Volume 5, Issue 7, July 2015 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

A Comparative Study of CryptDB and Adaptive Encryption Architectures for Cloud Data Base Services

K. Sunitha
Assistant Professor, CSE, MGIT, Gandipet,
HYD, TS., India

Abstract - Online applications are vulnerable to theft of sensitive information because adversaries can exploit software bugs to gain access to private data, and curious or malicious administrators may capture and leak data. Outsourcing the user data to third parties is a threat to the confidentiality and privacy of user data. Cloud Data Base-as-a –Service (DBaaS) is a model that supports many internet based applications and poses several security challenges.

This paper focuses on two different architectures such as CryptDB and Adaptive encryption architectures. CryptDB only empowers the server to execute queries that the users requested, and achieves maximum privacy given the mix of queries issued by the user. Some encryption schemes that allow the execution of SQL operations over encrypted data are inapplicable or some may have performance limits or require the choice of which encryption scheme must be adopted for each data base column and SQL operation. So the Adaptive Encryption architecture is guarantees at runtime the best level of data confidentiality for any database work load, even when the set of SQL queries dynamically changes.

The adaptive encryption of public cloud databases that offers an interesting alternative to the trade-off between the required data confidentiality level and the flexibility of the cloud Database structures at design time. Finally, a comparative study of these two efficient architectures has been carried out and given here.

Keywords: Cloud database, CryptDB, Adaptive encryption, Confidentiality and privacy.

I. INTRODUCTION

The companies have started relying on the cloud computing for several reasons and a trend has started by adopting cloud computing services for better and faster availability of the information rather than setting up an individual data center for each organization or the company.

The organizations always look for the ways that is effective and is cost saving. The same is the process with the database. Earlier the organizations set up their own data center and have their traditional database. Now the cloud database has evolved a new dimension Database as a Service (DBaaS). This allows the companies and organizations to use the resources of the DBaaS providers and without any hassle to invest and maintain the hardware and software for their data centers that hold all the information in the database. They get services from DBaaS provider and enjoy the freedom of 24/7 available database. There are advantages and disadvantages as well; however the adoption of cloud database has proven that the advantages are more than the disadvantages. The cloud database services have offered many benefits and different companies are in the race. The organization chooses the one that suits its requirements.

A. CryptDB:

CryptDB is a system that provides practical, provable confidentiality and privacy that guarantees without having to trust the DBMS server or the DBAs who maintain and tune the DBMS.

CryptDB is the first private system to support all of standard SQL over encrypted data without requiring any client-side query processing, modifications to existing DBMS codebases, changes to legacy applications and offloads virtually all query processing to the server. CryptDB works by rewriting SQL queries, storing encrypted data in regular tables, and using an SQL user-defined function (UDF) to perform server-side cryptographic operations.

It works by *executing SQL queries over encrypted data* using a collection of efficient SQL-aware encryption schemes. CryptDB can also *chain encryption keys to user passwords*, so that a data item can be decrypted only by using the password of one of the users with access to that data. As a result, a database administrator never gets access to decrypted data, and even if all servers are compromised, an adversary cannot decrypt the data of any user who is not logged in.

B. An Adaptive Encryption

Adaptive encryption of public cloud databases offers a proxy-free alternative to the CryptDB system described in [1]. It provides the best level of data confidentiality for any database workload, even when the set of SQL queries dynamically changes.

The adaptive encryption scheme, which was initially proposed for applications not referring to the cloud, encrypts each plain column to multiple encrypted columns, and each value is encapsulated in different layers of encryption, so that the outer layers guarantee higher confidentiality but support fewer computation capabilities with respect to the inner layers.

The outer layers are dynamically adapted at runtime when new SQL operations are added to the workload. Although this adaptive encryption architecture is attractive because it does not require define at design time which database operations are allowed on each column.

The design of first proxy-free architecture for adaptive encryption of cloud databases that does not limit the availability, elasticity and scalability of a plain cloud database because multiple clients can issue concurrent operations without passing through some centralized component as in alternative architectures [1].

The cloud database must be able to execute SQL operations directly over encrypted data without accessing any decryption key.

Naive solutions:

- Encrypt the whole database through some standard encryption algorithms that do not allow executing any SQL operation directly on the cloud. As a consequence, the tenant has

Two alternatives:

- download the entire database, decrypt it, execute the query and, if the operation modifies the database , encrypt and upload the new data;
- Decrypt temporarily the cloud database, execute the query, and re-encrypt it.

The former solution is affected by huge communication and computation overheads, and consequent costs that would make cloud database services quite inconvenient;

The latter solution does not guarantee data confidentiality because the cloud provider obtains decryption keys.

The right alternative is to execute SQL operations directly on the cloud database, without giving decryption keys to the provider.

Encryption algorithms characterized by acceptable computational complexity support a subset of SQL operators [2], [3], [4].the database administrator cannot know at design time which database operations will be required over each database column. This issue is in part addressed in [1] by proposing an adaptive encryption architecture that is founded on an intermediate and trusted proxy. This tenant's component, which mediates all the interactions between the clients and a possibly untrusted DBMS server, is fine for a locally distributed architecture but it cannot be applied to a cloud context.

Indeed, any centralized component at the tenant side reduces the scalability and availability that are among the most important features of cloud services. A solution to this problem was presented in [5]: the proposed architecture allows multiple clients to issue concurrent SQL operations to an encrypted database without any intermediate trusted server, but it assumes that the set of SQL operations does not change after the database design.

II. ARCHITECTURE DESIGN

A. Design of CryptDB

CryptDB comprises of three major components, namely the ***Application Server, Proxy Server and DBMS Sever***. Application Server is the main server that runs CryptDB's database proxy and also the DBMS Server. The Proxy Server stores a secret Master Key, the database Schema and the onion layers of all the columns in the database. The DBMS server has access to the anonymized database schema, encrypted database data and also some cryptographic user-defined functions (UDF's). The DBMS server uses these cryptographic UDF's to carry out certain operations on the encrypted data (ciphertext).

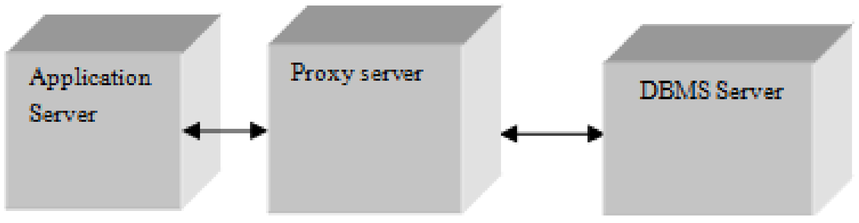

Fig. 1: CryptDB Architecture

Below we describe the steps involved in processing a query inside CryptDB.

1) In the first step a query is issued by the application server. The proxy server receives this query and it anonymizes the table name and each of the column names. The proxy server also encrypts all the constants in the query using the stored secret master key. The encryption layers or the onion layers are also adjusted based on the type of operation required by the issued query. For example if the query has to perform some equality checks then the deterministic encryption Scheme (DET)is applied to encrypt all the values in that particular column (on which equality check is to be performed).

2) The encrypted user query is then passed on to the DBMS server. The DBMS server executes these queries using standard SQL and also invokes UDF's to perform certain operations like token search and aggregation. The queries are executed on the encrypted database data.

3) The DBMS server performs computations on the encrypted data and forwards the encrypted results back to the proxy server.

4) The proxy server decrypts the encrypted query result obtained and returns it to the application server.

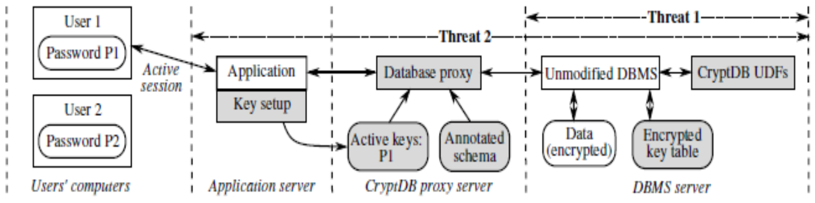

Figure 2: CryptDB's architecture consisting of two parts: a *proxy* and an unmodified *DBMS*. CryptDB uses user-defined functions (UDFs) to perform cryptographic operations in the DBMS. Rectangular and rounded boxes represent processes and data, respectively. Shading indicates components added by CryptDB. Dashed lines indicate separation between users' computers, the application server, a server running CryptDB's proxy (which is usually the same as the application server), and the DBMS server. The scope of the two threats CryptDB addresses is shown as dotted lines.

Threat 1: DBMS server compromise
CryptDB provides *confidentiality for the content of the data* and for names of columns and tables, but does not hide the overall table structure, the number of rows, the types of columns, or the approximate size of data in bytes. The only information that CryptDB reveals to the DBMS server is relationships among data items corresponding to *classes of computation* that queries perform on the database, such as comparing items for equality, sorting, or performing word search.
The granularity at which CryptDB allows the DBMS to perform a class of computations is an entire column (or a group of joined columns, for joins), which means that even if a query requires equality checks for a few rows, executing that query on the server would require revealing that class of computation for an entire column.

CryptDB provides the following properties:

- Sensitive data is never available in plaintext at the DBMS server.
- If the application requests no relational predicate filtering on a column, nothing about the data content leaks (other than its size in bytes).
- If the application requests equality checks on a column, CryptDB's proxy reveals which items repeat in that column, but not the actual values.
- If the application requests order checks on a column, the proxy reveals the order of the elements in the column.

Threat 2: arbitrary confidentiality attacks on any servers
The second threat is where the application server, proxy, and DBMS server infrastructures may be compromised arbitrarily.
The approach in threat 1 is insufficient because an adversary can now get access to the keys used to encrypt the entire database. The solution is to encrypt different data items (e.g., data belonging to different users) with different keys. To determine the key that should be used for each data item, developers annotate the application's database schema to express finer-grained confidentiality policies. A curious DBA still cannot obtain private data by snooping on the DBMS server (threat 1), and in addition, an adversary who compromises the application server or the proxy can now decrypt only data of currently logged-in users (which are stored in the proxy). Data of currently inactive users would be encrypted with keys not available to the adversary, and would remain confidential.
In this configuration, CryptDB provides strong guarantees in the face of arbitrary server-side compromises, including those that gain root access to the application or the proxy. CryptDB leaks at most the data of currently active users for the duration of the compromise, even if the proxy behaves in a Byzantine fashion. By "duration of a compromise", we mean the interval from the start of the compromise until any trace of the compromise has been erased from the system. For a read SQL injection attack, the duration of the compromise spans the attacker's SQL queries.

B. Design of Adaptive Encryption Architecture

In Adaptive encryption for public cloud database services, distributed and concurrent clients can issue direct SQL operations. By avoiding an architecture based on intermediate servers [1] between the clients and the cloud database, the proposed solution guarantees the same level of scalability and availability of the cloud service.

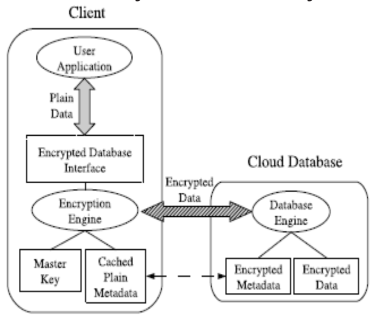

Fig:3 Encrypted cloud database architecture.

Fig.3 shows a scheme of an Adaptive encryption architecture where each client executes an encryption engine that manages encryption operations. This software module is accessed by external user applications through the encrypted database interface. Adaptive encryption architecture manages five types of information:

- **plain data** : represent the tenant information;
- **encrypted data** : are the encrypted version of the plain data, and are stored in the cloud database;
- **plain metadata** : represent the additional information that is necessary to execute SQL operations on encrypted data;
- **encrypted metadata:** are the encrypted version of the plain metadata, and are stored in the cloud database;
- **Master key** is the encryption key of the encrypted metadata, and is known by legitimate clients.

All data and metadata stored in the cloud database are encrypted. Any application running on a legitimate client can transparently issue SQL operations (e.g., SELECT, INSERT, UPDATE and DELETE) to the encrypted cloud database through the encrypted database interface.

Data transferred between the user application and the encryption engine are not encrypted, whereas information is always encrypted before sending it to the cloud database. When an application issues a new SQL operation, the encrypted database interface contacts the encryption engine that retrieves the encrypted metadata and decrypts them with the master key.

To improve performance, the plain metadata are cached locally by the client. After obtaining the metadata, the encryption engine is able to issue encrypted SQL statements to the cloud database, and then to decrypt the results. The results are returned to the user application through the encrypted database interface.

The architecture guarantees data confidentiality in a security model in which: the network is untrusted; tenant users are trusted, that is, they do not reveal information about plain data, plain metadata, and the master key; the cloud provider administrators are defined semi-honest or honest-but-curious [6], that is, they do not modify tenant's data and results of SQL operations, but they may access tenant's information stored in the cloud database.

III. ENCRYPTION SCHEMES

CryptDB and Adaptive encryption architectures uses various encryption schemes to process SELECT, RANGE, JOIN, and simple aggregation queries over various forms of encrypted data:

- **Randomized Symmetric Encryption (RND):** AES is used with an IV in CBC mode. Due to randomization, the scheme is semantically secure. It does not support any SQL operator, and it is used only for data retrieval.
- **Deterministic Symmetric Encryption (DET):** AES in ECB mode is used to support conditional selection queries via equality checks on cipher texts. While this type of encryption allows comparisons, it also opens the door to frequency analysis attacks.
- **Order Preserving Encryption (OPE):** Boldyreva's OPE scheme is used to support range queries. This scheme reveals the minimum, maximum, order and the most significant digit. It supports the comparison SQL operators (i.e. =; <; ≤; ≥; >).
- **Additive Homomorphic Encryption (HOM):** Paillier's encryption scheme [7] is used to support simple aggregation queries and arithmetic operations such as ADD, INCREMENT, SUM, and AVERAGE.
- **Search:** it supports equality check on full strings (i.e., the LIKE operator).
- **Plain:** it does not encrypt data, but it is useful to support all SQL operators on non confidential data.

If each column of the database was encrypted with only one algorithm, then the database administrator would have to decide at design time which operations must be supported on each database column. However, this solution is impractical for scenarios in which the database workload changes over time.

This issue can be addressed through ***adaptive encryption schemes*** that support at runtime SQL operations while preserving the highest possible data confidentiality level on the encrypted data. To this purpose, the encryption algorithms are organized into structures called ***onions***, where each onion is composed by an ordered set of encryption algorithms, called (encryption) layers. Outer layers guarantee higher level of data confidentiality and support fewer operations on encrypted data. Hence, each onion supports a specific set of operations.

The following onions are designed:

- **Onion-Eq:** it supports the equality operator, and integrates Plain, Det and Rand layers.
- **Onion-Ord:** it supports the comparison operators (i.e.,=; <; ≤; > ; ≥), and integrates Plain, Ope and Rand layers.
- **Onion-Sum**: it supports the sum operator, and integrates Plain, Sum and Rand layers.
- **Onion-Search:** it supports the string equality operator (LIKE), and integrates the Plain, Search and Rand layers.
- **Onion-Single-Layer:** this is a special type of onion that supports only one encryption layer.

Each plaintext column is converted into one or more encrypted columns, each one corresponding to an onion. Each plaintext value is encrypted through all the layers of its onions.

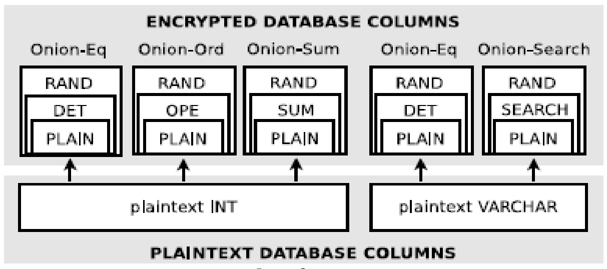

Fig: 4 Example of onion structures.

IV. METADATA STRUCTURE IN ADAPTIVE ENCRYPTION

Metadata include all information that allows a legitimate client knowing the master key to execute SQL operations over an encrypted database. They are organized and stored at table-level granularity to reduce communication overhead for retrieval, and to improve management of concurrent SQL operations [8].

We define all metadata information associated with a table as table metadata. Table metadata include the correspondence between the plain table name and the encrypted table name because each encrypted table name is randomly generated. Moreover, for each column of the original plain table they also include a set of column metadata containing the name and the data type of the corresponding plain column (e.g., integer, varchar, date time). Each set of column metadata is associated with as many sets of onion metadata as the number of onions associated with the column. Onion metadata describe all the encryption information about an onion and its layers, hence they are organized in a data structure that contains the following attributes:

- The ***encrypted name*** is the name of the encrypted column (i.e., the onion) in the encrypted cloud database;
- the ***actual encryption layer*** is the name of the most external layer of the encrypted data (e.g., Rand) stored in the column;
- the ***field confidentiality*** denotes which set of keys must be used to encrypt a column data, because only columns that share the same encryption keys can be joined; we identify three types of field confidentiality parameters: *self* denotes a private set of keys for the column, *multi-column* identifies the sharing of the same set of keys among two columns, database imposes the same set of keys on all columns of the same data type. the ***onion*** parameter identifies the type of onion that is used to encrypt data (e.g., Onion-Eq).

Each set of onion metadata is associated with as many sets of layer metadata as the number of layers required by the onion type. Each set of layer metadata includes an encryption key and a label identifying the corresponding encryption algorithm. The set of encryption keys for each onion is generated according to the field confidentiality parameter imposed on each encrypted column.

V. ENCRYPTED DATABASE MANAGEMENT

There are three main operations involved in the encrypted database management:
Database creation, execution of SQL operations, and adaptive layer removal.

A. Database Creation

In the setup phase, the database administrator generates a master key, and uses it to initialize the architecture metadata. The master key is then distributed to legitimate clients. Table creation requires the insertion of a new row in the metadata table. For each table creation, the administrator adds a column by specifying the column name, data type and confidentiality parameters. If the administrator does not specify the confidentiality parameters of a column, they are automatically chosen by the encryption engine with respect to some tenant's policy. Typically, the default policy assumes that each column is associated with all the compatible onions, and the starting layer of each onion is set to the strongest encryption algorithm.

B. Execution of SQL Operations

When a user/application wants to execute an operation on the cloud database, the client encryption engine analyzes the SQL command structure and identifies which tables, columns and SQL operators are involved. The client issues a request for the table metadata for each involved table, and decrypts the metadata with the master key. Then, the client determines whether the SQL operators are supported by the actual layers of the onions associated with the involved columns. If required, the client issues a request for layer removal in order to support the SQL operators at runtime. By using the information stored in the table metadata, the client is able to encrypt the parameters of the SQL operations: tables and columns names, and constant values.

The client issues this new statement called encrypted SQL operation to the cloud database which transparently executes it over encrypted data. The encrypted results are decrypted using information contained in the metadata.

C. Adaptive Layer Removal

The adaptive layer removal is the process that dynamically removes the external layer of an onion in order to adaptively support SQL operations issued by legitimate clients.

The cloud database can execute the adaptive layer removal if and only if a legitimate client invokes the stored procedure and gives to it the decryption key of the most external encryption layer. As each layer has a different encryption key, the data remain encrypted and the cloud provider cannot access plaintext data. For security reasons, we also assume that the adaptive layer removal mechanism does never expose the Plain layer of an onion.

VI. LIMITATIONS OF CRYPTDB

CryptDB has certain limitations. For example,

- ***It does not support both computation and comparison in the same predicate***, such as WHERE salary > age*2+10. CryptDB can facilitate processing such a query, but it would require a little bit of processing on the frontend. To use CryptDB, the query can be rewritten into a sub query that selects a whole column, SELECT age*2+10 FROM . . ., which CryptDB computes using HOM, then re-encrypting the results in the frontend, creating a new column (call it aux) at the server consisting of the newly-encrypted values, and finally running the original query with the predicate WHERE salary > aux.
- ***The current CryptDB prototype does not support stored procedures or other user-defined functions on the server.*** Supporting stored procedures written in SQL should be straightforward, by having the frontend rewrite the SQL statements inside of the stored procedure as it would for any other query.
- ***CryptDB's design cannot support the execution of arbitrary user defined Functions (not written in SQL) over encrypted data.***
- ***Finally, our current prototype handles server-side auto-increment columns by leaving the column values in plaintext on the server.***

VII. IMPROVING SECURITY AND PERFORMANCE OF CRYPTDB

Although CryptDB can operate with an unmodified and unannotated schema, its security and performance can be improved through several optional optimizations, as described below.

A. Security Improvements

- *Minimum onion layers:* Application developers can specify the lowest onion encryption layer that may be revealed to the server for a specific column. In this way, the developer can ensure that the proxy will not execute queries exposing sensitive relations to the server. For example, the developer could specify that credit card numbers should always remain at RND or DET.
- ***In-proxy processing****:* Although CryptDB can evaluate a number of predicates on the server, evaluating them in the proxy can improve security by not revealing additional information to the server.
  One common use case is a SELECT query that sorts on one of the selected columns, without a LIMIT on the number of returned columns. Since the proxy receives the entire result set from the server, sorting these results in the proxy does not require a significant amount of computation, and does not increase the bandwidth requirements. Doing so avoids revealing the OPE encryption of that column to the server.
- *Training mode:* CryptDB provides a training mode, which allows a developer to provide a trace of queries and get the resulting onion encryption layers for each field, along with a warning in case some query is not supported. The developer can then examine the resulting encryption levels to understand what each encryption scheme leaks. If some onion level is too low for a sensitive field, she should arrange to have the query processed in the proxy, or to process the data in some other fashion, such as by using a local instance of SQLite.
- *Onion re-encryption*: In cases when an application performs infrequent queries requiring a low onion layer (e.g., OPE), CryptDB could be extended to re-encrypt onions back to a higher layer after the infrequent query finishes executing. This approach reduces leakage to attacks happening in the time window when the data is at the higher onion layer.

B. Performance Optimizations

- *Developer annotations:* By default, CryptDB encrypts all fields and creates all applicable onions for each data item based on its type. If many columns are not sensitive, the developer can instead provide explicit annotations indicating the sensitive fields and leave the remaining fields in plaintext.
- *Known query set*: If the developer knows some of the queries ahead of time, as is the case for many web applications, the developer can use the training mode described above to adjust onions to the correct layer a priori, avoiding the overhead of runtime onion adjustments. If the developer provides the exact query set, or annotations that certain functionality is not needed on some columns, CryptDB can also discard onions that are not needed.
- *Cipher text pre-computing and caching*: The proxy spends a significant amount of time encrypting values used in queries with OPE and HOM. To reduce this cost, the proxy pre-computes (for HOM) and caches (for OPE) encryptions of frequently used constants under different keys. Since HOM is probabilistic, cipher texts cannot be reused. Therefore, in addition, the proxy pre-computes HOM's Paillier r^n randomness values for future encryptions of any data. This optimization reduces the amount of CPU time spent by the proxy on OPE encryption, and assuming the proxy is occasionally idle to perform HOM pre-computation, it removes HOM encryption from the critical path.

VIII. CONCLUSIONS

This paper presented differences between two different architectures for running SQL queries over encrypted data.

A. CryptDB

CryptDB ***is*** a practical and novel system for ensuring data privacy on an untrusted SQL DBMS server. CryptDB uses three novel ideas to achieve its goal: an SQL-aware encryption strategy, adjustable query-based encryption, and onion encryption.

CryptDB requires no changes to application or DBMS server code, and uses user-defined functions to perform cryptographic operations inside an existing DBMS engine, including both Postgres and MySQL.

CryptDB cannot provide confidentiality in the face of vulnerabilities that trick the user's client machine into issuing unwanted requests (such as cross-site scripting or cross-site request forgery vulnerabilities in web applications).

B. Adaptive encryption architecture

It is the first proxy-free architecture for adaptive encryption of cloud databases that does not limit the availability, elasticity and scalability of a plain cloud database because multiple clients can issue concurrent operations without passing through some centralized component.

It supports data confidentiality in cloud database environments without requiring any intermediate trusted proxy.

Adaptive encryption mechanisms have two main benefits:

They guarantee at runtime the maximum level of data confidentiality for any SQL workload, and they simplify database configuration at design time. If the workload is characterized by many similar operations, the present alternative is to accept this cost when data confidentiality is more important than performance.

Hence we can say the adaptive encryption performance efficiency more compare to CryptDB.

REFERENCES

[1] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB: Protecting confidentiality with encrypted query processing," in Proc. 23rd ACM Symp. Operating Systems Principles, Oct. 2011, pp. 85–100.

[2] A. Boldyreva, N. Chenette, and A. O'Neill, "Order-preserving encryption revisited: Improved security analysis and alternative solutions," in Proc. 31st Annu. Int. Conf. Adv. Cryptology, Aug.2011, pp. 578–595.

[3] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in Proc. 17th Int. Conf. Theory Appl. Cryptographic Tech., May 1999, pp. 223–238.

[4] D. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in Proc. IEEE Symp. Security Privacy, May 2000, pp. 44–55.

[5] L. Ferretti, M. Colajanni, and M. Marchetti, "Distributed, concurrent, and independent access to encrypted cloud databases," IEEE Trans. Parallel Distrib. Syst., vol. 25, no. 2, pp. 437–446, Feb. 2014.

[6] O. Goldreich, Foundations of Cryptography: Volume 2, Basic Applications. Cambridge, U.K.: Cambridge Univ. Press, 2004

[7] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in Proc. 17th Int. Conf. Theory Appl. Cryptographic Tech., May 1999, pp. 223–238.

[8] L. Ferretti, M. Colajanni, and M. Marchetti, "Supporting security and consistency for cloud database," in Proc. 4th Int. Symp. Cyberspace Safety Security, Dec. 2012, pp. 179–193.