Modeling and Assessing Inference Exposure in Encrypted Databases

ALBERTO CESELLI, ERNESTO DAMIANI, and
SABRINA DE CAPITANI DI VIMERCATI
Università di Milano
SUSHIL JAJODIA
George Mason University
STEFANO PARABOSCHI
Università di Bergamo
and
PIERANGELA SAMARATI
Università di Milano

The scope and character of today's computing environments are progressively shifting from traditional, one-on-one client-server interaction to the new cooperative paradigm. It then becomes of primary importance to provide means of protecting the secrecy of the information, while guaranteeing its availability to legitimate clients. Operating online querying services securely on open networks is very difficult; therefore many enterprises outsource their data center operations to external application service providers. A promising direction toward prevention of unauthorized access to outsourced data is represented by encryption. However, data encryption is often supported for the sole purpose of protecting the data in storage while allowing access to plaintext values by the server, which decrypts data for query execution.

In this paper, we present a simple yet robust single-server solution for remote querying of encrypted databases on external servers. Our approach is based on the use of indexing information attached to the encrypted database, which can be used by the server to select the data to be

This paper extends the previous work by the authors appeared under the title "Balancing Confidentiality and Efficiency in Untrusted Relational DBMSs" in *Proc. of the 10th ACM Conference on Computer and Communication Security (CCS 2003)*, Oct. 2003, Washington, DC, USA.
This work was supported in part by the European Union within the PRIME Project in the FP6/IST Programme under contract IST-2002-507591 and by the Italian MIUR within the KIWI and MAPS projects.
Authors' addresses: Alberto Ceselli, Ernesto Damiani, Sabrina De Capitani di Vimercati, and Pierangela Samarati, Dipartimento di Tecnologie dell'Informazione, Università di Milano, Via Bramante, 65, 26013 Crema, Italy; email: {ceselli,damiani,decapita,samarati}@dti.unimi.it; Sushil Jajodia, George Mason University, Fairfax, VA 22030-4444; email: jajodia@gmu.edu; Stefano Paraboschi, Dipartimento di Ingegneria Gestionale e dell'Informazione, Università di Bergamo, Viale Marconi, 5, 24044 Dalmine—Italy; email: parabosc@unibg.it.

returned in response to a query without the need of accessing the plaintext database content. Our indexes balance the trade-off between efficiency requirements in query execution and protection requirements due to possible inference attacks exploiting indexing information. We investigate quantitative measures to model inference exposure and provide some related experimental results.

Categories and Subject Descriptors: H.2.4 [**Database Management**]: Systems—*Relational databases*; H.2.7 [**Database Management**]: Database Administration—*Security, integrity, and protection*; H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing—*Indexing methods*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Query formulation*

General Terms: Security, Design

Additional Key Words and Phrases: Cryptography, database service, indexing, inference

1. INTRODUCTION

In most organizations, databases hold a critical concentration of sensitive information. Ensuring an adequate level of protection to databases' content is therefore an essential part of any comprehensive security program. *Database encryption* [Davida et al. 1981] is a time-honored technique that introduces an additional layer to conventional network and application-level security solutions, preventing exposure of sensitive information even if the database server is compromised. Database encryption prevents unauthorized users, including intruders breaking into a network, from seeing the sensitive data in databases; similarly, it allows database administrators to perform their tasks without accessing sensitive information (e.g., sales or payroll figures) in plaintext. Furthermore, encryption protects data integrity, as possible data tampering can be recognized and data correctness restored (e.g., by means of backup copies).

While much research has been done on the mutual influence of data and transmission security on organizations' overall security strategy [Walton 2002], the influence of service outsourcing on data security has been less investigated. Conventional approaches to database encryption have the sole purpose of protecting the data in storage and assume trust in the server, which decrypts data for query execution. This assumption is less justified in the new cooperative paradigm, where multiple *Web services* cooperate exchanging information in order to offer a variety of applications. Effective cooperation between Web services and *content providers* often requires critical information to be made continuously available for online querying by other services or final users. To name but a few, *telemedicine* applications involve network transfers of medical data, *location-based services* require availability of users' cartographic coordinates, while *e-business decision support systems* often need to access sensitive information such as credit ratings.

Customers, partners, regulatory agencies, and even suppliers now routinely need access to information originally intended to be stored deep within companies' information systems. Operating online querying services reliably on open networks is very difficult. For this reason, many enterprises prefer to outsource their data center operations to external *application providers*. *Remote storage technologies* (e.g., storage area networks [Ward et al. 2002]) are used to place sensitive and even critical company information at a provider's site, on systems

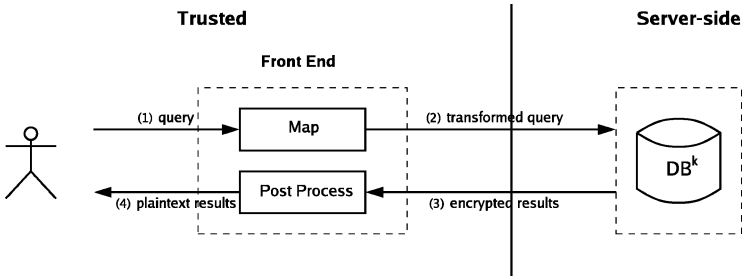


Fig. 1. Overall scenario.

whose architecture is specifically designed for database publishing and access is controlled by the provider itself.

As a consequence of this trend toward data outsourcing, highly sensitive data are now stored on systems run in locations that are not under the data owner's control, such as leased space and untrusted partners' sites. Therefore, data confidentiality and even integrity can be put at risk by outsourcing data storage and management.

The requirement that the database content remains secret to the database server itself introduces several new interesting challenges. Conventional encrypted DBMSs assume trust in the DBMS, which can then decrypt data for query execution. In an outsourced environment scenario, such an assumption is not applicable anymore as the party to which the service is being outsourced cannot be granted full access to the plaintext data. Since confidentiality demands that data decryption must be possible only at the client side, techniques are needed enabling servers to execute queries directly on encrypted data. A first proposal toward the solution of this problem was presented in Hacigümüs et al. [2002a], where the authors proposed storing, together with the encrypted database, additional *indexing* information. Such indexes can be used by the DBMS to select the data to return in response to a query. The basic idea is illustrated in Figure 1. Each plaintext query (1) is mapped onto a corresponding query (2) on the indexing information and executed in that form at the server side. The server returns the encrypted result (3), which is then decrypted at the trusted front end. If the mapping between indexing information and the original database plaintext is not exact, an additional query (4) may need to be executed to eliminate spurious tuples that do not belong to the result set.

A major challenge in this scenario is how to compute and represent indexing information. Two conflicting requirements challenge the solution of this problem: on one side, the indexing information should be related with the data well enough to provide for an effective query execution mechanism; on the other side, the relationship between indexes and data should not open the door to inference and linking attacks that can compromise the protection granted by encryption [Denning 1982]. The indexing information provided in Hacigümüs et al. [2002a], based on using as indexes names of sets containing value intervals, proves limited in this respect (see Section 2).

In this paper, we provide an approach to indexing encrypted data constructed with efficiency and confidentiality in mind, providing a balance between these

two requirements. A trade-off can be observed between the degree of protection that our family of techniques is able to offer, and a corresponding decrease in efficiency that can be produced by the use of these protection measures. Then, a general motivation of our investigation is an assessment of the degree of protection provided by different indexing techniques (each of which affects efficiency in query execution in a different way). It turns out that the data protection (and its relation with efficiency) cannot be synthesized by simple mathematical formulae. Instead, the paper proposes a family of abstract models for solving the inference problem using algorithmic techniques. Our analysis supports a sequence of experiments showing the behavior of different indexing solutions.

The contributions of this paper can be summarized as follows. First, we propose an approach to indexing encrypted data based on direct encryption and hashing. Second, we define a suite of graph theoretical models supporting quantitative evaluations of the inference exposure of the two approaches. Third, we present the result of a set of experiments that quantify the protection increase that hashing is able to provide.

2. RELATED WORK

Database encryption has been proposed since long as a fundamental tool for providing strong security for "data at rest." Thanks to recent advances in processors' capabilities and to the development of fast encryption techniques, the notion of encrypted database is nowadays well recognized, and several commercial products reached the market. However, developing a sound security strategy including database encryption still involves many open issues. Key management and security are of paramount importance in any encryption-based system and were therefore among the first issues to be investigated in the framework of database encryption [Davida et al. 1981; Hacigümüs and Mehrotra 2004]. Later, techniques have been developed aimed at efficiently querying encrypted databases [Song et al. 2000], some of them related to parallel efforts by the text retrieval community [Klein et al. 1989] for executing *hidden queries*, that is, queries where only the ciphertext of the query arguments is made available to the DBMS. On the other hand, architectural research investigated optimal sharing of the encryption burden between secure storage, communication channels and the application where the data originates [Jensen 2000], looking for a convenient trade-off between data security and application performance. Recently, much interest was devoted to secure handling of database encryption in distributed, Web-based execution scenarios, where data management is outsourced to external services [Bouganim and Pucheral 2002]. The main purpose of this line of research is to find techniques for delegating data storage and the execution of queries to external servers while preserving efficiency. The *index of range* technique proposed in Hacigümüs et al. [2002a] in the framework of a *database-service-provider* architecture relies on partitioning the domains of attributes in client tables into sets of intervals. The value of each remote table attribute is stored as the index countersigning the interval to which the corresponding plain value belongs. Indexes may be ordered or not, and the intervals may be chosen so that they have all the same length, or

are associated with the same number of tuples. This representation supports efficient evaluation on the remote server of both equality and range predicates; however, it makes it awkward to manage the correspondence between intervals and the actual values present in the database. In Damiani et al. [2004], 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. *Privacy homomorphism* has also been proposed for allowing the execution of aggregation queries over encrypted data [Hacigümüs et al. 2004]. The proposed approach is based on the technique introduced by Rivest et al. [1978] according to which an encrypted function $E()$ is homomorphic if given $E(x)$ and $E(y)$, one can obtain $E(x\theta y)$ without decrypting x and y for some operation θ. In this case, the server stores an encrypted table with an index for each aggregation attribute (i.e., an attribute on which the aggregate operator can be applied) obtained from the original attribute with privacy homomorphism. An operation on an aggregation attribute can then be evaluated by computing the aggregation at the server site and by decrypting the result at the client side. Other work on privacy homomorphism illustrates techniques for performing arithmetic operations ($+, -, \times, \div$) on encrypted data and does not consider comparison operations [Boyens and Günter 2003; Domingo-Ferrer 1996; Domingo-Ferrer and Herrera-Joanconmartí 1998]. In Agrawal et al. [2004], an order preserving encryption schema (OPES) is presented to support equality and range queries as well as `max`, `min`, and `count` queries over encrypted data. The basic idea is that given a target distribution, the plaintext values are transformed by using an order-preserving transformation and in such a way that the transformed values follow the target distribution. While our technique is applicable to any kind of data and robust against different class of attacks (i.e., known-plaintext attacks and ciphertext-only attacks), OPES is only applicable to numeric data and is secure against ciphertext-only attacks. A distinct, though related solution is proposed in Bouganim and Pucheral [2002], where smart cards are used for key management.

On a different line of related work, we note that the protection/exposure given by hashing can resemble the generalization approach for microdata protection; and correspondingly inference attacks exploiting it can resemble record linkage techniques examined in that context [Samarati 2001]. However, the two problems turn out to be quite different: while generalization replaces values in a given interval with a single identifier and preserves some information on plaintext values, hashing replaces uncorrelated values with a single bucket identifier.

Also, it is important to note that the problem we consider differs from existing approaches protecting encrypted data, which investigated solutions for the private information retrieval problem (protecting the query search criteria, that is, the information the user is looking for) or for the problem of limiting the amount of data that users can acquire.

3. DATA ORGANIZATION

We consider a relational DBMS where data are organized in tables, where the underlined attribute represents the key of the table (e.g., see table ACCOUNTS in

ACCOUNTS

Account	Customer	Balance
Acc1	Alice	100
Acc2	Alice	200
Acc3	Bob	300
Acc4	Chris	200
Acc5	Donna	400
Acc6	Elvis	200
Acc7	Fred	300

(a)

ENC_ACCOUNTS1

Enc_tuple	I_A	I_C	I_B
x4Z3tfX2ShOSM	π	α	μ
mNHg1oC010p8w	ϖ	α	κ
WsIaCvfyF1Dxw	ξ	β	η
JpO8eLTVgwV1E	ϱ	γ	κ
qctG6XnFNDTQc	ς	δ	θ
4QbqC3hxZHklU	Γ	ϵ	κ
3jO9DfoaU78Hs	τ	ϕ	η

(b)

ENC_ACCOUNTS2

Enc_tuple	I_A	I_C	I_B
x4Z3tfX2ShOSM	π	α	μ
mNHg1oC010p8w	ϖ	α	κ
WsIaCvfyF1Dxw	ξ	δ	θ
JpO8eLTVgwV1E	ϱ	α	κ
qctG6XnFNDTQc	ς	β	κ
4QbqC3hxZHklU	Γ	β	κ
3jO9DfoaU78Hs	τ	δ	θ

(c)

Fig. 2. A plaintext relation (a) and the corresponding encrypted relations with direct encryption (b) and hashing (c).

Figure 2). In principle, different granularity choices are possible for database encryption, such as encrypting at the level of whole tables, columns (i.e., attributes), rows (i.e., tuples), and cells (i.e., elements). Encrypting at the level of tables (columns resp.) implies that the whole table (column resp.) involved in a query should always be returned, providing therefore no means for selecting the data of interest and leaving to the client the burden of query execution on a possibly huge amount of data. On the other hand, supporting encryption at the finest granularity of single cells is also inapplicable as it would severely affect performance, since the client would be required to execute a potentially very large number of decrypt operations to interpret the results of queries [Hacigümüs et al. 2002b]. In the same line as Hacigümüs et al. [2002a], we assume encryption to be performed at the tuple level. To provide the server with the ability to select a set of tuples to be returned in response to a query, we associate with each encrypted tuple a number of indexing attributes. An *index* can be associated with each attribute in the original table on which conditions need to be evaluated in the execution of queries.

Each plaintext table is represented as a table with an attribute for the encrypted tuple and as many attributes as indexes to be supported. More specifically, each plaintext tuple $t(A_1, \ldots, A_n)$ is mapped onto a tuple $t'(T_k, I_1, \ldots, I_m)$, where $m \leq n$, $t'[T_k] = E_k(t)$, with $E_k()$ denoting an invertible encryption function over key k, and each I_i corresponds to the index over some A_j. Figure 2 illustrates an example of a plaintext table ACCOUNTS and the corresponding encrypted/indexed[1] table ENC_ACCOUNTS1 where `Enc_tuple` contains

[1]In the remainder of the paper, for the sake of simplicity, we shall designate this table format with the term encrypted.

the encrypted triples, while $\mathtt{I_A}$, $\mathtt{I_C}$, and $\mathtt{I_B}$ are indexes over attributes `Account`, `Customer`, and `Balance`, respectively. For the sake of readability, we use easy-to-understand names for the attributes and table names in the encrypted schema and Greek letters as index values. Of course, in a real example, attributes and table names would be obfuscated and actual values for indexes would be the results of an invertible encryption function and would then look like the ones reported for the encrypted tuples in Figure 2.

Let us now discuss how to represent indexing information.

An approach providing the same fine-grained selection capability as using plaintext values is to use their corresponding encrypted values as indexes. Then, for each indexed cell, the outcome of an invertible encryption function over the cell value is used, that is, $t[I_i] = E_k(t[A_i])$. Query execution is simple: each plaintext query can be translated into a corresponding query on encrypted data by simply applying the encryption function to the values mentioned in the query. For instance, with reference to the tables in Figure 2, query "SELECT * FROM ACCOUNTS WHERE CUSTOMER = `Alice`" would be translated into "SELECT ENC_TUPLE FROM ENC_ACCOUNTS1 WHERE $\mathrm{I}_C = \alpha$." This solution has the advantage of preserving plaintext distinguishability, together with precision and efficiency in query execution, as all the tuples returned belong to the query set of the original query. In particular, the solution is convenient for queries involving equality constraints over the attributes. Also, since equality predicates are almost always used in the computation of joins, a join between two tables that use the same encryption function on the join attribute can be computed precisely.

As a drawback, however, in this approach encrypted values reproduce exactly the plaintext values distribution with respect to values' *cardinality* (i.e., the number of distinct values of the attribute) and frequencies. This opens the door to frequency-based attacks (see next section).

An alternative approach to counter these attacks is to use as index the result of a *secure hash function* over the attribute values rather than straightforwardly encrypting the attributes; this way, the attribute values' distribution can be *flattened* by the hash function. A flexible characteristic of a hash function is the cardinality of its codomain B, which allows us to adapt it to the granularity of the represented data. When B is small compared with the cardinality of the attribute, the hash function can be interpreted as a mechanism that distributes tuples in B *buckets*; a good hash function (and a secure hash has to be good) distributes uniformly the values in the buckets. For instance, the ACCOUNTS table in Figure 2 can be indexed by considering three buckets (α, β, δ) for $\mathtt{I_C}$ and three buckets (μ, κ, θ) for $\mathtt{I_B}$. The encrypted relation ENC_ACCOUNTS2 in Figure 2 can then be obtained when `Alice` and `Chris` are both mapped onto α, `Donna` and `Elvis` are both mapped onto β, while `Bob` and `Fred` are both mapped onto δ. Also, `200` and `400` are both mapped to κ, `100` is mapped onto μ, and `300` is mapped onto θ. With respect to direct encryption, hash-based indexing provides more protection as different plaintext values are mapped onto the same index.

Using attribute hashes in remote tables permits an efficient evaluation of equality predicates within the remote server. If the same hash function is used to compute values of two attributes of different tables on which the equality predicate must be evaluated in the context of a join query, the join query itself

can be efficiently computed at the remote server simply by combining all of the pairs of tuples characterized by the same hash value.

When direct encryption is used for indexing, the results returned by a query on the encrypted table is exactly the query set of the original query. The only task left for the front end is then decryption. By contrast, when hashing is used, the results will often include spurious tuples (all those belonging to the same bucket of the index) that will have to be removed by the front end receiving them. In this case, the additional burden on the front end consists in purging from the result returned by the remote server all the pairs of tuples that, once brought back in plaintext form, do not satisfy the equality predicate on the join attribute. Intuitively, every query Q of the front end corresponds to a query Q' to be passed onto the server-side DBMS for execution over the encrypted database and a query Q'' to be executed at the front end on the results of Q'. As an example, consider the encrypted table ENC_ACCOUNTS2 in Figure 2 and the user query Q "SELECT BALANCE FROM ACCOUNTS WHERE CUSTOMER=Bob." The query is translated as Q' = "SELECT ENC_TUPLE FROM ENC_ACCOUNTS2 WHERE $\mathrm{I}_C = \delta$" for execution by the server-side DBMS which returns the third and seventh encrypted tuples. The trusted front end then decrypts the result obtaining the third and seventh tuples of the original table ACCOUNTS and re-executes on them the original query, eliminating the seventh one (whose presence was due to index collision).[2]

Incidentally, we observe that the trusted front end is a relatively complex piece of software, as it has to integrate most of the functionalities of a relational engine.

4. INFERENCE EXPOSURE

Being closely related to plaintext data, indexing information could open the door to inferences that exploit data analysis techniques to reconstruct the database content and/or break the indexing code. It is important to be able to evaluate quantitatively the level of exposure associated with the publication of certain indexes and therefore to determine the proper balance between index efficiency and protection.

There are different ways in which inference attacks could be modeled. In this paper, we consider two cases that differ in the assumption about the attacker's prior knowledge. In common, the two attack models have the fact that the attacker has complete access to the encrypted relations.

In the first case, which we call $\mathbf{Freq} + \mathbf{DB}^K$ scenario, we assume the attacker is aware of the distribution of plaintext values (Freq) in the original database in addition to knowing the encrypted database (DB^K). This knowledge can be exact (e.g., in a database storing accounting information, the account holder list can be fully known) or approximate (e.g., the ZIP codes of the geographical areas of the account holders can be estimated based on population data). For the sake of simplicity, in the following we will assume exact knowledge (which represents the worst case scenario). In this scenario, there are two possible

[2]We refer the reader to Damiani et al. [2003] for an illustration of the working of the query translation and the query evaluation processes.

	Direct encryption	**Hashing**
Freq + DB^k	Section 5	Section 6
DB + DB^k	Section 7	Section 8

Fig. 3. Synoptic guide to the exposure evaluation on the four attack scenarios.

inferences that the attacker can draw: (i) the *plaintext content* of the database, that is, determine the existence of a certain tuple (or *association* of values) in the original database, and/or (ii) the *indexing function*, that is, determine the correspondence between plaintext values and indexes.

In the second case, which we call **DB**+**DB**^{K} scenario, we assume the attacker has both the encrypted (DB^K) and the plaintext database (DB). In this case the attacker aims at *breaking the indexing function*, thus establishing the correlation between plaintext data and the corresponding index values. The hosting server will then be able to decode additional encrypted tuples that are inserted into the database. This attack may, for example, occur when the database owner switches from a remote plaintext database to the use of encryption.

The combination of the two attack models and the two encryption solutions gives us the four different scenarios illustrated in Figure 3, which will be investigated in subsequent sections, and for which we will propose abstract models and algorithmic solutions, using them in the experiments to evaluate exposure to inference.

5. FREQ+DB^K WITH DIRECT ENCRYPTION

To illustrate this scenario, let us consider the example in Figure 2. The attacker knows the encrypted table ENC_ACCOUNTS1; also, she knows that all values for attribute `Account` have only one occurrence, and she knows the values (and their occurrences) appearing independently in attributes `Customer` and `Balance`, namely,

`Customer` = {`Alice`, `Alice`, `Bob`, `Chris`, `Donna`, `Elvis`, `Fred`}
`Balance` = {`100`, `200`, `200`, `200`, `300`, `300`, `400`}.

Although the attacker does not know which index corresponds to which plaintext attribute, she can determine the actual correspondence by comparing their occurrence profiles. In particular, she can determine that $\mathtt{I_A}$, $\mathtt{I_C}$, and $\mathtt{I_B}$ correspond to attributes `Account`, `Customer` and `Balance`, respectively. The attacker can then infer that κ represents value `200` and index α represents value `Alice` (*indexing inference*). She can also infer that the plaintext table contains a tuple associating values `Alice` and `200` (*association inference*). The other occurrence of the index value corresponding to `Alice` (i.e., α) is associated with a balance other than `200`. Since there are only three other possible values, the probability of guessing it right is 1/3. In other terms, the probability of each association depends on the combination of occurrences of its values.

Intuitively, the basic protection from inference in the encrypted table is that values with the same number of occurrences are indistinguishable to the attacker. For instance, all customers but `Alice` are indistinguishable from one another, as well as all amounts but `200`. By contrast, `Alice` and `200` stand out

being, respectively, the only customer appearing twice and the only balance appearing three times.

The exposure of an encrypted relation to indexing inference can then be thought of in terms of an equivalence relation where indexes (and plaintext values) with the same number of occurrences belong to the same equivalence class. For instance, denoting each equivalence class with a dot notation showing the attribute name and its number of occurrences (e.g., class A.1 contains all the values of attribute A that occur once), we obtain

$$\text{A.1} = \{\pi, \varpi, \xi, \varrho, \varsigma, \Gamma, \tau\} = \{\texttt{Acc1}, \texttt{Acc2}, \texttt{Acc3}, \texttt{Acc4}, \texttt{Acc5}, \texttt{Acc6}, \texttt{Acc7}\}$$
$$\text{C.1} = \{\beta, \gamma, \delta, \epsilon, \phi\} = \{\texttt{Bob}, \texttt{Chris}, \texttt{Donna}, \texttt{Elvis}, \texttt{Fred}\}$$
$$\text{C.2} = \{\alpha\} = \{\texttt{Alice}\}$$
$$\text{B.1} = \{\mu, \theta\} = \{100, 400\}\ \text{B.2} = \{\eta\} = \{300\}\ \text{B.3} = \{\kappa\} = \{200\}.$$

The quotient of the encrypted table with respect to the equivalence relation defined above is the following:

Quotient Table

$\mathbf{qt}_A$	$\mathbf{qt}_C$	$\mathbf{qt}_B$
A.1	C.2	B.1
A.1	C.2	B.3
A.1	C.1	B.2
A.1	C.1	B.3
A.1	C.1	B.1
A.1	C.1	B.3
A.1	C.1	B.2

IC Table

$\mathbf{ic}_A$	$\mathbf{ic}_C$	$\mathbf{ic}_B$
1/7	1	1/2
1/7	1	1
1/7	1/5	1
1/7	1/5	1
1/7	1/5	1/2
1/7	1/5	1
1/7	1/5	1

The *exposure* of the encrypted table to inference attacks can then be evaluated by looking at the distinguishable characteristics in the quotient table. In particular, the association $\langle \texttt{Alice}, 200 \rangle$ (and its correspondence $\langle \alpha, \kappa \rangle$) can be spotted with certainty being composed by two *singleton* equivalence classes (C.2 and B.3). For the other values, probabilistic considerations can be made by looking at the *IC table,* that is, the table of the inverse of the cardinalities of the equivalence classes. In fact, the probability of disclosing a specific association is the product of the inverses of the cardinalities. The exposure of the whole relation (or projection of it) can then be estimated as the average exposure of each tuple in it.

We chose the average because it is a simple operator that best represents the intuition on the degree of exposure that characterizes a database; an alternative would be the use of the maximum exposure of a database tuple as the overall exposure index, but this choice would present a poor behavior and would often produce high ($= 1$) exposure values, in most cases due to the presence, among a multitude of unrecognizable values, of a single value characterized by a unique cardinality. In the paper, we always use the average to compute the overall database exposure from the exposure of single values.

Formally, we can write the *exposure coefficient* $\mathcal{E}$ associated with an encrypted table with inverse cardinality table IC as

$$\mathcal{E} = \frac{1}{n} \sum_{i=1}^{n} \prod_{j=1}^{k} \text{IC}_{i,j}. \tag{1}$$

	α	β	δ	
Alice				**2**
Bob				**1**
Chris				**1**
Donna				**1**
Elvis				**1**
Fred				**1**
	3	**2**	**2**	

(a)

	α	β	δ	
Alice	1	0	0	**2**
Bob	0	0	1	**1**
Chris	1	0	0	**1**
Donna	0	1	0	**1**
Elvis	0	1	0	**1**
Fred	0	0	1	**1**
	3	**2**	**2**	

(b)

	α	β	δ	
Alice	0	1	0	**2**
Bob	0	0	1	**1**
Chris	1	0	0	**1**
Donna	1	0	0	**1**
Elvis	1	0	0	**1**
Fred	0	0	1	**1**
	3	**2**	**2**	

(c)

Fig. 4. Problem 1 for attribute Customer and table ACCOUNTS: the correct solution (b), and an incorrect solution (c)

Here, i ranges over the tuples while j ranges over the columns.

With reference to our example, we have a value of $\mathcal{E} = \frac{1}{7} \cdot \frac{12}{35}$ for the protection of the whole relation, and a value of $\frac{1}{7} \cdot \frac{12}{5}$ for the pair $\langle$Customer, Balance$\rangle$.

Note how a long-tailed distribution of values (i.e., distributions composed of many values having few occurrences) can decrease the exposure to association attacks. This reflects the fact that while the attacker has information on many values, they all fall into the same equivalence class resulting indistinguishable from one another.

Since each index value corresponds to a single plaintext one, the exposure computed above may be regarded as a lower bound to vulnerability to association inference.

6. FREQ+DB^K WITH HASHING

As we already noticed, collisions due to hashing increase protection from inference. For a simple query using a hash value for selection, the average increase in the result size is equal to *cf* (collision factor), that is, the number of distinct attribute values that on average collide on the same hash value. If the query uses the index to evaluate a join between two tables, the increase in the query size would be cf^2.

For the sake of simplicity, in the following, we consider examples related to attributes taken individually, that is, focus on breaking (or reducing the uncertainty on) the indexing function. Association inference can then be modeled as the combination of information obtained from individual attributes.

Breaking the indexing function can be modeled as the problem of finding a mapping λ from the plaintext values A to indexing values H that satisfies the constraints given by the attacker's prior knowledge, which is represented by the occurrences of each plaintext value and of each hashed one. In the following, given any value v in a set of possible values V (either plaintext values in the original table or hash values in the encrypted table), we use $count(v)$ to denote the number of occurrences of v in the corresponding table. For instance, with reference to our example, $count(\texttt{Alice}) = 2$ and $count(\alpha) = 3$.

Function λ can be represented graphically by a table Λ (see Figure 4(a)), with plaintext values as rows and hash values as columns and where $\Lambda[i, j] = 1$ if $\lambda(i) = j; \Lambda[i, j] = 0$ otherwise. For instance, the table in Figure 4(b) illustrates the mapping for attribute Customer in our example. The bold numbers

appearing at the end of each row i and column j represent the attackers knowledge, that is, $count(i)$ and $count(j)$, respectively. Finding a mapping satisfying the cardinality constraints on the plaintext/hash values equates to finding different ways to fill this table subject to the constraints. For instance, Figures 4(b) and (c) illustrate two possible solutions for the table in Figure 4(a). Note that Figure 4(b) is the correct mapping, which is however indistinguishable, from the attacker's point of view, from the incorrect mapping in Figure 4(c).

This case can be stated as finding solutions to the following problem.

Problem 1. Let A and H be the set of plaintext values and hash values, respectively. Find all the solutions Λ such that

(1) $\forall j \in H : \sum_{i \in A} \Lambda[i, j] \cdot count(i) = count(j)$;
(2) $\forall i \in A$: $\sum_{j \in H} \Lambda[i, j] = 1$;
(3) $\forall i \in A, \forall j \in H : \Lambda[i, j] \in \{0, 1\}$.

The first constraint states that the number of plaintext values mapped to each hash value must coincide with the number of occurrences of that hash value. The second and third constraints trivially state that each plaintext value is mapped to exactly one hash value.

Note that since the attacker has to determine the mapping, a first evaluation of the strength of the encryption could be done by measuring how many solutions Problem 1 has. Intuitively, if the problem has exactly one solution, the encryption function is completely exposed to inference. Counting the number of solutions is, however, not sufficient as different values can be exposed in different ways (e.g., if all solutions have the same mapping for a specific value v, such a value is completely exposed). For this reason it is important to explicitly enumerate the solutions, rather than simply counting them.

Problem 1 is a well-known $\mathcal{NP}$-Hard problem addressed in the literature as the *multiple subset sum problem* [Caprara et al. 2000]. A few algorithms have been proposed, but only for the optimization version of the problem [Caprara et al. 2003].

Evaluating the inference exposure requires then to enumerate all the possible solutions to Problem 1. Fortunately, we can reduce the number of solutions to be enumerated by exploiting indistinguishable characteristics of both plaintext and hash values.

6.1 Reducing the Problem by Exploiting Indistinguishability of Plaintext Values

As already noticed in previous section, from the point of view of the attacker, plaintext values with the same cardinality are indistinguishable. Therefore, each instance of Problem 1 has several solutions that differ only for the order in which plaintext elements with the same number of occurrences are considered. If two solutions are in such a relationship, we say that they are *symmetric*. The number of symmetric solutions grows combinatorially in the size of the plaintext values. For instance, suppose that there are n plaintext values $v_1, \ldots, v_n$ with the same number of occurrences in the original table and k possible hash

	α	β	δ	
C.1				**5**
C.2				**1**
	3	**2**	**2**	

(a)

	α	β	δ	
C.1	1	2	2	**5**
C.2	1	0	0	**1**
	3	**2**	**2**	

(b)

	α	β	δ	
C.1	3	0	2	**5**
C.2	0	1	0	**1**
	3	**2**	**2**	

(c)

Fig. 5. Problem 2 for equivalence classes of attribute `Customer` in table ACCOUNTS (a): the correct solution (b), and an incorrect solution (c).

values $h_1, \ldots, h_k$ for them. If there exists a solution to Problem 1 mapping to each h_i a number n_i of values in $v_1, \ldots, v_n$, then each partition of the set of the n plaintext values in k subsets of cardinality $n_1, \ldots, n_k$ also represents a solution to Problem 1. The number of these partitions can be expressed as a multinomial coefficient $\binom{n}{n_1,\ldots,n_k}$. For instance, in the correct mapping, the five values of equivalence class C.1 are partitioned in a subset of one value and two subsets of two values. Hence, the number of symmetric solutions is $\binom{5}{1,2,2} = \frac{5!}{(1!\cdot 2!\cdot 2!)} = 30$.

Our algorithm exploits this symmetry of the solutions by enumerating only one solution in each of such a symmetry class. This behavior corresponds to stating the problem with reference to equivalence classes of plaintext values (like in the previous section), grouping values with the same number of occurrences.

In the following, we therefore consider a variation of the problem where matrix Λ has as rows equivalence classes of the plaintext values. For instance, for attribute `Customer` of our example, which we abbreviate as C in the following, there are two equivalence classes:

C.1 = {`Bob`, `Chris`, `Donna`, `Elvis`, `Fred`};
C.2 = {`Alice`}

and the matrix expressing the constraints becomes as illustrated in Figure 5(a).

Note that the number in bold at the end of each row i now corresponds to the number of elements in equivalence class i, which we denote by $|i|$. Also, we denote with $count(i)$ the (equal) number of occurrences of the elements in i. For instance, for class C.1, $|\text{C.1}| = 5$ and $count(\text{C.1}) = 1$.

The problem of finding a solution, with reference to equivalence classes, is stated as follows:

Problem 2. Let X and H be the set of equivalence classes of plaintext values and of the set of hash values, respectively. Find all the solutions Λ such that

(1) $\forall j \in H : \sum_{i \in A} \Lambda[i, j] \cdot count(i) = count(j)$;
(2) $\forall i \in A$: $\sum_{j \in H} \Lambda[i, j] =\mid i \mid$;
(3) $\forall i \in A, \forall j \in H : \Lambda[i, j] \in \mathcal{Z}^+$.

The first constraint states that the sum of the occurrences associated with elements of the equivalence classes mapped to a hash value must coincide with the number of occurrences of that hash value. The second and third constraints state that all plaintext values in each equivalence class are mapped to some hash value.

	$I_C.3(\alpha)$	$I_C.2\ (\beta,\delta)$	
C.1			**5**
C.2			**1**
	(3)	**(2,2)**	

(a)

	$I_C.3(\alpha)$	$I_C.2\ (\beta,\delta)$	
C.1	(1)	(2,2)	**5**
C.2	(1)	(0,0)	**1**
	(3)	**(2,2)**	

(b)

	$I_C.3(\alpha)$	$I_C.2\ (\beta,\delta)$	
C.1	(3)	(0,2)	**5**
C.2	(0)	(1,0)	**1**
	(3)	**(2,2)**	

(c)

Fig. 6. Problem 3 for equivalence classes of attribute `Customer` in table ACCOUNTS and equivalence classes of hash values (a): the correct solution (b), and an incorrect solution (c)

6.2 Reducing the Problem by Exploiting Indistinguishability of Hash Values

Following an approach similar to the one used for indistinguishability of plaintext values, we can exploit symmetry also among hash values. In particular, we avoid computing all the different combinations that would map different plaintext values in different hash values which are indistinguishable since they have the same number of occurrences (e.g., β and δ in our example). Therefore, instead of considering individual hash values we collapse together, in a single equivalence class, hash values with the same number of occurrences. For our I_C attribute, we have

$$\mathrm{I_C}.2 = \{\beta, \delta\}$$
$$\mathrm{I_C}.3 = \{\alpha\}.$$

Intuitively, with reference to the matrix we are now collapsing columns together. Note however a difference here, since this time we cannot simply collapse multiple columns and consider a single value for them, as this would correspond to a single hash value with a number of occurrences equal to the sum of the occurrences of the collapsed columns. Instead, we need to keep track, within each single column, of the different hash values it combines.

For our example, the matrix of Figure 5(a) would be transformed into the matrix of Figure 6(a). Note that each cell in the matrix is now a vector. For the sake of simplicity, with a notational abuse, given a vector j, we now use $count(j)$ to denote the vector of the occurrences of the single elements of j. For instance, $count(\mathrm{I_C}.2)$ represents now the vector $(2, 2)$.

We then restate the problem as follows.

Problem 3. Let X and Y be the set of equivalence classes of plaintext values and of hash values, respectively. Find all the solutions Λ such that

(1) $\forall j \in Y : \sum_{i \in X} \Lambda[i, j] \cdot count(i) = count(j)$;
(2) $\forall i \in X : \sum_{j \in H} \sum_{k \in |j|} \Lambda[i, j][k] = |i|$;
(3) $\forall i \in A, \forall j \in H, \forall k = 1, \ldots |j| : \Lambda[i, j][k] \in \mathcal{Z}^+$.

The constraints intuitively extend the constraints in Problem 2 taking into account equivalence classes of hash values.

The exposure coefficient can now be computed by enumerating the different solutions to Problem 3. Figure 7 illustrates the recursive enumeration algorithm we used in our experiments. Given a statement of Problem 3, our algorithm computes the different solutions $\Lambda_1, \ldots, \Lambda_n$ to the problem, starting by enumerating all the different columns that can appear in a solution by solving

```
Algorithm: (Enumeration of solutions for Problem 3)
Input:   The set X of equivalence classes of plaintext values and the set Y of equivalence classes of
         hash values
Output:  The set Λ_1 ··· Λ_n of solutions to Problem 3

/* For each j ∈ Y, use the Pisinger's algorithm [Pisinger 1995] to compute the set Cols[j] of solutions
   to the corresponding subset sum problem (i.e., all the ways in which value count(j) can be obtained
   as a sum of the count(i) values by using each count(i) value at most |i| times). Each solution
   col ∈ Cols[j] is a column that can appear in a Λ matrix. Each component col[i]
   represents how many plaintext values in the equivalence class i appear in that column. */

MAIN
begin
  for j ∈ Y do Cols[j] := subsetSum(X, count(j))
  for i ∈ X do leftValues[i] := |i| /* Number of plaintext values in i that are not assigned */
  for j ∈ Y do LeftCols[j] := Cols[j] /* Columns in Cols[j] left to consider */
  BMSSP(Λ, 1, 1, LeftCols, leftValues)
end

BMSSP(Λ, j, k, LeftCols, leftValues) /* j ranges over Y, k is the number of column selected for j */
begin
if (j > |Y|) then output(Λ) /* All hash values have been considered; a solution is found */
else
  TempCols := ∅ /* Store the considered columns */
  for col ∈ LeftCols[j] do
      full := FALSE
      for i ∈ X do
          if col[i] > leftValues[i] then full := TRUE /* col violates the constraints */
      if full == FALSE then
          Λ[∗, j][k] := col /* Select column col and update the number of remaining plaintext values */
          for i ∈ X do leftValues[i] := leftValues[i] − col[i]
          for j' ∈ Y, j' ≥ j do /* Check if a column can be found for each subsequent equivalence */
              classfull := TRUE /* class that does not violate the cardinality constraints */
              for col ∈ LeftCols[j'] do
                  full := FALSE
                  for i ∈ X do
                      if col[i] > l_i then full := TRUE
                  if full == FALSE then
                      classfull := FALSE
                      break
              if classfull == TRUE then break
          if classfull == FALSE then /* Recursive call */
              if (k < |j|) then /* Select more columns for the equivalence class j */
                  BMSSP(Λ, j, k + 1, LeftCols, leftValues)
              else /* Equivalence class j is completed, select columns for the subsequent classes */
                  BMSSP(Λ, j + 1, 1, LeftCols, leftValues)
          for i ∈ X do leftValues[i] := leftValues[i] + col[i] /* Backtrack */
      /* All solutions using column col have been enumerated, discard column col */
      LeftCols[j] := LeftCols[j] \ {col}
      TempCols := TempCols ∪ {col}
  LeftCols[j] := LeftCols[j] ∪ TempCols
end

subsetSum(X, count(j)) see [Pisinger 1995]
```

Fig. 7. Algorithm for computing the different solutions $\Lambda_1, \ldots, \Lambda_n$ of Problem 3.

the corresponding subset sum problem. This is done using an adaptation of Pisinger's algorithm for the subset sum problem [Pisinger 1995]. Pisinger's algorithm relies on dynamic programming: the original version runs in pseudo-polynomial time (i.e., it runs in time polynomial in the dimension of the problem and the magnitudes of the data, rather than the logarithm of their magnitudes [Garey and Johnson 1979]). However, a dominance criterion has to be removed to obtain all the feasible solutions. Each solution Λ_m then corresponds to different combinations of the columns found.

Once, we have the different solutions, we can evaluate the exposure coefficient as illustrated in the following subsection.

6.3 Computing the Exposure Coefficient

While Problem 3 has a number of solutions, which we denote as $\Lambda_1, \ldots, \Lambda_n$, only one corresponds to the correct mapping used for indexing. In the following, we use Λ' to denote the correct solution in $\{\Lambda_1, \ldots, \Lambda_n\}$.

For each equivalence class $i \in X$ and solution Λ, let $\Lambda[i, *]$ be the part of the solution of Λ describing the mapping of plaintext values in i (i.e., row i in matrix Λ). For instance, with reference to Figure 6(b), $\Lambda[i, *] = [(1), (2, 2)]$.

To evaluate the exposure coefficient $\mathcal{E}(i)$, we first introduce the concept of exposure coefficient for each solution Λ, which we denote as $\mathcal{E}(i, \Lambda[i, *])$.

We distinguish two separate cases depending on whether the solution is correct or incorrect with respect to i.

Let us first consider the case where Λ is the (unique) correct mapping Λ'. Here, the uncertainty remaining for the attacker is only due to the cardinality of the equivalence classes for both the plaintext values and for the hash values. To measure this uncertainty, we first define the set of potential equivalence classes $Pot\_eq(i)$ of hash values to which values of i could be mapped, that is, for which there is at least a nonzero value in a cell. Formally,

$$Pot\_eq(i) = \left\{ j \,|\, \sum_k \Lambda[i, j][k] > 0 \right\}.$$

In our example, $Pot\_eq(\mathrm{C}.1) = \{\mathrm{I_C}.2, \mathrm{I_C}.3\}$ while $Pot\_eq(\mathrm{C}.2) = \{\mathrm{I_C}.3\}$.

Then, for each individual value in i, the number of potential mappings $Pot(i)$ can be computed by adding up the cardinality of all equivalence classes of hash values j in $Pot\_eq(i)$, that is, the ones in which it could be mapped. Formally,

$$Pot(i) = \sum_{j \in Pot\_eq(i)} |j|.$$

In this case, $Pot(C.1) = |\mathrm{I_C}.2| + |\mathrm{I_C}.3| = 3$ and $Pot(C.2) = |\mathrm{I_C}.2| = 1$.

We are now ready to give the formula for the exposure coefficient $\mathcal{E}(i, \Lambda[i, *])$ with reference to the correct solution $\Lambda = \Lambda'$ as follows:

$$\mathcal{E}(i, \Lambda[i, *]) = \frac{1}{|i|} \cdot \sum_{j \in Pot\_eq(i)} \frac{1}{Pot(i)} \cdot \sum_k \Lambda[i, j][k] \cdot \frac{count(i)}{count(j)},$$

where the first factor models the uncertainty on the assignments of each member of i to the hash values; the second factor is the number of members of equivalence class i assigned to hash value in equivalence class j; while the third expresses, for each of those elements, the probability of guessing it right.

For instance, with respect to the correct matching, we have the following entries in the table illustrated above: $(\mathrm{C}.1, \mathrm{I_C}.3) = 1$, $(\mathrm{C}.1, \mathrm{I_C}.2) = 4$, $(\mathrm{C}.2, \mathrm{I_C}.3) = 1$, $(\mathrm{C}.2, \mathrm{I_C}.2) = 0$. A plaintext value in class C.1 can match with any hash value. When it matches with a hash value in class $\mathrm{I_C}.3$, there is a probability 1/3 of correctly identifying the value of a tuple, when it matches

with a hash value in class $I_C.2$, there is a probability $1/2$ of guessing the right value. Therefore, the exposure coefficient for values in equivalence class C.1 is

$$\mathcal{E}(\text{C.1}, \Lambda_1[\text{C.1}, *]) = \frac{1}{5} \cdot \left(1 \cdot \frac{1}{3} \cdot \frac{1}{3} + 4 \cdot \frac{1}{3} \cdot \frac{1}{2}\right).$$

Let us now consider the case of a solution Λ which is not the correct solution with respect to class i. Since Λ is not the correct solution either too few or too many elements of i might have been assigned to some j. If there are too many, there is no way to map the exceeding ones. If there are too few, the missing ones correspond to exceeding values mapped to some other classes. In either case, we need to consider only the number of values that have been correctly mapped, which corresponds to the ones in the correct solution Λ', the first case, and to the ones in the current solution Λ in the second case. This gives us the formula

$$\mathcal{E}(i, \Lambda[i, *]) = \frac{1}{|i|} \cdot \sum_{j \in Pot\_eq(i)} \frac{1}{Pot(i)} \cdot \min\left\{\sum_k \Lambda[i, j][k], \sum_k \Lambda'[i, j][k]\right\} \cdot \frac{count(i)}{count(j)}.$$

As an example, consider the second matching illustrated above, where $(\text{C.1}, I_C.3) = 3$, $(\text{C.1}, I_C.2) = 2$, $(\text{C.2}, I_C.3) = 0$, $(\text{C.2}, I_C.2) = 1$. Two of the three plaintext values in class C.1 matched with the hash value in class $I_C.3$ cannot be assigned to the correct value, and the third plaintext value can be identified with probability $1/3$. Moreover, the attacker has probability $1/2$ of correctly identifying each plaintext value in C.1 assigned to a hash value in $I_C.2$. In this case, the exposure coefficient for values in equivalence class C.1 is

$$\mathcal{E}(\text{C.1}, \Lambda_2[\text{C.1}, *]) = \frac{1}{5} \cdot \left(1 \cdot \frac{1}{3} \cdot \frac{1}{3} + 2 \cdot \frac{1}{3} \cdot \frac{1}{2}\right).$$

Finally, we observe that the exposure coefficient $\mathcal{E}(i)$ of values in equivalence class i can be obtained by averaging all the $\mathcal{E}(i, \Lambda[i, *])$ computed for *distinct* values of $\Lambda[i, *]$. Formally,

$$\mathcal{E}(i) = \frac{1}{|V|} \cdot \sum_{\{\Lambda[i,*] \in V\}} \mathcal{E}(i, \Lambda[i, *]),$$

where $V = \{\Lambda_1[i, *], \ldots, \Lambda_n[i, *]\}$ is the set (i.e., with elimination of duplicates) of all possible solutions. For instance, the exposure coefficient for class C.1 is

$$\mathcal{E}(\text{C.1}) = \mathcal{E}(i, \Lambda_1[\text{C.1}, *]) + \mathcal{E}(i, \Lambda_2[\text{C.1}, *]) = \frac{1}{5} \cdot \frac{1}{2} \cdot \left(\frac{7}{9} + \frac{4}{9}\right).$$

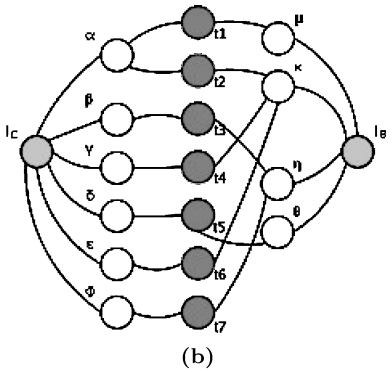
7. DB+DB^K WITH DIRECT ENCRYPTION

We now consider the situation where the attacker knows both the encrypted and the plaintext database. A typical scenario for this attack occurs when the content owner switches from no encryption to the use of encryption with indexing on the outsourced database. A malicious user with access to the database server may then be interested in reconstructing the correspondence between the plaintext and index values, to monitor the evolution of the database and

ENC_ACCOUNTS1

Enc_tuple	I_A	I_C	I_B
x4Z3tfX2ShOSM	π	α	μ
mNHg1oC010p8w	ϖ	α	κ
WsIaCvfyF1Dxw	ξ	β	η
JpO8eLTVgwV1E	ϱ	γ	κ
qctG6XnFNDTQc	ς	δ	θ
4QbqC3hxZHklU	Γ	ϵ	κ
3jO9DfoaU78Hs	τ	ϕ	η

(a)



(b)

Fig. 8. Relation ENC_ACCOUNTS1 (a) and the corresponding RCV-graph (b).

keep access to most of its content, independently of the strength of the encryption function adopted.

In this scenario, the attacker knows precisely the values' distribution and the relationships among them. Our model of the attack is based on the definition of RCV-graphs that we introduce in the following.

7.1 The RCV-Graph

Given a table $\mathcal{T}$ with attributes $A_1, A_2, \ldots, A_n$ and tuples $t_1, t_2, \ldots, t_m$, we build a three-colored undirected graph $G = (V, E)$ called the RCV-graph (i.e., the row–column-value-graph) as follows. The set of vertices V contains one vertex for every attribute (all of color "column"), one vertex for every tuple (all of color "row"), and one vertex for every distinct value in each of the attributes (all of color "value"); if the same value appears in different attributes, a distinct vertex is introduced for every attribute in which the value appears. The set of edges E is built as follows. First, we add edges connecting each vertex representing a value with the vertex representing the column in which the value appears. Second, we add edges connecting each vertex representing a value with the vertices representing tuples in which the value appears. To illustrate, consider table ENC_ACCOUNTS1 in Figure 2, which is repeated in Figure 8(a) for convenience, restricted to attributes `Customer` and `Balance`. There are two vertices labeled I_C and I_B for the attributes, seven vertices labeled $t_1 \ldots t_7$ for the tuples, and ten vertices labeled $\alpha \ldots \theta$ for the distinct values appearing in the attributes. The addition of all the edges produces the RCV-graph depicted in Figure 8(b).

An important property is that the RCV-graph built starting from the plaintext database is identical to the RCV-graph built starting from the encrypted database, since the cryptographic function only realizes a biunivocal mapping between plaintext and index values (in the relational model, the order of tuples, and the order of attributes within a relation are irrelevant). The identification of the correspondence between plaintext and index values requires then to establish a correspondence between the vertex labels and the plaintext values discussed in the following section.

7.2 RCV-Graph Automorphism

The identification of the correspondence between the labels on the graph $G = (V, E)$ and the plaintext values, when the plaintext database is known, can exploit information on the topological structure of the data that permits a more precise reconstruction than the one possible when the only information available is the distribution of values in each attribute (as in Section 5). In the example, it is possible to correctly identify the correspondence between label $\mathtt{I_C}$ and attribute `Customer`, label $\mathtt{I_B}$ and attribute `Balance`, and it is also possible to correctly identify the correspondence among all values but β and ϕ (t_3, t_7) and γ and ε (t_4, t_6). For the remaining vertices, it is only possible to obtain a probabilistic estimate of the correspondence.

While in the above example, we found the correspondence simply by inspecting the graph, in the general case, the complexity of this search is related to the number of automorphisms in the RCV-graph. An automorphism of a graph is an isomorphism of the graph with itself. Formally, an automorphism of a graph is a permutation Γ of the graph labels such that $G(V, E) = G(V^\Gamma, E)$ (i.e., $\forall e(v_i, v_j) \in E, e(v_i^\Gamma, {v_j}^\Gamma) \in E$). If the graph is colored (as in our case), nodes with different color cannot be exchanged by the permutation. The identical permutation trivially satisfies the relationship; then, at least one automorphism exists for any graph. When the RCV-graph presents only the trivial automorphism, the correspondence between the vertex labels and the plaintext values can be fully determined and the knowledge of the plaintext database permits a full reconstruction of the correspondence between plaintext and index values. When there are several automorphisms in the RCV-graph, the identification of a vertex can be uncertain, as there are many alternative ways to reconstruct the correspondence between the vertices. The RCV-graph shown in Figure 8(b) presents four automorphisms, which we represent here by the permutations of labels that characterize them. Each permutation is represented by a different order of the symbols in the following sequences.

$$
\begin{aligned}
&A_1.\{I_C, I_B, t_1, t_2, t_3, t_4, t_5, t_6, t_7, \alpha, \beta, \gamma, \delta, \epsilon, \phi, \mu, \kappa, \eta, \theta\}\\
&A_2.\{I_C, I_B, t_1, t_2, t_3, t_6, t_5, t_4, t_7, \alpha, \beta, \epsilon, \delta, \gamma, \phi, \mu, \kappa, \eta, \theta\}\\
&A_3.\{I_C, I_B, t_1, t_2, t_7, t_4, t_5, t_6, t_3, \alpha, \phi, \gamma, \delta, \epsilon, \beta, \mu, \kappa, \eta, \theta\}\\
&A_4.\{I_C, I_B, t_1, t_2, t_7, t_6, t_5, t_4, t_3, \alpha, \phi, \epsilon, \delta, \gamma, \beta, \mu, \kappa, \eta, \theta\}
\end{aligned}
$$

The four automorphisms derive from the choice in the order of the two vertices sets (t_4, γ, κ)-(t_6, ϵ, κ) and (t_3, β, η)-(t_7, ϕ, η).

The number of automorphisms is not a good measure of the protection against inference attacks, since situations with evidently different protection may be characterized by the same number of automorphisms.

For instance, starting from 1000 distinct values we can consider two distinct situations: (1) 900 values are fixed and 100 can interchange ($100! = 9.3 \times 10^{153}$ automorphisms), (2) there are 500 pairs of values that can interchange ($(2!)^{500} = 3.3 \times 10^{150}$ automorphisms); case (1) has a greater number of automorphisms, but higher exposure index (0.9 for case (1), 0.5 for case (2)).

Also, the number of automorphisms increases exponentially with the size of the graph and may reach considerable (and inexpressive) values even for

graphs of limited size. A more precise measure of protection, which considers the number of alternatives that are offered for the value of each label, is the following. For each value in a tuple in the database, there is a given probability of guessing it based on the knowledge of the plaintext database: if all the RCV-graph automorphisms do not permute the corresponding vertex, there is a probability ($p = 1$) of identifying its correct value. In general, if there are K automorphisms for the RCV-graph and in k of them the label assigned to vertex v_i is correct, there is a probability $p_i = k/K$ of correctly identifying the vertex (i.e., row and column vertices are ignored). Since the identification of the correspondence is of interest only for the vertices representing attribute values, the computation of the exposure coefficient is limited to these nodes. The probability of guessing right a generic value can be estimated by computing the average on all the vertices of the probability p_i of guessing each of the v_i correctly, thus obtaining the *attribute exposure coefficient* $\mathcal{E}(v_i) = \sum_{i=1}^{m} p_i/m$, where m is the number of vertices.

The automorphism problem has been extensively studied in the context of graph theory, and many results can be directly applied to our context. First, the set of automorphisms of a graph constitute a group (called the *automorphism group* of the graph), which, for undirected graphs like these, can be described by the coarsest *equitable partition* [McKay 1981] of the vertices, where each element of the partition (each subset appearing in the partition) contains vertices that can be substituted one for the other in an automorphism. The *Nauty* algorithm that identifies the automorphism group of the graph [McKay 1981] starts from a partition on the vertices that can be immediately derived grouping all the vertices with the same color and connected by the same number of edges. This partition is then iteratively refined, and a concise representation of all the automorphisms is produced. From the structure of the partition, it derives that all the vertices appearing in the generic partition element C_j are equivalently substitutable in all the automorphisms; from this observation, it derives that the probability p_i of a correct identification of a vertex $v_i \in C_j$ is equal to the inverse of the cardinality of C_j, $1/|C_j|$.

Then, for the identification of the $\mathcal{E}$, it is sufficient to identify the number of elements in the equitable partition and the total number of attribute vertices (i.e., it is not necessary to keep track of the number of vertices in each partition). In fact, with $|C_j|$ vertices in the partition element C_j, n elements in the equitable partition and a total number m of vertices, the exposure coefficient $\mathcal{E}(T)$ of the table is

$$\sum_{i=1}^{m} p_i/m = \sum_{j=1}^{n} \sum_{v_i \in C_j} p_i/m = \sum_{j=1}^{n} \sum_{v_i \in C_j} 1/(|C_j| \ m) = \sum_{j=1}^{n} 1/m = n/m. \qquad (2)$$

In the example, the equitable partition for attribute vertices is $\{(\alpha)(\beta, \phi)(\gamma, \epsilon)(\delta)(\mu)(\eta)(\kappa)(\theta)\}$, which gives $\mathcal{E} = 8/10 = 4/5$. As a check, the reader can verify on the RCV-graph that the vertices appearing in singleton elements are associated with $p_i = 1$ and those in the remaining elements are associated with $p_i = 1/2$. The average of p_i on all the vertices is therefore $2/3$.

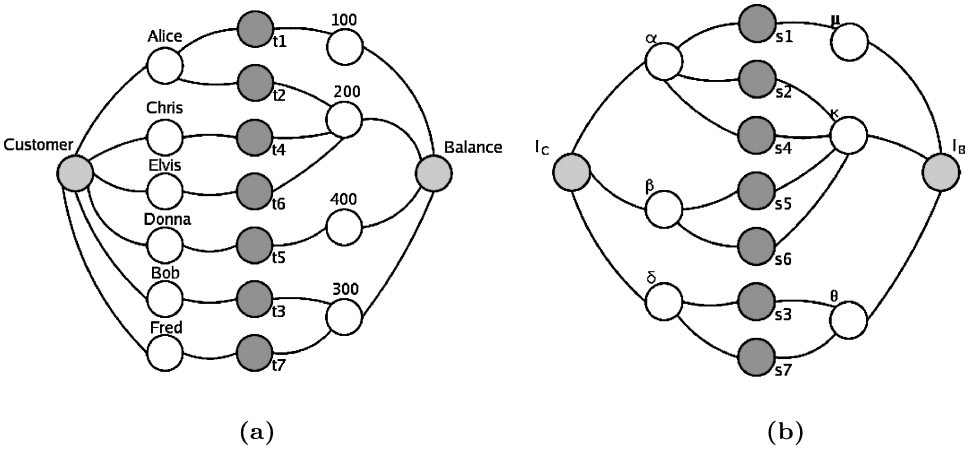


Fig. 9. A RCV representation of the plaintext table (a) and the corresponding encrypted table (b) illustrated in Figures 2(a)–(c), respectively.

When the structure of the database is completely obscured, as it occurs when all the attribute values appear once in the database, the $\mathcal{E}$ is minimal at $1/m$. The contribution of the knowledge of the plaintext database increases when the structure of the RCV-graph derived from it can impose restrictions that limit the number of options for a vertex, increasing the exposure coefficient.

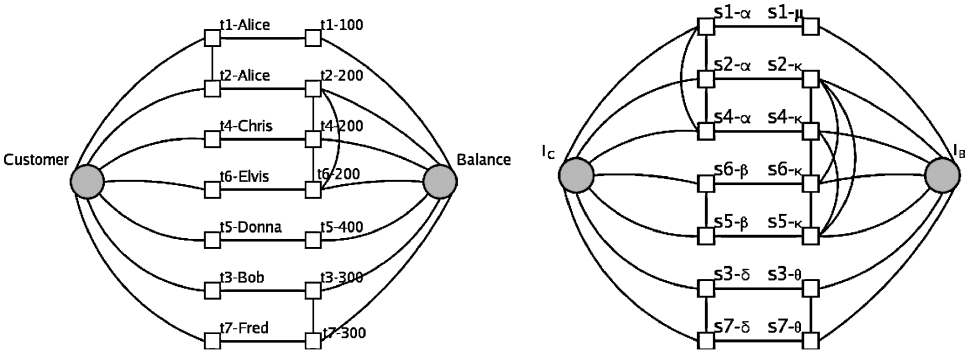
8. DB+DB^K WITH HASHING

In this scenario the attacker knows both the encrypted and plaintext databases. Our abstract models for computing the exposure coefficient extend the RCV-graph described in Section 7.

8.1 RCV Graphs and RCV Line Graphs

Given a table $\mathcal{T}$ with attributes $A_1, A_2, \ldots, A_n$ and tuples $t_1, t_2, \ldots, t_m$, we build a three-colored undirected graph, named the RCV-graph, as described in Section 7. As before, identifying the correct correspondence between plaintext and hash values requires finding a matching between each vertex of the plaintext RCV-graph (G_A) and a vertex of the corresponding encrypted RCV-graph (G_I). When collisions occur, the two graphs are not identical, as different vertices of the G_A may collapse to the same G_I vertex. For instance, Figure 9 illustrates the G_A and G_I for the tables in Figure 2(a) and Figure 2(c), respectively.

We can observe that the number of edges connecting row vertices to value vertices in G_A and G_I is the same. Therefore, the problem can be viewed as finding a correct matching between the edges of the G_A and the edges of the G_I. Following this observation, we substitute both G_A and G_I, with the exclusion of the column vertices, with their *line graphs*. The line graph $L(G)$ of a graph G is obtained by associating a vertex with each edge of the graph and connecting two vertices with an edge if and only if the corresponding edges of G meet at one or both endpoints [Whitney 1932]. Figure 10 illustrates the line graphs corresponding to the G_A and G_I in Figure 9.



(a) (b)

Fig. 10. L(G_A) (a) and L(G_I) (b) of the graphs illustrated in Figure 9.

The problem of finding a mapping between each vertex of L(G_A) and each vertex of L(G_I) is thus a graph–subgraph monomorphism, that is, the problem of finding whether L(G_A) is a (partial, or not-induced) subgraph of L(G_I). It is easy to verify that each graph monomorphism between these line graphs corresponds to a feasible mapping between the vertices of the RCV-graphs. Each value vertex of a RCV-graph with degree k corresponds to a clique of cardinality k in the corresponding line graph. Then we can assign a label i to each vertex in a clique corresponding to the plaintext value i. Each clique in the L(G_A) has to be mapped in a clique in the L(G_I) with greater than or equal cardinality (i.e., vertices in L(G_A) labeled with the same plaintext value have to be matched with vertices in L(G_I) labeled with the same hash value). This corresponds to mapping a set of plaintext value vertices $v_1, \ldots, v_k$ in a hash value vertex j whose degree is the sum of the degrees of $v_1, \ldots, v_k$. The number of L(G_A) and L(G_I) vertices is the same, so each plaintext value has to be matched with a hash value in a feasible graph monomorphism. Furthermore, the order of the vertices in the graph monomorphism describes a unique order between the edges of the RCV-graphs. Hence, each graph monomorphism corresponds to exactly one matching between the G_A and the G_I. The converse is also true: each feasible matching between the G_A and the G_I has a corresponding graph monomorphism between the L(G_A) and the L(G_I) in which the cliques representing each plaintext value are mapped in the clique representing the corresponding hash value.

The presence of apparently identical tuples in the encrypted table (tuples with the same value for every indexing attribute), such as β, κ, represents a symmetry condition. Also in this case, the number of symmetric solutions grows very quickly, making it hard for a search algorithm to enumerate all the feasible matchings. This problem can be handled in our line graph representation of the tables as follows. First, we can establish an order between the tuples in the plaintext table, leaving the uncertainty about the order of the tuples to the corresponding encrypted table. This can be done by representing the plaintext table with a directed graph: each edge in the L(G_A) is replaced by an arc, directed toward vertices belonging to previous tuples. Then, we can establish an order between the identical tuples in the L(G_I) as before, by substituting

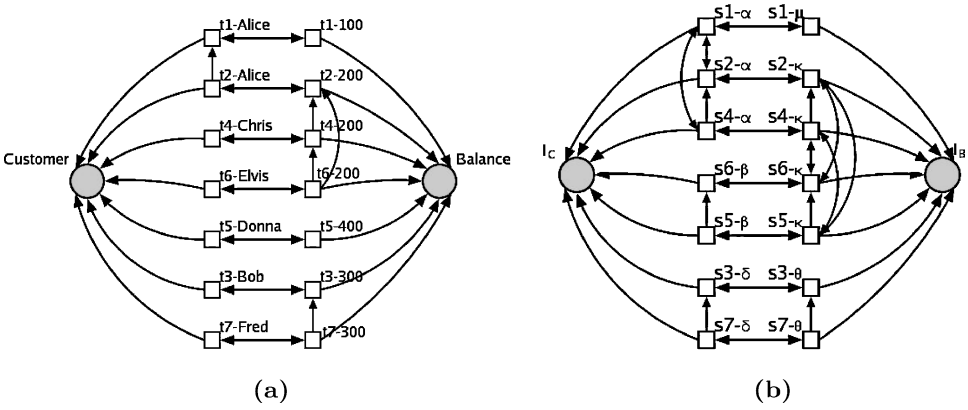


Fig. 11. Directed $L(G_A)$ (a) and $L(G_I)$ (b) corresponding to the line graphs in Figure 10.

each edge connecting the values of these tuples in the same column with an arc, directed toward the previous tuples. For instance, with respect to Figure 10(b), the edges connecting vertices s_3-δ and s_7-δ and vertices s_3-θ s_7-θ with two arcs. The remaining edges are replaced by two arcs, connecting the same pair of nodes, with opposite directions, except for the edges incident to column vertices, that can be replaced by an arc directed toward the column vertex. Both line graphs may be changed according to the above criterion, and the resulting directed graphs are reported in Figure 11.

Our efficient enumeration technique (Section 8.3) allows us to efficiently evaluate the exposure corresponding to specific solutions. Namely, whenever a monomorphism between the line graphs is found, a matching between each value v_i in the plaintext table and a value v_j in the encrypted table can be identified just by looking at the labels of vertices. As before, let $count(v_i)$ and $count(v_j)$ be the number of occurrences of each plaintext value v_i and each hash value v_j, respectively. If the matching is correct, an attacker has probability $count(v_i)/count(v_j)$ of identifying the plaintext value v_i of a tuple assigned with a hash value v_j. Let k_i be the number of different hash values to which plaintext value i can be assigned in a matching: the probability of guessing the right plaintext–hash value correspondence is $1/k_i$. The exposure of each plaintext value i is

$$\mathcal{E}(v_i) = \frac{1}{k_i} \frac{count(v_i)}{count(v_j)}.$$

Then, we define the attribute exposure coefficient as the average of these values

$$\mathcal{E} = \frac{\sum_{i \in I} \mathcal{E}(v_i)}{|I|},$$

where I is the set of the plaintext values. When there are no collisions, $count(v_i)/count(v_j) = 1$ and therefore the $\mathcal{E}$ reduces to the expression reported in Section 7. On the opposite, if we have m columns and n plaintext values in each column, when all the plaintext values of each column collide in the same hashed value, each $\mathcal{E}(v_i)$ value is $count(v_i) / (n \cdot m)$.

8.2 RCV Line Graphs Monomorphism

There is a theoretical difference between graph isomorphism and graph–subgraph isomorphism: neither a polynomial time algorithm is known for the graph isomorphism problem, nor the problem is known to be NP-complete. On the contrary, the graph–subgraph isomorphism was proved to be NP-complete for general graphs [Garey and Johnson 1979]. The graph–subgraph isomorphism problem is polynomial-time solvable for certain classes of graphs (e.g., planar graphs, trees, or bounded degree graphs), but our L(G_A) and L(G_I) do not have such a nice structure. When only one column is involved in the matching, the problem reduces to a multiple subset sum problem that is known to be NP-hard (see Section 6). Several enumeration (exponential time) algorithms for the graph–subgraph isomorphism problem have been proposed: Ullman [1976] devised a widely known enumeration method, which required $O(N!N^3)$ time and $O(N^3)$ space. More recently, Cordella et al. [1999, 2001] presented an algorithm, called VF2, that reduced the space complexity to $O(N)$ and the time complexity to $O(N!N)$. Their method allows to also manage directed graphs and attribute-relational graphs (ARGs), that is, graphs with semantical attributes associated with each vertex and edge. The matching between vertex and edges with *incompatible* attributes is not considered during the enumeration process. Also, the notion of incompatibility can be customized via user-defined comparing functions. Hence, each vertex of our linear graphs corresponding to an edge connecting a row vertex with a value vertex in the corresponding RCV-graphs can be labeled with color V (value), each column vertex can be labeled with color C (column), each arc connecting vertices in the same column can be labeled with color V, each arc connecting vertices in different columns can be labeled with color T (tuple), and each arc connecting V vertices with C vertices can be labeled with color C.

8.3 Efficiently Pruning the Search Tree Through Feasibility Conditions

Although our linear graphs do not have special properties, the corresponding RCV-graphs have a particular structure. They have $(2n + 1)$ layers: there is a layer for the vertices of color T; n layers for the vertices with color V, corresponding to values of the same column; and n layers made by each vertex labeled with color C. Furthermore, each V layer is only connected to the T layer and the corresponding C vertex. By removing the T layer, all the topological information about the structure of the database is lost. Therefore, the problem disaggregates in n^2 independent multiple subset sum problems (all the pairs between plaintext and encrypted columns have to be checked), that can be solved with the algorithm described in Figure 7. The idea is to match vertices of G_A with vertices of G_I just by looking at their degree: the sum of the degrees of a set of vertices $i_1, i_2, \ldots, i_k$ in G_A associated with the same vertex j in G_I has to be equal to the degree of j.

The solutions of these multiple subset sum problems can be computed before starting the enumeration process for identifying graph monomorphisms. Let S_j be the set of plaintext values $v_1, v_2, \ldots, v_k$ assigned to the hash value v_j in some solution of the multiple subset sum problems. An additional semantic

	Direct encryption	Hashing
Freq + DB^k	Quotient table	Multiple subset sum problem
DB + DB^k	RCV-graph	RCV line graph

Fig. 12. Abstract models supporting computation of exposure in the four attack scenarios.

attribute i is added to each vertex in the L(G_A), where i is the vertex label defined in Section 8.1. Analogously, an attribute S_j is added to each vertex in the L(G_I) marked with label j. A vertex labeled i of L(G_A) is compatible with a vertex labeled S_j of L(G_I) only if $i \in S_j$. This feasibility test greatly reduces the computation time required to enumerate graph monomorphisms, supporting experimental quantification of exposure for specific solutions.

9. EXPERIMENTAL RESULTS

The abstract models presented above, and summarized in Figure 12, can be used to obtain an indication of the exposure that characterizes generic databases, depending on the information available to the attacker and on the use of hashing. The results of three families of experiments are presented, corresponding to the scenarios considered, respectively, in Section 6, Section 7, and Section 8. The scenario "*Freq+DB^k* with direct encryption" presented in Section 5 has not been the subject of experiments, since it is not interesting from an algorithmic point of view (the behavior of an arbitrary database in this scenario can be precisely obtained by the analysis of the cardinality of the attribute values), and its behavior is dominated by the results, presented in Section 9.1, of the more complex scenario of Section 6.

9.1 Freq+DB^K with Hashing

We used an implementation in C of the algorithm for the subset-sum problem [Pisinger 1995] presented in Section 6. Other routines needed by the algorithm used for these experiments were implemented in ANSI C.

For the experimental analysis, we generated a series of random database instances considering two features of each database: the number N of distinct plaintext values and the collision factor cf, defined as the ratio $\frac{N}{M}$ between the number N of plaintext values and the number M of hash values. The collision factor shows how many values, on average, collide on the same hash value, which represents the expected increase in the size of the encrypted results.

In a first set of experiments, databases were randomly generated whose number of occurrences of each plaintext value followed a Zipf distribution. Databases have been considered with 15, 20, 25, and 40 plaintext values. The graph in Figure 13 represents the $\mathcal{E}$ for these databases as the collision factor increases. At the right-hand border of the graph, signs indicate the value $1/N$, that is, the optimal exposure of each database, occurring, for example, when all plaintext values collide on the same hash value. The $\mathcal{E}$ decreases from a value near 0.4 when no collision occurs to a value near 0.1 for a collision factor near to 3.0. A value near 0.1 is certainly adequate in this context, since the optimal is not very far.

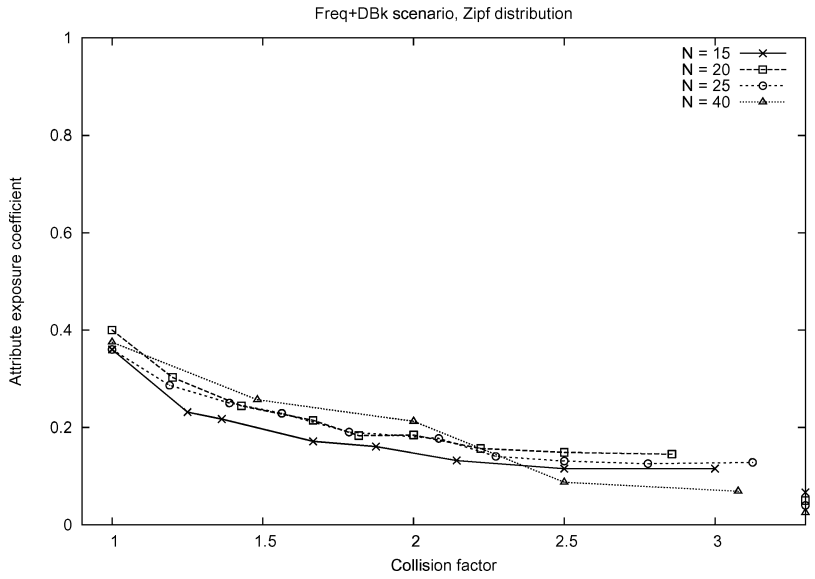


Fig. 13. Freq+DB^k scenario: $\mathcal{E}$ for databases whose number of occurrences of each value follows a Zipf distribution.

As a worst-case analysis, a second set of experiments was run, randomly generating databases with about 1200 tuples, where the number of occurrences f_i for each plaintext value was drawn as a random integer from a uniform distribution in $R = [1, \ldots, r]$.

Databases with a high $\frac{N}{r}$ ratio show very low exposure values even for a collision factor 1.0 (no collisions), due to the contribution given by large equivalence classes between plaintext values. To evaluate the protection from exposure in the most difficult scenario, the experiments considered instances with a ratio $\frac{N}{r}$ less than or equal to 1.0. These instances have, in general, singleton equivalence classes between plaintext values. Figure 14 presents a graph describing the $\mathcal{E}$ for these databases for increasing values of the collision factor. As before, the lowest exposure value $1/N$ appears on the right-hand border of the graph. The exposure of the database decreases as the ratio N/r increases (in fact, to high N/r values correspond to a high number of plaintext values in the same equivalence class). A collision factor of about 1.6 and 2.2 is sufficient to come close to the lowest exposure coefficient for databases with 50 and 35 plaintext values, respectively, while a higher collision factor is required for the databases with 20 values and number of occurrences in a wider range.

The results support the claim that a relatively modest increase in the cost of query execution, due to the use of a hash index with a low collision factor, produces a significant benefit in terms of protection from inference attacks.

The fact that the exposure is relatively high in the experiments when hashing is not used ($cf = 1$) mostly depends on the fact that the experiments have used databases characterized by a distribution of attribute values difficult to protect (this observation is confirmed by the low exposure that characterizes in Figure 15 the databases where only two indexes are used). Then, a possible

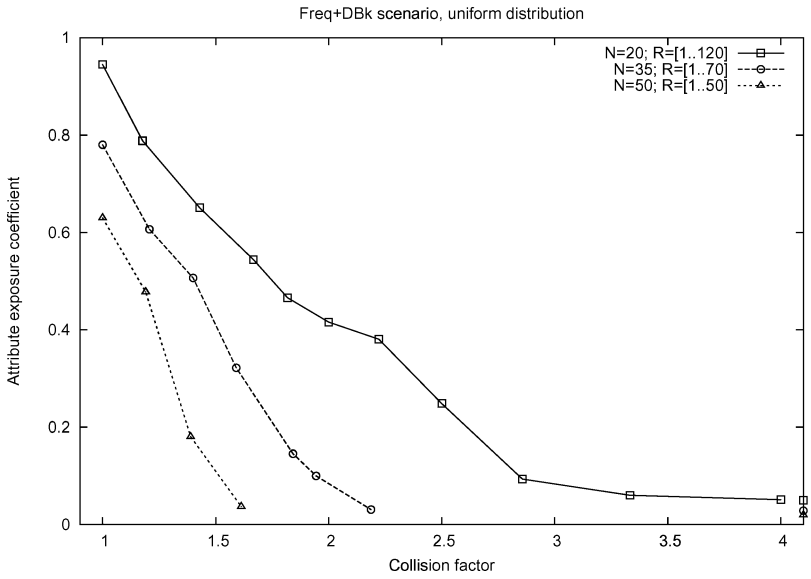


Fig. 14. Freq+DB^k scenario: $\mathcal{E}$ for databases whose number of occurrences of each value follows a uniform distribution.

strategy for the Freq+DB^k scenario may first evaluate the exposure coefficient in the specific database using the immediate formula in Section 6; if the result is above a chosen threshold, the database owner may switch to the use of a hashing function with a limited collision factor, always able to effectively increase the protection from inference attacks where the attacker knows the distribution of attribute values.

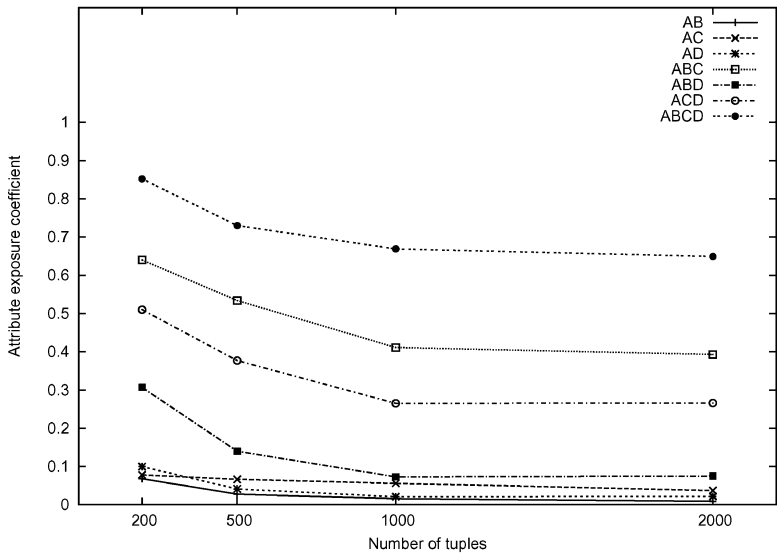
9.2 DB+DB^K with Direct Encryption

In this section we present the results of the experiments on the scenario where the attacker knows both the encrypted and the plaintext database. In Section 9.3 we shall consider the scenario with hashing at variable collision factor. Since direct encryption corresponds to the use of hashing with collision factor equal to 1, the results of this section could be considered redundant. The reason for carrying out a separate experiment on direct encryption is twofold. First, the greater efficiency allows us to analyze the behavior of a relatively complex database, illustrating the variation of exposure in this scenario with the increase in database size and with the variation in the number of indexed attributes. Second, it allows us to show the application of a tool specific for this scenario, which is much more efficient than the tool used for the analysis in Section 9.3. To optimize the exploration of a large database size, we considered in the experiments only one database instance following the Zipf distribution.

The tool we implemented for the experiments of this section takes as input a relational database and builds the RCV-graph that models it, with the construction presented in Section 7. The tool then invokes the Nauty algorithm [McKay 1981] on the RCV-graph, which is able to compute efficiently the automorphism

Attributes	Number of tuples 200	500	1,000	2,000
{A,B}	14/206	14/507	16/1007	18/2006
{A,C}	21/269	39/585	61/1093	83/2234
{A,D}	21/209	21/509	21/1009	44/2019
{A,B,C}	176/275	316/592	452/1100	881/2241
{A,B,D}	66/215	72/516	74/1017	152/2027
{A,C,D}	141/278	224/594	292/1103	599/2255
{A,B,C,D}	242/284	439/601	743/1110	1467/2262

(a)



(b)

Fig. 15. DB+DB^k with direct encryption scenario: Tabular representation of the experimental results (a) and their graphical representation (b) (curve labels refer to the initial of the attributes).

group (around 15 min on a 700 MHz Pentium III PC running Linux, for the greatest RCV-graph derived from a 2000 4-tuple table containing 2262 distinct values). The output of the program is then analyzed to reconstruct the equi-table partition that permits to determine the attribute exposure coefficient of the table.

In the experiments we used a table of four attributes A, B, C, and D and we applied the tool using a progressively greater number of tuples, up to 2000 tuples. We considered all the combinations of attributes containing attribute A; this choice was due to the fact that the analysis is meaningful only if at least two attributes are present in the table (otherwise, no correlation among attributes can be observed) and it was useful to keep a common attribute in all the experiments. The results appear in tabular form in Figure 15(a) and graphically in Figure 15(b).

The main result of these experiments is that the number of attributes used for the index has a great impact on the attribute exposure. With only two attributes, exposure coefficients tend to be quite low; when all the four attributes are used as index, the exposure is considerable.

Another question answered by the experiments is how the exposure evolves with an increase in the database size. What we observe is that the exposure decreases with the size of the database. The explanation is that as the number of tuples increases, a greater number of values become characterized by a distinct profile and are identifiable. At the same time, the new tuples introduce new values that are infrequent and indistinguishable, and this component prevails over the former.

Also, the increase in size of the database will generally make the task of the attacker computationally more expensive.

9.3 DB+DB^k with Hashing

For this experimentation, the VFLib2.0 C++ library [Foggia 2001] was used, implementing the VF2 and other algorithms for graph morphisms. This library is available on the Web, and it was used to develop a program that computes the exposure coefficient when the DB+DB^k scenario with hashing is analyzed. The experiments in Section 9.2 show that the exposure coefficient decreases as the number of tuples in the database increases; the following analysis for small databases can be considered as a worst-case scenario.

Two set of tests have been run. The first test analyzed the $\mathcal{E}$ for databases where the number of occurrences of each plaintext value follows the Zipf distribution. Figure 16 reports a graph representing the exposure $\mathcal{E}$ for a database with 12 plaintext values (about 30 tuples), when an increasing number of columns is considered. The lower bound on the exposure value $1/(NK)$, where N is the number of distinct plaintext values for each column and K is the number of columns, is indicated to the right-hand border of the graph. The $\mathcal{E}$ in all cases halves for a collision factor of about 3.0. As discussed before, the exposure grows as the number of considered columns increases, due to the larger amount of topological information available to the attacker. The $\mathcal{E}$ is still far from the lowest exposure value, but for small databases the correct matching between plaintext and encrypted indexes can always be identified, unless all the plaintext values collide in the same hash value.

The second test analyzed databases where the number of occurrences of each plaintext value follows a uniform distribution (Figure 17). Two kinds of databases have been considered: an instance with 8 plaintext values and a range [1..5] for the number of occurrences and an instance with 12 plaintext values and a range [1..4]. For both databases, the cases in which two or three columns are available to the attacker are considered. As before, these instances represent highly exposed databases. Although the exposure is still high when four columns are considered, the $\mathcal{E}$ is near to the lowest possible value when only two columns are available to the attacker.

The experiments show that the use of several index columns continues to have a significant impact on the exposure index. The use of hashing is able to

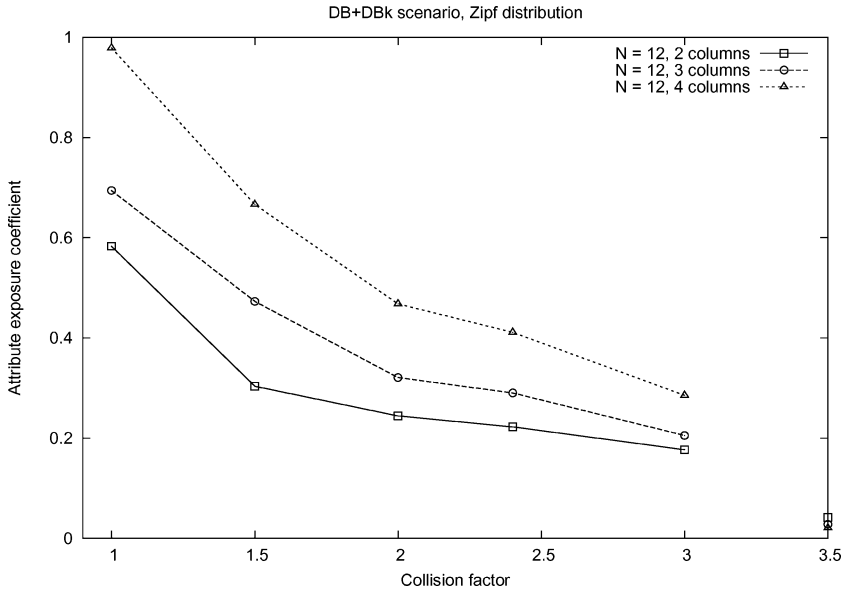


Fig. 16. DB+DB^k with hashing scenario: $\mathcal{E}$ for databases whose number of occurrences of each value follows a Zipf distribution.

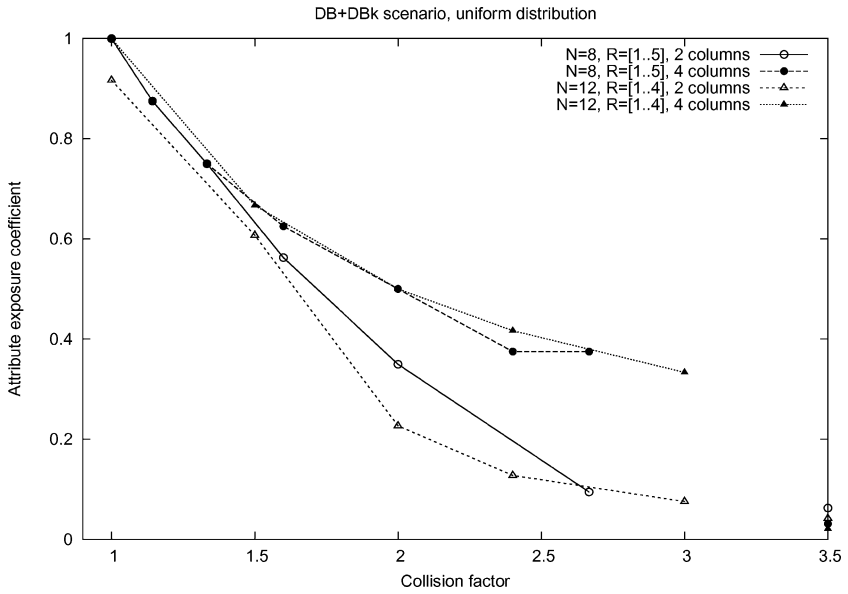


Fig. 17. DB+DB^k with hashing scenario: $\mathcal{E}$ for databases whose number of occurrences of each value follows a uniform distribution.

reduce considerably the exposure, but relatively high values persist, even with collision factors that in the Freq+DB^k scenario are able to protect the database from inference.

We conclude with an observation on the impact of database updates. The experiments have considered a static database and the results of the analysis permit to evaluate the collision factor that is adequate for a specific database. As the database evolves the results of the analysis may have to be reconsidered. In this situation, it should be appropriate to use an approach similar to the one typically adopted for the physical design of databases, where a representative database situation is used as a basis for the identification of the physical design, and a monitoring activity is executed that identifies when the changes to the database go beyond a given threshold, requiring a reconsideration of the physical design choices.

10. CONCLUSIONS

In this paper, we proposed a solution to the problem of secure database outsourcing on remote servers by providing a hash-based method for database encryption suitable for selection queries. Also, we gave a quantitative model for evaluating our method's vulnerability in different scenarios, showing that even straightforward direct encryption can provide an adequate level of protection against inference attacks, as long as a limited number of index attributes are used. To achieve a higher degree of protection against inference, it is convenient to use a hash function to encode index values. Indeed, our experiments show that even a hash function with a low collision factor, that is, with a limited impact on the size of the result set, can substantially decrease exposure to inference attacks as measured by a suitable exposure index. Our quantitative approach paves the way to the definition of a *query cost model*, capable of estimating the performance impact of the choice of attributes to be indexed and of the collision factors of their associated hash functions. Such a cost model should use as input a representative description of the set of queries that the server will have to manage, returning as output a quantitative evaluation of the decrease in performance introduced by the protection measures for that specific query set. The database designer will thus be able to compare this performance degradation to the degree of protection required for the database. When physically designing databases, database designers usually iteratively apply a cost model of the estimated workload, recalibrating physical design parameters until performance reaches a satisfactory level. Much in the same way, we expect that the database designer will use a query cost model to interact with the system in order to iteratively identify, the configuration that best balances the implicit protection requirements and database performance. Ideally, the database designer could be assisted by a more complex program, which will integrate performance and exposure quantitative models and use them to identify a solution optimizing overall system behavior. While such a system looks feasible in principle, we believe that in many environments characterized by complex security and performance requirements an interactive solution keeping the human

designer "in the loop" would be preferable. Another line of research stemming from this paper is defining of the set of queries that can efficiently be performed in the outsourced database setting. A well-known problem of hashing indexes is that they do not efficiently support execution of *range queries*, that is, queries that select tuples based on an interval of attribute values. Alternative indexing structures have been proposed in Hacigümüs et al. [2002a] and Damiani et al. [2003] that are able to support interval queries. The support for range queries presented in Hacigümüs et al. [2002a] unfortunately has a significant impact on inference exposure: due to the fact that the same index is associated with all the attribute values belonging to an interval, the number of possible assignments of attribute values to indexes is greatly reduced, permitting an easier reconstruction of the correspondence between values. It would be certainly interesting to consider the concrete impact that these techniques would have on the exposure index, extending the approach used in this paper. The problem with the approach using B+-trees presented in Damiani et al. [2003] is the deterioration in performance that it can introduce, due to need to execute of a series of queries to navigate the B+-tree in order to identify the tuples belonging to the interval. An alternative client-based solution was presented in Damiani et al. [2004]. A final consideration can be made on the defense against inference attacks where the attacker controls the encrypted database and knows plaintext updates that are applied to it. For instance, this may happen when the attacker is able to add a specific record to the encrypted database (e.g., an attacker working at the organization where an outsourced database of bank accounts is stored could open a bank account, thus triggering a known update the outsourced database) or when the attacker knows that the data owner will execute a specific update at a given moment. While in this paper we only took static analysis into account, it must be underlined that dynamic attacks could possibly rely on additional information that could facilitate the reconstruction of the correspondence between encrypted and cleartext values. A strategy that can be used to mitigate this attack is inserting random delays between queries, evenly distribute the database load, and even introducing a number of fake queries in order to hide the "real" activities of the database. This line of research is another interesting extension of our work.

REFERENCES

AGRAWAL, R., KIERNAN, J., SRIKANT, R., AND XU, Y. 2004. Order-preserving encryption for numeric data. In *Proceedings of the ACM SIGMOD 2004 Conference*, Paris, France. ACM Press, New York.

BOUGANIM, L. AND PUCHERAL, P. 2002. Chip-secured data access: Confidential data on untrusted servers. In *Proceedings of the 28th International Conference on Very Large Data Bases*, Hong Kong, China. Morgan Kaufmann, San Mateo, CA, 131–142.

BOYENS, C. AND GÜNTER, O. 2003. Using online services in untrusted environments—a privacy-preserving architecture. In *Proceedings of the 11th European Conference on Information Systems (ECIS '03)*, Naples, Italy.

CAPRARA, A., KELLERER, H., AND PFERSCHY, U. 2000. A ptas for the multiple subset sum problem with different knapsack capacities. *Inf. Process. Lett. 73*, 111–118.

CAPRARA, A., KELLERER, H., AND PFERSCHY, U. 2003. A 3/4-approximation algorithm for multiple subset sum. *J. Heuristics 9*, 99–111.

CORDELLA, L., FOGGIA, P., SANSONE, C., AND VENTO, M. 1999. Performance evaluation of the VF graph matching algorithm. In *Proceedings of the 10th International Conference on Image Analysis and Processing*, Venice, Italy. IEEE Computer Society Press, Los Alamitos, CA.

CORDELLA, L., FOGGIA, P., SANSONE, C., AND VENTO, M. 2001. An improved algorithm for matching large graphs. In *Proceedings of the 3rd IAPR TC-15 Workshop on Graph-Based Representations in Pattern Recognition*, Ischia, Italy, J. Jolion, W. Kropatsch, and M. Vento, Eds. 149–159.

DAMIANI, E., DE CAPITANI DI VIMERCATI, S., JAJODIA, S., PARABOSCHI, S., AND SAMARATI, P. 2003. Balancing confidentiality and efficiency in untrusted relational dbmss. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, Washington, DC. ACM Press, New York.

DAMIANI, E., DE CAPITANI DI VIMERCATI, S., PARABOSCHI, S., AND SAMARATI, P. 2004. Computing range queries on obfuscated data. In *Proceedings of the Information Processing and Management of Uncertainty in Knowledge-Based Systems*, Perugia, Italy.

DAVIDA, G., WELLS, D., AND KAM, J. 1981. A database encryption system with subkeys. *ACM Trans. Database Syst. 6,* 2 (June), 312–328.

DENNING, D. 1982. *Cryptography and Data Security*. Addison-Wesley, Reading, MA.

DOMINGO-FERRER, J. 1996. A new privacy homomorphism and applications. *Inf. Process. Lett. 60,* 5 (Dec.), 277–282.

DOMINGO-FERRER, J. AND HERRERA-JOANCONMARTÍ, J. 1998. A privacy homomorphism allowing field operations on encrypted data. *Jornades de Matemàtica Discreta i Algorísmica*.

FOGGIA, P. 2001. The vflib graph matching library, version 2.0. http://amalfi.dis.unina.it/graph/db/vflib-2.0/doc/vflib.html.

GAREY, M. AND JOHNSON, D. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York.

HACIGÜMÜS, H., IYER, B., LI, C., AND MEHROTRA, S. 2002a. Executing SQL over encrypted data in the database-service-provider model. In *Proceedings of the ACM SIGMOD'2002*, Madison, WI. ACM Press, New York.

HACIGÜMÜS, H., IYER, B., AND MEHROTRA, S. 2002b . Providing database as a service. In *Proceedings of the 18th International Conference on Data Engineering*, San Jose, CA. IEEE Computer Society Press, Los Alamitos, CA.

HACIGÜMÜS, H., IYER, B., AND MEHROTRA, S. 2004. Efficient execution of aggregation queries over encrypted relational databases. In *Proceedings of the 9th International Conference on Database Systems for Advanced Applications*. Springer, Jeju Island, Korea.

HACIGÜMÜS, H. AND MEHROTRA, S. 2004. Performance-conscious key management in encrypted databases. In *Proceedings of the 18th Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, Sitges, Catalonia, Spain. Kluwer Academic Publishers, Dordrecht.

JENSEN, C. 2000. Cryptocache: a secure sharable file cache for roaming users. In *Proceedings of the 9th Workshop on ACM SIGOPS European Workshop: Beyond the PC. New Challenges for the Operating System*, Kolding, Denmark. 73–78.

KLEIN, S., BOOKSTEIN, A., AND DEERWESTER, S. 1989. Storing text retrieval systems on CD-ROM: compression and encryption considerations. *ACM Trans. Inf. Syst. 7,* 3 (July), 230–245.

MCKAY, B. 1981. Practical graph isomorphism. *Congressus Numerantium 30*, 45–87.

PISINGER, D. 1995. Algorithms for knapsack problems. Ph.D. thesis, DIKU, University of Copenhagen, Copenhagen, Denmark. Report 95/1.

RIVEST, R., ADLEMAN, L., AND DERTOUZOS, M. 1978. Data banks and privacy homomorphisms. In *Foundations of Secure Computation*. Academic Press, Orlando, FL. 169–179.

SAMARATI, P. 2001. Protecting respondent's privacy in microdata release. *IEEE Trans. Knowledge Data Eng. 13,* 6 (Nov./Dec.), 1010–1017.

SONG, D., WAGNER, D., AND PERRIG, A. 2000. Practical techniques for searches on encrypted data. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, Oakland, CA. IEEE Computer Society Press, Los Alamitos, CA. 44–55.

ULLMAN, J. 1976. An algorithm for subgraph isomorphism. *J. ACM 23*, 31–42.

WALTON, J. 2002. Developing an enterprise information security policy. In *Proceedings of the 30th Annual ACM SIGUCCS Conference on User Services*, Providence, RI. ACM Press, New York.

WARD, J., O'SULLIVAN, M., SHAHOUMIAN, T., AND WILKES, J. 2002. Appia: Automatic storage area network fabric design. In *Proceedings of the Conference on File and Storage Technologies (FAST 2002)*. The USENIX Association, Monterey, CA.

WHITNEY, H. 1932. Congruent graphs and the connectivity of graphs. *Am. J. Math. 54*, 150–168.

Received May 2004; revised September 2004; accepted September 2004