Supporting Security and Consistency for Cloud Database

Luca Ferretti, Michele Colajanni, and Mirco Marchetti

Department of Information Engineering
University of Modena and Reggio Emilia
{luca.ferretti,michele.colajanni,mirco.marchetti}@unimore.it

Abstract. Typical Cloud database services guarantee high availability and scalability, but they rise many concerns about data confidentiality. Combining encryption with SQL operations is a promising approach although it is characterized by many open issues. Existing proposals, which are based on some trusted intermediate server, limit availability and scalability of original cloud database services. We propose an alternative architecture that avoids any intermediary component, thus achieving availability and scalability comparable to that of unencrypted cloud database services. Moreover, our proposal guarantees data consistency in scenarios in which independent clients concurrently execute SQL queries, and the structure of the database can be modified.

1 Introduction

Cloud-based solutions for database services are now considered as an appealing alternative thanks to their scalability and availability attributes. Nevertheless, outsourcing critical data to untrusted cloud providers still poses many security concerns [1, 9]. One interesting research goal is to allow customers to leverage cloud infrastructures while guaranteeing data confidentiality by avoiding that cloud providers may access customer data.

In the so called database-as-a-service (DBaaS) model [7] it is impossible to guarantee confidentiality by naively encrypting customer data because traditional encryption schemes prevent the execution of SQL queries through a DBMS engine.

Previous works [8, 13] addressed this issue through encryption schemes that allow the execution of SQL queries over encrypted data. These architectures are based on a trusted intermediate proxy, that accesses the database on behalf of the clients. This design choice is suitable to web clients that access the DBMS through other intermediate servers [13], but the reliance on a trusted proxy limits availability and scalability of the encrypted database. Hence, existing proxy-based architectures do not suit the cloud database context, where possibly distributed clients can access the remote DBMS.

This paper proposes a novel architecture that allows cloud customers to leverage untrusted DBaaS with the guarantee of data confidentiality. Unlike previous solutions, our architecture does not rely on a trusted proxy, and allows multiple

Y. Xiang et al. (Eds.): CSS 2012, LNCS 7672, pp. 179–193, 2012.

distributed clients to execute SQL queries concurrently and independently on the same encrypted database. All the encryption and decryption operations are carried out by a software module that is executed on each client machine. Our design choice does not introduce any bottleneck and single point of failure because clients connect directly to the cloud database. Moreover, our architecture guarantees the same availability, scalability and elasticity of the unencrypted DBaaS and it is applicable to any commercial DBaaS because it does not require modifications to the database.

On the other hand, our support to concurrent execution of queries from independent clients requires novel solutions to guarantee data consistency. In this paper, we identify several common usage scenarios and, for each scenario, we analyze the consistency issues [2] that may arise from the execution of concurrent queries. We show that our solution guarantees consistency of customer data in all these contexts through standard isolation mechanisms already implemented in popular DBMS engines. We remark that this result cannot be achieved naively in existing proxy-based solutions [8, 13] just by implementing multiple proxies because their encryption management strategies are not designed for being distributed among independent proxy instances that would require novel synchronization algorithms and protocols.

The remaining part of this paper is structured as following. Section 2 discusses previous work in the field of secure cloud database services. Section 3 describes the novel architecture proposed in this paper. Section 4 discusses how it is possible to guarantee data consistency in different usage contexts. Section 5 concludes the paper by summarizing its main contributions and future work.

2 Related Work

This paper proposes a novel architecture that is different from any previous work in the field of security for cloud database services.

Cryptographic file systems and secure storage solutions represent the earliest works to guarantee confidentiality and integrity of data outsourced to untrusted cloud storage services. We do not detail the several papers and products in this field (e.g., [6, 10, 11]) because they do not allow any computation on encrypted data. Hence they cannot be applied to the context of cloud DBaaS.

Some DBMS engines offer the possibility of encrypting data at the file system level through the so called Transparent Data Encryption (TDE) feature [3, 12]. This feature makes it possible to build a trusted DBMS over untrusted storage. However, in the DBaaS context the DBMS engine is not trusted because it is controlled by the cloud provider, hence the TDE approach is not applicable to cloud database services.

An approach to preserve data confidentiality in scenarios where the DBMS is not trusted is proposed in [5]. However it requires a modified DBMS engine that is not compatible with commercial and open source DBMS software adopted by cloud providers. On the other hand, the architecture we propose is compatible with standard DBMS engines, and allows customers to build a secure cloud database by leveraging cloud DBaaS readily available.

The proposal in [4] uses encryption to control accesses to encrypted data stored in a cloud database. This solution is not applicable to usage contexts in which the structure of the database changes, and does not support concurrent accesses from multiple clients possibly distributed on a geographical scale.

Our proposal is related to [8] and [13] that preserve data confidentiality in an untrusted DBMS through encryption techniques that allow the execution of SQL queries over encrypted data and are compatible with common DBMS engines. These architectures are based on an intermediate and trusted proxy that mediates all the interactions between clients and the untrusted DBMS server. The reliance on a trusted proxy that characterizes both [8] and [13] facilitates the implementation of a secure DBaaS, but causes several drawbacks. A detailed comparison between the proxy-less architecture proposed in this paper and previous architectures based on a trusted proxy is in Section 3.

The architecture we propose moves away from existing solutions because it allows multiple and independent clients to connect directly to the untrusted cloud DBaaS without any intermediate server. To the best of our knowledge this is the first paper that identifies consistency issues related to concurrent execution of queries over encrypted data and to propose viable solutions for different usage contexts, including data manipulation, modification to the database structure, and data re-encryption.

3 Architecture Design

This paper proposes a novel architecture that allows customers to use cloud DBaaS while preserving confidentiality of outsourced data. In particular, we aim to:

- maintain the benefits of cloud solutions in terms of availability, scalability and elasticity;
- support direct access from multiple clients, possibly distributed on a geographical scale;
- allow concurrent execution of SQL operations including those modifying data and the structure of the database.

The architecture proposed in this paper guarantees data confidentiality together with the ability to execute SQL operations over customer data by using SQL-aware encryptions schemes similar to those already proposed in [8, 13]. Regardless of the particular encryption algorithm used to cipher customer data, all the solutions based on cryptography depend on *metadata*. Metadata consist of information required to encrypt and decrypt customer data and to translate plaintext SQL statements to SQL statements over encrypted data. Hence, guaranteeing metadata confidentiality is as critical as guaranteeing confidentiality of customer data in the cloud.

We investigate three types of architectures:

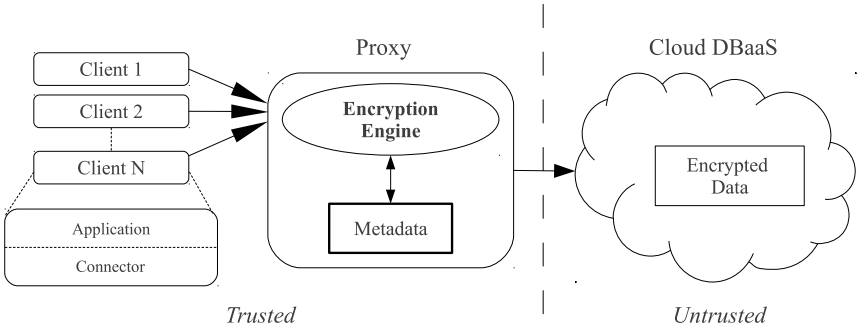

Fig. 1. A proxy-based architecture

- proxy-based architectures, proposed in [8, 13];
- proxy-less architectures that store metadata in the clients, proposed in [4];
- proxy-less architectures that store metadata in the cloud database, proposed in this paper.

The most popular solutions [8, 13] for the confidentiality of data outsourced to untrusted database propose a proxy-based architecture, that is represented in Figure 1.

Clients access the database by issuing unmodified SQL queries to the proxy through a standard database connector. The proxy executes the *encryption engine*, that is the module responsible of applying encryption strategies on customer data, and manages all metadata. The cloud database stores only encrypted customer data, hence the cloud provider cannot access plaintext customer data nor metadata that are required to decrypt encrypted customer data.

These proxy-based architectures do not satisfy our design requirements because the proxy is a bottleneck and a single-point-of-failure that limits availability, scalability and elasticity of the cloud DBaaS. Since the proxy must be trusted, it cannot be outsourced to the cloud and has to be deployed and maintained locally. Moreover, proxy-based architectures cannot scale trivially by increasing the number of proxies. Such a naive solution would imply the replication of metadata among all the proxies, but this would require synchronization algorithms and protocols to guarantee consistency among all the proxies that are not considered in [8, 13].

A different approach proposed in [4] is shown in Figure 2. Here, the architecture does not use an intermediate proxy and metadata are stored in the clients. Since clients connect directly to the cloud database, this architecture achieves availability, scalability and elasticity comparable to those of the original DBaaS. However, each client has its own encryption engine and manages a local copy of metadata. Hence, this solution can represent a sub-case of the proxy-based architecture, in which a different proxy is deployed within each client. As a consequence, a similar architecture for cloud accesses would suffer from the same consistency issues of proxy-based architectures. Guaranteeing metadata consistency

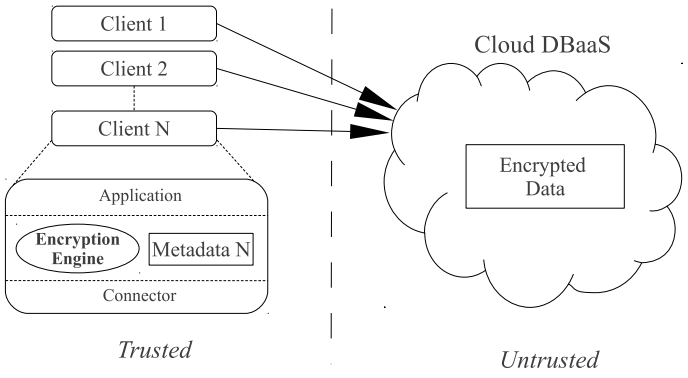

Fig. 2. A proxy-less architecture with metadata in the clients

in the face of concurrent query execution would require novel synchronization algorithms and protocols among all the clients.

The novel proxy-less architecture represented in Figure 3 is proposed as a solution to meet all the design requirements outlined at the beginning of this section. The main idea is to move metadata to the cloud database, while the encryption engine is executed by each client. Since metadata are not shared among clients there is no need of synchronization mechanisms. Client machines execute a *client software component* that allows a user to connect and issue queries directly to the cloud DBaaS. This component retrieves the necessary metadata from the untrusted database through SQL statements and makes them available to the encryption engine. Multiple clients can access the untrusted cloud database independently, with the guarantee of the same level of availability, scalability and elasticity of cloud-based services.

The proposed proxy-less architecture overcomes the main drawbacks of proxy-based solutions, however it introduces new issues with respect to *metadata security* and *data consistency*. Previous proposals solve metadata security issues by storing and managing them on trusted components. Since they do not take into account the concurrent management of metadata by multiple components, they do not address any consistency issues related to data and metadata.

Our proposal guarantees security of metadata when at rest, in motion and in use by encrypting metadata stored in the cloud. Only clients that know the encryption key can decrypt metadata. Therefore, only these clients can access data that are stored in an encrypted form in the cloud DBaaS. The proposed architecture does not limit the applicability of any well-known system for key distribution, ranging from simple pre-shared key to the use of dedicated authentication servers. Describing the deployment of a specific system is out of the scope of this paper, even because this choice does not influence our proposal.

In the proposed architecture the plaintext database is transformed into an encrypted database by translating each plaintext table into a corresponding encrypted table. Each encrypted table is associated with a set of metadata that

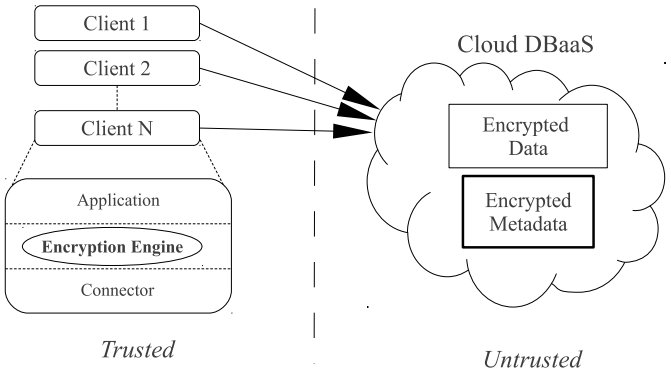

Fig. 3. The novel proxy-less architecture with encrypted metadata in the cloud

contains all management information required to encrypt and decrypt data belonging to that table. Metadata associated with different tables are independent.

We discuss data consistency using the example represented in Figure 4, where we consider a database composed by two tables $T1$ and $T2$, that are stored encrypted in the two corresponding tables $T1_{enc}$ and $T2_{enc}$. Each table is associated with a set of metadata, respectively $M1$ and $M2$, that are independent of each other. All metadata are encrypted and stored in the database as $M1_{enc}$ and $M2_{enc}$. In this context, let us consider that clients A, B and C are concurrently accessing the database:

- client A executes queries on tables $T1$ and $T2$. Hence, it reads $M1_{enc}$ and $M2_{enc}$, decrypts them and maintains temporary local versions $M1^{temp}$ and $M2^{temp}$;
- client B executes queries on table $T2$, hence it retrieves $M2_{enc}$ and maintains $M2^{temp}$;
- client C executes queries on table $T1$, hence it retrieves $M1_{enc}$ and maintains $M1^{temp}$.

Clients B and C access the database independently from each other, since they handle independent metadata. Hence, they do not cause any consistency issues. This design choice makes it possible to avoid conflicts when modifications on metadata associated with different tables occur. However, client A accesses both $M1_{enc}$ and $M2_{enc}$, and modifications to any of them can cause consistency issues with respect to temporary versions of clients C and B. In the proposed design, concurrent accesses on the same table can still cause consistency issues depending on the types of SQL queries that are concurrently executed. Consistency issues caused by concurrent modifications and related solutions are discussed in Section 4.

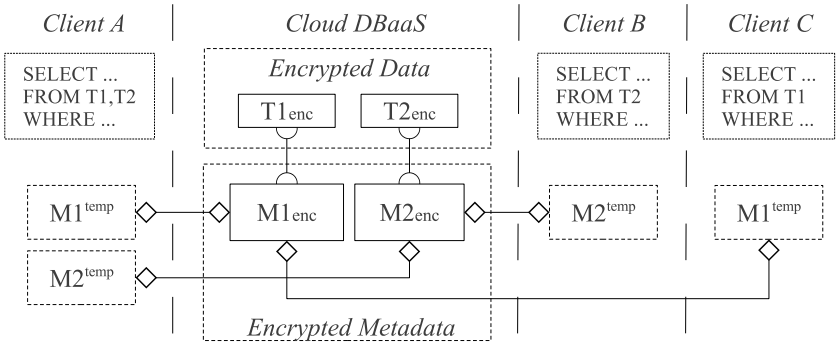

Fig. 4. Metadata structure

4 Concurrent Operation Management

The support to concurrent execution of SQL statements issued by multiple independent (possibly geographically distributed) clients is one of the most important benefits with respect to state-of-the-art solutions that require clients to issue queries to database through an intermediate proxy. Our architecture must guarantee consistency among encrypted customer data and encrypted metadata, because corrupted or out-of-date metadata would prevent clients from decoding encrypted customer data with permanent data loss consequences. In such a way, clients can transform plaintext SQL statements into SQL operations that leverage transactions and isolation mechanisms provided by any relational database engine and cloud DBaaS.

Problems and solutions depend on the use of the database and on related types of queries. We present consistency issues and adopted solutions in relation to four contexts:

- Data manipulation;
- Structure modifications;
- Data re-encryption;
- Unrestricted operations.

4.1 Data Manipulation

In the Data Manipulation context clients can read and write encrypted customer data stored in the untrusted cloud database through the execution of SELECT, INSERT, DELETE and UPDATE commands. This set of SQL operations is indicated by the *DML* acronym. In this scenario, clients cannot modify the structure of the database by creating new tables or altering or dropping existing tables. We assume tables are created by the database administrator during a set-up period. Since only one client can access the cloud database while tables are being created, no concurrency issues arise here because multiple and independent clients can access the cloud database only after all tables have been created.

Plaintext SQL commands issued by users are translated by clients into queries that operate over encrypted customer data. The client analyzes plaintext SQL commands to identify which plaintext tables are involved. Then, it issues a SELECT query to retrieve the metadata associated with the corresponding encrypted tables. A client generates exactly one translated SQL command for each plaintext SQL command issued by the users.

In this context, there are no consistency issues related to metadata management because metadata never change. However, multiple clients executing concurrent read and write commands over the same data set can lead to inconsistencies over customer data. These issues can be addressed by leveraging standard concurrency isolation mechanisms provided by the DBMS server used to provision the cloud database service. Each user can enclose several SQL statements within a transaction by issuing BEGIN, COMMIT and ABORT commands. In this context, clients forward these commands to the cloud database without any modifications. Hence the cloud database executes concurrent transactions of translated queries in the same way as a traditional cloud DBaaS executes concurrent transactions of plaintext SQL commands. Consistency guarantees derive from the isolation level chosen by the database administrator among those implemented by the database, and are not influenced by the encryption and decryption operations.

4.2 Structure Modifications

A popular context that has not been considered by previous proposals about secure cloud databases is the possibility of modifying the structure of the database. Our architecture supports the execution of CREATE, DROP and DML SQL commands. Unlike the previous scenario, in this context database metadata can change, hence clients cannot rely on a cached copy of metadata. Our architecture requires clients to translate each SQL command into a database transaction containing:

- the SQL queries necessary to retrieve the up-to-date metadata;
- the translated SQL commands that correspond to the original SQL command.

Each plaintext SQL command executed in unencrypted databases is an atomic operation. However, we translate each atomic command into a sequence of multiple commands enclosed in a transaction. Hence, consistency is guaranteed by choosing a sufficient transaction isolation level among those offered by the cloud database.

If the isolation level is not sufficient, consistency issues may arise from the execution of operations belonging to different but concurrent transactions. If concurrent transactions operate just on encrypted customer data, metadata are not modified and we return to the data manipulation context analyzed in Section 4.1, in which the database administrator can choose the isolation level among those provided by the DBMS. On the other hand, consistency issues may arise when

a concurrent transaction contains commands that modify metadata. Among the considered SQL commands, only CREATE and DROP operations modify metadata, hence consistency issues may arise if concurrent executions of the following commands occur:

- DROP and DML;
- CREATE and DML;
- any concurrent CREATE and DROP.

We analyze these contexts by using the notation in Table 1, that is similar to that proposed in [2].

Table 1. Notation for transactions and SQL queries

B_t	BEGIN operation of transaction t
C_t	COMMIT operation of transaction t
A_t	ABORT operation of transaction t
$R_t[T_{enc}, U_{enc}]$	Read (SELECT) operation on tables T_{enc},U_{enc} in transaction t
$W_t[T_{enc}, U_{enc}]$	Write (INSERT, UPDATE, DELETE) operation on tables T_{enc},U_{enc} in transaction t
$SM_t[T_{enc}]$	Structure Modification (CREATE or DROP) operation on table T_{enc} in transaction t
$M_t^R[T]$	Read operation on metadata related to table T in transaction t
$M_t^W[T]$	Write operation on metadata related to table T in transaction t

DROP and DML. The database may generate errors if the DML command is executed after the table has been dropped. For example, we consider the following two transaction histories.

The former represents the execution of a table DROP, while a data read is being executed on the same table:

$$B_1 B_2 M_1^R[T] M_2^R[T] M_2^W[T] SM_2[T_{enc}] C_2 R_1[T_{enc}] A_1 \tag{1}$$

Transaction 1 obtains metadata that are necessary to create the translated read command $R_1[T_{enc}]$ and to decrypt its result. The DROP command ($SM_2[T_{enc}]$) issued by transaction 2 is executed before the translated data read issued by transaction 1. The table T_{enc} does not exist anymore and the read command issued by transaction 1 fails.

Now we consider the concurrent execution of a DROP and a write command:

$$B_1 B_2 M_1^R[T] M_2^R[T] M_2^W[T] SM_2[T_{enc}] C_2 W_1[T_{enc}] A_1 \tag{2}$$

In this context, the write command executed by transaction 1 fails because T_{enc} was deleted by the DROP command ($SM_2[T_{enc}]$).

CREATE and DML. The database may generate errors if the DML command is executed before the creation of the table. As an example, we consider the following two transaction histories.

The former represents the execution of a table CREATE, while a data read is being executed on the same table:

$$B_1 B_2 M_2^R[T] M_2^W[T] M_1^R[T] R_1[T_{enc}] A_1 SM_2[T_{enc}] C_2 \tag{3}$$

The read command executed by transaction 1 fails because T_{enc} was not yet created by the CREATE ($SM_2[T_{enc}]$).

Now we consider the concurrent execution of a CREATE and a write command:

$$B_1 B_2 M_2^R[T] M_2^W[T] M_1^R[T] W_1[T_{enc}] A_1 SM_2[T_{enc}] C_2 \tag{4}$$

The write command executed by transaction 1 fails because T_{enc} was not yet created by the CREATE ($SM_2[T_{enc}]$).

In all these cases, the client software handles the error notification generated by the remote database. We highlight that none of the considered errors cause consistency issues to the encrypted customer data or metadata.

Any concurrent CREATE and DROP. The database may generate errors if two commands that modify the structure of the database are executed concurrently. For example, if two CREATE (DROP) commands insist on the same table, then an error is generated as soon as the second transaction insert (delete) the related metadata, as represented by the following history case.

$$B_1 B_2 M_1^R[T] M_2^R[T] M_1^W[T] M_2^W[T] A_2 SM_1[T_{enc}] C_1 \tag{5}$$

If a CREATE and a DROP are executed concurrently over the same table, an error can be generated because the DROP is executed on a table that does not exist yet, or because a client creates an already existing table. The following history represents a failed CREATE (DROP) command by the transaction 2 executed before the other DROP (CREATE) command by the transaction 1.

$$B_1 B_2 M_1^R[T] M_1^W[T] M_2^R[T] M_2^W[T] SM_2[T_{enc}] A_2 SM_1[T_{enc}] C_1 \tag{6}$$

Since the transaction 2 aborts, its previous modification on related metadata ($M_2^W[T]$) are rolled back (A_2).

In this context the use of implicit transactions is enough to guarantee data consistency, hence the database administrator can freely choose the preferred isolation level among those provided by the database.

4.3 Data Re-encryption

The proposed proxy-less architecture guarantees data confidentiality by independently encrypting tables. In the data re-encryption context, we analyze the

consistency issues that arise when clients re-encrypt data stored in the cloud database. This occurs when it is required to change encryption keys, or to use a different encryption algorithm to guarantee confidentiality. Our architecture handles both cases through the execution of the following transaction:

$$BM^R[T]M^W[T]R[T_{enc}]W[T_{enc}]C \tag{7}$$

As an example we consider a re-encryption command that modifies the encryption key that is used to encrypt customer data stored in the table T_{enc}. The client first reads the current metadata ($M^R[T]$) associated with the encrypted customer data to retrieve all the information related to their encryption policy, including current encryption keys. Then, it updates the metadata ($M^W[T]$) according to the new encryption policy, by changing the encryption keys. Hence, the client needs to read all the data ($R[T_{enc}]$), to decrypt them with the old encryption keys, to encrypt them with the new encryption keys and to write new data to the encrypted table ($W[T_{enc}]$). Decryption and encryption operations have to be performed locally by a trusted client because the client never exposes plaintext data to the untrusted cloud database.

Consistency issues may arise in the following cases:

- concurrent execution of a re-encryption and data read;
- concurrent execution of a re-encryption and data write;
- concurrent execution of multiple re-encryptions.

Re-encryption and data read. The database may return data that are not accessible by the client, if a data read command is executed concurrently to a re-encryption command. We consider the case in which a data read command requires a set of data whose encryption key is being modified by a concurrent re-encryption command, as represented by the following transaction history:

$$B_1B_2M_1^R[T]M_2^R[T]M_2^W[T]R_2[T_{enc}]W_2[T_{enc}]C_2R_1[T_{enc}]C_1 \tag{8}$$

In this example, transaction 1 reads metadata ($M_1^R[T]$). Then transaction 2 executes sequentially all operations included in the re-encryption command as defined in (7). Finally, transaction 1 reads the set of data ($R_1[T_{enc}]$). However, it obtains data that are encrypted through a new encryption key, hence it cannot decrypt them. This concurrency issue is an instance of the well known *read skew* anomaly defined in [2].

Re-encryption and data write. Inconsistent data may be written if the data write command and a re-encryption command are executed concurrently. We consider the case in which a data write command stores a set of data whose encryption key is being modified by a concurrent re-encryption command. This scenario is represented by the following transaction history.

$$B_1B_2M_1^R[T]M_2^R[T]M_2^W[T]R_2[T_{enc}]W_2[T_{enc}]C_2W_1[T_{enc}]C_1 \tag{9}$$

In this example, transaction 1 reads metadata ($M_1^R[T]$), then transaction 2 executes sequentially all operations included in the re-encryption command as

defined in (7). Finally, transaction 1 writes the set of data ($W_1[T_{enc}]$). However, it writes data that are encrypted by means of the old encryption key that is not stored anymore in metadata related to table T_{enc}. As a consequence, these data are inaccessible.

The consistency anomaly that affects the above history may differ on the basis of the considered write command. We distinguish two main cases: UPDATE or DELETE commands, and INSERT commands.

In the case of an UPDATE or a DELETE command the data write command ($W_1[T_{enc}]$) insists on a set of data also interested by the re-encryption command ($W_2[T_{enc}]$). Hence, the concurrency issue is an instance of the *lost update* phenomenon, as defined in [2].

In the INSERT case the data write command insists on a set of data that did not exist when the re-encryption command was executed, but that is included in the predicate of the update sequence of the re-encryption command ($R_2[T_{enc}]W_2[T_{enc}]$). This concurrency issue is an instance of the *phantom* anomaly as defined in [2].

We highlight that an alternative example of the above transaction history is to swap the order of the last writes operations, as represented by the following history:

$$B_1 B_2 M_1^R[T] M_2^R[T] M_2^W[T] R_2[T_{enc}] W_1[T_{enc}] C_1 W_2[T_{enc}] C_2 \tag{10}$$

In the case of an UPDATE or DELETE command, the database is still consistent and completely accessible. However, newly written data have been lost, due to the *lost update* phenomenon.

Multiple re-encryptions. We consider the case in which two fields of the same table are re-encrypted, as represented in the following history:

$$\begin{aligned} &B_1 B_2 M_1^R[T] M_2^R[T] M_1^W[T] M_2^W[T] \\ &R_1[T_{enc}] R_2[T_{enc}] W_1[T_{enc}] C_1 W_2[T_{enc}] C_2 \end{aligned} \tag{11}$$

Since both transactions modify the same metadata ($M_1^W[T]$ $M_2^W[T]$), the execution of concurrent re-encryptions may cause a *lost update* anomaly.

Finally, we can define the consistency requirements of the *re-encryption* context. The DBMS isolation level must avoid *read skew*, *lost update* and *phantom* concurrency anomalies. Since *lost update* is a sub-case of a *read skew*, as discussed in [2], it is possible to trace back the two anomalies to only *read skew*.

The proposed proxy-less architecture guarantee data consistency by leveraging the appropriate isolation level. The *read skew* anomaly is avoided by the snapshot isolation level, that does not guarantee consistency with respect to the *phantom* anomaly. Besides the highest ANSI *serializable*, no standard isolation level with similar guarantees have been defined yet. However, several well-known DBMS engines extend snapshot isolation through predicate locking mechanisms, thus avoiding also *phantom* anomalies. We call the set of snapshot isolation levels that also avoid *phantom* anomalies as *snapshot isolation plus*.

If the required isolation level is set on the cloud database, the proposed proxy-less architecture lets several clients execute DML commands concurrently while one client executes re-encryption operations on a table with no consistency issues. An isolation level that suits our requirements with low overhead has been proposed in [14].

4.4 Unrestricted Operations

This context does not pose any limitation to the nature of the commands that can be issued concurrently by clients to the cloud database. It is possible to execute any data definition language (DDL) command, as well as DML commands, and re-encryptions that modify the database structure and encryption policies. Since the behavior of DDL is not formalized in any standard, each DBMS implements different DDL locking mechanisms and DDL transaction policies. Hence, identify one isolation level that does not depend on the database and that guarantee data consistency. A possible solution is to impose the isolation level *serializable* [2] together with the support to rollback of DDL operations that are included in the transactions.

If the constraints are satisfied, the proposed proxy-less architecture guarantees data consistency in any execution context. Since these constraints are not met by all the DBMS engines, another solution is to explicitly handle concurrency issues at the application level. This problem is out of the scope of this paper because it would depend on the guarantees provided by the remote database.

4.5 Discussion

In scenarios characterized by a static database structure (as described in Section 4.1) the proposed architecture allows multiple, independent and possibly geographically distributed clients to issue concurrent SQL commands to read, write and update data stored in an encrypted cloud database. It is worth to observe that:

- in the data manipulation context, that is the one taken into account by previous proposals [4, 8, 13], negligible overhead is generated. Clients can read metadata only and cache them locally without consistency issues;
- in all contexts the proposed architecture does not introduce any new consistency issue with respect to unencrypted databases;
- any underlying mechanisms implementing database operations are transparent to the users.

Some inevitable overhead is caused by the computational cost related to data encryption and decryption operations. However, this cost is inherent in any encrypted database solution that does not want to expose plaintext data to the cloud provider.

We highlight also that the novel proxy-less architecture is the first solution that allows concurrent and direct accesses to the cloud database and that supports even modifications to the database structure. Depending on the type of

modification, higher isolation levels are required with consequential overheads. If we have to support operations such as CREATE and DROP tables (Section 4.2) or data re-encryption (Section 4.3), then the proposed solution introduces some additional operations to implement implicit transactions.

5 Conclusions

This paper proposes a novel solution that guarantees confidentiality of data saved into cloud databases that are untrusted by definition. All data outsourced to the cloud provider are encrypted through cryptographic algorithms that allow the execution of standard SQL queries on encrypted data. This is the first solution that allows direct, independent and concurrent access to the cloud database and that supports even changes to the database structure. It does not rely on a trusted proxy that represents a single point of failure and a system bottleneck, and that limits the availability and scalability of cloud database services. Concurrent read and write operations that do not modify the structure of the encrypted database are supported with minimal overhead. More dynamic scenarios characterized by (concurrent) modifications of the database structure are supported but at the price of higher overhead and stricter transaction isolation levels. This should be considered an initial paper on a long-term research that will include implementation on different cloud platforms, experimentations, and evaluation of performance and overheads.

Acknowledgments. The authors acknowledge the support of MIUR-PRIN project DOTS-LCCI Dependable Off-the-Shelf based middleware system for Large-scale Complex Critical Infrastructures.

References

1. Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al.: A view of cloud computing. Communications of the ACM 53(4), 50–58 (2010)
2. Berenson, H., Bernstein, P., Gray, J., Melton, J., O'Neil, E., O'Neil, P.: A critique of ansi sql isolation levels. SIGMOD Rec. 24(2), 1–10 (1995)
3. Cattaneo, G., Catuogno, L., Sorbo, A.D., Persiano, P.: The design and implementation of a transparent cryptographic file system for unix. In: Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference, pp. 199–212. USENIX Association, Berkeley (2001)
4. Damiani, E., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Metadata Management in Outsourced Encrypted Databases. In: Jonker, W., Petković, M. (eds.) SDM 2005. LNCS, vol. 3674, pp. 16–32. Springer, Heidelberg (2005)
5. Damiani, E., De Capitani di Vimercati, S., Jajodia, S., Paraboschi, S., Samarati, P.: Balancing confidentiality and efficiency in untrusted relational dbmss. In: Proceedings of the 10th ACM Conference on Computer and Communications Security, CCS 2003, pp. 93–102. ACM, New York (2003)

6. Feldman, A., Zeller, W., Freedman, M., Felten, E.: Sporc: Group collaboration using untrusted cloud resources. OSDI (October 2010)
7. Hacigümüş, H., Iyer, B., Mehrotra, S.: Providing database as a service. In: Proceedings of the 18th International Conference on Data Engineering, pp. 29–38 (2002)
8. Hacigümüş, H., Iyer, B., Li, C., Mehrotra, S.: Executing sql over encrypted data in the database-service-provider model. In: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, SIGMOD 2002, pp. 216–227. ACM, New York (2002)
9. Jansen, W., Grance, T.: Guidelines on security and privacy in public cloud computing. NIST Special Publication 800–144(2011)
10. Li, J., Krohn, M., Mazières, D., Shasha, D.: Secure untrusted data repository (sundr). In: Proceedings of the 6th Symposium on Operating Systems Design and Implementation, pp. 91–106 (2004)
11. Mahajan, P., Setty, S., Lee, S., Clement, A., Alvisi, L., Dahlin, M., Walfish, M.: Depot: Cloud storage with minimal trust. ACM Trans. Comput. Syst. 29(4), 12:1–12:38 (2011)
12. Oracle corporation: Oracle advanced security (October 2012), `http://www.oracle.com/technetwork/database/options/advanced-security`
13. Popa, R.A., Redfield, C.M.S., Zeldovich, N., Balakrishnan, H.: CryptDB: protecting confidentiality with encrypted query processing. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, SOSP 2011, pp. 85–100. ACM, New York (2011)
14. Yabandeh, M., Gómez Ferro, D.: A critique of snapshot isolation. In: Proceedings of the 7th ACM European Conference on Computer Systems, pp. 155–168. ACM (2012)