Efficient Query Integrity for Outsourced Dynamic Databases

Qingji Zheng
Department of Computer Science
University of Texas at San Antonio
qzheng@cs.utsa.edu

Shouhuai Xu
Department of Computer Science
University of Texas at San Antonio
shxu@cs.utsa.edu

Giuseppe Ateniese
Department of Computer Science
Sapienza-University of Rome and
Johns Hopkins University
ateniese@di.uniroma1.it

ABSTRACT

As databases are increasingly outsourced to the cloud, data owners require various security assurances. This paper investigates one particular assurance, *query integrity*, by which a database querier (either the data owner or a third party) can verify that its queries were faithfully executed by the cloud server with respect to the outsourced database. Query integrity is investigated in the setting of dynamic databases, where the outsourced databases can be updated by the data owners as needed. We present a formal security definition of query integrity and a provably-secure efficient construction. Our solution improves upon the state-of-the-art solutions by additionally allowing aggregate queries and more flexible join queries. In addition, we provide better performance by eliminating a linear factor in the extra storage complexity for security purpose. Our solution also achieves a trade-off between computational and communication complexities.

Categories and Subject Descriptors

C.2.4 [**Communication Networks**]: Distributed Systems; H.2 [**DATABASE MANAGEMENT**]:

General Terms

Security

Keywords

Dynamic outsourced database, query integrity, authenticated data structure.

1. INTRODUCTION

When databases are outsourced to the cloud, security issues arise. The concern that outsourced data may be modified or (partially) deleted has led to novel solutions to assuring the *storage integrity* of outsourced data [2, 12, 3, 26, 8]. However, *query integrity*, verifying whether or not queries against outsourced data are faithfully executed, has not been adequately addressed. Intuitively, query integrity aims to assure the queriers, which can be the data owners and third parties (e.g., the data owners' business partners), that their queries are executed against the outsourced data (i.e., neither a portion of it nor a modified version of it). Despite some previous studies [11, 13, 17, 18, 16, 19, 23], the problem of query integrity largely remains open.

1.1 Our Contributions

We present a formal security definition and an efficient construction for query integrity in the setting of outsourced dynamic database. Our solution can be characterized from three perspectives: (i) functionality, (ii) security, and (iii) efficiency. From the perspective of (i) functionality, our solution supports four kinds of queries — selection, projection, join, and aggregate. Whereas, the state-of-the-art solutions [13, 23] only support selection, projection and join queries, but do not support aggregate queries (see Section 5.4 for details). Moreover, our solution supports strictly more *flexible* join queries, namely that the queries do not have to be defined with respect to pre-defined keyword attributes. In contrast, the state-of-the-art solutions [13, 23] only support join queries with respect to pre-defined keyword attributes.

From the perspective of (ii) security, our solution is provably secure as long as the two underlying building-blocks are provably secure. The first building-block is called *Authenticated Outsourced Ordered Data Set*, and the second building-block is called *Homomorphic Linear Tag*. Although our concrete solution is based on our specific constructions of these building-block, its security analysis can be directly applied to solutions that use other (perhaps more efficient) building-blocks as long as the building-blocks satisfy their respective security definitions. This is due to our modular construction and “compiler"-like security analysis.

From the perspective of (iii) efficiency, our solution is characterized as follows. Let m be the number of attributes and n be the number of tuples.

- Our solution incurs an $O(n)$ storage complexity at the cloud side for security purpose, in contrast to the $O(mn)$ of [13, 23].
- For selection query, our solution incurs $O(n)$ exponentiations at the querier side, which is not as efficient as the $O(n)$ hash operations of [13] but more efficient than the $O(n)$ exponentiation operations on bilinear map of [23].

  Our solution incurs communication of $O(n)$ tags, which is less efficient than the $O(\log n)$ hash values of [13] but comparable to the $O(n)$ of [23].
- For projection query, our solution incurs $O(n)$ modular exponentiations at the querier side. This is not as efficient as the $O(n)$ hash operations of [13], but much more efficient than the $O(nk)$ exponentiation operations on bilinear map

CCSW'12, October 19, 2012, Raleigh, North Carolina, USA.
Copyright 2012 ACM 978-1-4503-1665-1/12/10 ...$15.00.

of [23], where $k \leq m$ is the number of attributes involved in the projection operation.

Our solution incurs an $O(n+m)$ communication complexity, which is the same as in [23] but much more efficient than the $O((m-k)n)$ of [13], where $k \leq m$ is the number of attributes involved in the projection operation.

- For join queries with respect to two tables of n tuples and m attributes, our solution incurs $O(n)$ modular exponentiations at the querier side, which is not as efficient as the $O(n \log n)$ hash operations of [13], but more efficient than $O(n)$ exponentiation operations on bilinear map of [23].

  Our solution incurs the communication complexity of $O(n+m)$ tags, which is more efficient than the $O(n(\log n))$ hash values of [13] and comparable to the $O(n)$ of [23].

The efficiency of our solution mainly comes from the second building-block mentioned above, which is weaker than the Homomorphic Linear Authenticator introduced in [2] and may be of independent value.

1.2 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 in order 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 so as to facilitate range queries, and publishes a certified bitmap corresponding to every update so as to facilitate dynamic updates. These cause a large storage and communication overhead while including many exponentiations and pairing operations.

There are studies that are somewhat related to the theme of the present paper as well. 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].

Paper outline.

The rest of the paper is organized as follows. Section 2 presents the functional and security definitions of outsourced dynamic database with the requirement of query integrity. Section 3 describes the first building-block, and Section 4 describes the second building-block. Section 5 presents the main construction of authenticated outsourced dynamic database and analyzes its security and efficiency. Section 6 presents an extension of the construction to accommodate storage integrity of outsourced dynamic database. Section 7 concludes the paper with future research directions.

2. QUERY INTEGRITY FOR OUTSOURCED DYNAMIC DATABASES: DEFINITIONS

In the context of the present paper, a relational database consists of multiple tables, and each table has multiple tuples and multiple attributes. As shown in Figure 1, an outsourced database system has three participants: data owner (who outsources its database to the cloud), database server (i.e., the cloud), and database queriers (e.g., business partners of the data owner). The data owner uses a *management interface* to outsource its database to the cloud, including dynamic updates of the database. There is also a *query interface*, which can be used by any third party, including the data owner itself if desired.

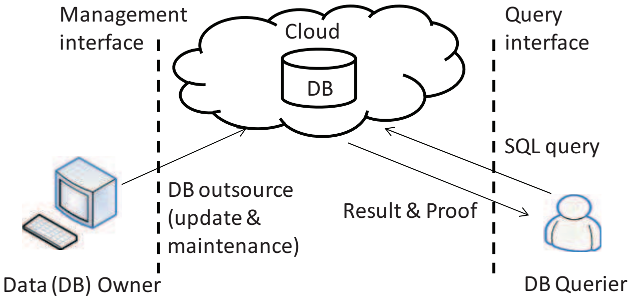


Figure 1: Outsourced dynamic database system model.

Intuitively, query integrity means that any query qry is faithfully executed with respect to the database D. If we treat a query qry as a function, the querier should be able to verify that the answer to its query is indeed $\mathsf{qry}(D)$. The concern is legitimate because the cloud may execute the query qry with respect to D', where $D' \neq D$ because (for example) the cloud vendor may use an outdated version of D rather than the up-to-date one, or $D' \subset D$ because the cloud vendor wants to spend less resources on searching the entire D. Moreover, the cloud may return the answer to a modified query qry' on database D or even some $D' \neq D$. As a concrete example, a query qry asks for the tuples with some attribute values that belong to the interval $[10, 100]$, but the cloud actually returns the tuples whose attribute values belong to the smaller interval $[10, 20]$. Without assuring query integrity, the querier cannot tell whether the returned answer is indeed $\mathsf{qry}(D)$ or some $\mathsf{qry}'(D')$.

In what follows, we present the functional and security definitions of Authenticated Outsourced Dynamic Database (AuthDDB), which was somewhat inspired by the definitions of Authenticated

Data Structures that allow verifiable queries over dynamic sets [24, 25].

Definition 1. (AuthDDB) Let D be a database outsourced to the server. An AuthDDB scheme consists of the following algorithms:

- KeyGen: This algorithm takes as input the primary security parameter ℓ, and outputs a pair of private and public keys (sk, pk). We denote this by
$$(sk, pk) \leftarrow \mathsf{KeyGen}(1^{\ell}).$$
- SetUp: This algorithm is executed by a data owner O before outsourcing its database D to the server. By taking as input the private key sk and the database D, this algorithm outputs some cryptographic auxiliary information Au and state information State. Both D and Au will be outsourced to the server and State will be made public (so as to allow third parties to verify the query answers). We denote this by
$$(\mathsf{State}, \mathsf{Au}, D) \leftarrow \mathsf{SetUp}(sk, D)$$
- Update: This protocol is executed between a data owner O and the server S to perform update operations, the detail of which is described by Upd. By taking as input the private key sk and the current state information State, the data owner interacts with the server, which takes as input the stored data D and the cryptographic auxiliary information Au. The data owner O updates its state information to State$'$ from the update information Upd, and the server obtains Au$'$ and D' by updating the stored database accordingly. We denote the protocol by
$$(\mathsf{Au}', \mathsf{State}', D') \leftarrow (O(sk, \mathsf{State}, \mathsf{Upd}) \leftrightarrow S(\mathsf{Au}, D))$$
- QueryVrfy: This is a protocol between a querier Q, which issues a SQL query qry, and the server S, which answers the query with the result Rst and a proof Prf. The querier verifies the result Rst with Prf, and outputs reject if Rst is not valid with respect to the query qry and the state State; otherwise, the querier accepts Rst and Prf. We denote the protocol by
$$\{(\mathsf{reject}), (\mathsf{accept}, \mathsf{Rst}, \mathsf{Prf})\} \leftarrow (Q(pk, \mathsf{qry}, \mathsf{State}) \leftrightarrow S(\mathsf{Au}, D))$$

We require an AuthDDB scheme to be correct, meaning that for any honest server, $(sk, pk) \leftarrow \mathsf{KeyGen}(1^{\ell})$, $(\mathsf{State}, \mathsf{Au}, D) \leftarrow \mathsf{SetUp}(sk, D)$, polynomial-many executions of the Update protocol, and a query qry, it holds that

$$(\mathsf{accept}, \mathsf{Rst}, \mathsf{Prf}) \leftarrow (Q(pk, \mathsf{qry}, \mathsf{State}) \leftrightarrow S(\mathsf{Au}, D))$$

We require an AuthDDB scheme to be sound, meaning that no malicious server can return incorrect query answers without being detected by the querier. Specifically, we say an AuthDDB scheme is sound if for any query qry on database D, the server can not return an incorrect Rst such that

$$(\mathsf{accept}, \mathsf{Rst}, \mathsf{Prf})\} \leftarrow (Q(pk, \mathsf{qry}, \mathsf{State}) \leftrightarrow S(\mathsf{Au}, D)).$$

Formally,

Definition 2. (soundness of AuthDDB) Let $\Lambda = (\mathsf{KeyGen}, \mathsf{SetUp}, \mathsf{Update}, \mathsf{QueryVrfy})$ be an AuthDDB scheme and $\mathcal{A}$ be a probabilistic polynomial-time adversary. Consider the following security game between a challenger and $\mathcal{A}$.

- The challenger runs $(sk, pk) \leftarrow \mathsf{KeyGen}(1^{\ell})$ and give pk to the adversary $\mathcal{A}$.
- $\mathcal{A}$ makes oracle access to SetUp, by presenting a database D_0. The challenger computes
$$(\mathsf{State}_0, \mathsf{Au}_0, D_0) \leftarrow \mathsf{SetUp}(sk, D_0),$$
and gives $\mathsf{State}_0, \mathsf{Au}_0$ to $\mathcal{A}$. The challenger makes State_0 public.
- $\mathcal{A}$ asks for updating D_0 adaptively with Upd_i, $i \geq 0$. The challenger computes
$$(\mathsf{Au}_{i+1}, \mathsf{State}_{i+1}, D_{i+1}) \leftarrow (O(sk, \mathsf{State}_i, \mathsf{Upd}_i, \mathsf{Au}_i, D_i) \leftrightarrow S(\mathsf{Au}_i, D_i)).$$
- $\mathcal{A}$ may execute QueryVrfy polynomial-many times. Eventually, $\mathcal{A}$ outputs a query qry and a query result Rst with proof Prf.
- $\mathcal{A}$ wins the game if
$$(\mathsf{accept}, \mathsf{Rst}, \mathsf{Prf})\} \leftarrow (Q(pk, \mathsf{qry}, \mathsf{State}_k) \leftrightarrow S(\mathsf{Au}_k, D_k))$$
for some $k \geq 0$ and $\mathsf{Rst} \neq \mathsf{localRst}$, where $\mathsf{localRst} \leftarrow \mathsf{LocalQuery}(\mathsf{qry}, D_k)$ is produced by the challenger that faithfully executes query qry on database D_k.

We say that Λ is sound if any polynomial-time algorithm $\mathcal{A}$ can win the game with at most a negligible probability.

3. BUILDING-BLOCK I: AUTHENTICATED OUTSOURCED ORDERED DATA SET (AUTHODS)

In this section, we introduce a building block for assuring range query integrity on ordered data set that is outsourced to the server. This building-block is called Authenticated Outsourced Ordered Data Set (AuthODS), which is similar to AuthDDB.

3.1 Definition of AuthODS

Definition 3. (AuthODS) Let E be an ordered data set. An AuthODS scheme consists of the following algorithms, which are similar to those in Definition 1:

- KeyGen: This key generation algorithm generates the public/private key as KeyGen in Definition 1.
- SetUp: This setup algorithm is the same as SetUp in Definition 1, except that the database is replaced with an ordered set E.
- Update: This update protocol proceeds is the same as Update in Definition 1, except that the update operations are element insertion/deletion/update on the ordered data set E.
- QueryVrfy: This query protocol is the same as QueryVrfy in Definition 1, except that it only supports range query $\mathsf{qry}(a, b)$ that asks for all elements in the interval $[a, b]$.

The correctness of AuthODS can be defined similar to that of AuthDDB scheme.

Definition 4. (soundness of $\mathsf{AuthODS}$) For an $\mathsf{AuthODS}$ scheme, $\Lambda = (\mathsf{KeyGen}, \mathsf{SetUp}, \mathsf{Update}, \mathsf{QueryVrfy})$, we consider the security game as in Definition 2, except that (i) the initial database is replaced with an ordered set E, (ii) the update operation is element insertion, deletion or update on the ordered data set, and (iii) the queries are only range queries $\mathsf{qry}(a, b)$ that ask for elements in the interval $[a, b]$. We say that Λ is sound if any polynomial-time algorithm $\mathcal{A}$ can win the game with at most a negligible probability.

3.2 Construction and Analysis of AuthODS: Merkle B-Tree

Now we describe an $\mathsf{AuthODS}$ scheme, which is a Merkle B-tree (MB-tree) and has been extensively studied in [13, 17]. Merkle B-tree applies the basic idea of Merkle tree on a B^+ tree structure, where the operations on Merkle B-tree (e.g., insertion and deletion) are similar to those on B^+ tree. The primary advantage of B^+ tree is that it has a large fan-out, which can reduce the number of I/O operations when searching for an element [13]. Let $\mathsf{Sig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ be a secure signature scheme. Let E be an ordered set. The Merkle B-tree scheme consists of algorithms as follows:

- $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\ell)$: This algorithm runs $\mathsf{Sig.KeyGen}$ (1^ℓ) to obtain a pair of private and public keys (sk, pk).

- $(\mathsf{State}, \mathsf{Au}) \leftarrow \mathsf{SetUp}(sk, E)$: This algorithm outputs a succinct signature which can be used for verification. The structure of Merkle B-tree $\mathcal{T}$ is similar to B^+ tree, where the leaves store elements in the ordered set E, and the values of internal nodes are computed from the concatenation of the values of their children via an appropriate hash function. The root of the tree will be signed to produce the state information, denoted by $\mathsf{State} = \mathsf{Sig.Sign}(\mathcal{T})$ and $\mathsf{Au} = \mathcal{T}$.

- Update: The update protocol fulfills update operations. For simplicity, we consider the example of the replacement operation while assuming that the replacement preserves the order of the elements. We refer to [13] for details about the insertion and deletion operations. Suppose Upd = "update the element E_i to E_i'". Upon receiving Upd from the data owner, the server updates E to E' by replacing E_i with E_i', and updates $\mathcal{T}$ to $\mathcal{T}'$. The server provides a proof, a path of E_i in $\mathcal{T}$, namely a sequence including values of the nodes from E_i to the root of MB-tree as well as the values of these nodes' siblings. The data owner can hash the path of E_i from the bottom to the top and verify whether the root is valid with respect to state State or not. If so, the data owner updates the path from the bottom to the top by replacing E_i with E_i', which will result in a new root, signs the new root, and sets $\mathsf{State}' = \mathsf{Sig.Sign}(\mathcal{T}')$; otherwise, the data owner aborts.

- $\mathsf{QueryVrfy}$: Given a range query $\mathsf{qry}(a, b)$, the server outputs a proof Prf showing that Rst contains all elements in $[a, b]$.
  - If Rst is empty, which means there exists some s, such that $E_s < a, b < E_{s+1}$. The server returns the proof Prf including two paths: a path of E_s and a path of E_{s+1}. The querier hashes each path from bottom to the top, and verify whether the roots match the state State, and E_s is neighbor to E_{s+1}. If so, the querier returns the null set Rst, Prf, and accept. Otherwise, abort.
  - If Rst is not null, suppose the query result is $(E_s, \ldots, E_t), s \leq t$. The server returns the proof Prf including two paths: one path of the left-most neighbor leaf of E_s, and the other path of the right-most leaf of E_t. Then the querier uses Prf and the result Rst to construct a B^+ tree, and verifies whether the root of the this B^+ tree is valid for $\mathsf{State} = \mathsf{Sig.Sign}(\mathcal{T})$. If so, the querier returns $(\mathsf{Rst}, \mathsf{Prf}, \mathsf{accept})$; otherwise, the querier aborts.

THEOREM 1. *Assuming that* Sig *is a secure signature scheme and the hash function is collision resistant, the Merkle B-tree scheme is sound with respect to Definition 4.*

4. BUILDING-BLOCK II: HOMOMORPHIC LINEAR TAG (HLT)

Now we present the second building block, HLT. Intuitively, HLT offers the following property: If messages $M_1, \ldots, M_n$ are respectively tagged with $\sigma_1, \ldots, \sigma_n$ using some cryptographic function, then for coefficients $c_1, \ldots, c_n$ in a pre-defined coefficient space, the aggregate message $M = \sum_{i=1}^{n} c_i M_i$ can be verified via the aggregate tag σ of $\sigma_1, \ldots, \sigma_n$ and the coefficients $c_1, \ldots, c_n$. HLT can be divided into two types:

- Publicly verifiable HLT: It allows anyone (without knowing any secret) to verify the validity of tags. In order to allow any third party to verify query integrity, this type of HLT is needed for the purpose of the present paper.

- Privately verifiable HLT: It allows someone who knows the relevant secret to verify the validity of tags. Putting this into the context of the present paper, this type of HLT can be used to allow the data owner (but not third parties) to verify query integrity. Therefore, this type of HLT will not be discussed further in the paper.

The concept of HLT was inspired by the notion of Homomorphic Linear Authenticator (HLA), which was formally introduced in [3]. The difference between them is that HLT is weaker than HLA because HLT only considers attacks that do not attempt to tamper the individual tags (which is dealt with by another layer of protection for free, namely by the first building-block); whereas, HLA explicitly accommodates attacks that aim to tamper the individual tags. This makes it possible to construct HLT schemes that are more efficient that their HLA counterparts. It is worthwhile to point out the following feature of HLT and HLA: the aggregated message M and the aggregated tag σ are sufficient to allow the verifier to test their validity *without* knowing the individual messages $M_1, \ldots, M_n$. This is not the case for aggregate signatures [5], batch RSA [9], and condensed RSA [18], which are not sufficient for the purpose of HLT or HLA.

4.1 Definitions of HLT

Definition 5. (publicly verifiable HLT) A publicly verifiable HLT scheme consists of the following algorithms:

- $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\ell)$: This algorithm takes as input a security parameter ℓ, and outputs a pair of public and private keys (pk, sk). It may optionally specify a coefficient domain $\mathcal{C}$ and a message space $\mathcal{M}$.

- $\sigma_i \leftarrow \mathsf{TagGen}(sk, M_i)$: This algorithm takes as input the private key sk and a message $M_i \in \mathcal{M}$, and outputs a tag σ_i for M_i.

- $\sigma \leftarrow \mathsf{HLTAgg}(\vec{c}, \vec{\mathsf{Tag}})$: This linear aggregation algorithm takes as input a vector of tags $\vec{\mathsf{Tag}} = (\sigma_1, \ldots, \sigma_n)$ with respect to a vector of messages $\vec{M} = (M_1, \ldots, M_n)$ and a vector of coefficients $\vec{c} = (c_1, \ldots, c_n)$. It outputs an aggregate tag σ with respect to the aggregated message $M = \sum_{i=1}^{n} c_i M_i$.

- $\{0, 1\} \leftarrow \mathsf{Vrfy}(pk, M', \sigma')$: This deterministic algorithm takes as input the public key pk, a candidate message M', and a tag σ'. It outputs 1 if σ' is valid with respect to M', and outputs 0 otherwise.

We require a HLT scheme to be correct, meaning that any faithfully aggregated message M and tag σ are always accepted as valid. Formally, this means that for $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\ell)$, $\vec{M} = (M_1, \ldots, M_n) \in \mathcal{M}^n$, $\vec{\mathsf{Tag}} = (\sigma_1, \ldots, \sigma_n)$ where $\sigma_i \leftarrow \mathsf{TagGen}(sk, M_i)$ for $1 \le i \le n$, and $\vec{c} = (c_1, \ldots, c_n) \in \mathcal{C}^n$, then $\sigma \leftarrow \mathsf{HLTAgg}(\vec{c}, \vec{\mathsf{Tag}})$ implies $1 \leftarrow \mathsf{Vrfy}(pk, \sum_{i=1}^{n} c_i M_i, \sigma)$.

The intuition behind the following security definition of HLT is: for any tag σ generated for message M, there is no probabilistic polynomial time adversary that can present $M' \neq M$ such that $1 \leftarrow \mathsf{Vrfy}(pk, M', \sigma)$. Formally, we have:

Definition 6. (security of HLT) Let $\Lambda = (\mathsf{KeyGen}, \mathsf{TagGen}, \mathsf{HLTAgg}, \mathsf{Vrfy})$ be a HLT and $\mathcal{A}$ be a probabilistic polynomial-time adversary. Consider the following security game between a challenger and $\mathcal{A}$:

1. The challenger runs $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\ell)$ and gives pk to $\mathcal{A}$. The optional coefficient domain $\mathcal{C}$ and the message space $\mathcal{M}$ are specified by KeyGen.

2. $\mathcal{A}$ may make oracle queries to TagGen by adaptively selecting $M_1, \ldots, M_n$ from $\mathcal{M}$. The challenger computes $\sigma_i \leftarrow \mathsf{TagGen}(sk, M_i)$ for $1 \le i \le n$ and returns tags $(\sigma_1, \ldots, \sigma_n)$ to $\mathcal{A}$. The challenger keeps the lists of messages and tags: $(M_1, \ldots, M_n)$ and $(\sigma_1, \ldots, \sigma_n)$.

3. $\mathcal{A}$ may make oracle queries to HLTAgg by selecting a vector of coefficients $\vec{c} = (c_1, \ldots, c_n)$, obtain the aggregate tag σ, and run Vrfy with the aggregate tag σ and the aggregated message $\sum_{i=1}^{n} c_i M_i$. This can be performed polynomially many times.

4. Eventually, $\mathcal{A}$ selects a vector of coefficients $\vec{c} = (c_1, \ldots, c_n)$, where $c_i \in \mathcal{C}$, and some $M' \in \mathcal{M}$.

5. The adversary $\mathcal{A}$ wins the game if $1 \leftarrow \mathsf{Vrfy}(pk, M', \sigma)$ and $M' \neq \sum_{i=1}^{n} c_i M_i$, where $\sigma \leftarrow \mathsf{HLTAgg}(\vec{c}, \vec{\mathsf{Tag}})$ was computed by the challenger, where $\vec{\mathsf{Tag}} = (\sigma_1, \ldots, \sigma_n)$ corresponds to the message vector $(M_1, \ldots, M_n)$ that can be identified by the coefficient vector $\vec{c} = (c_1, \ldots, c_n)$ provided by the adversary $\mathcal{A}$.

We say Λ is secure if no probabilistic polynomial-time algorithm $\mathcal{A}$ can win the game with a non-negligible probability in the security parameter ℓ.

From the security game, we observe that the adversary $\mathcal{A}$ is only allowed to manipulate the messages $M_1, \ldots, M_n$ but not the tags. This further explains why HLT is weaker than the aforementioned HLA (Homomorphic Linear Authenticator) [2, 3, 26], where the adversary can manipulate *both* messages and tags. This can be stated as:

LEMMA 1. *Any secure HLA scheme as defined in [3] is also a secure* HLT *scheme as defined above.*

4.2 Construction and Analysis of HLT

We present a HLT scheme whose security is based on the Discrete Logarithm (DLOG) problem. The scheme consists of the following algorithms.

- $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\ell)$:
  1. Let q be a $\ell-$bit prime and p be another large prime such that $q|(p-1)$.
  2. Select v_1 and v_2 uniformly at random from Z_p^* such that the order of v_1 and v_2 is q
  3. Select s_{j1}, s_{j2} uniformly at random from Z_q^* and set $z_j = v_1^{-s_{j1}} v_2^{-s_{j2}} \mod p$, for $1 \le j \le m$.
  4. Let $sk = \{(s_{11}, s_{12}), \ldots, (s_{m1}, s_{m2})\}$ and $pk = \{v_1, v_2, z_1, \ldots, z_m\}$.
  5. The coefficient domain $\mathcal{C}$ is $[0, q)$ and the message space is $\mathcal{M} = [0, q)^m$.

- $\sigma_i \leftarrow \mathsf{TagGen}(sk, M_i)$: For $M_i \in \mathcal{M}$, the tag σ_i is computed by selecting r_1, r_2 uniformly at random from Z_q^* and:

$$\begin{aligned} x &= v_1^{r_1} v_2^{r_2} \mod p, \\ y_1 &= r_1 + \sum_{j=1}^{m} M_i[j] s_{j1} \mod q, \\ y_2 &= r_2 + \sum_{j=1}^{m} M_i[j] s_{j2} \mod q. \end{aligned}$$

  Let $\sigma_i = (x, y_1, y_2)$.

- $\sigma \leftarrow \mathsf{HLTAgg}(\vec{c}, \vec{\mathsf{Tag}})$: Given tags $\vec{\mathsf{Tag}} = (\sigma_1, \ldots, \sigma_n)$ with $\sigma_i = (x_i, y_{i1}, y_{i2})$, and $\vec{c} = (c_1, \ldots, c_n)$, the aggregate tag $\sigma = (x, y_1, y_2)$ is computed as:

$$\begin{aligned} x &= \prod_{i=1}^{n} x_i^{c_i} \mod p, \\ y_1 &= \sum_{i=1}^{n} c_i y_{i1} \mod q, \\ y_2 &= \sum_{i=1}^{n} c_i y_{i2} \mod q. \end{aligned}$$

- $\{0, 1\} \leftarrow \mathsf{Vrfy}(pk, M, \sigma)$: To verify that M is valid with respect to tag σ, check whether:

$$x \overset{?}{=} v_1^{y_1} v_2^{y_2} \prod_{j=1}^{m} z_j^{M[j]} \mod p.$$

  If it holds, return 1; otherwise, return 0.

It can be verified that $M = \sum_{i=1}^{n} c_i M_i$ matches the aggregated tag σ because

$$\begin{aligned} v_1^{y_1} v_2^{y_2} \prod_{j=1}^{m} z_j^{M[j]} &= v_1^{\sum_{i=1}^{n} c_i y_{i1}} v_2^{\sum_{i=1}^{n} c_i y_{i1}} \prod_{j=1}^{m} z_j^{\sum_{i=1}^{n} c_i M_i[j]} \\ &= \prod_{i=i}^{n} v_1^{c_i y_{i1}} \prod_{i=i}^{n} v_2^{c_i y_{i2}} \prod_{j=1}^{m} z_j^{\sum_{i=1}^{n} c_i M_i[j]} \\ &= \prod_{i=i}^{n} (v_1^{c_i y_{i1}} v_2^{c_i y_{i2}} \prod_{j=1}^{m} z_j^{c_i M_i[j]}) \\ &= \prod_{i=i}^{n} x_i^{c_i} = x \end{aligned}$$

THEOREM 2. *Assuming DLOG problem is hard, the* HLT *scheme is secure according to Definition 6.*

PROOF. Let $M_1, \ldots, M_n$ be the messages adaptively selected by $\mathcal{A}$ and $\sigma_1 = (x_1, y_{11}, y_{12}), \ldots, \sigma_n = (x_n, y_{n1}, y_{n2})$ be the corresponding tags generated by the challenger. Assume the adversary wins the security game with a non-negligible probability. That is, it outputs a vector of coefficients $\vec{c} = \{c_1, \ldots, c_n\}$ and a message $M' \in \mathcal{M}$, such that $M' \neq M = \sum_{i=1}^{n} c_i M_i$ but $1 \leftarrow \mathsf{Vrfy}(pk, M', \sigma)$, where $\sigma \leftarrow \mathsf{HLTAgg}(\vec{c}, \vec{\mathsf{Tag}})$, and $\vec{\mathsf{Tag}} = (\sigma_1, \ldots, \sigma_n)$. We show that if $\mathcal{A}$ wins the security game with a non-negligible probability, then we can solve the DLOG problem: given v_1, v_2 randomly selected from Z_p^*, find $\log_{v_2}(v_1)$.

Suppose $\sigma = (x, y_1, y_2)$. Since $1 \leftarrow \mathsf{Vrfy}(pk, M', \sigma)$, we have

$$x = v_1^{y_1} v_2^{y_2} \prod_{j=1}^{m} z_j^{M'[j]}.$$

On the other hand, as $\sigma \leftarrow \mathsf{HLTAgg}(\vec{c}, \vec{\mathsf{Tag}})$, we have

$$x = v_1^{y_1} v_2^{y_2} \prod_{j=1}^{m} z_j^{M[j]},$$

where $M = \sum_{i=1}^{n} c_i M_i$. Therefore, we have

$$\prod_{j=1}^{m} z_j^{M'[j]} = \prod_{j=1}^{m} z_j^{M[j]},$$

namely

$$\prod_{j=1}^{m} z_j^{M'[j]-M[j]} = 1.$$

As $M' \neq M$, let $\Delta M[j] = M'[j] - M[j]$ for $1 \leq j \leq m$. Since $z_j = v_1^{-s_{j1}} v_2^{-s_{j2}}$, we have

$$v_1^{\sum_{j=1}^{m} -s_{j1} \Delta M[j]} v_2^{\sum_{j=1}^{m} -s_{j2} \Delta M[j]} = 1.$$

We claim that $\sum_{j=1}^{m} -s_{j1} \Delta M[j] \mod q = 0$ with negligible probability because s_{j1} for $1 \leq j \leq n$ are kept secret. Then we have

$$v_1 = v_2^{\frac{\sum_{j=1}^{m} s_{j2} \Delta M[j]}{\sum_{j=1}^{m} -s_{j1} \Delta M[j]}}.$$

□

Performance.

As stated in Lemma 1, any secure HLA scheme is also a secure HLT scheme. Now we show that HLT constructions can be significantly more efficient than HLA schemes. Specifically, we compare our HLT with two HLA schemes presented in [2, 26]. We use comparable parameters that offer the same level of security. Specifically, the parameter q is 140-bit and p is 512-bit in our HLT scheme, p is 160-bit in [26] and N is 1024-bit in [2]. We consider n messages, namely $M_i = (M_i[1], \ldots, M_i[m])$ for $1 \leq i \leq n$, and compare the costs of the respective operations.

As shown in Table 1, the HLA scheme presented in [26] has the shortest tag but incurs the most expensive computation. Recall that exponentiations and multiplications in pairing groups are much less efficient than those in integer groups (e.g., the cost of one pairing is about that of 6-20 exponentiations [4]).

	HLT	HLA [26]	HLA [2]
assumption	DLOG	CDH	Factoring
pairing-based?	No	Yes	No
tag size	790 bits	160 bits	1024 bits
tagGen	$2n$ Ex+ mnMu	mnEx+ mnMu	mnEx+ mnMu
verify (single)	mEx	2Pairing + mEx	mEx
verify (aggregate)	mEx+ mn Mu	2Pairing + $(m+n)$Ex + mn Mu	$(m+n)$Ex +mn Mu
tagAggregate	nEx+ $2n$ Mu	nEx+ n Mu	nEx+ n Mu

Table 1: Performance of HLT and HLA, where Ex denotes exponentiation and Mu denotes multiplication.

5. QUERY INTEGRITY FOR OUTSOURCED DYNAMIC DATABASES: CONSTRUCTION AND ANALYSIS

In this section, we begin with a discussion on the solution design space. Then, we present the main construction and analyze its security. Finally, we discuss its efficiency with a comparison to the state-of-the-art solutions.

5.1 Solution Space

As discussed in the related work section, the state-of-the-art solutions to the query integrity problem fall into two approaches. The first approach is *tree-based* [13]. This approach incurs the least computational complexity because of the hash functions, but also incurs $O(n \log n)$ communication overhead. The second approach is *signature-based* [23]. This approach incurs high computational complexity of $O(kn)$ bilinear map exponentiations and communication complexity of $O(n)$ bitmaps (a small constant bits). Both approaches incur $O(mn)$ extra storage complexity in the cloud.

Our solution is based on a third approach. It reduces the extra complexity at the cloud side from $O(mn)$ to $O(n)$. It achieves a balanced trade-off between computational and communication communications. Specifically, it is less efficient than the tree-based solution in terms of computational complexity but substantially more efficient than the tree-based solution in terms of communication complexity. It is also substantially more efficient than the signature-based solution in terms of computational complexity but less efficient than the signature-based solution in terms of communication complexity. Perhaps more importantly, our solution can accommodate aggregate queries, which are not supported by the state-of-the-art solutions [13, 23].

The high-level idea of our solution is the following: The HLT scheme generates a tag for each tuple in the table, and the AuthODS scheme can be built on those tags, which are ordered by the search key. Intuitively, the AuthODS scheme provides two functionalities: one is to enable range query, and the other is to guarantee tag integrity (i.e., preventing HLT tags from being manipulated). The performance gain comes from the HLT scheme because only one aggregate tuple is needed to verify the integrity of (parts of) tuples. This is critical for the projection query because its query result only contains a portion of attributes from all tuples.

5.2 Proposed Construction

Let R be a table of n tuples with schema $(A_1, \ldots, A_m)$ and $r_1, \ldots, r_n$ be tuples ordered by search key A_1. Let L and U be the lower and upper bounds of the search key A_1, respectively.

Let $\Lambda_{\mathsf{RS}} = (\mathsf{KeyGen}, \mathsf{SetUp}, \mathsf{QueryVrfy}, \mathsf{Update})$ be a secure AuthODS scheme and $\Lambda_{\mathsf{HLT}} = (\mathsf{KeyGen}, \mathsf{Tag}, \mathsf{Vrfy}, \mathsf{HLTAgg})$ be a secure HLT scheme. The AuthDDB scheme is described as follows:

- KeyGen: Given the primary security parameter ℓ, the data owner obtains two secondary security parameters ℓ_1 and ℓ_2, and generates a pair of private and public keys (sk, pk),

1. Compute $(\Lambda_{\mathsf{RS}}.sk, \Lambda_{\mathsf{RS}}.pk) \leftarrow \Lambda_{\mathsf{RS}}.\mathsf{KeyGen}(1^{\ell_1})$.
2. Compute $(\Lambda_{\mathsf{HLT}}.sk, \Lambda_{\mathsf{HLT}}.pk) \leftarrow \Lambda_{\mathsf{HLT}}.\mathsf{KeyGen}(1^{\ell_2})$.
3. $sk = \{\Lambda_{\mathsf{RS}}.sk, \Lambda_{\mathsf{HLT}}.sk\}$ and $pk = \{\Lambda_{\mathsf{RS}}.pk, \Lambda_{\mathsf{HLT}}.pk\}$.
4. $\Lambda_{\mathsf{HLT}}.\mathsf{KeyGen}$ specifies the coefficient domain $\mathcal{C}$ and the message space $\mathcal{M}$, s.t. $(r_i.A_1, \ldots, r_i.A_m) \in \mathcal{M}$ for $r_i \in R, 1 \le i \le n$.

- SetUp: The data owner takes as input the private key sk and a table R, and obtains State and Au as follows:

  1. Let r_0 and r_{n+1} be two tuples added at both ends of table R in order to facilitate range query, where $r_0.A_1 = L$ and $r_{n+1}.A_1 = U$.
  2. Compute $\sigma_i \leftarrow \Lambda_{\mathsf{HLT}}.\mathsf{TagGen}(\Lambda_{\mathsf{HLT}}.sk, r_i)$ for tuple $r_i, 0 \le i \le n+1$.
  3. Let E_{RS} be the ordered data set, such that $E_{\mathsf{RS}} = \{E_0, \ldots, E_{n+1}\}$ where $E_i = (r_i.A_1, \sigma_i)$ for $0 \le i \le n+1$ and E_{RS} is ordered by A_1. Compute $(\mathsf{State}_{\mathsf{RS}}, \mathsf{Au}_{\mathsf{RS}}, E_{\mathsf{RS}}) \leftarrow \Lambda_{\mathsf{RS}}.\mathsf{SetUp}(\Lambda_{\mathsf{RS}}.sk, E_{\mathsf{RS}})$.
  4. Let $\mathsf{State} = \mathsf{State}_{\mathsf{RS}}$ and $\mathsf{Au} = (\mathsf{Au}_{\mathsf{RS}}, E_{\mathsf{RS}})$. R and Au will be outsourced to the server, and State will be made public.

- Update: The data owner interacts with the server to update the stored table with the update information Upd.

  Insertion: Suppose Upd is "insert the tuple r into R where $r_s.A_1 < r.A_1 < r_{s+1}.A_1, 0 \le s \le n$":

  1. The data owner computes $\sigma \leftarrow \Lambda_{\mathsf{HLT}}.\mathsf{TagGen}(\Lambda_{\mathsf{HLT}}.sk, r)$.
  2. Let $\mathsf{Upd}_{\mathsf{RS}}$ be "add an element of $E = (r.A_1, \sigma)$ between E_s and E_{s+1}". The data owner takes as input $\mathsf{Upd}_{\mathsf{RS}}, \Lambda_{\mathsf{RS}}.sk$ and $\mathsf{State}_{\mathsf{RS}}$, runs protocol $\Lambda_{\mathsf{RS}}.\mathsf{Update}$ with the server, who takes as input $\mathsf{Upd}_{\mathsf{RS}}$ and $\mathsf{Au}_{\mathsf{RS}}$. Eventually, the data owner outputs $\mathsf{State}'_{\mathsf{RS}}$ and the server updates $\mathsf{Au}_{\mathsf{RS}}$ to $\mathsf{Au}'_{\mathsf{RS}}$ and E_{RS} to E'_{RS}.
  3. The data owner delivers Upd to the server, and the server updates R to R'.

  Replacement: Suppose Upd is "update the tuple r with r'":

  1. The data owner fetches the tag σ for the tuple r from the server.
  2. The data owner computes $\sigma' \leftarrow \Lambda_{\mathsf{HLT}}.\mathsf{TagGen}(\Lambda_{\mathsf{HLT}}.sk, r')$.
  3. Let $\mathsf{Upd}_{\mathsf{RS}}$ be "update the element $(r.A_1, \sigma)$ with $(r'.A_1, \sigma')$". The data owner takes as input $\mathsf{Upd}_{\mathsf{RS}}, \Lambda_{\mathsf{RS}}.sk$ and $\mathsf{State}_{\mathsf{RS}}$, runs protocol $\Lambda_{\mathsf{RS}}.\mathsf{Update}$ with the server, who takes as input $\mathsf{Upd}_{\mathsf{RS}}$ and $\mathsf{Au}_{\mathsf{RS}}$. Eventually, the data owner outputs $\mathsf{State}'_{\mathsf{RS}}$ and the server updates $\mathsf{Au}_{\mathsf{RS}}$ to $\mathsf{Au}'_{\mathsf{RS}}$ and E_{RS} to E'_{RS}.
  4. The data owner delivers Upd to the server, and the server updates R to R'.

  Deletion: Suppose Upd is "delete the tuple r":

  1. The data owner fetches the tag σ for the tuple r from the server.
  2. Let $\mathsf{Upd}_{\mathsf{RS}}$ be "delete the element $(r.A_1, \sigma)$". The data owner takes as input $\mathsf{Upd}_{\mathsf{RS}}, \Lambda_{\mathsf{RS}}.sk$ and $\mathsf{State}_{\mathsf{RS}}$, runs protocol $\Lambda_{\mathsf{RS}}.\mathsf{Update}$ with the server, who takes as input $\mathsf{Upd}_{\mathsf{RS}}$ and $\mathsf{Au}_{\mathsf{RS}}$. Eventually, the data owner outputs $\mathsf{State}'_{\mathsf{RS}}$ and the server updates $\mathsf{Au}_{\mathsf{RS}}$ to $\mathsf{Au}'_{\mathsf{RS}}$ and E_{RS} to E'_{RS}.
  3. The data owner delivers Upd to the server, and the server updates R to R'.

We present the construction of QueryVrfy protocol based on the query type. Recall that $\mathsf{State} = \mathsf{State}_{\mathsf{RS}}$ and $\mathsf{Au} = (\mathsf{Au}_{\mathsf{RS}}, E_{\mathsf{RS}})$.

QueryVrfy **on Selection Query**.

Suppose the selection query is qry ="select * from R where $A_1 \ge a$ and $A_1 \le b$". There are two scenarios.

- If the result Rst is not null, assume $\mathsf{Rst} = \{r_s, \ldots, r_t\}, 1 \le s \le t \le n$, where $r_{s-1}.A_1 < a \le r_s.A_1$ and $r_t.A_1 \le b < r_{t+1}.A_1$. The protocol proceeds as follows:

  1. The server sets $\mathsf{Rst} = \{r_s, \ldots, r_t\}$, and sends Rst to the querier.
  2. The querier runs protocol $\Lambda_{\mathsf{RS}}.\mathsf{QueryVrfy}$ with the server for range query $\mathsf{qry}(a, b)$. If the output is reject, the querier aborts; otherwise, the querier obtains range query result $\mathsf{Rst}_{\mathsf{RS}} = ((r_s.A_1, \sigma_s), \ldots, (r_t.A_1, \sigma_t))$ and $\mathsf{Prf}_{\mathsf{RS}}$.
  3. The querier randomly selects a vector of coefficients $\vec{c} = (c_s, \ldots, c_t)$, computes $\sigma \leftarrow \Lambda_{\mathsf{HLT}}.\mathsf{HLTAgg}(\vec{c}, \vec{\mathsf{Tag}})$ where $\vec{\mathsf{Tag}} = (\sigma_s, \ldots, \sigma_t)$, and runs $\Lambda_{\mathsf{HLT}}.\mathsf{Vrfy}(\Lambda_{\mathsf{HLT}}.pk, \sum_{i=s}^{t} c_i r_i, \sigma)$. If the output is 1, the querier returns (accept, $\mathsf{Prf} = (\mathsf{Rst}_{\mathsf{RS}}, \mathsf{Prf}_{\mathsf{RS}})$, Rst); otherwise, the querier returns reject.

- If the result Rst is null, there exist two tuples $r_s, r_{s+1}, 0 \le s \le n$ such that $r_s.A_1 < a, b < r_{s+1}.A_1$. The querier can verify this fact by running protocol $\Lambda_{\mathsf{RS}}.\mathsf{QueryVrfy}$ with range query $\mathsf{qry}(a, b)$, which should return accept and $\mathsf{Rst}_{\mathsf{RS}}$ is null.

QueryVrfy **on Projection Query**.

Suppose the projection query is qry ="select $A_1, \ldots, A_k$ from R" ($k \ge 1$). The protocol proceeds as follows:

1. The server sets $\mathsf{Rst} = \{(r_i.A_1, \ldots, r_i.A_k), 1 \le i \le n\}$ and passes it to the querier.

2. The querier runs protocol $\Lambda_{\mathsf{RS}}.\mathsf{QueryVrfy}$ with the server on range query (L, U). If the output is reject, the querier aborts; otherwise, the querier obtains the range query result $\mathsf{Rst}_{\mathsf{RS}} = ((r_0.A_1, \sigma_0), \ldots, (r_{n+1}.A_1, \sigma_{n+1}))$ and proof $\mathsf{Prf}_{\mathsf{RS}}$.

3. The querier randomly selects a vector of coefficients $\vec{c} = (c_1, \ldots, c_n)$ and sends it to the server.

4. The server computes $r.A_j = \sum_{i=1}^{n} c_i r_i.A_j, k+1 \le j \le m$ and sends $(r.A_{k+1}, \ldots, r.A_m)$ to the querier as part of Prf.

5. The querier computes $r.A_j = \sum_{i=1}^{n} c_i r_i.A_j, 1 \le j \le k$ from $\mathsf{Rst} = \{(r_i.A_1, \ldots, r_i.A_k), 1 \le i \le n\}$ and the aggregated tag $\sigma = \Lambda_{\mathsf{HLT}}.\mathsf{HLTAgg}(\vec{c}, \vec{\mathsf{Tag}})$, where $\vec{\mathsf{Tag}} = (\sigma_1, \ldots, \sigma_n)$.

6. The querier computes $\Lambda_{\mathsf{HLT}}.\mathsf{Vrfy}(\Lambda_{\mathsf{HLT}}.pk, M, \sigma)$ where $M = (r.A_1, \ldots, r.A_m)$. If the output is 1, the querier returns

$$(\mathsf{accept}, \mathsf{Rst}, \mathsf{Prf} = (\mathsf{Rst}_{\mathsf{RS}}, \mathsf{Prf}_{\mathsf{RS}}, r.A_{k+1}, \ldots, r.A_m));$$

otherwise, the querier returns reject.

QueryVrfy **on Join Query**.

Let P be another table with schema $(B_1, \ldots, B_m)$ and be processed by SetUp, where B_1 is the search key. For convenience, suppose P has n tuples, and A_2 and B_2 are the respective primary key for tables R and P. Suppose the join query is qry ="select $R.^*, P.^*$ from R, P where $R.A_s = P.B_t$" $(1 \le s, t \le m)$. The protocol proceeds as follows:

1. The server sets $\mathsf{Rst} = (R^*, P^*)$ and passes it to the querier, where R^* and P^* are the tuples in R and P such that $R.A_s = P.B_t$.

2. The querier runs QueryVrfy on projection queries with qry_R = "select A_2, A_s from R" and qry_P = "select B_2, B_t from P", respectively. If either execution outputs 0, the querier aborts; otherwise, the querier obtains $\{(r_i.A_2, r_i.A_s, \sigma_i), 1 \le i \le n\}$ and $\{(p_j.B_2, p_j.B_t, \sigma'_j), 1 \le j \le n\}$.

3. The querier identifies tuples satisfying $R.A_s = P.B_t$ from $\{(r_i.A_2, r_i.A_s, \sigma_i), 1 \le i \le n\}$ and $\{(p_j.B_2, p_j.B_t, \sigma'_j), 1 \le j \le n\}$. Specifically, let α and β be two sets of indices such that $\alpha \subseteq \{1, \ldots, n\}$, $\beta \subseteq \{1, \ldots, n\}$ and $i \in \alpha, j \in \beta, r_i.A_s = p_j.B_t$. Then, the querier obtains two sets of tuples $\{(r_i.A_2, r_i.A_s, \sigma_i), i \in \alpha\}$ and $\{(p_j.B_2, p_j.B_t, \sigma'_j), j \in \beta\}$, where $i \in \alpha, j \in \beta, r_i.A_s = p_j.B_t$. The querier verifies that the number of tuples in R^* equals to the number of tuples in $\{(r_i.A_2, r_i.A_s, \sigma_i), i \in \alpha\}$, and the number of tuples in P^* equals to the number of tuples in $\{(r_j.B_2, p_j.B_t, \sigma'_j), j \in \beta\}$. If both are true, the querier continues; otherwise, the querier aborts.

4. The querier randomly selects a vector of coefficients $\vec{c} = (c_1, \ldots, c_{|\alpha|})$, computes σ by aggregating tags $\{\sigma_i, i \in \alpha\}$, and executes $\Lambda_{\mathsf{HLT}}.\mathsf{Vrfy}$ with $\vec{c}$, σ, R^* and $\Lambda_{\mathsf{HLT}}.pk$. The same is executed with respect to P^*. If both executions output 1, the querier returns

$$(\mathsf{accept}, \mathsf{Rst}, \mathsf{Prf} = (\mathsf{Rst}_{\mathsf{RS},R}, \mathsf{Prf}_{\mathsf{RS},R}, \mathsf{Rst}_{\mathsf{RS},P}, \mathsf{Rst}_{\mathsf{RS},P}));$$

otherwise, the querier returns reject. Here $(\mathsf{Rst}_{\mathsf{RS},R}, \mathsf{Prf}_{\mathsf{RS},R})$ are the query result and proof when executing QueryVrfy on projection query qry_R, and that $(\mathsf{Rst}_{\mathsf{RS},P}, \mathsf{Prf}_{\mathsf{RS},P})$ are the query result and proof when executing QueryVrfy on projection query qry_P.

QueryVrfy **on Aggregate Query**.

Suppose the aggregate query is qry ="select SUM(A_2) from R where $A_1 \ge a$ and $A_1 \le b$". Suppose $1 \le s \le t \le n$, $r_{s-1}.A_1 < a \le r_s.A_1$ and $r_t.A_1 \le b < r_{t+1}.A_1$. The protocol proceeds as follows:

1. The server sets $r.A_j = \sum_{i=s}^{t} r_i.A_j$ for $j = 1, \ldots, m$, sets $\mathsf{Rst} = r.A_2$, and passes Rst and $(r.A_1, r.A_3, \ldots, r.A_m)$ to the querier.

2. The querier runs protocol $\Lambda_{\mathsf{RS}}.\mathsf{QueryVrfy}$ on range query $\mathsf{qry}(a, b)$ with the server. If the output is reject, the querier aborts; otherwise, the querier obtains $\mathsf{Rst}_{\mathsf{RS}} = ((r_s.A_1, \sigma_s), \ldots, (r_t.A_1, \sigma_t))$ and $\mathsf{Prf}_{\mathsf{RS}}$ for the range query.

3. The querier computes $\sigma \leftarrow \Lambda_{\mathsf{HLT}}.\mathsf{HLTAgg}(\vec{c}, \vec{\mathsf{Tag}})$, where $\vec{c}$ is a vector of 1's and $\vec{\mathsf{Tag}} = (\sigma_s, \ldots, \sigma_t)$. The querier computes $\Lambda_{\mathsf{HLT}}.\mathsf{Vrfy}(\Lambda_{\mathsf{HLT}}.pk, M, \sigma)$, where $M = \{r.A_1, \cdots, r.A_m\}$. If the output is 1, the querier returns

$$(\mathsf{accept}, \mathsf{Rst}, \mathsf{Prf} = (\mathsf{Rst}_{\mathsf{RS}}, \mathsf{Prf}_{\mathsf{RS}}, r.A_1, r.A_3, \ldots, r.A_m));$$

otherwise, the querier returns reject.

REMARK 1. *In selection/projection/join query, we use randomly selected $\vec{c}$ to prevent* aggregate attack. *To see this, let us consider the case without using $\vec{c}$, namely $\vec{c}$ is composed of 1s. The server has $r'_i = r_i, s-1 \le i \le t+1$, and manipulates two tuples $r_e, r_{e+1}, s \le e \le t-1$, to obtain $r'_e = (r_e.A_1, r_e.A_2 + 1, r_e.A_3, \ldots)$ and $r'_{e+1} = (r_e.A_1, r_e.A_2 - 1, r_e.A_3, \ldots)$, which makes $\sum_{i=s}^{t} r_i = \sum_{i=s}^{t} r'_i$. Hence, the server could have $\Lambda_{\mathsf{HLT}}.\mathsf{Vrfy}$ output 1 with manipulated $\{r'_s, \ldots, r'_t\}$.*

REMARK 2. *Note that our solution toward the aggregate query supports only SUM queries and weighted SUM queries.*

5.3 Security Analysis

It is easy to check the correctness of the AuthDDB scheme. In what follows we focus on its security.

THEOREM 3. *Assume Λ_{RS} is a secure* AuthODS *scheme and Λ_{HLT} is a secure*HLT *scheme, where the coefficient space is large enough (e.g. $1/|\mathcal{C}|$ is negligible). The proposed* AuthDDB *scheme attains the soundness with respect to the selection, projection, join and aggregate queries.*

The basic idea to prove the soundness is to show that if there exists a probabilistic polynomial-time adversary $\mathcal{A}$ that breaks soundness of the AuthDDB scheme, we can break either soundness of Λ_{RS} or security of HLT.

PROOF. We show our proof through a sequence of games between a challenger, who plays the role of the data owner , and adversary A, who acts as the malicious server.
Game 0: Game 0 is defined as in Definition 2, where the challenger only keeps the relevant public/private keys and the latest state information State_k.
Game 1: Game 1 is defined as in Definition 2, where the challenger keeps the relevant public/private keys, the latest state information State_k and the latest auxiliary information Au_k. We can see that the probability that $\mathcal{A}$ wins Game 1 is at most negligibly less than the probability that $\mathcal{A}$ wins Game 0.
Game 2: Game 2 is defined as in Definition 2, where the challenger keeps the relevant public/private keys, the latest state information State_k, the latest auxiliary information Au_k, and the latest database D_k. We can see that the probability that $\mathcal{A}$ wins Game 2 is at most negligibly less than the probability that $\mathcal{A}$ wins Game 1.

Let $\mathsf{State}_k, \mathsf{Au}_k$ and $D_k = R$ be the latest version of the state information, the auxiliary information and the database, where $\mathsf{State}_k = \mathsf{State}_{\mathsf{RS}}, \mathsf{Au}_k = (\mathsf{Au}_{\mathsf{RS}}, E_{\mathsf{RS}})$, and $E_{\mathsf{RS}} = \{(L, \sigma_0), (r_1.A_1, \sigma_1), \ldots, (r_n.A_1, \sigma_n), (U, \sigma_{n+1})\}$.

Soundness of Selection Query Suppose the adversary $\mathcal{A}$ finds a selection query qry with query result Rst and proof Prf, and wins Game 2. In other words, given qry ="select * from R where $A_1 \ge a$ and $A_1 \le b$", $\mathcal{A}$ returns $\mathsf{Rst} = \{r'_s, \ldots, r'_t\}$ and $\mathsf{Prf} = \{\mathsf{Rst}_{\mathsf{RS}}, \mathsf{Prf}_{\mathsf{RS}}\}$, and wins Game 2, where $\mathsf{Rst}_{\mathsf{RS}} = \{(r'_s.A_1, \sigma'_s), \ldots, (r'_t.A_1, \sigma'_t)\}$. Let $\mathsf{localRst} \leftarrow \mathsf{LocalQuery}(\mathsf{qry}, D_K)$ and $\mathsf{localRst}_{\mathsf{RS}} \leftarrow \Lambda_{\mathsf{RS}}.\mathsf{LocalQuery}(\mathsf{qry}(a, b), E_{\mathsf{RS}})$, which are produced by the challenger with stored $\mathsf{State}_k, \mathsf{Au}_k$ and $D_k = R$.

Since $\mathcal{A}$ wins Game 3, we have

$$(\mathsf{accept}, \mathsf{Rst}_{\mathsf{RS}}, \mathsf{Prf}_{\mathsf{RS}}) \leftarrow \\ (Q(\Lambda_{\mathsf{RS}}.pk, \mathsf{qry}(a,b), \mathsf{State}_k) \leftrightarrow S(\mathsf{Au}_{\mathsf{RS}}, E_{\mathsf{RS}})).$$

This means that given range query $\mathsf{qry}(a,b)$, $\mathsf{Rst}_{\mathsf{RS}}$ is the query result on the ordered data set E_{RS} with respect to state State_k. On the other hand, if $\mathsf{localRst}_{\mathsf{RS}} \neq \mathsf{Rst}_{\mathsf{RS}}$, there exist two different query results with respect to $\mathsf{qry}(a,b)$, which contradicts with the soundness of Λ_{RS}. Since Λ_{RS} is sound, we have $\mathsf{Rst}_{rs} = \mathsf{localRst}_{\mathsf{RS}}$. Therefore, we can assume $\mathsf{localRst} = \{r_s, \ldots, r_t\}$.

Suppose $\vec{c} = (c_s, \ldots, c_t)$ is the coefficient vector sent from the challenger to $\mathcal{A}$. The challenger computes $\sigma' \leftarrow \Lambda_{\mathsf{HLT}}.\mathsf{HLTAgg}(\vec{c}, \vec{\mathsf{Tag}}')$, where $\vec{\mathsf{Tag}}' = (\sigma'_s, \ldots, \sigma'_t)$. Since the adversary wins Game 2, it should satisfy:

$$1 \leftarrow \Lambda_{\mathsf{HLT}}.\mathsf{Vrfy}(\Lambda_{\mathsf{HLT}}.pk, \sum_{i=s}^{t} c_i r'_i, \sigma').$$

If $\mathsf{localRst} \neq \mathsf{Rst}$, there exist some $i, s \leq i \leq t, r_i \neq r'_i$. So, we have $\sum_{i=s}^{t} c_i r'_i = \sum_{i=s}^{t} c_i r_i$ with negligible probability because $c_1, \ldots, c_n$ are randomly selected from $\mathcal{C}$ and $\frac{1}{|\mathcal{C}|}$ is negligible. That is, we have another equation

$$1 \leftarrow \Lambda_{\mathsf{HLT}}.\mathsf{Vrfy}(\Lambda_{\mathsf{HLT}}.pk, \sum_{i=s}^{t} c_i r_i, \sigma'),$$

which allows us to break security of Λ_{HLT} if $\mathsf{localRst} \neq \mathsf{Rst}$. This means that if $\mathcal{A}$ breaks the soundness of $\mathsf{AuthDDB}$, we can break either the soundness of Λ_{RS} or the security of Λ_{HLT}.

Soundness of Projection Query suppose the adversary $\mathcal{A}$ finds a projection query qry with query result Rst and proof Prf, and wins Game 2. In other words, given qry =“select $A_1, \ldots, A_k$ from R” ($k \geq 1$), $\mathcal{A}$ returns query result $\mathsf{Rst} = \{(r'_i.A_1, \ldots, r'_i.A_k), 1 \leq i \leq n\}$ and $\mathsf{Prf} = (\mathsf{Rst}_{\mathsf{RS}}, \mathsf{Prf}_{\mathsf{RS}}, r'.A_{k+1}, \ldots, r'.A_m)$. Let $\mathsf{localRst} \leftarrow \mathsf{LocalQuery}(\mathsf{qry}, D_k)$ and $\mathsf{localRst}_{\mathsf{RS}} \leftarrow \Lambda_{\mathsf{RS}}.\mathsf{LocalQuery}(\mathsf{qry}(a,b), E_{\mathsf{RS}})$, which are produced by the challenger with stored $\mathsf{State}_k, \mathsf{Au}_k$ and $D_k = R$.

Since $\mathcal{A}$ wins Game 2, we have

$$(\mathsf{accept}, \mathsf{Rst}_{\mathsf{RS}}, \mathsf{Prf}_{\mathsf{RS}}) \leftarrow \\ (Q(\Lambda_{\mathsf{RS}}.pk, \mathsf{qry}(L,U), \mathsf{State}_k) \leftrightarrow S(\mathsf{Au}_{\mathsf{RS}}, E_{\mathsf{RS}})).$$

This means that given range query $\mathsf{qry}(L,U)$, $\mathsf{Rst}_{\mathsf{RS}}$ is the query result on the ordered data set E_{RS} with respect to state State_k. On the other hand, if $\mathsf{localRst}_{\mathsf{RS}} \neq \mathsf{Rst}_{\mathsf{RS}}$, there exist two different query results with respect to $\mathsf{qry}(a,b)$, which contradicts with the soundness of Λ_{RS}. Since Λ_{RS} is sound, we have $\mathsf{Rst}_{rs} = \mathsf{localRst}_{\mathsf{RS}}$. Therefore, we can assume $\mathsf{localRst} = \{r_1, \ldots, r_n\}$.

Suppose $\vec{c} = (c_1, \ldots, c_n)$ is the coefficient vector sent from the challenger to $\mathcal{A}$. The challenger computes $\sigma' \leftarrow \Lambda_{\mathsf{HLT}}.\mathsf{HLTAgg}(\vec{c}, \vec{\mathsf{Tag}}')$, where $\vec{\mathsf{Tag}}' = (\sigma'_s, \ldots, \sigma'_t)$ and $r'.A_j = \sum_{i=1}^{n} c_i r_i.A_j, 1 \leq j \leq m$. Since $\mathcal{A}$ wins Game 2, we have

$$1 \leftarrow \Lambda_{\mathsf{HLT}}.\mathsf{Vrfy}(\Lambda_{\mathsf{HLT}}.pk, (r'.A_1, \ldots, r'.A_m), \sigma').$$

If $\mathsf{localRst} \neq \mathsf{Rst}$, there exist some $i, j, 1 \leq i \leq n, 1 \leq j \leq k$ such that $r_i.A_j \neq r'_i.A_j$. This means that $\sum_{i=1}^{n} c_i r'_i.A_j = \sum_{i=1}^{n} c_i r_i.A_j$ with negligible probability. Therefore, we obtain another equation

$$1 \leftarrow \Lambda_{\mathsf{HLT}}.\mathsf{Vrfy}(\Lambda_{\mathsf{HLT}}.pk, (\sum_{i=1}^{n} c_i r_i.A_1, \ldots, \sum_{i=1}^{n} c_i r_i.A_m), \sigma'),$$

which allows us to break the security of Λ_{HLT} if $\mathsf{localRst} \neq \mathsf{Rst}$. This means that if $\mathcal{A}$ breaks the soundness of $\mathsf{AuthDDB}$, we can break either the soundness of Λ_{RS} or the security of Λ_{HLT}.

Soundness of Join Query This holds because of the soundness of the projection query and the security of Λ_{HLT}.

Soundness of Aggregate Query Suppose the adversary $\mathcal{A}$ finds an aggregate query qry with query result Rst and proof Prf, and wins Game 2. In other words, given qry=“select SUM(A_2) from R where $A_1 \geq a$ and $A_1 \leq b$”, $\mathcal{A}$ returns query result $\mathsf{Rst} = r'.A_2$ and $\mathsf{Prf} = \{r'.A_1, r'.A_3, \ldots, r'.A_m, \mathsf{Rst}_{\mathsf{RS}}, \mathsf{Prf}_{\mathsf{RS}}\}$, where $\mathsf{Rst}_{\mathsf{RS}} = \{(r'_s.A_1, \sigma'_s), \ldots, (r'_t.A_1, \sigma'_t)\}$ for the range query $\mathsf{qry}(a,b)$. Let $\mathsf{localRst} \leftarrow \mathsf{LocalQuery}(\mathsf{qry}, D_K)$ and $\mathsf{localRst}_{\mathsf{RS}} \leftarrow \Lambda_{\mathsf{RS}}.\mathsf{LocalQuery}(\mathsf{qry}(a,b), E_{\mathsf{RS}})$, which are produced by the challenger with stored $\mathsf{State}_k, \mathsf{Au}_k$ and $D_k = R$.

since the adversary wins Game 2, we have

$$(\mathsf{accept}, \mathsf{Rst}_{\mathsf{RS}}, \mathsf{Prf}_{\mathsf{RS}}) \leftarrow \\ (Q(\Lambda_{\mathsf{RS}}.pk, \mathsf{qry}(a,b), \mathsf{State}_k) \leftrightarrow S(\mathsf{Au}_{\mathsf{RS}}, E_{\mathsf{RS}}))$$

This means that given range query $\mathsf{qry}(a,b)$, $\mathsf{Rst}_{\mathsf{RS}}$ is the query result on the ordered data set E_{RS} with respect to state State_k. On the other hand, if $\mathsf{localRst}_{\mathsf{RS}} \neq \mathsf{Rst}_{\mathsf{RS}}$, there exist two different query results with respect to $\mathsf{qry}(a,b)$, which contradicts with the soundness of Λ_{RS}. Since Λ_{RS} is sound, we have $\mathsf{Rst}_{rs} = \mathsf{localRst}_{\mathsf{RS}}$, which means that any tuple r_i in R, where $s \leq i \leq t$, satisfies $a \leq r_i.A_1 \leq b$. The challenger computes $\sigma' \leftarrow \Lambda_{\mathsf{HLT}}.\mathsf{HLTAgg}(\vec{c}, \vec{\mathsf{Tag}}')$, where $\vec{\mathsf{Tag}}' = (\sigma'_s, \ldots, \sigma'_t)$ and $\vec{c}$ is all 1's. If $\mathcal{A}$ wins Game 2, it should satisfy

$$1 \leftarrow \Lambda_{\mathsf{HLT}}.\mathsf{Vrfy}(\Lambda_{\mathsf{HLT}}.pk, (r'.A_1, \ldots, r'.A_m), \sigma').$$

If $\mathsf{localRst} \neq \mathsf{Rst}$, we have $\sum_{i=s}^{t} r_i.A_2 \neq r'.A_2$. Therefore, we obtain another equation

$$1 \leftarrow \Lambda_{\mathsf{HLT}}.\mathsf{Vrfy}(\Lambda_{\mathsf{HLT}}.pk, (\sum_{i=s}^{t} r_i.A_1, \ldots, \sum_{i=s}^{t} c_i r_i.A_m), \sigma'),$$

which allows us to break the security of Λ_{HLT} if $\mathsf{localRst} \neq \mathsf{Rst}$. This means that if $\mathcal{A}$ breaks the soundness of $\mathsf{AuthDDB}$, we can break either the soundness of Λ_{RS} or the security of Λ_{HLT}. □

5.4 Performance

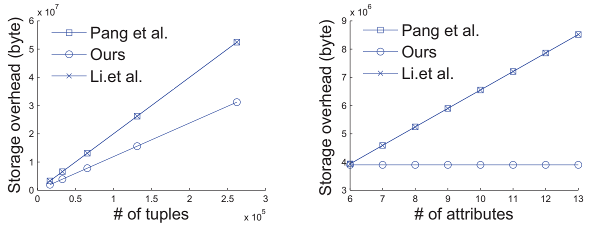
We compare the asymptotic performance of our solution with that of the two state-of-the-art solutions [13, 23]. As shown in Table 2, our solution is more expressive because it additionally supports aggregate queries, such as: “select SUM(A_2) from R where $A_1 > a$." Moreover, our solution allows the join query with respect to arbitrary attributes, such as: “select $R.^*, P.^*$ from R, P where $R.A_3 = P.B_4$" *without* requiring that $R.A_3$ and $P.B_4$ be search keys. Whereas, this type of join queries cannot be handled by the state-of-the-art solutions [13, 23].

Regarding pre-processing the database before outsourcing it to the cloud, our solution is more efficient than [23], and as efficient as [13]. In addition, our solution incurs the least extra storage complexity. To see this, we compare the three solutions with parameters in Table 1. Figure 2(a) shows that our solution is storage-space more efficient than [13, 23] as long as the number of attributes is greater than Tag/Hash, which is often the case. Moreover, from Figure 2(b) we can see that the storage-space requirement in our solution is independent of the number of attributes; in contrast, the storage-space complexity of [13, 23] increases linearly with respect to the number of attributes.

Regarding selection queries, projection queries and join queries, our solution incurs respective computational complexity $O(\log n)$

		Li et al. [13]	Pang et al. [23]	This paper
Functions		Selection, Projection, Join	Selection, Projection, Join	Selection, Projection, Join, Aggregate
Technique		Merkle-based Hash Tree	Aggregate Signature with Chaining	Merkle-based Hash Tree and HLT
Security		Sound	Sound	Sound
Data PreProcessing		$O(n)$Hash	$O(mn)$Ex	$O(n)$Hash+ $O(n)$Ex
Storage Overhead		$O(mn)$Hash	$O(mn)$AggSig	$O(n)$Hash + $O(n)$Tag
Selection	Computation$_S$	N/A	$O(n)$Mu	N/A
	Communication	$O(\log n)$Hash	$O(n)$Bitmap	$O(\log n)$Hash + $O(n)$Tag
	Computation$_V$	$O(n)$Hash	$O(n)$Ex	$O(\log n)$Hash + $O(n)$Ex
Projection	Computation$_S$	N/A	$O(kn)$Mu	$O(n)$Mu
	Communication	$O((m-k)n)$attribute	$O(n)$Bitmap	$O(n+m)$Tag
	Computation$_V$	$O(n)$Hash	$O(kn)$Ex	$O(n)$Ex
Join	Computation$_S$	N/A	$O(n)$Mu	$O(n)$Mu
	Communication	$O(n\log(n))$Hash + R^*	$O(n)$Bitmap + R^*	$O(n+m)$Tag
	Computation$_V$	$O(n\log(n))$Hash	$O(n)$Ex	$O(n)$Ex
Aggregate	Computation$_S$	N/A	N/A	N/A
	Communication	N/A	N/A	$O(\log n)$Hash + $O(\lambda)$Tag
	N/A	N/A	N/A	$O(\log n)$Hash + $O(\lambda)$Ex
Update	Computation$_S$	$O(\log n)$Hash	N/A	$O(\log n)$Hash
	Communication	O(1)	O(1)	O(1)
	Computation$_O$	$O(\log n)$Hash	$O(m)$Ex	$O(\log n)$Hash

Table 2: Comparison of asymptotic performance, where Hash **is 160 bits,** Sig **is 1024 bits,** AggSig= **160 bits,** Tag= **792 bits,** Bitmap **is a small constant,** Ex **denotes modular exponentiation,** Mu **denotes modular multiplication** Pairing **denotes pairing operation,** k **is the number of attributes in projection query,** attribute **is an attribute value in** R, R^* **denotes unmatched tuples in** R**, and assume** $|R| = |P| = n$ **in join query. Note that our solution supports aggregate queries and more flexible join queries, and we do not count the basic search operation in the comparison.**



(a) $m = 10$

(b) $n = 2^{15}$

Figure 2: Comparison of storage overhead

Hash plus $O(n)$Ex, $O(n)$Ex, and $O(n)$Ex at the querier side in order to aggregate HLT tags. However, our solution still outperforms [23], which incurs respective computational complexity $O(n)$Ex, $O(kn)$Ex, and $O(n)$Ex on bilinear group.

Regarding projection queries and join queries, our solutions requires $O(n+m)$ tags. In contrast, [13] requires $O((m-k)n)$ attribute values for projection queries, and $O(n \log n)$ hash values plus those unmatched tuples in R for join queries. Although [23] only requires $O(n)$ certified bitmap (recording updated tuples in on update period) for projection queries, it requires at least $O(n)$ certified bitmap plus those unmatched tuples in R for join queries. It is due to the fact that [13, 23] have to fetch at least one table (either R or P) for join queries.

Regarding aggregate queries, the computational and communication complexities are the same regardless of the number of attributes the querier wants to aggregate. For example, our solution incurs the same complexity for queries such as: "select SUM(A_1), . . ., SUM(A_k) from R where $A_1 > a$ and $A_1 < b$" and "select SUM(A_1) from R where $A_1 > a$ and $A_1 < b$".

6. INTEGRATE QUERY INTEGRITY AND STORAGE INTEGRITY

The accompanying concept to query integrity is *storage integrity*, namely the assurance that the outsourced data is kept intact in the cloud. Elegant solutions to storage integrity include Provable Data Possession (PDP) [2] and Proof of Retrievability (POR) [12]. In particular, PDP can achieve *constant* computational and communication complexities in the static setting [2], and *logarithmic* computational and communication complexities in the dynamic setting [8].

A systematic solution should assure both query complexity and storage complexity. Intuitively, query integrity is more demanding than storage integrity because storage integrity does not have to deal with the structure of database. However, one cannot simply adapt PDP/POR techniques to the setting of outsourced database because they deal with unstructured data. In what follows, we sketch a solution that integrates PDP-flavor storage integrity with respect to the *logical* structure of the outsourced database (rather than the physical structure of the database). The solution is not optimal because it incurs communication complexity of $O(n)$ tags and computational complexity of $O(n)$ exponentiations. We defer a detailed analysis of the following solution to an expanded version of this paper.

Specifically, for a table R with schema $(A_1, \ldots, A_m)$, where A_i is the primary key (ID) that uniquely identifies a tuple. In order to ensure storage integrity, the database storage integrity auditor (e.g., the data owner or a third party) can perform the procedure shown in Figure 3. Since the auditor has no knowledge about R, it fetches

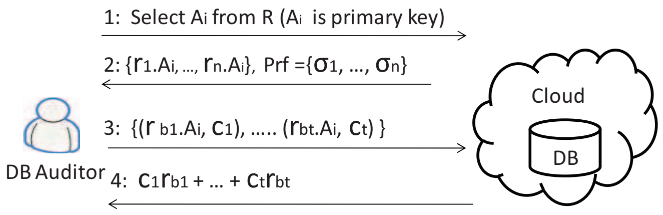

1: Select Ai from R (Ai is primary key)
2: {r1.Ai, …, rn.Ai}, Prf ={σ1, …, σn}
Cloud
3: {(r b1.Ai, c1), ….. (rbt.Ai, ct) }
DB
DB Auditor
4: c1rb1 + … + ctrbt

Figure 3: Procedures to ensure storage integrity

and verifies integrity of all primary keys (IDs) in R and their tags, which is showed in Steps 1 and 2. Then the auditor randomly selects t primary IDs from the set of IDs and t coefficients, denoted by $(r_{b_1}.A_i, c_1), \ldots, (r_{b_t}.A_i, c_t)$, and asks the server to compute an aggregate tuple $r = \sum_{i=1}^{t} c_i r_{b_i}$. Suppose σ is the aggregated tag of $\sigma_{b1}, \ldots, \sigma_{b_t}$ with coefficients $c_1, \ldots, c_t$, the auditor can verify storage integrity by running $\Lambda_{\mathsf{HLT}}.\mathsf{Vrfy}(\Lambda_{\mathsf{HLT}}.pk, r, \sigma)$. If the output is 1, the storage assurance can be guaranteed with $(1-f)^t$ confidence, where f is the fraction of the corrupted tuples.

7. CONCLUSION

We presented an efficient solution to the problem of query integrity in the setting of outsourced dynamic databases. Query integrity allows a querier, the data owner or a third party, to verify that its queries were faithfully executed by the cloud server. Compared with the state-of-the-art solutions, our solution is: (i) more powerful by additionally supporting aggregate queries (in addition to selection, projection, and join queries), and (ii) more efficient by eliminating a logarithmic (or even linear) multiplication factor from the overall cost (depending on the type of the queries).

Our solution still incurs linear complexity. A notable direction for future research is to address the following open problem: Can we attain query integrity logarithmic (or constant) complexity as in the case of assuring storage integrity?

Acknowledgement

Qingji Zheng and Shouhuai Xu were supported in part by an AFOSR MURI grant and a NSF grant. Giuseppe Ateniese was supported in part by a Google Research Award and an IBM Faculty Award.

8. REFERENCES

[1] B. Applebaum, Y. Ishai, and E. Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In S. Abramsky, C. Gavoille, C. Kirchner, F. Meyer auf der Heide, and P. Spirakis, editors, *Automata, Languages and Programming*, volume 6198 of *Lecture Notes in Computer Science*, pages 152–163. Springer Berlin / Heidelberg, 2010. 10.1007/978-3-642-14165-2_14.

[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*, CCS '07, pages 598–609, New York, NY, USA, 2007. ACM.

[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*, ASIACRYPT '09, pages 319–333, Berlin, Heidelberg, 2009. Springer-Verlag.

[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*, pages 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*, EUROCRYPT'03, pages 416–432, Berlin, Heidelberg, 2003. Springer-Verlag.

[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*, CRYPTO'10, pages 483–501, Berlin, Heidelberg, 2010. Springer-Verlag.

[7] P. Devanbu, M. Gertz, C. Martel, and S. G. Stubblebine. Authentic data publication over the internet. *J. Comput. Secur.*, 11(3):291–314, Apr. 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*, CCS '09, pages 213–222, New York, NY, USA, 2009. ACM.

[9] A. Fiat. Batch rsa. In *Proceedings on Advances in cryptology*, CRYPTO '89, pages 175–185, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

[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*, CRYPTO'10, pages 465–482, Berlin, Heidelberg, 2010. Springer-Verlag.

[11] M. Goodrich, R. Tamassia, and N. Triandopoulos. Super-efficient verification of dynamic outsourced databases. In T. Malkin, editor, *Topics in Cryptology ÍC CT-RSA 2008*, volume 4964 of *Lecture Notes in Computer Science*, pages 407–424. Springer Berlin / Heidelberg, 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*, CCS '07, pages 584–597, New York, NY, USA, 2007. ACM.

[13] 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.

[14] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin. Authenticated index structures for aggregation queries. *ACM Trans. Inf. Syst. Secur.*, 13(4):32:1–32:35, Dec. 2010.

[15] R. C. Merkle. A certified digital signature. In *Proceedings on Advances in cryptology*, CRYPTO '89, pages 218–238, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

[16] 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.

[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*, 2(2):107–138, May 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*, DASFAA'06, pages 420–436, Berlin, Heidelberg, 2006. Springer-Verlag.

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

[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*, DBSec'10, pages 177–192, Berlin, Heidelberg, 2010. Springer-Verlag.

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

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

[24] C. Papamanthou, R. Tamassia, and N. Triandopoulos. Authenticated hash tables. In *Proceedings of the 15th ACM conference on Computer and communications security*, CCS '08, pages 437–448, New York, NY, USA, 2008. ACM.

[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*, CRYPTO'11, pages 91–110, Berlin, Heidelberg, 2011. Springer-Verlag.

[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*, ASIACRYPT '08, pages 90–107, Berlin, Heidelberg, 2008. Springer-Verlag.

[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*, VLDB '07, pages 782–793. VLDB Endowment, 2007.

[29] J. XU and E.-C. CHANG. Authenticating aggregate range queries over multidimensional dataset. Cryptology ePrint Archive, Report 2010/050, 2010. `http://eprint.iacr.org/`.

[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*, SIGMOD '09, pages 5–18, New York, NY, USA, 2009. ACM.

[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*, ICDE '08, pages 1082–1091, Washington, DC, USA, 2008. IEEE Computer Society.