Dynamic Group Key Management in Outsourced Databases

Alla Lanovenko and Huiping Guo

***Abstract*—Database outsourcing is becoming increasingly popular, which leads to a new scenario, called database-as-a-service where an organization's database is stored at an external service provider[14]. In such a scenario, it's very important to control the access to the database if the data owner wishes to publish his or her data for external use. Previous researchers had made many efforts in designing and querying encrypted outsourced databases aiming at protecting data on the server side. However, protecting outsourced database from the unauthorized access on the client side is still an open issue. Since, knowledge of the decryption key allows clients to not only access authorized data, but also the entire outsourced database, which violates data confidentiality and owners privacy. This paper addresses one of the major deficiencies of the outsourced database model: confidentiality. We propose a key management schema which is suitable for the dynamic environment. This schema can be efficiently applied to the outsourced databases and is based on the widely used Rivest-Shamir-Adelman (RSA) cryptographic algorithm.**

***Index Terms*— database-as-a-service(DAS), security, access control, outsourced database.**

I. Introduction

The Internet has made it possible for all computers to be connected to one another. It has facilitated an opportunity to provide data usage over the Internet, and has led to a new category of businesses called "application service providers" or ASP [2]. ASPs make it possible that worldwide can use data over the Internet. They can also provide storage and file access as services, that is, database as a service. Database as a service inherits all the advantages of the ASP model since a great number of organizations have their own database management systems (DBMS). The challenge is the feasibility of providing the next value-add layer in data management.

The solution is outsourced database model, which allows organizations to reduce cost by using the related hardware and software services provided by the service providers, without having to develop their own. In this model, database owners (e.g., an organization) outsource their data management needs to an external service provider. Organizations rely on the premises of the provider for the storage, maintenance, and retrieval of their data. A third-party database service provider hosts client's databases and offers mechanisms to efficiently create, update and query outsourced data.

Database outsourcing is becoming increasingly popular in recent years because it offers great advantages of cost reduction, team of expert resources and also better database protection to the organizations that outsource their data. First, it provides significant cost savings over the full time resources and consultants. Second, organizations receive better support system for the databases. Most of the outsource companies provides all day long support for critical databases, which ensures higher availability and more effective disaster protection. Perhaps more importantly, it offers a way to share the expertise of database professionals, thereby cutting the cost of managing a complex information infrastructure, which is important both for industrial and academic organizations [4]. For the public sector, this is effective solution for the unclassified type projects with limited budgets.

From the technological angle, the DAS model introduces numerous research challenges and thus has rapidly become one of the hot topics in the research community [14, 5, 11, 6, 13].

As mentioned, in DAS model, a database owner stores its private data at an external service provider, who is typically not fully trusted. Therefore, securing outsourced data, i.e. making it confidential, is one of the foremost challenges in this model. Basically, regardless of the untrusted server at the provider's side, the final goal that database owner's want is that they can use the outsourced database service as an in-house one. This includes a requirement that users can operate on outsourced data without a leak of sensitive information. This requirement in turn poses several additional challenges related to privacy-preserving for user's queries as well as for the outsourced data during the execution of operations at the untrusted server.

Overall, with an assumption that server side is not trusted and users of the database is trusted only with the data that they have privilege to access; the following security requirements must be met:

- Data confidentiality: outsiders and the server's operators (database administrators) cannot see the owner's outsourced data contents in any cases. Users of the database have only partial access to the outsourced data, they can not access whole database, but only the part that owner allowed them to see.
- Owner privacy: owner does not want the server to know about their queries and the returned results.
- Authentication and data integrity: database users must be ensured that data returned from the untrusted server is originated from the data owner and has not been tampered with.

The authors are with the Computer Science Department, California State University at Los Angeles, Los Angeles, CA 90032. Email: hpguo@calstatela.edu

The above security requirements are different from the traditional database security issues [29, 30] and can not be taking care of by build-in functions in existing database products such as Oracle, Microsoft SQL Server or IBM DB2.

In this paper, we concentrate on addressing the first two security objectives for the outsourced databases that come together with access control as discussed below.

These first two security objectives are obviously related to each other. To deal with the data confidentiality issue, outsourced data is usually encrypted before being stored at the external server. By encrypting the data, the database owner can ensure that no one except the permitted users can read the data. Although this solution can protect the data from outsiders as well as the server, it introduces difficulties in querying process: it is hard to protect the database owner privacy as performing queries over encrypted data.

The existing proposals for outsourced databases security only focus on protecting data on the server side, and assume the client has complete access to the query result [1, 8, 9, 11]. In these schemes, a single key is used to encrypt all tuples, so the knowledge of the key grants complete access to the whole database. Obviously, this type of assumption would not fit in real world, where the data owners often require enforcing access restrictions to different sets of users. In the real world, we can't allow users to see the whole database. There is some restricted information such as credit card information, bank account information, social security information of other users, which can be misused. Rather than using a single key to encrypt whole database, we can use different keys to encrypt different data. To access the encrypted data, users need to know the decryption algorithm and the specific decryption key to decrypt the data.

The existing group key management for dynamic environment suggests the group key has to be re-encrypted each time a user leaves or joins the group [1, 5]. However, re-encrypting the database whenever there is a change in the set of users is not feasible. Hence, this approach would be highly inefficient.

In this paper, we propose a secure group key management schema that is suitable for dynamically changing group environment. Our schema performs scalable encryption/decryption algorithm both at the server side and the client side using key pairs generated from the group keys based on most widely used Rivest-Shamir-Adelman (RSA) cryptographic algorithm. Without re-encrypting outsourced data each time group membership is changed, this schema ensures security and confidentiality to the database. It protects outsourced data not only from the untrusted server, but also solves the problem of unauthorized access of the group members that are evicted from the group and other intruders. We also present the experimental results, which demonstrate the applicability of our proposal.

The rest of this paper is organized as follows: Section 2 presents previous and related work. Section 3 is dedicated to presenting our contributions to solve the problem radically. Experimental results are shown in section 4 and section 5 concludes the paper.

II. REALTED WORK

Outsourced databases are typically composed of three entities that are database owner, database users (clients) and server. Database-as-a-service (DAS) model poses many significant challenges. In such model owner's database is stored at a third-party database service provider (server), which is not trusted. Untrusted server hosts owner's databases and offers mechanisms to efficiently query outsourced data.

The main problem with DAS model is that sensitive data are stored on a third party site which is not under the data owner's direct control; thus, data privacy and security can be put at risk. Ensuring an adequate level of protection to databases' content is therefore an essential part of any comprehensive security program. Database encryption 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 [13]. Database encryption prevents unauthorized users, including intruders braking into a network, from seeing sensitive data in databases; similarly, it allows database administrators to perform their task without being able to access sensitive information (e.g., sales or payroll figures) in plaintext. However, a simple solution that store only the encrypted database on the external server does not work, because it disables the ability of the external server to query the encrypted database. Since the external server is not trusted, it is required that data only be decrypted on the client side. To solve this problem, we need techniques which enable external servers to execute queries on encrypted data. In this way, all the relations involved in a query do not have to be sent to the client for query execution. To execute query over encrypted data different indexing methods have been proposed, each one suitable for the remote execution of a particular kind of query. The index of range technique proposed [2] to support efficient evaluation on the remote server of both equality and range predicate query. This work relies on partitioning client tables' attributes' domains 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.

In [10, 13] the authors present a method for databases based on hash. The proposed schemes are suitable for selection queries, but they do not support well interval-based queries. To enable interval-based queries, the authors in [3] adopt the B+-tree structures which are typically used in DBMSs. Furthermore, to allow aggregation queries on encrypted databases, privacy homomorphism is proposed in [15, 12]. In these schemes, an encrypted table is stored on the server with an index for each aggregation attribute, which is an attribute on which the aggregate operator is applied. By computing the aggregation on the server side and decrypting the result on the client side, an operation on an aggregation attribute can be evaluated. In this way, arithmetic operations but not comparison operations can be performed[7]. In order to support range queries on encrypted tables, an order preserving encryption schema is proposed in

[3]. This scheme only applies to integer values and the query results are complete but contain redundant tuples.

In the existing schemes for designing encrypted outsourced databases [2, 8, 9, 14], it is assumed that the entire database is encrypted with a single key and the users are granted the key. The assumption is only limited to protecting data on the server side and the users have complete access to the database. However, in real world, complete access to the encrypted outsourced data is not acceptable. It is desirable that the users can only have selective access to the encrypted data. Though DAS scenario has been studied in depth in the last few years, the problem of guaranteeing an efficient mechanism for implementing selective access to the remote database is still an open issue, despite the fact that the access control is an important entity in outsourced database security. Proposed access control mechanism [1] exploit data encryption by including authorization in the encrypted data themselves. In this way it enforces access restriction to deferent sets of users. The scheme uses different encryption keys to encrypt different data. Users have to use the encryption algorithm and the specific key to decrypt the encrypted data in order to access them. If the decryption key is differentiated based on the users' identity, different users are given different access rights.

This paper proposed a different method where users are grouped with the same access privileges and each tuple (or group) is encrypted with the key associated with the set of users that can access it. The proposed scheme is described in detail in the next section.

III. Dynamic Group Key Management in Outsourced Database

Previously proposed access control mechanism for the DAS model on the client side is limited to the static groups. Whenever group membership changes, outsourced database has to be re-encrypted in order to protect database contents. This solution is not efficient and does not fit real world applications where authorizations, users and objects can be dynamically changed. In this section, we present a novel group key management schema, which is suitable for dynamic groups. Our group key distribution schema is based on modification of widely used Rivest-Shamir-Adelman (RSA) cryptographic algorithm. This schema solves the database re-encrypting problem in the event when group membership changes dynamically (i.e, eviction of the group member). It efficiently solves the security problem of data confidentiality and owner privacy.

Our key distribution scheme is designed to keep database encrypted with the same group key while protecting database from the access of unauthorized users. We conduct group key distribution scheme in order to avoid the needs of database re-encryption in case of group evict or join events.

Below, we list the main notations (Table 3.1) used in the paper.

Notation	Description
N	Universe of all users
K	Set of all possible secrets
GR0	Initial user group
U	User
I	Group or user index or identifier
Ke	Group encryption key
Kd	Group decryption key
Kd1	Group subkey for the group users
Kd2	Group subkey for the server
DAS	Database-as-a-service model
C	Ciphertext or encrypted text
C'	Ciphertext
Ee	Encrypt with group encryption key
Dd	Decrypt with group decryption key
ASP	Application service provider
RSA	Rivest-Shamir-Adelman cryptographic algorithm
Mod	Modulus
n	A big number

Table 3.1 Notations

A. System description

All the users of the outsourced database are divided into different groups. Users with the same database access privilege are grouped together and can access the same part of the outsourced data. Each group of database users has a pair of keys and modulus. One key is called encryption key Ke and another key is a decryption key Kd. These keys and the modulus n (mod n) are generated by the database owner using RSA algorithm.

Encryption key Ke and decryption key Kd are kept secret by the database owner and never should be revealed neither to the group members, nor to the untrusted server.

- Group encryption key Ke is used by the database owner to encrypt a set of tuples in the database based on group access privilege such that

$$C = EKe [Data] \bmod n \quad (3.1)$$

Where C is ciphertext (or encrypted data), EKe is the encryption function that takes encryption key and plain text (Data), mod n, and produces ciphertext C.

- Group decryption key Kd is used by the owner to randomly generate a pair of group subkeys Kd1 and Kd2 such that

$$DKd [C] \bmod n \Leftrightarrow DKd1 [DKd2 [C]] \bmod n \quad (3.2)$$

Where function DKd2 [C] takes ciphertext and subkey. Kd2 and generates ciphertext C', and function DKd1 [C'] takes ciphertext C' and subkey Kd1, mod n and produces the same result as it would been produced by the function DKd [C] mod n, or by applying the decryption key Kd mod n. Thus, by applying decryption key Kd in a form of key pair Kd1 and Kd2 outsourced data can be decrypted on the user side:

$$Data = DKd [C] \bmod n \quad (3.3)$$

or the same result can be acquired as follows:

$$Data = DKd1 [DKd2 [C]] \bmod n \quad (3.4)$$

By hiding both encryption key and decryption key from the group members and from the untrusted server in this way, we are solving a major problem of access control technique

described before. This problem involves re-encrypting the outsourced databases by the owner in the case of dynamic group of users. In particular, whenever a user leaves a group, the database owner has to re-encrypt the database with a new key to prevent leak of information, so that the evicted user can not access the database anymore. This involves a lot of performance overhead and becomes practically impossible for large databases accessed by a dynamic group of users. To solve this problem, there should be some way of preventing the access of the database from the evicted user without having to re-encrypt the whole database frequently. Instead of disclosing the whole decryption key to the users only a decryption subkey is disclosed to them. This subkey can be changed and securely transmitted to all the authorized users whenever a group user is evicted from the group. Hence according to the proposed schema, by keeping the encryption and decryption keys secret from the group users, an evicted member will never be able to access the decrypted sensitive data from the encrypted outsourced database anymore.

B. *System architecture*

Our group key distribution schema uses centralized setting. There is a single trusted entity, database owner that decides and manages the system keys, including system setup and keys update. This setting is mostly suitable for one-to-many group where there is a single sender and a large number of receivers. Thus, it perfectly fits in the DAS model.

Dynamic group key management schema includes tree entities data owner, server and user, as shown in Figure 3.1:

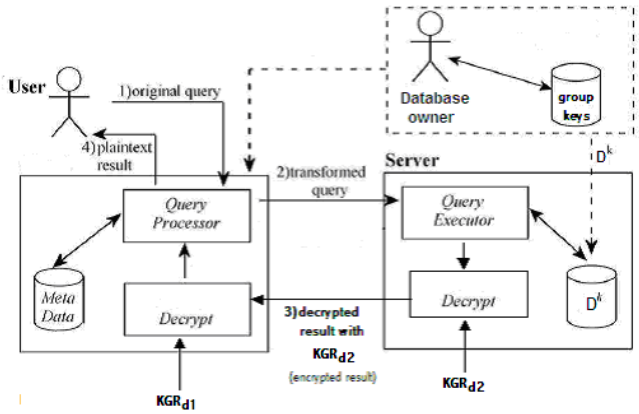

Figure 3.1: Dynamic group key management schema

The roles of the DAS entities in this dynamic group key management schema are as follows:

- Database owner: is responsible for producing, distributing, managing and updating group keys.
- Group User: decrypts the result from the server using the first part of the group decryption subkey Kd1 in the decryption algorithm in order to get the plaintext result.
- Server: is responsible for producing the query result on the encrypted database, decrypting the result with the second part of the group decryption subkey Kd2 and sending encrypted result to the group user.

C. *Group Key Distribution Model*

Let the set N denote the universe of all users, and let K denote the set of all possible secret keys in the system. There is a trusted entity, called the database owner that initializes the system, and an initial group of users GR0 = {U1, U2,… , Un} $\subseteq$ N. Each user Ui$\in$ GR0 holds a secret key K(GRi) $\subset$K. The index i is a unique group number identifier. In the groups initialization, database owner generates group keys, gives secret information to each initial user and the server by using secure communication protocol (ex. Diffie-Hellman Key agreement protocol).

After the initialization phase, the system is dynamic and its lifetime consists of lifetime of all the groups. Group keys GRi$\in$ K are shared by users belonging to a particular group i, while unauthorized users not belonging to this group do not share those keys. The collection of group keys {GR1, GR2, …, GRn} $\subset$K.

There are three events in the system: initialization, adding new group member, and evicting existing group member. These events will be described in following steps:

1) Initialization phase

Initialization phase establishes group keys. A group initialization is performed by the database owner. Recall that GRi is the initial group of n0 users.

a) The database owner uses RSA cryptographic algorithm to generate two keys: group encryption key (Ke , n) and group decryption key (Kd , n).
b) The database owner randomly splits decryption key Kd on two parts and produces two group subkeys Kd1 and Kd2.
c) The database owner sends to each user Ui $\in$ GRi a subkey Kd1 and modulus n. Group members hold a group subkey Kd1 and mod n as their secret key.
d) The database owner sends to the server a group subkey Kd2. The Server holds a group subkey Kd2 as group GRi secret key.
e) Database owner encrypts set of tuples to which group members have access with group GRi encryption key (Ke, mod n) and store them in the outsourced database.

2) Adding a group member

Adding a new member to a group does not require re-keying the group keys. While adding a new member Ui to a group GRi, users' group subkey Kd1 is sent by the database owner to the new group member through the secure channel. Upon receiving a group key Kd1 the new group member can access the server, query the outsourced database and using the group key Kd1 to decrypt the result received from the server. Table 3.2, shows the process of adding a new group member:

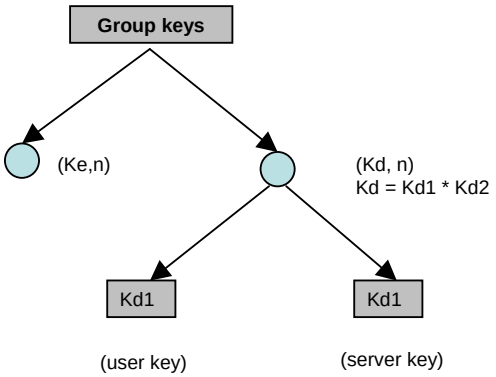

Figure 3.2: Group keys tree

If U_i = Add	
Input:	U_i , GR_i
Process:	adding a new user
Output:	$U_i \in GR_i$ obtains secret group sub key $K_{GR_{d1}}$

Table 3.2: Add event

3) Evicting a group member

The evict operation is most demanding because it necessitates a compromise re-keying, where the group subkeys have to be changed in order to prevent the evicted member of a group from accessing the restricted data. When a group member Ui is evicted from the group GRi database owner has to go through three steps:

a) In this step database owner has to establish new group subkeys Kd1 and Kd2 in order to prevent evicted user from accessing constrained data. The database owner generates a new pair of group GRi subkeys Kd1' and Kd2' from the existing group GRi decryption key Kd , such that Kd1' ≠ Kd1 and Kd2' ≠ Kd2.

b) The database owner sends the new users' group GRi subkey Kd1' to all the group members.

c) The database owner sends new group GRi subkey Kd2' to the server.

Every time group keys has to be send by the database owner to the database users or to the server through unsecured channels, they have to be secured with one of the cryptographic protocols (e.g. Diffie-Hellman key exchange cryptographic protocol).

If U_i = Evict	
Input:	U_i , GR_i , K_{GR_d}
Process:	evicting an existing user
Output:	$U_i \notin GR_i$
	$K_{GR_{d1}}'$, $K_{GR_{d2}}'$
	all $U_i \in GR_i$ obtain secret group subkey $K_{GR_{d1}}'$
	server obtains secret group subkey $K_{GR_{d2}}'$

Table 3.3: Evict event

Whenever a group user is evicted from the group, the database owner does not have to re-key the group decryption key Kd and group encryption key Ke because both keys are secret to the group members. Only the group subkeys Kd1 and Kd2 need to be changed since the knowledge of these subkeys grants the evicted group member access to the database. The advantage of this proposed key distribution schema is that the outsourced database owners do not have to re-encrypt all or part of the database every time a user is evicted from the group to protect the database from unauthorized access by evicted users. Re-encrypting outsourced database each time a group membership changes (member eviction) is not efficient. Our proposal solves this problem; hence, it improves the outsourced database performance.

D. Query processing

User Ui is belongs to a particular group GRi. Whenever user Ui needs to retrieve data from the outsourced database he or she generates a query (1). Each query is mapped onto a query on encrypted data, which is based on the use of indexing information associated with each relation in the encrypted database [2, 13] and (2) it is then sent to the server. Server on receiving queries from the group member Ui processes the query [2, 13]. The result of this query is a set of encrypted tuples, which had been encrypted by the database owner with the group GRi encryption key Ke such that C = EKe [Data] mod n.

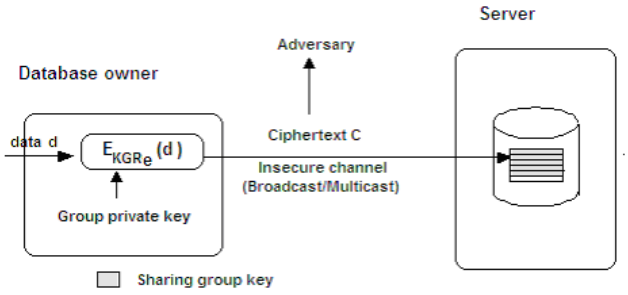

Figure 3.3: Owner encrypts database with group encryption key

The query result is decrypted by the server with group subkey Kd2 that is stored on the server such that C' = DKd2 [C] (C is the encrypted result from the database, C' is the result of the decryption function performed by the server that is still encrypted data, D is decryption). The user Ui on receiving the

result from the server, using decryption algorithm and group GRi subkey Kd1, decrypts data as follow: Data = DKd1 [C'], discard spurious tuples that may be part of the result and gets final plaintext result. All the mappings are based on catalogs stored at the user side that describe the structure of the remote database [4].

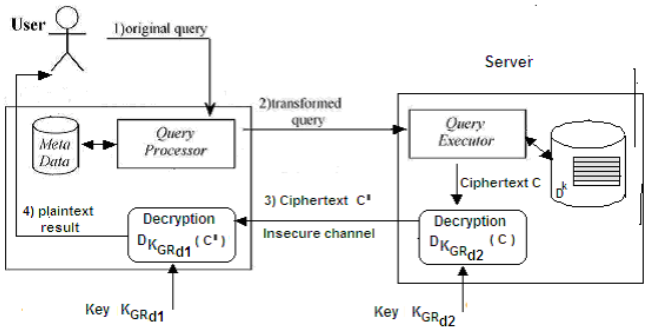

Figure 3.4: Query processing

IV. Experimental Results

For the experiment we used the programming language Java with the following characteristics: JDK1.4. We also used Microsoft Access database for the data storage. Real data are encrypted and stored in the database.In table Customer^k {etuple, CustomerInd, AccountInd, AmountInd, Subkey}, etuple column is an encrypted data, CustomerInd, AccountInd, AmountInd columns are indexes that used to retrieve the data and Subkey column is used to decrypt tuples in the table with group GR_i subkey $KGRd_2$. There are six tuples in the table. The members in group 1 can see the first four tuples while members in group 2 can only see the last two tuples.

The query "Select * from the Customer" of a group 1 member is mapped to the query "Select etuple from the Customer* and send to the server. The result of this query, all the tuples in the table are decrypted with group 1 subkey 2 and send to the user. On receiving the result from the server, user decrypts the query result with the group subkey 1 stored on the user side.

First four tuples in this test are accessible for the group 1 members and tuple five and six are accessible for group 2 members.

The above test is shown that even group 1 members can retrieve all the tuples in the table they can only view tuples 1, 2, 3, and 4, the tuples that they have privilege to see. The other tuples (e.g. tuples 5 and 6) are still unreadable to the group 1 members.

V. Conclusion

Selective access to the encrypted database is a very important issue in the DAS scenario, especially the encrypted database is published on un-trusted third party's server for external use. In this paper, we propose a scheme which enables users selective access to the encrypted database. Based on the modification of the RSA cryptographic algorithm, we present a key management schema for outsourced databases. This schema applies to the dynamic environment where authorizations and users can dynamically change.

References

[1] Ernesto Damiani, S. De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, Pierangela Samarati, Key Management for Multi-User Encrypted Databases, Proceedings of the 2005 ACM workshop on Storage security and survivability, November 2005.

[2] H. Hacig¨um¨us, B. Iyer, S. Mehrotra, and C. Li. Executing SQL over encrypted data in the database-service-provider model. In Proc. of the ACM SIGMOD'2002, Madison, WI, USA, June 2002.

[3] R. Agrawal, J. Kierman, R. Srikant, and Y. Xu. Order preserving encryption for numeric data. In Proc. of ACM SIGMOD 2004, Paris, France, June 2004.

[4] E. Damiani, S. De Capitani di Vimercati, S. Foresti, . Jajodia, S.Paraboschi, and P.Samarati. Metadata management in outsourced encrypted databases. In Proc. of the 2nd VLDB Workshop on Secure Data Management (SDM'05), Trondheim, Norway, September 2005.

[5] Alan T. Sherman and David A. McGrew. Key Establishment in Large Dynamic Groups Using One-Way Function Trees. IEEE Transactions on Software Eng., 29(5):444–458, 2003.

[6] D. Boneh and M. Franklin. Identity-based encryption from the weil pairing. In Proc. CRYPTO 01, pages 213–229, 2001.

[7] C. Boyens and O. Gunter. Using online services in untrusted environments – a privacy-preserving architecture. In Proc. of the 11th European Conference on Information Systems (ECIS '03), Naples, Italy, June 2003.

[8] R. Brinkman, J. Doumen, and W. Jonker. Using secret sharing for searching in encrypted data. In Proc. of the Secure Data Management Workshop, Toronto, Canada, August 2004.

[9] A. Ceselli, E. Damiani, S. De Capitani di Vimercati, S. Jajodia, S.Paraboschi, and P. Samarati. Modeling and assessing inference exposure in encrypted databases. ACM Transactions on Information and System Security (TISSEC), 8(1):119–152, February 2005.

[10] E. Damiani, S. De Capitani di Vimercati, M. Finetti, S. Paraboschi, P.Samarati, and S. Jajodia. Implementation of a storage mechanism for untrusted DBMSs. In Proc. of the Second International IEEE Security in Storage Workshop, Washington DC, USA, May 2003

[11] E. Mykletun, M. Narasimha, and G. Tsudik. Authentication and integrity in outsourced database. In Proc. of the 11th Annual Network and Distributed System Security Symposium, San Diego, CA, USA, February 2004.

[12] H. Hacig¨um¨us and S. Mehrotra. Performance-conscious key management in encrypted databases. In DBSec, pages 95–109, 2004.

[13] E. Damiani, S. De Capitani di Vimercati, S. Jajodia, S. Paraboschi, and P.Samarati. Balancing confidentiality and efficiency in untrusted relational DBMSs. In Proc. of the 10th ACM Conference on Computer and Communications Security, Washington, DC, USA, October 27-31 2003.

[14] E. Mykletun, M. Narasimha, G. Tsudik. Authentication and Integrity in Outsourced Databases. The 11th Annual Network and Distributed System Security Symposium – NDSS2004, San Diego, California, USA, February 5-6, 2004

[15] H. Hacig¨um¨us, B. Iyer, and S. Mehrotra. Efficient execution of aggregation queries over encrypted relational databases. In Proc. of the 9th International Conference on Database Systems for Advanced Applications, Jeju Island, Korea, March 2004.