Secure Cloud Database with Distribution and Concurrent, Independent Access Methods

Nallapaneni Suresh Babu [#1], Maddirala Syam Kumar [*2], Sk.Subhani [*3]

[#] *M.Tech Student, CSE, Narasaraopeta Engineering College,Narasaraopet, Guntur (D.T), A.P*

[*] *Asst.Professor, CSE, Narasaraopeta Engineering College,Narasaraopet, Guntur (D.T), A.P*

[1] Nallapanenisuresh501@gmail.com,

[2] syam.jnsa@gmail.com

[3] subhaniskn@gmail.com

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, database

I. 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, and 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], [4], [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, and 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, elasti-city, and scalability of a typical cloud DBaaS with data confidentiality is 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], [9], [10], [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, and elasticity) of a database service deployed on a cloud platform

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 cloud database. Workloads including modifications to the data-base 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 follows: Section 2 compares our proposal to existing solutions related to confidentiality in cloud database services. Sections 3 and 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, which can be found on the Computer Society Digital Library http://doi.ieeecomputersociety.org/10.1109/TPDS.2013.154, to de-scribe our solutions to concurrency and data consistency problems in Appendix B, available in the online supple-mental material, to detail the prototype architecture in Appendix C, available in the online supplemental material.

II. RELATED WORKS

SecureDBaaS provides several original features that differ-entiate 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 modifica-tions 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 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 solu-tions 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.

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 en-cryption to protect data managed by untrusted databases. In such a 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.
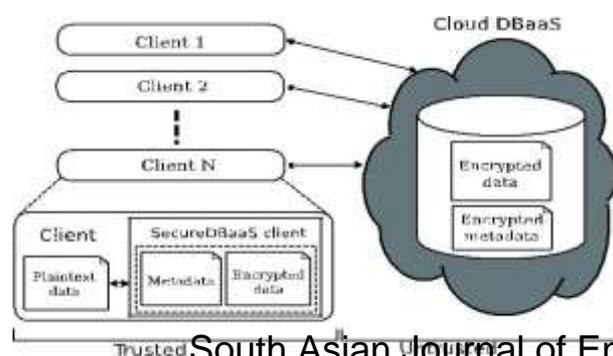

Fig. 1. SecureDBaaS architecture.

applicable to multitier 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 bench-mark [20], in addition to multiple clients and different client-cloud network latencies that were never evaluated in the literature.

III. ARCHITECTURE DESIGN

SecureDBaaS is designed to allow multiple and indepen-dent clients to connect directly to the untrusted cloud DBaaS without any intermediate server. Fig. 1 describes the overall architecture.
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.

The information managed by SecureDBaaS includes plaintext data, encrypted data, metadata, and encrypted metada-ta. 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 con-fidentiality 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.

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 indepen-dently.

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 sections.

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.

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.

SecureDBaaS allows tenants to leverage the computa-tional 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 with 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 of 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, available in the online supplemental material. 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 en-crypted representations. 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.

·   Multicolumn (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 makes 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. 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, available in the online supplemental material.

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 Fig. 2and 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 with 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 opens 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 confidenti-ality is guaranteed through encryption. The structure of the metadata storage table is shown in Fig. 3. This table uses one row for the database metadata, and one row for each table metadata.

Database and table metadata are encrypted through the same encryption key before being saved. This encryption key is called a 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.

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.

IV. 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 naï¨ve 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 encryp-tion keys for any combinations of data types and encryption types, and stores them in the metadata storage table after encryption through the master key.

In the following steps, the DBA creates the tables of the encrypted database. It must consider the three field confidentiality attributes (COL, MCOL, and DBC) intro-duced at the end of the Section 3. Let us describe this phase by referring to a simple but representative example shown in Fig. 4, where we have three secure tables named ST1, ST2, and ST3. Each table STi (i ¼ 1; 2; 3) includes an encrypted table Ti that contains encrypted tenant data, and a table metadata Mi. C1-CN.)

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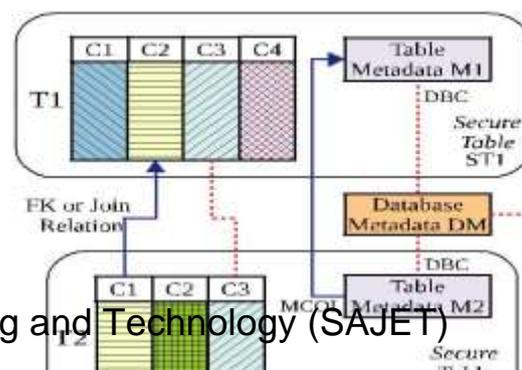
Fig. 4. Management of the encryption keys according to the field confidentiality parameter.

the database is characterized by the same secure type; we limit possible consistency issues in some scenarios characterized by concurrent clients (see Appendix B, available in the online supplemental material). For example, the columns T1.C3, T2.C3, and T3.C1 in Fig. 4 share the same secure type. Hence, they reference the database metadata, as represented by the dashed line, and use the encryption key associated with their data and encryption types. 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 optimi-zations and concurrency issues that will be discussed in Section 4.3 and Appendix B, available in the online supplemental material.

The first connection of the client with the cloud DBaaS is, 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 en-crypted 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 geographi-cally 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 en-crypted 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 Appendix B, available in the online supplemental material. Here, we remark the importance of distinguishing two classes of statements that are supported by SecureDBaaS: SQL opera-tions not causing modifications to the database structure, such as read, write, and update;

In scenarios characterized by a static database structure, SecureDBaaS allows clients to issue concurrent SQL com-mands to the encrypted cloud database without introducing any new consistency issues with respect to unencrypted databases. After metadata retrieval, a plaintext SQL com-mand 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 con-current and consistent accesses even when there are operations that can modify the database structure
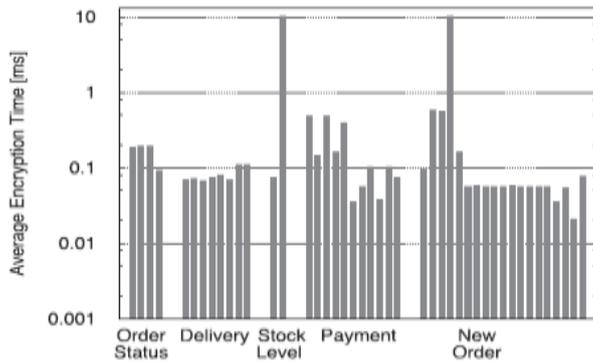
Fig. 5. Encryption times of TPC-C benchmark operations grouped by thetransaction class.

Fig. 6. Plain versus encrypted SELECT and DELETE operations

V. 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 code base. We refer to Appendix C, available in the online supplemental material, 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 require a manual setup of the database. The first group of cloud providers offers 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 evaluates the performance and the overheads of our prototype. We use the Emulab [26] test bed that provides us a controlled environment with several machines, ensuring 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]. of RAM. Clients are connected to the server through a LAN.
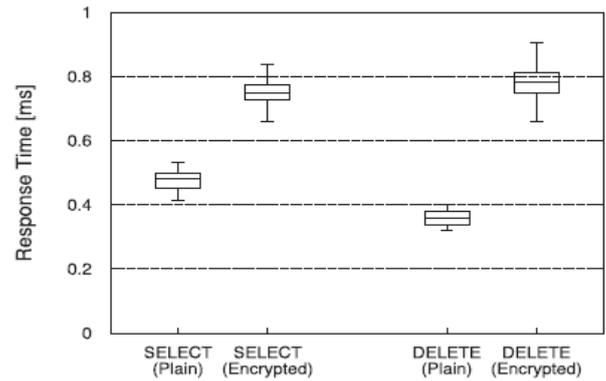
WAN connections that are typical of cloud services. The experiments evaluate the overhead of encryption, compare the response times of plain versus encrypted database operations, and analyze the impact of network latency. We consider two TPC-C compliant databases with 10 warehouses that contain the same number of tuples: plain tuples consist of 1,046 MB data, while SecureDBaaS tuples have size equal to 2,615 MB 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 Fig. 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, and New Order. From this figure, we can appreciate that the encryption time is below 0.1 ms for the majority of operations, and below 1 ms 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, available in the online supplemental material). 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 Figs. 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 UPDATE operations, the response times of SecureDBaaS SQL commands are 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.

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 20 to 150 ms in the client-server connection. These values are representative of round-trip times in continental (in the range of 40-60 ms) and intercontinental (in the range of 80-150 ms) 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 20, 40, and 80 ms latencies. The last column of this table also reports the absolute and percentage overhead intro-duced by SecureDBaaS.

TABLE 1
Response Times and Overheads of SQL Operations for Different Network Latencies

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%

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 percent to 1.31 percent and to 0.26 percent in correspondence of network latencies equal to 20 ms and 80 ms, respectively.

The last set of experiments assess the performance of Secure DBaaS in realistic cloud database scenarios, as well as its ability to support multiple, distributed, and independent clients. 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 en-cryption, 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.

Fig. 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 (>20 ms); can be realistically referred to a cloud database context.

Figs. 9 and 10 show the throughput for increasing numbers of concurrent clients in contexts characterized by 40 ms and 80 ms 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 demonstrat-ing that SecureDBaaS encrypted database does not affect scalability with respect to the plain database. Even more important, the network latencies tend to mask crypto-graphic overheads for any number of clients. For example, the overheads of SecureDBaaS with 40 concurrent clients decreases from 20 percent in a 40-ms scenario to 13 percent in a realistic scenario, where the client-server latency is equal to 80 ms. This result is important because it confirms that SecureDBaaS is a valid and practical solution for guaranteeing data confidentiality in real cloud data-base services.

VI. CONCLUSION

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 hetero-genous 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].

It is worth observing 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.

REFERENCES

[1] M. Armbrust et al., "A View of Cloud Computing," Comm. of the ACM, vol. 53, no. 4, pp. 50-58, 2010.

[2] W. Jansen and T. Grance, "Guidelines on Security and Privacy in Public Cloud Computing," Technical Report Special Publication 800-144, NIST, 2011.

[3] A.J. Feldman, W.P. Zeller, M.J. Freedman, and E.W. Felten, "SPORC: Group Collaboration Using Untrusted Cloud Re-sources," Proc. Ninth USENIX Conf. Operating Systems Design and Implementation, Oct. 2010

[4] J. Li, M. Krohn, D. Mazie`res, and D. Shasha, "Secure Untrusted Data Repository (SUNDR)," Proc. Sixth USENIX Conf. Opearting Systems Design and Implementation, Oct. 2004.

[5] P. Mahajan, S. Setty, S. Lee, A. Clement, L. Alvisi, M. Dahlin, and M. Walfish, "Depot: Cloud Storage with Minimal Trust," ACM Trans. Computer Systems, vol. 29, no. 4, article 12, 2011.

[6] H. Hacigu¨mu¨s¸, B. Iyer, and S. Mehrotra, "Providing Database as a Service," Proc. 18th IEEE Int'l Conf. Data Eng., Feb. 2002.

[7] C. Gentry, "Fully Homomorphic Encryption Using Ideal Lattices," Proc. 41st Ann. ACM Symp. 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," Proc. 23rd ACM Symp. Operating Systems Principles, Oct. 2011.

[9] H. Hacigu¨mu¨s¸, B. Iyer, C. Li, and S. Mehrotra, "Executing SQL over Encrypted Data in the Database-Service-Provider Model," Proc. ACM SIGMOD Int'l Conf.

Selected Paper from International Conference on Computing (NECICC-2k15)