E-ISSN 2581 – 7957
P-ISSN 2277 – 3916

CVR Journal of Science and Technology, Volume 22, June 2022
DOI: 10.32377/cvrjst2201

A Scheme for Verifying Integrity of SQL Query Processing on Encrypted Databases

K. Srinivasa Reddy[1] and K. Pranitha Kumari[2]
[1]Assoc. Professor, CVR College of Engineering/CSE Department, Hyderabad, India
Email: k.srinivasareddy@cvr.ac.in
[2]Assoc. Professor, CVR College of Engineering/CSE Department, Hyderabad, India
Email: k.pranithakumari@ cvr.ac.in

Abstract: **In the database-as-a-service model, a service provider hosts the clients' data and allows access to the data through the Internet. Database-as-a-service model offers considerable benefits to organizations with data management needs by allowing them to outsource their data management infrastructures. Yet, the model introduces many significant challenges, that of data privacy and security. Ensuring the integrity of the database, which is hosted by a service provider, is a critical and challenging problem in this context. We propose a novel Query Result Integrity Guarantee (QRIG) scheme over encrypted databases, which allow a database enquirer to verify that their queries were faithfully executed by the server. The experimental results obtained in this paper show the performance of proposed scheme.**

Index Terms: **data integrity, query result, database, encryption, security**

I. INTRODUCTION

Nowadays outsourced IT services have become very popular. Users no longer need to be concerned about providing infrastructure or scaling up software for their computation. With the advent of outsourced IT services like cloud computing any required infrastructure, software or services can be provided to users by third persons. Users can speed up their computation based on the load by boosting the software without the rise in cost, because using 1000 hours of a server does not cost more than using 1000 servers for an hour [1]. Mobility is the other advantage of cloud computing. Users can run their computation by logging into their account in cloud via their cell phones on their way to work. Cloud computing also reduces cost for users. Users will pay for the services that they receive during the service time. The other costs like software (in our case database licensing costs), hardware and maintenance are associated with service providers which makes cloud computing more attractive for users [2]. Although outsourced IT services and cloud computing are desirable for users but there is a concern that prevents most of users from using these services. Security is the main concern that stops users to delegate their computation to the cloud. Users want to ensure that services provided by third party are secure and trusted. Recent study on 500 IT administrators and IT managers in 17 countries revealed that 80% of them tend to use existing internal systems rather than using cloud infrastructures. The fear of losing control over protecting data was the main reason for them to stay with existing internal systems [3].

In this paper we focus on providing security for the outsourced database. Outsourced database security has two aspects: privacy and data integrity. Privacy in databases were studied in many research works and different approaches have been presented [4][5]. Data integrity recently became an interesting challenge for researchers. Data integrity can be provided at different levels of granularity. In principle, integrity checks can be at the level of a table, a column, a row (a record or a tuple of the table), or an individual attribute value. Providing integrity checks at the table (or column) level implies that the entire data pertaining to that table (or column) should be returned in the query reply for the client to verify the integrity of the query response. This is clearly impractical as it requires transferring large amounts of data to the client. Hence, we do not consider this to be a viable approach. On the other hand, computing integrity checks at the level of individual attribute values yields a very large number of signatures which is very expensive for the signer (owner) in terms of computation as well as for the server in terms of storage.

Therefore, the optimal choice is to provide integrity at the record level. This enables the server to return in response to a query any set of matching records along with their respective integrity checks. Of course, computing integrity checks over the entire record, as opposed to individual attributes, inferred that the smallest unit of data returned as a query reply is an entire record, even when the querying client is only interested in a single field.

Query result integrity has three aspects: Correctness, Completeness and Freshness. Correctness means that the results returned by server must be genuine records. Genuine records are those records that exist in original database and have not been modified by server. Completeness means that returned records are the results of executing the query over the entire database and all the possible records that satisfy the query are included. Freshness means that query result contains the records based on executing the query over the most updated database. In this paper, we propose a novel Query Result Integrity Guarantee (QRIG) scheme for verifying integrity of query processing over encrypted databases and to satisfy the three aspects of query result integrity: correctness, completeness, and freshness.

In this paper, we consider insert, delete, and update queries and the standard SQL queries involving *SELECT* clauses such as equality, range, join and aggregate queries. This paper focuses on providing *correctness* and *freshness* of query replies returned by the server. A related, and

Received on 09-08-2021, Revised on 18-10-2021, Accepted on 30-10-2021.

CVR Journal of Science and Technology, Volume 22, June 2022
DOI: 10.32377/cvrjst2201

equally important, issue is the *completeness* of query replies. Although we consider it debatable whether *completeness* is a security concern, we acknowledge that it poses a challenge which needs to be addressed in the outsourced databases. However, we consider *completeness* of query only with respect to SCOPE scheme [32], in which any sensitive column is encrypted and stored at trusted proxy.

The remaining part of this paper is structured as following. Section II describes the related work. Section III describes the proposed system model. Section IV describes the proposed scheme in detail. In Section V the results of experiments are summarized, and our conclusions are presented in Section VI.

II. Related Work

The problem of assuring query integrity in the context of outsourced data was fundamentally related to the concept of certified data structures [27], which presents some results that are conceptually important but not efficient. The state-of-the-art solutions to query integrity are due to [13][23], which are the only solutions that support selection, projection and join queries simultaneously. These two solutions follow two respective approaches to the query integrity problem.

The tree-based approach: Basically, this approach uses the Merkle hash tree [15] or its variants to index search keys [11][17][13][7][16][31][20][21]. As a result, this approach leads to logarithmic complexity in terms of both communication and verification, possibly with some further tricks (e.g., using the Merkle hash tree to maintain signatures at multiple hash tree levels [11]). The best solution in this approach is due to [13], which uses the Merkle B-tree and the Embedded Merkle B-tree to reduce I/O operations.

The signature-based approach: Basically, this approach uses the signature aggregation technique [5][18] to aggregate the validity of query answers [18][19][23][22]. As a result, this approach can lead to low (even constant) communication complexity but may require special treatment for handling more powerful (e.g., projection) queries and often leads to large storage and computational complexities. The best solution in this approach is due to [23], which uses aggregate signatures to sign each attribute and returns a single signature as the validity proof for projection queries. This solution uses a chaining signing technique to build the index for the search key to facilitate range queries and publishes a certified bitmap corresponding to every update to facilitate dynamic updates. These cause a large storage and communication overhead while including many enfolding and pairing operations.

There are studies that are somewhat related to the theme of the present paper as well [33][34][35]. These include authenticating the answers to set operations using accumulator [25], authenticating the answers to aggregate queries using authenticated prefix-sums trees [14], authenticating the answers to join queries [30], authenticating count queries with respect to multi-dimensional data while preserving privacy [29], and assuring probabilistic integrity in selection and join operations [28]. Query integrity is also somewhat related to outsourced verifiable computation [1][6][10].

III. System Model

The proposed system model is shown in Fig. 1 in which the trusted proxy hosts the creation and management of the encrypted database. All tenant database users can submit SQL queries directly to the encrypted cloud database. The whole tenant organization data is stored in an encrypted form in the cloud database. By using SQL-aware encryption schemes, the cloud database server can process user's SQL queries on encrypted data without decryption. The key management module at proxy handles generation, derivation, and revocation of cryptographic keys.

The trusted proxy has four significant responsibilities:

1. **Key Management:** The trusted proxy generates and manages the keys required to decrypt the data.
2. **Query Rewrite:** The queries written by the user contain plain text. Therefore, the plain text in the query has to be encrypted by the trusted proxy. Moreover, depending on the encryption scheme used to encrypt the data in the cloud, the queries written by the user might need to be restructured using metadata information.
3. **Decryption and Post-Processing:** Depending on the encryption scheme, the trusted proxy needs to decrypt and possibly post-process the encrypted query results.
4. **Verify query result integrity:** Proxy authenticates the origin and verifies the integrity of data returned by the service provider in response to a posed query.

The system model works as follows: The Trusted Proxy (TP) translates the tenant organization's access control policies into an access control matrix. TP distributes unique secret keys to the users at the creation of their accounts according to the access control matrix. These keys enable the users to access all and only the subsets of encrypted tenant data on which the users have legitimate access. When a user enters his credentials, TP validates the user's credentials. The TP takes as its input the original plaintext database and generates the tuple integrity code and produces the

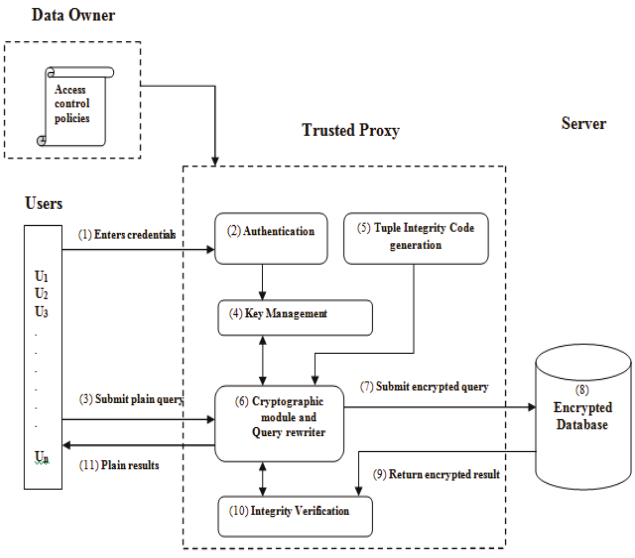

Figure 1. System model

encrypted tenant data. Each user can execute SQL operations through the TP. The TP takes as its inputs the user credentials and the encrypted metadata and translates plaintext SQL queries into encrypted SQL queries that can be executed on encrypted data at server. The TP decrypts the encrypted results returned by the server and verifies the integrity of the results. Finally, the verified plain results are returned to the user.

A. Generic Assumptions

Various assumptions that are made in this paper are as follows:

1. Server is not trusted. Hence proxy does not share its secret encryption keys with server.
2. Proxy is fully trusted and won't be compromised. If we remove this assumption, then security can never be guaranteed since adversary can compromise proxy and see all the data.
3. The communication channel between client, proxy and server is secure. This can be ensured by using various techniques such as TLS (Transport Layer Security) and SSL (Secure Sockets Layer).

All the encryption schemes used here are individually secure.

B. Adversary Model

The model used in this work is "malicious" adversary model. The assumption of this model is that the server can misbehave in any way, such as returning incorrect answers to the user query.

The assumptions made here allow our scheme to protect the data against:

1. An adversary that eavesdrops network traffic cannot access any plaintext information because SQL operations issued to the cloud database are protected by using standard encryption protocols (e.g., SSL).
2. An adversary that has breached the cloud database cannot access confidential information, because our scheme encrypts client data with semantically secure algorithms and the cloud provider never obtains the decryption keys.

IV. Query Result Integrity Guarantee Scheme

The proposed QRIG scheme works in two phases, the first phase is called the Tuple Integrity Code generation and the second phase is called Query Result Integrity Verification.

A. Tuple Integrity Code Generation

The tuple-level integrity represents that the content of a record has not been manipulated in an unauthorized manner. Although it may not be apparent, data encryption does not provide data integrity automatically. The owner of the decryption key can decrypt the encrypted messages, which were encrypted with the same key. But this does not guarantee that the encrypted message has not been manipulated by the adversary. The discussion of how encrypted messages can be manipulated undetectably can be found in [12]. This motivates the need for data integrity measures over encrypted data.

To provide tuple-level integrity we propose a scheme based on *Tuple Integrity Codes (TICs)*. TICs are specially computed representative images for each record with certain security and uniqueness measures. Fig. 2 shows the procedure that provides tuple-level data integrity. The data owner/user has a record r that will be inserted into the database, which is maintained by the server. The trusted proxy first computes the hash code of the record $H=h(r)$ by using a TIC algorithm [8], which produces *Tuple Integrity Code* (TIC). After this step, the trusted proxy integrates the

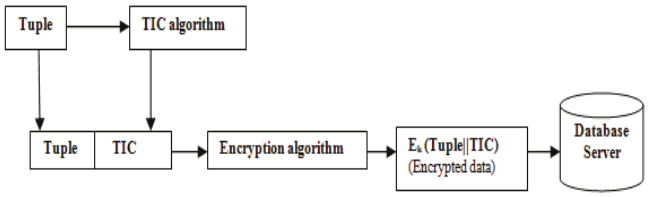

Figure 2. Tuple integrity code generation

hash code H with the original record text r and encrypts them together by using any deterministic encryption algorithm E [9] with secret key k i.e., the trusted proxy computes ciphertext $C= E_k (r||h(r))$ where || represents concatenation. The trusted proxy inserts ciphertext C as an *encTuple* into the database.

B. Query Result Integrity Verification

Whenever the user requests a record, the server sends back the corresponding *encTuple* in encrypted form to trusted proxy. To verify the integrity of the record, the trusted proxy first decrypts the *encTuple* recovering r' and H', which is the TIC, parts. Since only the trusted proxy has the secret key k for encryption algorithm no one else can decrypt. Then the trusted proxy independently computes $h(r')$ of received record r' and compares that with the hash code H'. If they are equivalent, this verifies that the received record is authentic and has data integrity, i.e., has not been manipulated in an unauthorized manner. The query result integrity verification process is depicted in Fig. 3.

C. Query Processing

Here, we discuss how a trusted proxy uses the QRIG scheme

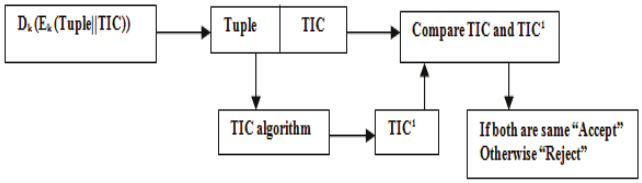

Figure 3. Query result integrity verification

and verifies the integrity of query results returned by the server when a user executes various SQL queries over an encrypted database. For example, consider an employee table in a database consisting of name and salary columns as given in Table I. The employee table is mapped to a corresponding encrypted table, shown in Table II, at the server. TID represents tuple identifier, the second column *encTuple* contains the string corresponding to the encrypted tuples in employee table.

For instance, the first tuple is encrypted to Enc (John, 32,000 || TIC), where *Enc* is a deterministic encryption algorithm [24] with key k. The third column corresponds to

the *eName* which is encrypted with deterministic encryption algorithm such as AES [26]. The fourth column represents the *eSalary* which is encrypted with SCOPE scheme [32], since order operations are usually applied on it.

SELECT Query

Suppose the user sends the select query Q = "select * from employee where salary ≥ x and salary ≤ y" to server. We consider correctness, completeness and freshness of Q while verifying the query result integrity.

The QRIG scheme for verifying the correctness of Q results works as follows:

1. The trusted proxy rewrites the Q using SCOPE scheme [32], to an encrypted query Q_E and sends it to server.
2. The server processes Q_E and returns the encrypted tuples result to the trusted proxy.
3. The trusted proxy applies the query result integrity verification process given in Section 4.2.
4. The trusted proxy returns the plaintext results to the user.

The QRIG scheme for verifying the completeness of Q results works as follows:

1. The trusted proxy temporarily stores the count of the data items that satisfies the condition in Q while rewriting the Q using SCOPE scheme. In SCOPE scheme, the trusted proxy stores the sensitive column (i.e., salary) in encrypted form.
2. The trusted proxy rewrites Q to an encrypted query Q_E and sends it to server.
3. The server processes Q_E and returns the encrypted tuples result to the trusted proxy.
4. The trusted proxy computes the count of the encrypted tuples returned by the server and compares it with the count value stored with it. If both count values are same then it accepts the result, otherwise it rejects the result.
5. The trusted proxy returns the plaintext results to the user.

If the SELECT query result satisfies correctness and completeness, then it automatically satisfies the freshness. The SELECT query includes equality, range, join and aggregate queries. Therefore, the QRIG scheme is

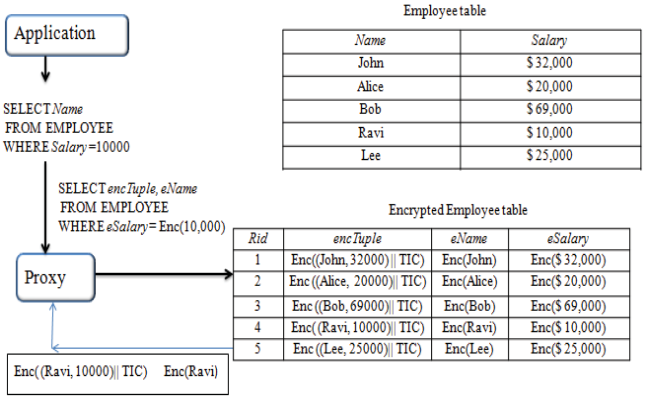

Figure 4. Equality query result integrity verification

applicable for verifying correctness, completeness, and freshness of all SELECT queries.

The example given in Fig. 4 shows how the QRIG scheme is applicable for verifying correctness, completeness, and freshness of equality query result.

The trusted proxy temporarily stores the count of the data items that satisfies the condition in the equality query EQ (count=1). The trusted proxy rewrites EQ to an encrypted query EQ_E and sends it to server. The server processes EQ_E and returns the encrypted tuples result to the trusted proxy. The trusted proxy applies the query result integrity verification process given in Section 4.2 for correctness and computes the count of the encrypted tuples returned by the server and compares it with the count value stored with it for completeness and returns the plaintext results to the user. Since the equality query result satisfies correctness and completeness then it automatically satisfies the freshness.

UPDATE Query

The trusted proxy interacts with the server to update the stored table with the update information. The following are the different cases of update operation:

1. *Insertion*: Suppose update is "insert the tuple *r* into table". The trusted proxy applies the tuple integrity code generation process given in Section 4.1. The trusted proxy delivers update information to the server and the server inserts *r* into table. The trusted proxy maintains the count *c* of the existing tuples in the table. The trusted proxy updates *c* whenever new tuples *t* is inserted into the table (i.e., c = c + t). Therefore, the trusted proxy can verify the completeness and freshness of tuples by getting the count *c′* of the total tuples from the server and comparing *c′* with *c*.
2. *Deletion*: Suppose update is "delete the tuple *r* from table". The trusted proxy delivers update information to the server and the server deletes *r* from table. The trusted proxy gets the count *d* of the tuples that are deleted from the table. The trusted proxy maintains the count *c* of the existing tuples in the table. The trusted proxy updates *c* whenever new tuples *t* is deleted from the table (i.e. c = c - t). Therefore, the trusted proxy can verify the completeness and freshness of tuples by getting the count *c′* of the tuples from the server and comparing *c′* with *c*.
3. *Modify*: Suppose update is "update the tuple *r* with *r′* ". The trusted proxy applies the tuple integrity code generation process given in Section 4.1 for *r′*. The trusted proxy delivers update information to the server and the server updates *r* to *r′*. The trusted proxy maintains the count *c* of the existing tuples in the table. Therefore, the trusted proxy can verify the completeness and freshness of tuples by getting the count *c′* of the tuples from the server and comparing *c′* with *c*.

The correctness of the tuples present in the table is verified during the data retrieval process which is described in Section 4.2.

V. Experimental Evaluation

In this section we evaluate the performance of QRIG scheme on an encrypted database. QRIG is implemented in Java over MySQL database server. We measured the performance of QRIG on a machine with a 2.27 GHz Intel Core i5 processor running with Windows 7 with only a single core enabled for consistency, running both the client and the server on the same machine and with 2GB of memory.

In the experiments, we consider employee database consisting of one table with two columns. Here the table columns are encrypted with AES and SCOPE to support equality and ordering operations over encrypted data respectively. The database columns not requiring any computation can be encrypted through standard algorithms such as AES with random initialization vectors.

QRIG scheme performances are examined by considering insert and retrieve operations on encrypted employee database. In our experiments, we varied the total database size between one hundred and one lakh entries. We were able to run experiments up to one lakh entries, limited only by the RAM size available on our workstation. The efficiency of QRIG is measured in terms of the time taken to generate tuple integrity codes during insert operations. During the retrieval operation, the efficiency of QRIG is measured in terms of the time taken for verifying query result integrity by applying a set of retrieval queries on encrypted employee database.

A. Database Insert And Retrieve

During the experiment, an insert query is applied on employee table by inserting 10^2,10^3,10^4 and 10^5 records at a time using QRIG scheme and the corresponding user time, system time, and CPU time are calculated and is clearly depicted in Table III. Next, a set of M range queries where M=10, 20, 30…100 are applied on 1000 records present in an encrypted employee database and the corresponding user time, system time, and CPU time are calculated. The results are described in Table IV.

Here, "user time" is the time spent running the application code, "system time" is the time spent running OS code on behalf of the application (such as for IO) and "CPU time" is user time plus system time. It is the total time spent using a CPU for your application. Here, the total insertion time (CPU time) = User time + System time.

VI. Conclusions

We propose a novel Query Result Integrity Guarantee scheme over encrypted databases, which allow a database querier to verify that their queries were faithfully executed by the server. The proposed scheme provides the security of the stored data against the malicious attacks as well as the database integrity features, which ensure the correctness, completeness and freshness of the data stored at the server. Our approach is efficient, and it only introduces small storage overhead.

TABLE I.
Employee Table

Name	Salary
John	$ 32,000
Alice	$ 20,000
Bob	$ 69,000
Ravi	$ 10,000
Lee	$ 25,000

TABLE II.
Encrypted Employee Table

Tid	encTuple	eName	eSalary
1	Enc (John, 32,000 || TIC)	Enc (John)	Enc (32,000)
2	Enc (Alice, 20,000 || TIC)	Enc (Alice)	Enc (20,000)
3	Enc (Bob, 69,000 || TIC)	Enc (Bob)	Enc (69,000)
4	Enc (Ravi, 10,000 || TIC)	Enc (Ravi)	Enc (10,000)
5	Enc (Lee, 25,000 || TIC)	Enc (Lee)	Enc (25,000)

TABLE III.
Applying Insertion Query on Employee Table

Number of Records Inserted	User time (seconds)	System time (seconds)	CPU time (seconds)
100	0.03	0.01	0.04
1000	0.09	0.02	0.11
10000	0.22	0.02	0.24
100000	1.05	0.08	1.13

TABLE IV.
Applying Insertion Query on Employee Table

Number of Records Inserted	User time (seconds)	System time (seconds)	CPU time (seconds)	Number of Records retrieved
10	0.15	0.01	0.16	5382
20	0.22	0.01	0.23	10764
30	0.24	0.02	0.26	16137
40	0.27	0.03	0.30	21430
50	0.30	0.04	0.34	26326
60	0.35	0.04	0.39	31708
70	0.40	0.04	0.44	37090
80	0.46	0.04	0.50	42463
90	0.49	0.05	0.54	47756
100	0.53	0.05	0.58	52652

References

[1] B. Applebaum, Y. Ishai, and E. Kushilevitz, "From secrecy to soundness: Efficient verification via secure computation," *Automata, Languages and Programming*, vol. 6198, pp. 152–163, 2010.

[2] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," *In Proceedings of the 14th ACM conference on Computer and communications security*, pp. 598–609, 2007.

[3] G. Ateniese, S. Kamara, and J. Katz, "Proofs of storage from homomorphic identification protocols," In Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology, pp. 319–333, 2009.

[4] M. Bellare and G. Neven, "Multi-signatures in the plain public-key model and a general forking lemma," *In ACM Conference on Computer and Communications Security*, pp. 390–399, 2006.

[5] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," *In Proceedings of the 22nd international conference on Theory and applications of cryptographic techniques*, pp. 416–432, 2003.

[6] K.-M. Chung, Y. Kalai, and S. Vadhan, "Improved delegation of computation using fully homomorphic encryption," *In Proceedings of the 30th annual conference on Advances in cryptology*, pp. 483–501, 2010.

[7] P. Devanbu, M. Gertz, C. Martel, and S. G. Stubblebine, "Authentic data publication over the internet," *J. Computer Security*, vol. 11(3), pp. 291–314, 2003.

[8] C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," *In Proceedings of the 16th ACM conference on Computer and communications security*, pp. 213–222, 2009.

[9] A. Fiat, "Batch rsa," *In Proceedings on Advances in cryptology*, pp. 175–185, 1989.

[10] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: outsourcing computation to untrusted workers," *In Proceedings of the 30th annual conference on Advances in cryptology*, pp. 465–482, 2010.

[11] M. Goodrich, R. Tamassia, and N. Triandopoulos, "Super-efficient verification of dynamic outsourced databases," *Topics in Cryptology*, vol. 4964, pp. 407–424, 2008.

[12] A. Juels and B. S. Kaliski, Jr., "Pors: proofs of retrievability for large files," *In Proceedings of the 14th ACM conference on Computer and communications security*, pp. 584–597, 2007.

[13] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin, "Dynamic authenticated index structures for outsourced databases," *In Proceedings of the ACM SIGMOD international conference on Management of data*, pp. 121–132, 2006.

[14] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin, "Authenticated index structures for aggregation queries," *ACM Trans. Inf. Syst. Secur.*, vol. 13(4), pp. 1–32, 2010.

[15] R. C. Merkle, "A certified digital signature," *In Proceedings on Advances in cryptology*, pp. 218–238, 1989.

[16] K. Mouratidis, D. Sacharidis, and H. Pang, "Partially materialized digest scheme: an efficient verification method for outsourced databases," *The VLDB Journal*, vol. 18(1), pp. 363–381, 2009.

[17] E. Mykletun, M. Narasimha, and G. Tsudik, "Providing authentication and integrity in outsourced databases using merkley hash trees," *In UCI-SCONCE Technical Report*, 2003.

[18] E. Mykletun, M. Narasimha, and G. Tsudik, "Authentication and integrity in outsourced databases," *Trans. Storage*, vol. 2(2), pp. 107–138, 2006.

[19] M. Narasimha and G. Tsudik, "Authentication of outsourced databases using signature aggregation and chaining," *In Proceedings of the 11th international conference on Database Systems for Advanced Applications*, pp. 420–436, 2006.

[20] G. Nuckolls, "Verified query results from hybrid authentication trees," In Proceedings of the 19th annual IFIP WG 11.3 working conference on Data and Applications Security, pp. 84–98, 2005.

[21] B. Palazzi, M. Pizzonia, and S. Pucacco, "Query racing: fast completeness certification of query results," *In Proceedings of the 24th annual IFIP WG 11.3 working conference on Data and applications security and privacy*, pp. 177–192, 2010.

[22] H. Pang, A. Jain, K. Ramamritham, and K.-L. Tan, "Verifying completeness of relational query results in data publishing," *In Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pp. 407–418, 2005.

[23] H. Pang, J. Zhang, and K. Mouratidis, "Scalable verification for outsourced dynamic databases," *Proc. VLDB Endow.*, vol. 2(1), pp. 802–813, 2009.

[24] C. Papamanthou, R. Tamassia, and N. Triandopoulos, "Authenticated hash tables," *In Proceedings of the 15th ACM conference on Computer and communications security*, pp. 437–448, 2008.

[25] C. Papamanthou, R. Tamassia, and N. Triandopoulos, "Optimal verification of operations on dynamic sets," *In Proceedings of the 31st annual conference on Advances in cryptology*, pp. 91–110, 2011.

[26] H. Shacham and B. Waters, "Compact proofs of retrievability," In Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology, pp. 90–107, 2008.

[27] R. Tamassia and N. Triandopoulos, "Certification and authentication of data structures" *In AMW*, 2010.

[28] M. Xie, H. Wang, J. Yin, and X. Meng, "Integrity auditing of outsourced data," *In Proceedings of the 33rd international conference on Very large data bases*, pp. 782–793, 2007.

[29] J. XU and E.-C. CHANG, "Authenticating aggregate range queries over multidimensional dataset," *Cryptology ePrint Archive*, 2010.

[30] Y. Yang, D. Papadias, S. Papadopoulos, and P. Kalnis, "Authenticated join processing in outsourced databases," *In Proceedings of the 35th SIGMOD international conference on Management of data*, pp. 5–18, 2009.

[31] Y. Yang, S. Papadopoulos, D. Papadias, and G. Kollios, "Spatial outsourcing for location-based services," *In Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, pp. 1082–1091, 2008.

[32] Srinivasa Reddy, K. and Ramachandram, S, "A secure, fast insert and efficient search order preserving encryption scheme for outsourced databases," *Int. J. Advanced Intelligence Paradigms*, Vol. 13, Nos. 1/2, pp.155–177, 2019.

[33] Reddy, K.S. and Ramachandram, S, "Cryptographic key management scheme for supporting multi-user SQL queries over encrypted databases," *Int. J. Computer Aided Engineering and Technology*, Vol. 12, No. 4, pp.461–479, 2020.

[34] Srinivasa Reddy, K. and Ramachandram, S, "A new randomized order preserving encryption scheme," *International Journal of Computer Applications*, Vol. 108, No. 12, pp.41–46, 2014.

[35] K.Srinivasa Reddy and Sirandas Ramachandram, "A Novel Dynamic Order-Preserving Encryption Scheme*," IEEE First International Conference on Networks & Soft Computing*, 2014.