Flexible and Publicly Verifiable Aggregation Query for Outsourced Databases in Cloud

Jiawei Yuan
Department of Computer Science
University of Arkansas at Little Rock, USA
Email: jxyuan@ualr.edu

Shucheng Yu
Department of Computer Science
University of Arkansas at Little Rock, USA
Email: sxyu1@ualr.edu

***Abstract*—For securing databases outsourced to the cloud, it is important to allow cloud users to verify that their queries to the cloud-hosted databases are correctly executed by the cloud. Existing solutions on this issue suffer from a high communication cost, a heavy storage overhead or an overwhelming computational cost on clients. Besides, only simple SQL queries (e.g., selection query, projection query, weighted sum query, etc) are supported in existing solutions. For practical considerations, it is desirable to design a client-verifiable (or publicly verifiable) aggregation query scheme that supports more flexible queries with affordable storage overhead, communication and computational cost for users. This paper investigates this challenging problem and proposes an efficient publicly verifiable aggregation query scheme for databases outsourced to the cloud. By designing a renewable polynomial-based authentication tag, our scheme supports a wide range of practical SQL queries including polynomial queries of any degrees, variance query and many other linear queries. Remarkably, our proposed scheme only introduces constant communication and computational cost to cloud users. Our scheme is provably secure under the Static Diffie-Hellman problem, the t-Strong Diffie-Hellman problem and the Computational Diffie-Hellman problem. We show the efficiency and scalability of our scheme through extensive numerical analysis.**

I. Introduction

By outsourcing databases to the cloud, cloud users enjoy data sharing across geographical boundaries in addition to other benefits such as cost saving, on-demand self-service, resource elasticity, etc [15]. Despite appealing advantages, outsourcing databases to cloud also raises security concerns even for non-confidential databases. In particular, clients who query database outsourced to the cloud may wonder whether or not their queries are always correctly executed by cloud servers. This is because large-scale cloud infrastructures may suffer from hardware/software failures, errors caused by human operations, and even malicious attacks [1], [2]. To specify, clients need to verify the *integrity* and *completeness* of their queries over the outsourced databases: for any query request, we need to assure that the query is executed by the cloud on correct data and the returned results have not been modified (integrity); the results shall include the complete data set (completeness), e.g., if the query is on range $[a, b]$, the results shall not be on other ranges $[a', b']$, where $a' \neq a$ and $b' \neq b$.

Related Work. A number of techniques [11], [13], [7], [9], [10], [14], [8], [12], [16] have been proposed, aiming to provide both *completeness* and *correctness* of queries to remote databases. These existing techniques can be mainly divided into two groups: tree-based techniques [11], [7], [8], [12] and signature-based techniques [13], [9], [10], [14]. Among existing tree-based techniques, the best one is proposed by Li et.al. [7], which introduces a novel embedded Merkle B^+-Tree structure and achieve integrity of the query result with simple hash operations. However, the communication complexity of ref. [7] is linear to the number of tuples and attributes associated with the query results, which limits its performance for queries results with large ranges. Compared with the tree-based techniques, signature-based techniques greatly reduce the communication complexity by aggregating signatures. Nevertheless, these techniques require large storage overhead for signatures, because each attribute of every tuple in the database needs a signature for verification purpose. In addition, existing tree-based and signature-based techniques only support simple verifiable SQL queries (i.e., selection, projection, join queries). Trivial extension of these techniques for enabling more aggregation queries inevitably introduces tremendous communication cost or storage overhead for the system. Using homomorphic authenticators, Zheng et.al [16] recently proposed an integrity checking scheme that supports weighted sum query in addition to simple SQL queries. Their signature design allow all attributes in one tuple to share one signature and make the storage overhead independent to the number of attributes. However, in this scheme tags of all tuples needed for calculating the sum shall be transmitted to the client, which represents a non-trivial communication cost. What is more, the client has to perform all integrity verification operations including expensive exponentiation operations, the number of which is linear to the number of tuples associated with the result. For large databases or queries over large ranges, such a computational complexity means a formidable computational burden to the client. Last but not least, ref. [16] does not support advanced aggregation queries such as high-degree polynomial queries. For practical considerations, more advanced aggregation queries shall be supported. Storage overhead, communication cost and computational cost on client side shall be minimal in order to accommodate a wide range client devices such as mobile phones.

In this paper, we design an efficient verifiable aggregation query scheme for outsourced databases. By uniquely incorporating our proposed renewable polynomial-based authentication tags, our scheme can efficiently check the integrity

and completeness of numerous aggregation queries, including polynomial queries of any degree, variance query and many other linear queries. In addition, the communication cost and computational cost for the client during integrity checking is constant in our scheme. We achieve this via our novel design on authentication tag aggregation. Moreover, our proposed scheme is featured by public integrity checking – every client is able to verify the integrity of his query result without any help from the database owner. Thus, the owner can stay off-line after outsourcing the database. Extensive analysis shows that our proposed scheme is efficient and scalable. Our proposed scheme is provably secure under the Computational Diffie-Hellman (CDH) problem, the t-Strong Diffie-Hellman (SDH) assumption and the Static Diffiee-Hellman problem.

We summarize the main contributions of this paper as below.

- An efficient, flexible and publicly verifiable aggregation query scheme for outsourced databases in cloud is proposed. For the first time, we achieve constant communication and computational cost for clients.
- Our proposed scheme allows verification of polynomial queries of any degree, variance query and many other flexible linear queries, in addition to all queries supported by existing schemes.
- Our proposed scheme is provably secure under standard assumptions.
- Our proposed polynomial based authentication tag can be used as an independent solution for other related application, such as database auditing, encrypted key word search, etc.

The rest of this paper is organized as follows: Section II describes the models of our scheme. In Section.III, we present the construction of our scheme. In Section IV, we analyze our proposed scheme in terms of security and performance. We conclude our paper in Section V.

II. Models

A. System Model

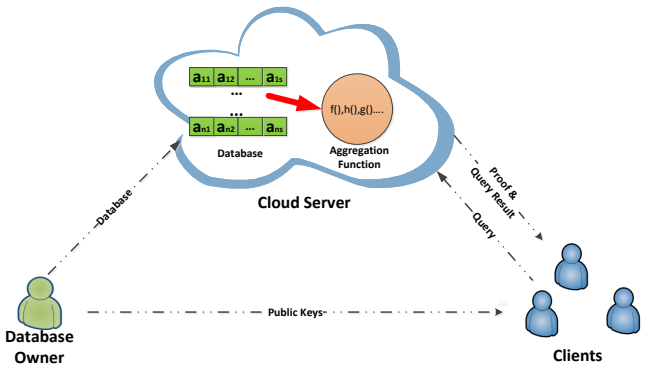

Fig. 1. System Model

We consider a system consisting of three major entities as shown in Fig.1: *Database Owner*, *Cloud Server* and *Client*. The database owner has a relational database with multiple tables, each of which consists of multiple tuples and multiples attributes. The owner outsources his databases to the cloud server together with the corresponding authentication tags. The client who has access to the database performs verifiable aggregation query without help of the owner. To check integrity and completeness of a query result, the client requests the cloud server to generate the proof information for the query result. Based on the proof information and public keys, the client verifies completeness and integrity of the query result.

B. Security Model

In this work, the cloud server is considered as untrusted and potentially malicious, which is consistent with previous schemes [8], [14], [16]. Completeness and integrity of query results from cloud servers are mainly affected by the following factors: 1) attacks corrupting the database; 2) attacks conducting queries on small ranges instead of the actual range; 3) attacks conducting simple aggregation queries instead of complex ones; 4) hardware/software failures, human errors. We assume that transmission channels among entities are well secured using existing techniques.

III. Our Construction

A. Design Details

In this section, we describe the design details of our scheme. We consider a database table TB consisting of n tuples $\{r_1, r_2, \cdots, r_n\}$, each of which has s attributes $\{a_0, a_1, \cdots, a_{s-1}\}$. For an attribute in a tuple, we denote it as $r_i.a_j$. TB is ordered by attribute a_0 (it can also be ordered by any other attributes). Let $e : G \times G \rightarrow G_T$ be a bilinear map and $H(\cdot)$ be the one-way hash function, where G is a multiplicative cyclic group of prime order p and u, g be two random generators of G. We define $f_{\vec{c}}(x)$ as a polynomial with coefficient vector $\vec{c} = (c_0, c_1, \cdots, c_{s-1})$.

KeyGen: A database owner first generates a random signing key-pair $((spk, ssk) \xleftarrow{R} SKG)$ using BLS signature [3]. Then, the owner chooses two random numbers $\alpha, \epsilon \xleftarrow{R} Z_p^*$. The owner also computes $\kappa \leftarrow g^{\epsilon}, \nu \leftarrow g^{\epsilon\alpha}$ and $\{g^{\alpha^j}\}_{j=0}^{s+1}$. The public and secret keys are:

$$PK = \{p, \kappa, \nu, u, spk, \{g^{\alpha^j}\}_{j=0}^{s+1}\}, SK = \{\epsilon, \alpha, ssk\}$$

Setup: The database owner randomly chooses a random table name $TB_{name} \in Z_p^*$. Let τ' be "$TB_{name}||n$"; the file tag τ is τ' together with a signature on τ' under ssk: $\tau \leftarrow \tau'||SKG_{ssk}(\tau')$. Suppose the database will support polynomials query with highest degree q (the value of q can be set according to the requirement of the owner and clients), for each tuple r_i in a table, the owner generates authentication tags as:

$$\sigma_{ix} = (u^{H(TB_{name}^x||i)} \cdot \prod_{j=0}^{s-1} g^{(r_i.a_j)^x \alpha^{j+2}})^{\epsilon}$$
$$= (u^{H(TB_{name}^x||i)} \cdot g^{f_{\vec{B}_{ix}}(\alpha)})^{\epsilon}$$

where $1 \leq x \leq q, \vec{\beta}_{ix} = \{0, 0, \beta_{ix,0}, \beta_{ix,1}, \cdots, \beta_{ix,s-1}\}$ and $\beta_{ix,j} = (r_i.a_j)^x$. The owner outsources the table and its corresponding authentication tags to cloud servers.

Query: To perform an aggregation query, a client first retrieves the table tag τ from the cloud and verifies the signature on it: if the signature is invalid, the client rejects it and halts; otherwise, suppose the client wants to perform a query $Qry =$ "select $SUM(c_i * (r_i.a_t)^{m_i})$ from TB_{name} where $L < a_0 < U$", where $\{c_i\}$ and $\{m_i\}$ are parameters assigned by the client, and L, U are the lower bound and upper bound of the query condition. The client first sends query $Qry_{ra} =$ "select $SUM(r_i.a_t)$ where $L < a_0 < U$" to the cloud server together with a random number ρ.

Completeness-Prove: On receiving the query request, the cloud first executes the query and gets LI and UI as the lower bound index and upper bound index of the query results. The cloud then computes $\sigma_{ra} = \prod \sigma_{i1}^{\rho^i}$ and $y_{ra} = f_{\vec{A}}(\rho)$, where $i \in \{UI+1, UI-1, LI+1, LI-1\}$ and $\vec{A} = \{0, 0, 0, \sum r_i.a_1 * (\rho^i), \cdots, \sum r_i.a_{s-1} * (\rho^i)\}$. As polynomials $f(x) \in Z[x]$ have the algebraic property that $(x-r)$ perfectly divides the polynomial $f(x) - f(r), r \overset{R}{\leftarrow} Z_p^*$. The cloud divides the polynomial $f_{\vec{A}}(x) - y_{ra}$ with $(x - \rho)$ using polynomial long division, and denotes the coefficients vector of the resulting quotient polynomial as $\vec{t} = (t_0, t_1, \cdots, t_{s+1})$, that is, $f_{\vec{t}}(x) \equiv \frac{f_{\vec{A}}(x) - f_{\vec{A}}(r)}{x-r}$. The cloud generates

$$\psi_{ra} = \prod_{j=3}^{s+1} (g^{\alpha^j})^{t_j} = g^{f_{\vec{t}}(\alpha)}$$

Finally, the cloud responds the client with the completeness proof information as $prf_{ra} = \{UI, LI, \psi_{ra}, \sigma_{ra}, y_{ra}, r_{LI-1}.a_0, r_{LI+1}.a_0, r_{UI-1}.a_0, r_{UI+1}.a_0\}$.

Completeness-Verify: Based on the completeness proof information Prf_{ra}, the client computes $\eta_{ra} = (g^{\alpha^2})^{\sum \rho^i * r_i.a_0} \cdot u^{\sum H(TB_{name}||i)\rho^i}$, where $i \in \{UI+1, UI-1, LI+1, LI-1\}$. The client then checks

$$e(\eta_{ra}, \kappa) \cdot e(\psi_{ra}, \nu \cdot \kappa^{-r}) \overset{?}{=} e(\sigma_{ra}, g) \cdot e(\kappa^{-y_{ra}}, g) \quad (1)$$

If Eq.1 holds, the client compares U with $r_{UI-1}.a_0, r_{UI+1}.a_0$ and L with $r_{LI-1}.a_0, r_{LI+1}.a_0$. If $r_{LI-1}.a_0 \geq L$ or $r_{UI-1}.a_0 \geq U$ or $r_{LI+1}.a_0 \leq L$ or $r_{UI+1}.a_0 \leq U$, the client rejects the result and holds; otherwise, the client then sends the query message QM to the cloud server. The message $QM = \{\rho, Qry\}$ contains a random number ρ and an aggregation query $Qry =$ "select $SUM(c_i * (r_i.a_t)^{m_i})$ from TB_{name} where $L < a_0 < U$".

Integrity-Prove: The cloud server first generates query result as

$$QRST = \sum_{i=LI}^{UI} c_i * (r_i.a_t)^{m_i}$$

Then, the cloud generates $y = f_{\vec{B}}(\rho)$, where $\vec{B} = \{0, 0, \sum_{i=LI}^{UI} c_i * (r_i.a_0)^{m_0}, \cdots, \sum_{i=LI}^{UI} c_i * (r_i.a_{t-1})^{m_{t-1}}, \sum_{i=LI}^{UI} c_i * (r_i.a_{t+1})^{m_{t+1}}, \cdots, \sum_{i=LI}^{UI} c_i * (r_i.a_{s-1})^{m_{s-1}}\}$. By dividing $f_{\vec{B}}(x) - y$ with $(x - \rho)$, the cloud gets coefficients vector of the resulting quotient polynomial $\vec{w} = (w_0, w_1, \cdots, w_{t-1}, w_{t+1}, \cdots, w_{s+1})$. Afterward, the cloud computes

$$\psi = \prod_{j=2, j \neq t}^{s+1} (g^{\alpha^{j+2}})^{w_j} = g^{f_{\vec{w}}(\alpha)}$$

The cloud server then aggregates authentication tags for query result as $\sigma = \prod_{i=LI}^{UI} \sigma_{ix}^{c_i}$, where $x = m_i$ for each i. Finally, the cloud computes responds the client with the query result $QRST$ and the corresponding proof information $Prf = \{\psi, \sigma, y\}$.

Integrity-Verify: Based on the proof information Prf, the client computes $\omega = \sum_{i=LI}^{UI} c_i * H(TB_{name}^{m_i}||i)$ and $\eta = u^{\omega}$. Then the client verifies the integrity of the query result as

$$e(\eta \cdot g^{\alpha^{t+2} QRST}, \kappa) \cdot e(\psi, \nu \cdot \kappa^{-\rho}) \overset{?}{=} e(\sigma, g) \cdot e(\kappa^{-y}, g) \quad (2)$$

If Eq.2 holds, the client accepts the query result; otherwise, it denies the result.

Correctness: We analyze the correctness of our construction based on Eq.2:

$$\begin{aligned} & e(\eta \cdot g^{\alpha^{t+2} QRST}, \kappa) \cdot e(\psi, \nu \cdot \kappa^{-\rho}) \quad (3) \\ &= e(u^{\sum_{i=LI}^{UI} c_i * H(TB_{name}^{m_i}||i)}, g^{\epsilon}) \\ &\cdot e(g^{\alpha^{t+2} (\sum_{i=LI}^{UI} c_i * (r_i.a_t)^{m_i})}, g^{\epsilon}) \cdot e(g^{f_{\vec{w}}(\alpha)}, g^{\epsilon(\alpha - r)}) \\ &= e(u^{\sum_{i=LI}^{UI} c_i * H(TB_{name}^{m_i}||i)}, g)^{\epsilon} \cdot e(g, g)^{f_{\vec{B}}(\alpha)\epsilon} \\ &\cdot e(g^{\alpha^{t+2} (\sum_{i=LI}^{UI} c_i * (r_i.a_t)^{m_i})}, g)^{\epsilon} \cdot e(g, g)^{-y\epsilon} \\ &= e(\sigma, g) \cdot e(\kappa^{-y}, g) \end{aligned}$$

The correctness of our scheme is obvious by Eq.3.

B. Support of Other SQL Queries

1) Other Aggregation Queries: Based on our general queries introduced in Section.III-A, we can construct many other verifiable aggregation queries. Here, we give an example of Variance Query, other queries can be flexibly supported similarly.

Considering any k numbers $c_i, 1 \leq i \leq k$, their variance is calculated as $Vari = \frac{\sum_{i=1}^{k} (c_i - c_m)^2}{k-1}$ and c_m is the mean value of the c_i. Suppose a client wants to conduct variance query $Qry =$"select $Vari(r_i.a_t)$ from TB_{name} where $L \leq r_i.a_0 \leq U$" on databases outsourced to the cloud server. The client first performs the completeness verification. Assume there are k tuples satisfying the query condition, the client then performs a query to get the mean value V_{mean} with integrity verification. The verifiable query of the mean value a_m is just a special case of our general query as "select $SUM(r_i.a_t)$ from TB_{name} where $L \leq a_t \leq U$" and divide the result with k, which is known to the client. As $(r_i.a_t - V_{mean})^2 = (r_i.a_t)^2 + V_{mean}^2 - 2V_{mean} * r_i.a_t$, the query of $(k-1) * Vari(r_i.a_t)$ can be decomposed of queries of $SUM((r_i.a_t)^2) - SUM(2V_{mean} * r_i.a_t) + SUM(V_{mean}^2)$ with the same query condition. The client performs verifiable queries for $Qrst' = SUM((r_i.a_t)^2)$ and $Qrst'' = SUM(-2V_{mean} * r_i.a_t)$ separately with the

verified mean value obtained before. Finally, the client can add up $\frac{Qrst'}{k-1}$, $\frac{Qrst''}{k-1}$ and $SUM(\frac{1}{k-1}V^2_{mean})$ to get the verified query result of $Vari(r_i.a_t)$.

2) Simple SQL Queries: Although we focus on aggregation queries in this work, our proposed scheme can also be efficiently applied to simple verifiable SQL queries supported in existing works. Specifically, suppose a client needs to perform a selection query Qry_s = "select * from TB_{name} where $L \leq r_i.a_0 \leq U$" or a projection query Qry_p ="select $a_0, \cdots, a_k$ from TB_{name}", where $1 \leq k \leq s-1$. The client first performs the completeness verification with the cloud using $Completeness - Prove$ and $Completeness - Verify$ algorithms. Then, the client sends the query message Qry_s or Qry_p together with a random number r to the cloud. Based on the query request, the cloud runs the $Integrity - Prove$ algorithm by setting $\{c_i = r^i\}$. In this case, the cloud does not need to remove element from the coefficient like aggregation query does. On receiving proof information from cloud server, the client can finally check the integrity of all query results by running the $Integrity - Verify$ algorithm. Considering join query, the only difference is the query results may come from different tables with different table names TB_{name}. As our designed authentication tags of different tables can also be aggregated, the client can still check integrity of all query results together with our proposed scheme.

IV. Analysis Of Our Proposed Scheme

A. Security Analysis

In this section, we give the security assumptions used in our scheme and then sketch its security proof based on these assumptions. Due to the space limitation, we leave the detailed proof is in the full version of this paper.

Definition IV.1. ***Computational Diffie-Hellman (CDH) Problem*** *[5]*

Let $x, y \xleftarrow{R} Z_p^*$. Given (g, g^x, g^y), it is computationally intractable to compute the value of g^{xy}, where G is a cyclic group of order q and g is a generator of G.

Definition IV.2. ***Static Diffie-Hellman Problem*** *[19]*

Let $a \xleftarrow{R} Z_p^*$. Given input as (g, g^a) and $h \in G$, where g is a generator of a cyclic group G of order q. It is computationally intractable to compute the value h^a.

Definition IV.3. ***t-Strong Diffie-Hellman (t-SDH) Problem*** *[3]*

Let $\alpha \xleftarrow{R} Z_q^*$. Given input as a $(t+1)-tuple$ $(g, g^\alpha, \cdots, g^{\alpha^t}) \in G^{t+1}$, where g is the generator of a cyclic group G of order q. For any probabilistic polynomial time adversary (Adv), the probability $Pr[Adv(g, g^\alpha, \cdots, g^{\alpha^t}) = (c, g^{\frac{1}{\alpha+c}})]$ is negligible for any value of $a \in Z_q^*/-\alpha$.

Theorem IV.4. *If there is a probabilistic polynomial time adversity Adv that can convince the client with an invalid query result in terms of completeness and integrity, by following ref.[18], we can construct a probabilistic polynomial time algorithm B using the Adv to break the CDH problem, the t-SDH problem or the Static Diffie-Hellman problem.*

Proof: Denote the forged completeness proof information $Prf'_{ra} = \{UI', LI', \psi'_{ra}, \sigma'_{ra}, y'_{ra}, r'_{LI-1}.a_0, r'_{LI+1}.a_0, r'_{UI-1}.a_0, r'_{UI+1}.a_0\}$, forged integrity proof information as $Prf' = (\psi', \sigma', y')$ and invalid query result as $QRST'$, where $Prf'_{ra} \neq Prf_{ra}$, $Prf \neq Prf'$ and $QRST' \neq QRST$. In each forged proof information, there is at least one element different from the valid proof information. Therefore, we can conduct case analysis for each element in integrity proof information first. Based on the CDH problem and static Diffie-Hellman problem, $\sigma' = \sigma$ can be proved. Using t-SDH problem, $y' = y$ and $\psi' = \psi$ can be proved. Similar to the unforgeability of integrity proof information, the unforgeability of completeness proof information Prf_{ra} can also be proved. Afterwards, it is easy to prove that $QRST' = QRST$. ■

B. Performance Evaluation

In this section, we numerically evaluate the performance of our proposed scheme in terms of computational complexity, communication complexity and storage overhead. We compare our proposed scheme with ref.[16] and show the results in Table.1. For simplicity, we denote the complexity of one multiplication operation and one exponentiation operation on Group G as MUL and EXP[1] respectively.

1) Computational Cost: In our scheme, $KeyGen$ and $Setup$ are database preparation processes, which can be conducted by the database owner off-line. In the $KeyGen$ algorithm, the data owner performs $(s+3)$ EXP operations to generate keys for the system, where s is the number of attributes in each tuple. To setup the system, the database owner needs $(s+2)nq$ EXP and snq MUL operations for each table, where n is the number of tuples in the table and q is the highest degree required in polynomial queries. Note that, these preparation processes are one-time cost and will not affect the later real-time query performance. In the $Query$ process, as the client only needs to generate a query request and choose the corresponding coefficient generation functions, the computational cost in this process is negligible. To generate the completeness proof information, the cloud server performs $(4+s+1)$ EXP and $(4+s)$ MUL operations. The verification of completeness of the query result costs the client 4 EXP, 4 MUL and 4 Pairing operations. In order to generate integrity proof information, the cloud server conducts $(k+s-1)$ MUL and $(s+k)$ EXP operations, where k is the number of tuples that satisfy the query condition. To finally check integrity of the query result, the client only needs 4 EXP, 4 MUL and 4 Pairing operations. This property is interesting because such a constant verification cost on clients can be affordable to many contemporary client side devices such as mobile phones. Notably, ref.[16] introduces k EXP operations for a client to verify the query result. For large databases or queries over

[1]When the operation is on the elliptic curve, EXP means scalar multiplication operation and MUL means one point addition operation.

	Computational Cost	Communication Cost	Storage Overhead
Our Scheme	8 EXP+8 MUL+8 Pairing	$4\|G\| + 10\|\lambda\|$	qne$\|G\|$
Ref.[16]	k EXP	k$\|G\|$	ne$\|G\|$

Table.1 Complexity Summary: in this table, e is the number of tables in the database, n is number of tuples in a table, q is the highest degree required in polynomial queries, $|G|$ is the size of a group element and $|\lambda|$ the size of security parameter of the system; EXP and MUL are one multiplication operation and one exponentiation operation on Group G respectively, Pairing is a bilinear pairing operation.

large ranges, k can be extremely large and the introduced EXP operations means a formidable burden to clients.

2) Communication Cost: According to Section.III-A, the communication cost in our scheme is mainly attributed to two random numbers and two query message Qry_{ra} and Qry in the challenging message, the completeness proof information $prf_{ra} = \{UI, LI, \psi_{ra}, \sigma_{ra}, y_{ra}, r_{LI-1}.a_0, r_{LI+1}.a_0, r_{UI-1}.a_0, r_{UI+1}.a_0\}$ and the integrity proof information $Prf = \{\sigma, \psi, y\}$. In the query procedure, only two random numbers are needed as additional communication cost. In the proof information, we only introduce 4 group elements, two polynomials, 4 indexes and 4 data attributes. Therefore, the total communication cost introduced by our completeness and integrity checking design is approximately $4|G| + 10|\lambda|$ bits, where $|G|$ is the size of a group element and $|\lambda|$ the size of security parameter of the system. Different from our scheme, the communication cost in ref.[16] is linear to the number of tuples satisfying the query condition, and thus greatly limiting its scalability.

3) Storage Overhead: To enable the integrity checking, our scheme requires the cloud server to store authentication tags for all tuples in the database. Therefore, the storage overhead is $qne|G|$ bits and e is the number of table in the database. Compared with ref. [16], our scheme achieves the same storage overhead to support same functionality ($q = 1$). Assuming the size of an authentication tag is 1024 bits in consistence with previous work [17], [18], a database with 8,000,000 tuples will introduce about 1GB storage overhead using our scheme. In real life, such storage overhead only costs \$0.095/month on Amazon S3 as of when this paper was written.

V. Conclusion

In this work, we present an efficient and publicly verifiable aggregation query scheme in the setting of outsourced databases in cloud. Our proposed scheme not only allows the cloud server to perform most computational tasks for the query verification, but also aggregates the proof information to achieve constant communication cost. Compared with existing solutions, our proposed scheme significantly reduces the expensive computational operations (e.g., exponentiation operation) on the client side from linear level to constant level. Besides the simple SQL query types supported in existing schemes, our proposed scheme also allows many powerful aggregation queries such as polynomial queries of any degrees, variance query and many other linear queries. Moreover, our proposed polynomial based authentication tag can be used as an independent technique for other related applications, such as database auditing, encrypted key word search, etc.

References

[1] "Amazon web service. summary of the amazon ec2 and amazon rds service disruption in the us east region," http://aws.amazon.com/message/65648/.

[2] "Dropbox. dropbox forums on data loss topic," http://forums.dropbox.com/tags.php?tag=data-loss.

[3] D. Boneh and X. Boyen. Short signatures without random oracles. In *Proceedings of the 23nd international conference on Theory and applications of cryptographic techniques*, EUROCRYPT'04 pages 56–73, 2004.

[4] 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*, EUROCRYPT'03, pages 416–432, Berlin, 2003.

[5] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theor.*, 22(6):644–654, Sept. 1976.

[6] D. Boneh and M. K. Franklin, "Identity-based encryption from the weil pairing," in *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '01. London, UK, UK: Springer-Verlag, 2001, pp. 213–229.

[7] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin. Dynamic authenticated index structures for outsourced databases. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, SIGMOD '06, pages 121–132, New York, NY, USA, 2006. ACM.

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

[9] E. Mykletun, M. Narasimha, and G. Tsudik. Authentication and integrity in outsourced databases. *Trans. Storage*, 2(2):107–138, May 2006.

[10] 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*, DASFAA'06, pages 420–436, Berlin, Heidelberg, 2006. Springer-Verlag.

[11] 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*, DBSec'05, pages 84–98, Berlin, Heidelberg, 2005. Springer-Verlag.

[12] B. Palazzi, M. Pizzonia, and S. Pucacco. Query racing: Fast completeness certification of query results. In S. Foresti and S. Jajodia, editors, *Data and Applications Security and Privacy XXIV*, volume 6166 of *Lecture Notes in Computer Science*, pages 177–192, 2010.

[13] 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*, SIGMOD '05, pages 407–418, New York, NY, USA, 2005. ACM.

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

[15] G. Timothy and M. M. Peter. The nist definition of cloud computing. NIST SP - 800-145, September 2011.

[16] Q. Zheng, S. Xu, and G. Ateniese. Efficient query integrity for outsourced dynamic databases. In *Proceedings of the 2012 ACM Workshop on Cloud computing security workshop*, CCSW '12, pages 71–82, New York, NY, USA, 2012. ACM.

[17] J. Yuan and S. Yu, "Proofs of retrievability with public verifiability and constant communication cost in cloud," in *Proceedings of the 2013 international workshop on Security in cloud computing*, ser. Cloud Computing '13. Hangzhou, China: ACM, 2013, pp. 19–26.

[18] J. Yuan and S. Y, "Secure and constant cost public cloud storage auditing with deduplication," in *IEEE Conference on Communications and Network Security 2013 (IEEE CNS 2013)*, Washington, USA, Oct. 2013.

[19] D. R. L. Brown and R. P. Gallant, "The static diffie-hellman problem," Cryptology ePrint Archive, Report 2004/306, 2004.