in Figure 5(a) and F_3^e in Figure 1(e). If the indexes are defined using a direct or bucket index function, the release of these

F_2^e

salt	et	Job	i_d
s_{21}	et_{21}	Nurse	α
s_{22}	et_{22}	Nurse	β
s_{23}	et_{23}	Teacher	α
s_{24}	et_{24}	Lawyer	γ
s_{25}	et_{25}	Clerk	γ
s_{26}	et_{26}	Teacher	β
s_{27}	et_{27}	Doctor	δ
s_{28}	et_{28}	Nurse	δ

(a)

F_3^e

salt	et	Disease	i_j
s_{31}	et_{31}	Flu	ϵ
s_{32}	et_{32}	Flu	θ
s_{33}	et_{33}	Flu	ζ
s_{34}	et_{34}	Diabetes	η
s_{35}	et_{35}	Diabetes	η
s_{36}	et_{36}	Gastritis	ζ
s_{37}	et_{37}	Arthritis	θ
s_{38}	et_{38}	Arthritis	ϵ

(b)

Fig. 5. Fragment F_2^e with index over `Disease` (a) and fragment F_3^e with index over `Job` (b)

two fragments would instead amplify the risks of information leakage described in Section III. With reference to the second case, we note that the index functions used for indexing the same attribute in different fragments should produce different mappings. Otherwise, indexes could be exploited to (loosely) join fragments, thus reconstructing sensitive associations. As an example, consider fragments F_1^e and F_2^e in Figures 1(c-d) and assume that they are complemented with a flattened index with collision over `Disease`, computed by the index function in Figure 4(a). These two fragments can be joined revealing that Adams and Brown are nurses. This violates the confidentiality of association $\{$`Name`, `Job`$\}$. On the contrary, the release of fragments F_1^e and F_2^e in Figure 4(b) and in Figure 5(a), respectively, does not cause such an inference as the index functions adopted produce different mappings.

V. Related Work

Several research efforts have been performed in the context of data outsourcing [11], also proposing indexing techniques for efficiently performing queries directly on outsourced encrypted data [2]. These indexing techniques support different kinds of selection conditions and SQL clauses (e.g., equality conditions [1], [12], [13], range conditions [8], [12], [14], keyword searches [15]). The proposal in [7] analyzes inference exposure caused by indexes complementing encrypted outsourced data. Although addressing a similar problem, the results illustrated in [7] are not applicable to our scenario as the authors assume the outsourced relation to be completely encrypted, while we consider a fragmentation scenario where data are also plaintext represented.

Data fragmentation has been studied as a solution to provide privacy guarantees in outsourcing scenarios without encrypting the whole outsourced dataset. The proposals that adopt fragmentation can be classified depending on the technique used to protect sensitive associations (i.e., encryption, fragmentation, or a combination of them) [3], [4], [5] Our work is complementary to these proposals since it combines fragmentation with indexing. The first solution in this direction has been introduced in [6], where the authors propose to store an index structure at the data owner side to efficiently evaluate queries including conditions on attributes plaintext represented in different fragments. This proposal however does not analyze the inference exposure possibly caused by complementing fragments with indexes.

VI. Conclusions

Our observations can provide a useful basis for future works, including: the investigation of index functions providing both flattening and collisions, the definition of metrics for assessing exposures due to indexes, and the consideration of different types of knowledge that observers could exploit and that could enable leakage of sensitive information.

Acknowledgment

This work was supported in part by the EC within the 7FP under grant agreement 257129 (PoSecCo), by the Italian Ministry of Research within PRIN 2008 project PEPPER (2008SY2PH4) and PRIN 2010-2011 project "GenData 2020" (2010RTFWBH), and by a Google Faculty Research Award on "Fine-Grained Access Control for Social Networking Applications". The work of Sushil Jajodia was partially supported by the US Air Force Office of Scientific Research under grant FA9550-09-1-0421.

References

[1] H. Hacigümüs, B. Iyer, S. Mehrotra, and C. Li, "Executing SQL over encrypted data in the database-service-provider model." in *Proc. of SIGMOD*, Madison, WI, USA, Jun. 2002.

[2] P. Samarati and S. De Capitani di Vimercati, "Data protection in outsourcing scenarios: Issues and directions," in *Proc. of ASIACCS*, Beijing, China, Apr. 2010.

[3] G. Aggarwal et al., "Two can keep a secret: A distributed architecture for secure database services," in *Proc. of CIDR*, Asilomar, CA, USA, Jan. 2005.

[4] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Combining fragmentation and encryption to protect privacy in data storage," *ACM TISSEC*, vol. 13, no. 3, pp. 22:1–22:33, Jul. 2010.

[5] ——, "Selective data outsourcing for enforcing privacy," *JCS*, vol. 19, no. 3, pp. 531–566, 2011.

[6] A. Steele and K. Frikken, "An index structure for private data outsourcing," in *Proc. of DBSec*, Richmond, VA, USA, Jul. 2011.

[7] A. Ceselli, E. Damiani, S. De Capitani di Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati, "Modeling and assessing inference exposure in encrypted databases," *ACM TISSEC*, vol. 8, no. 1, pp. 119–152, Feb. 2005.

[8] H. Wang and L. V. S. Lakshmanan, "Efficient secure query evaluation over encrypted XML databases," in *Proc. of VLDB*, Seoul, Korea, Sep. 2006.

[9] Z. J. Czech, G. Havas, and B. S. Majewski, "Fundamental study - perfect hashing," *Theor. Comput. Science*, vol. 182, no. 1, pp. 1–143, Aug. 1997.

[10] S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, G. Pelosi, and P. Samarati, "Efficient and private access to outsourced data," in *Proc. of ICDCS*, Minneapolis, MN, USA, Jun. 2011.

[11] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Encryption policies for regulating access to outsourced data," *ACM TODS*, vol. 35, no. 2, pp. 12:1–12:46, Apr. 2010.

[12] E. Damiani, S. De Capitani di Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati, "Balancing confidentiality and efficiency in untrusted relational DBMSs," in *Proc. of CCS*, Washington, DC, USA, Oct. 2003.

[13] B. Hore, S. Mehrotra, M. Canim, and M. Kantarcioglu, "Secure multidimensional range queries over outsourced data," *The VLDB Journal*, vol. 21, no. 3, pp. 333–358, Jun. 2012.

[14] R. Agrawal, J. Kierman, R. Srikant, and Y. Xu, "Order preserving encryption for numeric data," in *Proc. of SIGMOD*, Paris, France, Jun. 2004.

[15] C. Wang, N. Cao, K. Ren, and W. Lou, "Enabling secure and efficient ranked keyword search over outsourced cloud data," *IEEE TPDS*, vol. 23, no. 8, pp. 1467–1479, Aug. 2012.

[16] S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, G. Pelosi, and P. Samarati, "Supporting concurrency in private data outsourcing," in *Proc. of ESORICS*, Leuven, Belgium, Sep. 2011.