This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

Distributed, concurrent, and independent access to encrypted cloud databases

Luca Ferretti, Michele Colajanni, and Mirco Marchetti

Abstract—Placing critical data in the hands of a cloud provider should come with the guarantee of security and availability for data at rest, in motion, and in use. Several alternatives exist for storage services, while data confidentiality solutions for the Database as a Service paradigm are still immature. We propose a novel architecture that integrates cloud database services with data confidentiality and the possibility of executing concurrent operations on encrypted data. This is the first solution supporting geographically distributed clients to connect directly to an encrypted cloud database, and to execute concurrent and independent operations including those modifying the database structure. The proposed architecture has the further advantage of eliminating intermediate proxies that limit the elasticity, availability and scalability properties that are intrinsic in cloud-based solutions. The efficacy of the proposed architecture is evaluated through theoretical analyses and extensive experimental results based on a prototype implementation subject to the TPC-C standard benchmark for different numbers of clients and network latencies.

Index Terms—Cloud, Security, Confidentiality, SecureDBaaS.

1 Introduction

In a cloud context, where critical information is placed in infrastructures of untrusted third parties, ensuring data confidentiality is of paramount importance [1], [2]. This requirement imposes clear data management choices: original plain data must be accessible only by trusted parties that do not include cloud providers, intermediaries, Internet; in any untrusted context data must be encrypted. Satisfying these goals has different levels of complexity depending on the type of cloud service. There are several solutions ensuring confidentiality for the *storage as a service* paradigm (e.g., [3]–[5]), while guaranteeing confidentiality in the *database as a service* (DBaaS) paradigm [6] is still an open research area. In this context, we propose SecureDBaaS as the first solution that allows cloud tenants to take full advantage of DBaaS qualities, such as availability, reliability, elastic scalability, without exposing unencrypted data to the cloud provider.

The architecture design was motivated by a threefold goal: to allow multiple, independent, and geographically distributed clients to execute concurrent operations on encrypted data, including SQL statements that modify the database structure; to preserve data confidentiality and consistency at the client and cloud level; to eliminate any intermediate server between the cloud client and the cloud provider. The possibility of combining availability, elasticity, and scalability of a typical cloud DBaaS with data confidentiality are demonstrated through a prototype of SecureDBaaS that supports the execution of concurrent and independent operations to the remote encrypted database from many geographically distributed clients as in any unencrypted DBaaS setup. To achieve these goals, SecureDBaaS integrates existing cryptographic schemes, isolation mechanisms, and novel strategies for management of encrypted metadata on the untrusted cloud database. This paper contains a theoretical discussion about solutions for data consistency issues due to concurrent and independent client accesses to encrypted data. In this context, we cannot apply fully homomorphic encryption schemes [7] because of their excessive computational complexity.

The SecureDBaaS architecture is tailored to cloud platforms and does not introduce any intermediary proxy or broker server between the client and the cloud provider. Eliminating any trusted intermediate server allows SecureDBaaS to achieve the same availability, reliability and elasticity levels of a cloud DBaaS. Other proposals (e.g., [8]–[11]) based on intermediate server(s) were considered impracticable for a cloud-based solution because any proxy represents a single point of failure and a system bottleneck that limits the main benefits (e.g., scalability, availability, elasticity) of a database service deployed on a cloud platform. Unlike SecureDBaaS, architectures relying on a trusted intermediate proxy do not support the most typical cloud scenario where geographically dispersed clients can concurrently issue read/write operations and data structure modifications to a cloud database.

A large set of experiments based on real cloud platforms demonstrate that SecureDBaaS is immediately applicable to any DBMS because it requires no modification to the cloud database services. Other studies where the proposed architecture is subject to the TPC-C standard benchmark for different numbers of clients and network latencies show that the performance of concurrent read and write operations not modifying the SecureDBaaS database structure is comparable to that of unencrypted

• *University of Modena and Reggio Emilia.*
E-mail: {luca.ferretti,michele.colajanni,mirco.marchetti}@unimore.it

Digital Object Indentifier 10.1109/TPDS.2013.154

cloud database. Workloads including modifications to the database structure are also supported by SecureDBaaS, but at the price of overheads that seem acceptable to achieve the desired level of data confidentiality. The motivation of these results is that network latencies, which are typical of cloud scenarios, tend to mask the performance costs of data encryption on response time. The overall conclusions of this paper are important because for the first time they demonstrate the applicability of encryption to cloud database services in terms of feasibility and performance.

The remaining part of this paper is structured as following. Section 2 compares our proposal to existing solutions related to confidentiality in cloud database services. Section 3 and Section 4 describe the overall architecture and how it supports its main operations, respectively. Section 5 reports some experimental evaluation achieved through the implemented prototype. Section 6 outlines the main results. Space limitation requires us to postpone the assumed security model in Appendix A, to describe our solutions to concurrency and data consistency problems in Appendix B, to detail the prototype architecture in Appendix C.

2 Related work

SecureDBaaS provides several original features, that differentiate it from previous work in the field of security for remote database services.

- It guarantees data confidentiality by allowing a cloud database server to execute concurrent SQL operations (not only read/write, but also modifications to the database structure) over encrypted data.
- It provides the same availability, elasticity, and scalability of the original cloud DBaaS because it does not require any intermediate server. Response times are affected by cryptographic overheads that for most SQL operations are masked by network latencies.
- Multiple clients, possibly geographically distributed, can access concurrently and independently to a cloud database service.
- It does not require a trusted broker or a trusted proxy because tenant data and metadata stored by the cloud database are always encrypted.
- It is compatible with the most popular relational database servers, and it is applicable to different DBMS implementations because all adopted solutions are database agnostic.

Cryptographic file systems and secure storage solutions represent the earliest works in this field. We do not detail the several papers and products (e.g., Sporc [3], Sundr [4], Depot [5]) because they do not support computations on encrypted data.

Different approaches guarantee some confidentiality (e.g., [12], [13]) by distributing data among different providers and by taking advantage of secret sharing [14]. In such a way, they prevent one cloud provider to read its portion of data, but information can be reconstructed by colluding cloud providers. A step forward is proposed in [15], that makes it possible to execute range queries on data and to be robust against collusive providers. SecureDBaaS differs from these solutions as it does not require the use of multiple cloud providers, and makes use of SQL-aware encryption algorithms to support the execution of most common SQL operations on encrypted data.

SecureDBaaS relates more closely to works using encryption to protect data managed by untrusted databases. In such case, a main issue to address is that cryptographic techniques cannot be naïvely applied to standard DBaaS because DBMS can only execute SQL operations over plaintext data.

Some DBMS engines offer the possibility of encrypting data at the filesystem level through the so called Transparent Data Encryption feature [16], [17]. This feature makes it possible to build a trusted DBMS over untrusted storage. However, the DBMS is trusted and decrypts data before their use. Hence, this approach is not applicable to the DBaaS context considered by SecureDBaas, because we assume that the cloud provider is untrusted.

Other solutions, such as [18], allow the execution of operations over encrypted data. These approaches preserve data confidentiality in scenarios where the DBMS is not trusted, however they require a modified DBMS engine and are not compatible with DBMS software (both commercial and open source) used by cloud providers. On the other hand, SecureDBaaS is compatible with standard DBMS engines, and allows tenants to build secure cloud databases by leveraging cloud DBaaS services already available. For this reason, SecureDBaaS is more related to [9] and [8] that preserve data confidentiality in untrusted DBMSs through encryption techniques, allow the execution of SQL operations over encrypted data, and are compatible with common DBMS engines. However, the architecture of these solutions is based on an intermediate and trusted proxy that mediates any interaction between each client and the untrusted DBMS server. The approach proposed in [9] by the same authors of the DBaaS model [6], works by encrypting blocks of data instead of each data item. Whenever a data item that belongs to a block is required, the trusted proxy needs to retrieve the whole block, to decrypt it, and to filter out unnecessary data that belong to the same block. As a consequence, this design choice requires heavy modifications of the original SQL operations produced by each client, thus causing significant overheads on both the DBMS server and the trusted proxy. Other works [10], [11] introduce optimization and generalization that extend the subset of SQL operators supported by [9], but they share the same proxy-based architecture and its intrinsic issues. On the other hand, SecureDBaaS allows the execution of operations over encrypted data through SQL-aware encryption algorithms. This technique, initially proposed in CryptDB [8], makes it possible to execute operations

over encrypted data that are similar to operations over plaintext data. In many cases, the query plan executed by the DBMS for encrypted and plaintext data is the same.

The reliance on a trusted proxy that characterize [9] and [8] facilitates the implementation of a secure DBaaS, and is applicable to multi-tier Web applications, which are their main focus. However, it causes several drawbacks. Since the proxy is trusted, its functions cannot be outsourced to an untrusted cloud provider. Hence, the proxy is meant to be implemented and managed by the cloud tenant. Availability, scalability, and elasticity of the whole secure DBaaS service are then bounded by availability, scalability, and elasticity of the trusted proxy, that becomes a single point of failure and a system bottleneck. Since high availability, scalability and elasticity are among the foremost reasons that lead to the adoption of cloud services, this limitation hinders the applicability of [9] and [8] to the cloud database scenario. SecureDBaaS solves this problem by letting clients connect directly to the cloud DBaaS, without the need of any intermediate component and without introducing new bottlenecks and single points of failure.

A proxy-based architecture requiring that any client operation should pass through one intermediate server is not suitable to cloud-based scenarios, in which multiple clients, typically distributed among different locations, need concurrent access to data stored in the same DBMS. On the other hand, SecureDBaaS supports distributed clients issuing independent and concurrent SQL operations to the same database and possibly to the same data. SecureDBaaS extends our preliminary studies [19] showing that data consistency can be guaranteed for some operations by leveraging concurrency isolation mechanisms implemented in DBMS engines, and identifying the minimum isolation level required for those statements. Moreover, we now consider theoretically and experimentally a complete set of SQL operations represented by the TPC-C standard benchmark [20], in addition to multiple clients, and different client-cloud network latencies that were never evaluated in literature.

3 Architecture Design

SecureDBaaS is designed to allow multiple and independent clients to connect directly to the untrusted cloud DBaaS without any intermediate server. Figure 1 describes the overall architecture. We assume that a *tenant* organization acquires a cloud database service from an untrusted DBaaS provider. The tenant then deploys one or more machines (*Client 1* through *N*) and install a *SecureDBaaS client* on each of them. This client allows a user to connect to the cloud DBaaS to administer it, to read and write data, and even to create and modify the database tables after creation.

We assume the same security model that is commonly adopted by the literature in this field (e.g., [8], [9]), where: tenant users are trusted, the network is untrusted, and the cloud provider is honest-but-curious, that is, cloud service operations are executed correctly, but tenant information confidentiality is at risk. For these reasons, tenant data, data structures, and metadata must be encrypted before exiting from the client. A thorough presentation of the security model adopted in this paper is in Appendix A.

The information managed by SecureDBaaS includes *plaintext data*, *encrypted data*, *metadata*, and *encrypted metadata*. *Plaintext data* consist of information that a tenant wants to store and process remotely in the cloud DBaaS. To prevent an untrusted cloud provider from violating confidentiality of tenant data stored in plain form, SecureDBaaS adopts multiple cryptographic techniques to transform plaintext data into *encrypted tenant data*, and *encrypted tenant data structures* because even the names of the tables and of their columns must be encrypted. SecureDBaaS clients produce also a set of *metadata* consisting of information required to encrypt and decrypt data as well as other administration information. Even metadata are encrypted and stored in the cloud DBaaS.

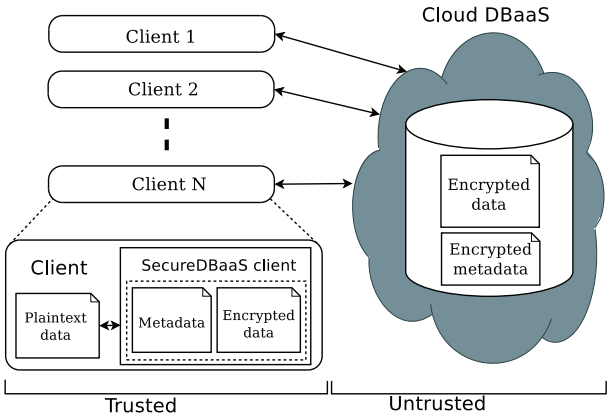

Fig. 1. SecureDBaaS architecture.

SecureDBaaS moves away from existing architectures that store just tenant data in the cloud database, and save metadata in the client machine [9] or split metadata between the cloud database and a trusted proxy [8]. When considering scenarios where multiple clients can access the same database concurrently, these previous solutions are quite inefficient. For example, saving metadata on the clients would require onerous mechanisms for metadata synchronization, and the practical impossibility of allowing multiple clients to access cloud database services independently. Solutions based on a trusted proxy are more feasible, but they introduce a system bottleneck that reduces availability, elasticity and scalability of cloud database services.

SecureDBaaS proposes a different approach where all data and metadata are stored in the cloud database. SecureDBaaS clients can retrieve the necessary metadata from the untrusted database through SQL statements, so that multiple instances of the SecureDBaaS client can access to the untrusted cloud database independently with the guarantee of the same availability and scalability

properties of typical cloud DBaaS. Encryption strategies for tenant data, and innovative solutions for metadata management and storage are described in the following two subsections.

3.1 Data management

We assume that tenant data are saved in a relational database. We have to preserve the confidentiality of the stored data and even of the database structure because table and column names may yield information about saved data. We distinguish the strategies for encrypting the database structures and the tenant data.

Encrypted tenant data are stored through *secure tables* into the cloud database. To allow transparent execution of SQL statements, each plaintext table is transformed into a secure table because the cloud database is untrusted. The name of a secure table is generated by encrypting the name of the corresponding plaintext table. Table names are encrypted by means of the same encryption algorithm and an encryption key that is known to all the SecureDBaaS clients. Hence, the encrypted name can be computed from the plaintext name. On the other hand, column names of secure tables are randomly generated by SecureDBaaS, hence even if different plaintext tables have columns with the same name, the names of the columns of the corresponding secure tables are different. This design choice improves confidentiality by preventing an adversarial cloud database from guessing relations among different secure tables through the identification of columns having the same encrypted name.

SecureDBaaS allows tenants to leverage the computational power of untrusted cloud databases by making it possible to execute SQL statements remotely and over encrypted tenant data, although remote processing of encrypted data is possible to the extent allowed by the *encryption policy*. To this purpose, SecureDBaaS extends the concept of *data type*, that is associated to each column of a traditional database by introducing the *secure type*. By choosing a *secure type* for each column of a secure table, a tenant can define fine-grained encryption policies, thus reaching the desired trade-off between data confidentiality and remote processing ability. A *secure type* is composed by three fields: *data type*, *encryption type*, and *field confidentiality*. The combination of the *encryption type* and of the *field confidentiality* parameters defines the *encryption policy* of the associated column.

The **data type** represents the type of the plaintext data (e.g., int, varchar). The **encryption type** identifies the encryption algorithm that is used to cipher all the data of a column. It is chosen among the algorithms supported by the SecureDBaaS implementation. As in [8], SecureDBaaS leverages several SQL-aware encryption algorithms that allow the execution of statements over encrypted data. It is important to observe that each algorithm supports only a subset of SQL operators. These features are discussed in Appendix C. When SecureDBaaS creates an encrypted table, the data type of each column of the encrypted table is determined by the encryption algorithm used to encode tenant data. Two encryption algorithms are defined *compatible* if they produce encrypted data that require the same column data type.

As a default behavior, SecureDBaaS uses a different encryption key for each column, hence equal values stored in different columns are transformed into different encrypted representations. This design choice guarantees the highest confidentiality level, because it prevents an adversarial cloud provider to identify data that are repeated in different columns. However, to allow remote processing of SQL statements over encrypted data, sometimes it is required to encrypt different columns by means of the same encryption key. Common examples are the *join* queries and the *foreign key* constraint.

The **field confidentiality** parameter allows a tenant to define explicitly which columns of which secure table should share the same encryption key (if any). SecureDBaaS offers three *field confidentiality* attributes:

- **Column** (COL) is the default confidentiality level that should be used when SQL statements operate on one column; the values of this column are encrypted through a randomly generated encryption key that is not used by any other column.
- **Multi-column** (MCOL) should be used for columns referenced by join operators, foreign keys, and other operations involving two columns; the two columns are encrypted through the same key.
- **Database** (DBC) is recommended when operations involve multiple columns; in this instance, it is convenient to use the special encryption key that is generated and implicitly shared among all the columns of the database characterized by the same *secure type*.

The choice of the field confidentiality levels make it possible to execute SQL statements over encrypted data while allowing a tenant to minimize key sharing.

3.2 Metadata management

Metadata generated by SecureDBaaS contain all the information that is necessary to manage SQL statements over the encrypted database in a way transparent to the user. Metadata management strategies represent an original idea because SecureDBaaS is the first architecture storing all metadata in the untrusted cloud database together with the encrypted tenant data. SecureDBaaS uses two types of metadata.

- **Database metadata** are related to the whole database. There is only one instance of this metadata type for each database.
- **Table metadata** are associated with one *secure table*. Each table metadata contains all information that is necessary to encrypt and decrypt data of the associated secure table.

This design choice makes it possible to identify which metadata type is required to execute any SQL statement

so that a SecureDBaaS client needs to fetch only the metadata related to the secure table/s that is/are involved in the SQL statement. Retrieval and management of database metadata are necessary only if the SQL statement involves columns having the field confidentiality policy equal to *database*. This design choice minimizes the amount of metadata that each SecureDBaaS client has to fetch from the untrusted cloud database, thus reducing bandwidth consumption and processing time. Moreover, it allows multiple clients to access independently metadata related to different secure tables, as we discuss in Section 4.3 and Appendix B.

Database metadata contain the encryption keys that are used for the secure types having the field confidentiality set to *database*. A different encryption key is associated with all the possible combinations of *data type* and *encryption type*. Hence, the database metadata represent a keyring and do not contain any information about tenant data.

The structure of a *table metadata* is represented in Figure 2. Table metadata contain the name of the related

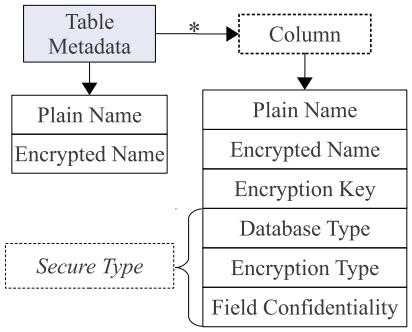

Fig. 2. Structure of table metadata.

secure table and the unencrypted name of the related plaintext table. Moreover, table metadata include *column metadata* for each column of the related secure table. Each column metadata contain the following information.

- **Plain name**: the name of the corresponding column of the plaintext table.
- **Coded name**: the name of the column of the secure table. This is the only information that links a column to the corresponding plaintext column because column names of secure tables are randomly generated.
- **Secure type**: the secure type of the column, as defined in Section 3.1. This allows a SecureDBaaS client to be informed about the data type and the encryption policies associated to a column.
- **Encryption key**: the key used to encrypt and decrypt all the data stored in the column.

SecureDBaaS stores metadata in the *metadata storage table* that is located in the untrusted cloud as the database. This is an original choice that augments flexibility, but opening two novel issues in terms of efficient data retrieval and data confidentiality. To allow SecureDBaaS clients to manipulate metadata through SQL statements, we save database and table metadata in a tabular form. Even metadata confidentiality is guaranteed through encryption. The structure of the metadata storage table is shown in Figure 3. This table uses one row for the database metadata, and one row for each table metadata.

Metadata Storage Table

ID	*Encrypted Metadata*	*Control Structure*
MAC('.'+Db)	Enc(Db metadata)	MAC(Db metadata)
MAC(T1)	Enc(T1 metadata)	MAC(T1 metadata)
MAC(T2)	Enc(T2 metadata)	MAC(T2 metadata)

Fig. 3. Organization of database metadata and table metadata in the metadata storage table.

Database and table *metadata* are encrypted through the same encryption key before being saved. This encryption key is called *master key*. Only trusted clients that already know the master key can decrypt the metadata and acquire information that is necessary to encrypt and decrypt tenant data. Each metadata can be retrieved by clients through an associated *ID*, which is the primary key of the metadata storage table. This ID is computed by applying a Message Authentication Code (MAC) function to the name of the object (database or table) described by the corresponding row. The use of a deterministic MAC function allows clients to retrieve the metadata of a given table by knowing its plaintext name.

This mechanism has the further benefit of allowing clients to access each metadata independently, which is an important feature in concurrent environments. In addition, SecureDBaaS clients can use caching policies to reduce the bandwidth overhead.

4 Operations

In this section we outline the setup setting operations carried out by a database administrator (DBA), and we describe the execution of SQL operations on encrypted data in two scenarios: a naïve context characterized by a single client, and realistic contexts where the database services are accessed by concurrent clients.

4.1 Setup phase

We describe how to initialize a SecureDBaaS architecture from a cloud database service acquired by a *tenant* from a cloud provider. We assume that the DBA creates the metadata storage table that at the beginning contains just the database metadata, and not the table metadata. The DBA populates the database metadata through the SecureDBaaS client by using randomly generated encryption keys for any combinations of *data types* and *encryption types*, and stores them in the *metadata storage table* after encryption through the *master key*. Then, the DBA distributes the *master key* to the legitimate users. User access control policies are administrated by the

DBA through some standard data control language as in any unencrypted database.

In the following steps, the DBA creates the tables of the encrypted database. He must consider the three field confidentiality attributes (COL, MCOL, DBC) introduced at the end of the Section 3. Let us describe this phase by referring to a simple but representative example shown in Figure 4, where we have three secure tables named ST1, ST2 and ST3. Each table STi ($i = 1, 2, 3$) includes an *encrypted table* Ti that contains *encrypted tenant data*, and a *table metadata* Mi. (Although in the reality the names of the columns of the secure tables are randomly generated, for the sake of simplicity, this figure refers to them through C1-CN.)

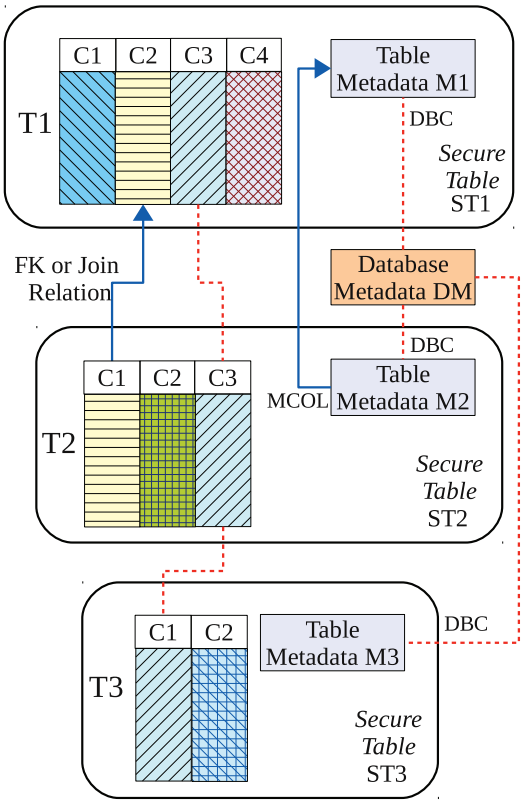

Fig. 4. Management of the encryption keys according to the field confidentiality parameter.

For example, if the database has to support a join statement among the values of T1.C2 and T2.C1, the DBA must use the MCOL field confidentiality for T2.C1 that references T1.C2 (solid arrow). In such a way, SecureDBaaS can retrieve the encryption key specified in the column metadata of T1.C2 from the metadata table M1 and can use the same key for T2.C1. The solid arrow from M2 to M1 denotes that they explicitly share the encryption algorithm and the key.

When operations (e.g., algebraic, order comparison) involve more than two columns, it is convenient to adopt the DBC field confidentiality. This has a twofold advantage: we can use the special encryption key that is generated and implicitly shared among all the columns of the database characterized by the same *secure type*; we limit possible consistency issues in some scenarios characterized by concurrent clients (see Appendix B). For example, the columns T1.C3, T2.C3 and T3.C1 in Figure 4 share the same *secure type*. Hence, they reference the *database metadata*, as represented by the dashed line, and use the encryption key associated to their data and encryption type. As they have the same data and encryption types, T1.C3, T2.C3 and T3.C1 can use the same encryption key even if no direct reference exists between them. The database metadata already contain the encryption key K associated with the data and the encryption types of the three columns, because the encryption keys for all combinations of data and encryption types are created in the initialization phase. Hence K is used as the encryption key of the T1.C3, T2.C3 and T3.C1 columns and copied in M1, M2 and M3.

4.2 Sequential SQL operations

We describe the SQL operations in SecureDBaaS by considering an initial simple scenario in which we assume that the cloud database is accessed by one client. Our goal here is to highlight the main processing steps, hence we do not take into account performance optimizations and concurrency issues that will be discussed in Section 4.3 and Appendix B.

The first connection of the client with the cloud DBaaS is for authentication purposes: SecureDBaaS relies on standard authentication and authorization mechanisms provided by the original DBMS server. After the authentication, a user interacts with the cloud database through the SecureDBaaS client. SecureDBaaS analyzes the original operation to identify which tables are involved and to retrieve their metadata from the cloud database. The metadata are decrypted through the master key and their information is used to translate the original plain SQL into a query that operates on the encrypted database.

Translated operations contain neither plaintext database (table and column names) nor plaintext tenant data. Nevertheless, they are valid SQL operations that the SecureDBaaS client can issue to the cloud database. Translated operations are then executed by the cloud database over the *encrypted tenant data*. As there is a one-to-one correspondence between plaintext tables and encrypted tables, it is possible to prevent a trusted database user from accessing or modifying some tenant data by granting limited privileges on some tables. User privileges can be managed directly by the untrusted and encrypted cloud database. The results of the translated query that includes encrypted tenant data and metadata are received by the SecureDBaaS client, decrypted, and delivered to the user. The complexity of the translation process depends on the type of SQL statement.

4.3 Concurrent SQL operations

The support to concurrent execution of SQL statements issued by multiple independent (and possibly geographically distributed) clients is one of the most important

benefits of SecureDBaaS with respect to state-of-the-art solutions. Our architecture must guarantee consistency among encrypted tenant data and encrypted metadata, because corrupted or out-of-date metadata would prevent clients from decoding encrypted tenant data resulting in permanent data losses. A thorough analysis of the possible issues and solutions related to concurrent SQL operations on encrypted tenant data and metadata is contained in the Appendix B). Here, we remark the importance of distinguishing two classes of statements that are supported by SecureDBaaS: SQL operations not causing modifications of the database structure, such as read, write, update; operations involving alterations of the database structure through creation, removal and modification of database tables (*Data Definition Layer* operators).

In scenarios characterized by a static database structure, SecureDBaaS allows clients to issue concurrent SQL commands to the encrypted cloud database without introducing any new consistency issues with respect to unencrypted databases. After metadata retrieval, a plaintext SQL command is translated into one SQL command operating on encrypted tenant data. As metadata do not change, a client can read them once and cache them for further uses, thus improving performance.

SecureDBaaS is the first architecture that allows concurrent and consistent accesses even when there are operations that can modify the database structure. In such cases, we have to guarantee the consistency of data and metadata through isolation levels, such as the snapshot isolation [21], that we demonstrate can work for most usage scenarios.

5 Experimental results

We demonstrate the applicability of SecureDBaaS to different cloud DBaaS solutions by implementing and handling encrypted database operations on emulated and real cloud infrastructures. The present version of the SecureDBaaS prototype supports PostgreSQL, MySql and SQL Server relational databases. As a first result, we can observe that porting SecureDBaaS to different DBMS required minor changes related to the database connector, and minimal modifications of the codebase. We refer to the Appendix C for an in-depth description of the prototype implementation.

Other tests are oriented to verify the functionality of SecureDBaaS on different cloud database providers. Experiments are carried out in Xeround [22], Postgres Plus Cloud Database [23], Windows SQL Azure [24], and also on an IaaS provider, such as Amazon EC2 [25], that requires a manual setup of the database. The first group of cloud providers offer ready-to-use solutions to tenants, but they do not allow a full access to the database system. For example, Xeround provides a standard MySql interface and proprietary APIs that simplify scalability and availability of the cloud database, but do not allow a direct access to the machine. This prevents the installation of additional software, the use of tools, and any customization. On the positive side, SecureDBaaS using just standard SQL commands can encrypt tenant data on any cloud database service. Some advanced computation on encrypted data may require the installation of custom libraries on the cloud infrastructure. This is the case of Postgres Plus Cloud that provides SSH access to enrich the database with additional functions.

The next set of experiments evaluate the performance and the overheads of our prototype. We use the Emulab [26] testbed that provides us a controlled environment with several machines, assuring repeatability of the experiments for the variety of scenarios to consider in terms of workload models, number of clients and network latencies.

As the workload model for the database, we refer to the TPC-C benchmark [20]. The DBMS server is PostgreSQL9.1 deployed on a quad-core Xeon having 12GB of RAM. Clients are connected to the server through a LAN where we can introduce arbitrary network latencies to emulate WAN connections that are typical of cloud services. The experiments evaluate the overhead of encryption, compare the response times of plain vs. encrypted database operations, and analyze the impact of network latency. We consider two TPC-C compliant databases with 10 warehouses that contain the same number of tuples: plain tuples consist of 1046MB data, while SecureDBaaS tuples have size equal to 2615MB because of encryption overhead. Both databases use repeatable read (snapshot) isolation level [27].

In the first set of experiments, we evaluate the overhead introduced when one SecureDBaaS client executes SQL operations on the encrypted database. Client and database server are connected through a LAN where no network latency is added.

To evaluate encryption costs, the client measures the execution time of the 44 SQL commands of the TPC-C benchmark. Encryption times are reported in the histogram of the Figure 5 that has a logarithmic Y-axis. TPC-C operations are grouped on the basis of the class of transaction: Order Status, Delivery, Stock Level, Payment, New Order. From this figure, we can appreciate that the encryption time is below 0.1ms for the majority of operations, and below 1ms for almost all operations but two. The exceptions are represented by two operations of the *Stock level* and *Payment* transactions where the encryption time is two orders of magnitude higher. This high overhead is caused by the use of the order preserving encryption that is necessary for range queries [28] (see Appendix C).

To evaluate the performance overhead of encrypted SQL operations, we focus on the most frequently executed SELECT, INSERT, UPDATE and DELETE commands of the TPC-C benchmark. In the Figures 6 and 7, we compare the response times of SELECT and DELETE, and UPDATE and INSERT operations, respectively. The Y-axis reports the boxplots of the response times expressed in ms (at a different scale), while the X-axis identifies the SQL operations. In SELECT, DELETE, and

Fig. 5. Encryption times of TPC-C benchmark operations grouped by transaction class.

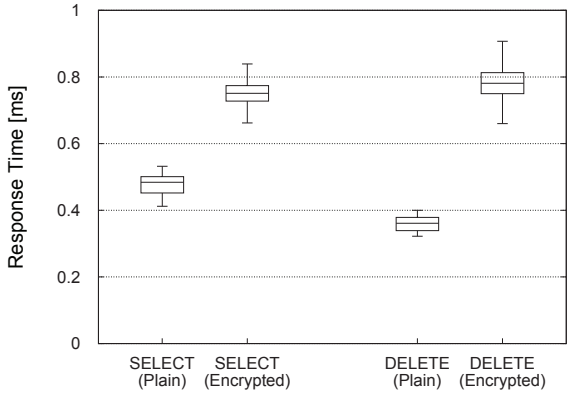

Fig. 6. Plain vs. encrypted SELECT and DELETE operations.

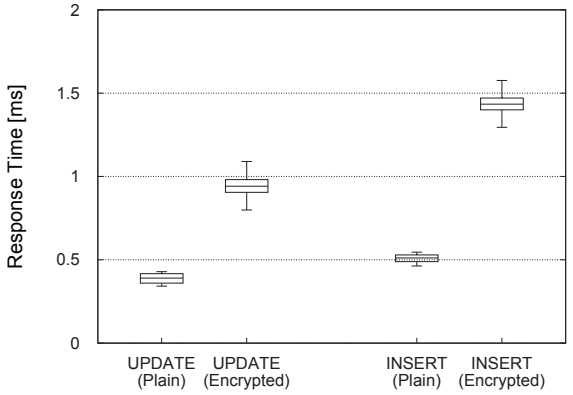

Fig. 7. Plain vs. encrypted UPDATE and INSERT operations.

Network delay	SQL command	Plaintext response time	Encrypted response time	Overhead (absolute and percentage)
LAN	SELECT	0.478 ms	0.753 ms	0.275 ms 57%
	DELETE	0.369 ms	0.783 ms	0.414 ms 112%
	UPDATE	0.397 ms	0.951 ms	0.554 ms 140%
	INSERT	0.517 ms	1.442 ms	0.925 ms 179%
20 ms	SELECT	20.67 ms	20.94 ms	0.27 ms 1.31%
	DELETE	20.66 ms	20.97 ms	0.31 ms 1.50%
	UPDATE	20.67 ms	21.12 ms	0.45 ms 2.18%
	INSERT	20.85 ms	21.61 ms	0.76 ms 3.65%
40 ms	SELECT	40.64 ms	40.90 ms	0.26 ms 0.64%
	DELETE	40.65 ms	40.92 ms	0.27 ms 0.66%
	UPDATE	40.62 ms	41.08 ms	0.46 ms 1.13%
	INSERT	40.82 ms	41.56 ms	0.74 ms 1.81%
80 ms	SELECT	80.76 ms	80.97 ms	0.21 ms 0.26%
	DELETE	80.67 ms	81.01 ms	0.34 ms 0.42%
	UPDATE	80.65 ms	81.09 ms	0.44 ms 0.55%
	INSERT	80.86 ms	81.63 ms	0.77 ms 0.95%

TABLE 1
Response times and overheads of SQL operations for different network latencies

UPDATE operations, the response times of SecureDBaaS SQL commands is almost doubled, while the INSERT operation is, as expected, more critical from the computational point of view and it achieves a tripled response time with respect to the plain version. This higher overhead is motivated by the fact that an INSERT command has to encrypt all columns of a tuple, while an UPDATE operation encrypts just one or few values.

The second set of the experiments is oriented to evaluate the impact of network latency and concurrency on the use of a cloud database from geographically distant clients. To this purpose, we emulate network latencies through the traffic shaping utilities available in the Linux kernel by introducing synthetic delays from 20ms to 150ms in the client-server connection. These values are representative of round-trip times in continental (in the ranges 40-60ms) and inter-continental (in the ranges 80-150ms) connections [29], that are expected when a cloud-based solution is deployed. Table 1 reports the response times of the most frequent SQL operations in the plain and encrypted cases for 20ms, 40ms and 80ms latencies. The last column of this table also reports the absolute and percentage overhead introduced by SecureDBaaS.

These experimental results demonstrate that the response times of the SQL operations issued to a remote database is dominated by network latencies even in well connected regions. Each response time is two orders of magnitude higher than the corresponding time of a plain SQL operation in a LAN environment. Thanks to this effect, the overhead of SecureDBaaS for the most common SELECT operation falls from 57% to 1.31% and to 0.26% in correspondence of network latencies equal to 20 ms and 80 ms, respectively.

The last set of experiments assess the performance of SecureDBaaS in realistic cloud database scenarios, as well as its ability to support multiple, distributed and independent clients. The testbed is similar to that described previously, but now the runs are repeated

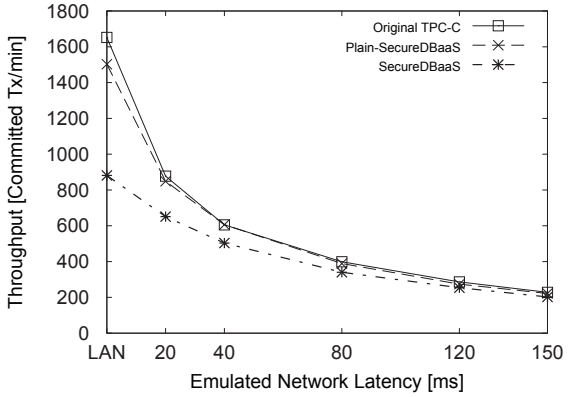

Fig. 8. TPC-C performance (20 concurrent clients)

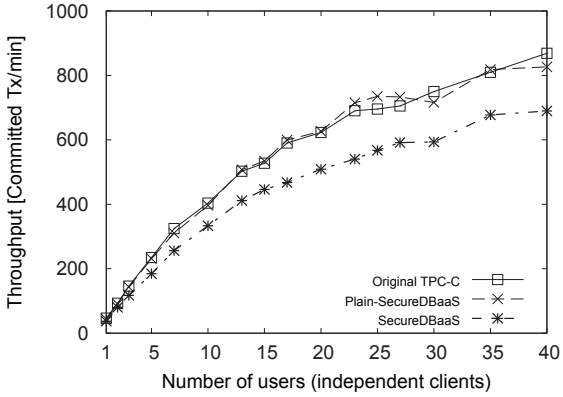

Fig. 9. TPC-C performance (latency equal to 40 ms)

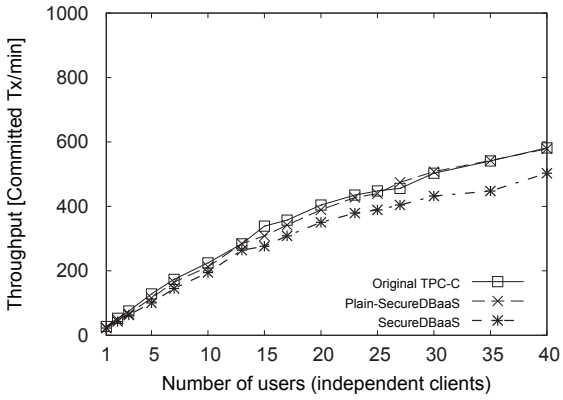

Fig. 10. TPC-C performance (latency equal to 80 ms)

by varying the number of concurrent clients (from 1 to 40) and the network latencies (from plain LAN to delays reaching 150 ms). All clients execute concurrently the benchmark for 300 seconds. The results in terms of throughput refer to three types of database operations:

- **Original TPC-C**: the standard TPC-C benchmark;
- **Plain-SecureDBaaS**: SecureDBaaS that use plain encryption, that is, all SecureDBaaS functions and data structures with no encryption; it allows us to evaluate the overhead of SecureDBaaS without the cost of cryptographic operations;
- **SecureDBaaS**: SecureDBaaS referring to the highest confidentiality level.

Figure 8 shows the system throughput referring to 20 clients issuing requests to SecureDBaaS as a function of the network latency. The Y-axis reports the number of committed transactions per minute during the entire experiment. This figure shows two important results:

- if we exclude the cryptographic costs, SecureDBaaS does not introduce significant overheads. This can be appreciated by verifying that the throughput of Plain-SecureDBaaS and Original TPC-C overlies for any realistic Internet delay (>20ms);
- as expected, the number of transactions per minute executed by SecureDBaaS are lower than those referring to Original TPC-C and Plain-SecureDBaaS, but the difference rapidly decreases as the network latency increases to the extent that is almost nullified in any network scenario that can be realistically referred to a cloud database context.

Figures 9 and 10 show the throughput for increasing numbers of concurrent clients in contexts characterized by 40ms and 80ms network latencies, respectively. These measures are optimistic representations of continental and intercontinental delays. The Y-axis represents the number of committed TPC-C transactions per minute executed by the clients. The trends of the SecureDBaaS lines are close to those of the Original TPC-C database, thus demonstrating that SecureDBaaS encrypted database does not affect scalability with respect to the plain database. Even more important, the network latencies tend to mask cryptographic overheads for any number of clients. For example, the overheads of SecureDBaaS with 40 concurrent clients decreases from 20% in a 40ms scenario to 13% in a realistic scenario where the client-server latency is equal to 80ms. This result is important because it confirms that SecureDBaaS is a valid and practical solution for guaranteeing data confidentiality in real cloud database services.

6 Conclusions

We propose an innovative architecture that guarantees confidentiality of data stored in public cloud databases. Unlike state of the art approaches, our solution does not rely on an intermediate proxy that we consider a single point of failure and a bottleneck limiting availability and scalability of typical cloud database services. A large part of the research includes solutions to support concurrent SQL operations (including statements modifying the database structure) on encrypted data issued by heterogenous and possibly geographically dispersed clients. The proposed architecture does not require modifications to the cloud database, and it is immediately applicable to existing cloud DBaaS, such as the experimented PostgreSQL Plus Cloud Database [23], Windows Azure [24] and Xeround [22]. There are no theoretical and practical

limits to extend our solution to other platforms and to include new encryption algorithms.

It is worth to observe that experimental results based on the TPC-C standard benchmark show that the performance impact of data encryption on response time becomes negligible because it is masked by network latencies that are typical of cloud scenarios. In particular, concurrent read and write operations that do not modify the structure of the encrypted database cause negligible overhead. Dynamic scenarios characterized by (possibly) concurrent modifications of the database structure are supported, but at the price of high computational costs. These performance results open the space to future improvements that we are investigating.

Acknowledgements

We would like to thank Prof. Lorenzo Alvisi of the University of Texas at Austin for his constructive comments on preliminary versions of this paper.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, 2010.

[2] W. Jansen and T. Grance, "Guidelines on security and privacy in public cloud computing," Tech. Rep. NIST Special Publication 800-144, 2011.

[3] A. J. Feldman, W. P. Zeller, M. J. Freedman, and E. W. Felten, "Sporc: group collaboration using untrusted cloud resources," in *Proc. of the 9th USENIX conference on Operating Systems Design and Implementation*, October 2010.

[4] J. Li, M. Krohn, D. Mazières, and D. Shasha, "Secure untrusted data repository (sundr)," in *Proc. of the 6th USENIX conference on Opearting Systems Design and Implementation*, October 2004.

[5] P. Mahajan, S. Setty, S. Lee, A. Clement, L. Alvisi, M. Dahlin, and M. Walfish, "Depot: Cloud storage with minimal trust," *ACM Transactions on Computer Systems*, vol. 29, no. 4, 2011.

[6] H. Hacigümüş, B. Iyer, and S. Mehrotra, "Providing database as a service," in *Proc. of the 18th IEEE International Conference on Data Engineering*, February 2002.

[7] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. of the 41st annual ACM symposium on Theory of computing*, May 2009.

[8] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB: protecting confidentiality with encrypted query processing," in *Proc. of the 23rd ACM Symposium on Operating Systems Principles*, October 2011.

[9] H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra, "Executing sql over encrypted data in the database-service-provider model," in *Proc. of the ACM SIGMOD international conference on Management of data*, June 2002.

[10] J. Li and E. Omiecinski, "Efficiency and security trade-off in supporting range queries on encrypted databases," in *Proc. of the 19th annual IFIP WG 11.3 working conference on Data and Applications Security*. Springer, August 2005.

[11] E. Mykletun and G. Tsudik, "Aggregation queries in the database-as-a-service model," in *Proc. of the 20th annual IFIP WG 11.3 working conference on Data and Applications Security*. Springer, July-August 2006.

[12] D. Agrawal, A. El Abbadi, F. Emekci, and A. Metwally, "Database management as a service: Challenges and opportunities," in *Proc. of the 25th IEEE International Conference on Data Engineering*, March-April 2009.

[13] V. Ganapathy, D. Thomas, T. Feder, H. Garcia-Molina, and R. Motwani, "Distributing data for secure database services," in *Proc. of the 4th ACM International Workshop on Privacy and Anonymity in the Information Society*, March 2011.

[14] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, 1979.

[15] M. Hadavi, E. Damiani, R. Jalili, S. Cimato, and Z. Ganjei, "AS5: A secure searchable secret sharing scheme for privacy preserving database outsourcing," in *Proc. of the 5th International Workshop on Autonomous and Spontaneous Security*. Springer, September 2013.

[16] Oracle corporation, "Oracle advanced security," http://www.oracle.com/technetwork/database/options/advanced-security, April 2013.

[17] G. Cattaneo, L. Catuogno, A. D. Sorbo, and P. Persiano, "The design and implementation of a transparent cryptographic file system for unix," in *Proc. of the FREENIX Track: 2001 USENIX Annual Technical Conference*, April 2001.

[18] E. Damiani, S. D. C. 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*, October 2003.

[19] L. Ferretti, M. Colajanni, and M. Marchetti, "Supporting security and consistency for cloud database," in *Proc. of the 4th International Symposium on Cyberspace Safety and Security*. Springer, December 2012.

[20] TPC-C, "Transaction processing performance council," http://www.tpc.org, April 2013.

[21] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil, "A critique of ansi sql isolation levels," in *Proc. of the ACM SIGMOD*, June 1995.

[22] Xeround, "Xeround: The cloud database," http://xeround.com, April 2013.

[23] EnterpriseDB, "Postgres Plus Cloud Database," http://enterprisedb.com/cloud-database, April 2013.

[24] Microsoft corporation, "Windows azure," http://www.windowsazure.com, April 2013.

[25] Amazon Web Services (AWS), "Amazon elastic compute cloud (amazon ec2)," http://aws.amazon.com/ec2, April 2013.

[26] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," in *Proc. of the 5th USENIX conference on Operating Systems Design and Implementation*, December 2002.

[27] A. Fekete, D. Liarokapis, E. O'Neil, P. O'Neil, and D. Shasha, "Making snapshot isolation serializable," *ACM Transactions on Database Systems*, vol. 30, no. 2, 2005.

[28] A. Boldyreva, N. Chenette, and A. O'Neill, "Order-preserving encryption revisited: Improved security analysis and alternative solutions," in *Proc. of the Advances in Cryptology – CRYPTO 2011*. Springer, August 2011.

[29] Verizon, "IP Latency Statistics," http://www.verizonbusiness.com/about/network/latency, April 2013.

Luca Ferretti is a Ph.D. student at the International Doctorate School in Information and Communication Technologies (ICT) of the University of Modena and Reggio Emilia, Italy. He received the Master Degree in computer engineering from the same University in 2012. His research focuses on information security, and cloud architectures and services. Home page: http://weblab.ing.unimo.it/people/ferretti

Michele Colajanni is full professor in computer engineering at the University of Modena and Reggio Emilia since 2000. He received the Master degree in computer science from the University of Pisa, and the Ph.D. degree in computer engineering from the University of Roma in 1992. He manages the Interdepartment Research Center on Security and Safety (CRIS), and the Master in "Information Security: Technology and Law". His research interests include security of large scale systems, performance and prediction models, Web and cloud systems. Home page: http://weblab.ing.unimo.it/people/colajanni

Mirco Marchetti received his Ph.D. in Information and Communication Technologies (ICT) in 2009. He holds a post-doc position at the Interdepartment Center for Research on Security and Safety (CRIS) of the University of Modena and Reggio Emilia. He is interested in intrusion detection, cloud security and in all aspects of information security. Home page: http://weblab.ing.unimo.it/people/marchetti