Privacy-Preserving Verification of Aggregate Queries on Outsourced Databases

Stuart Haber William G. Horne Tomas Sander Danfeng Yao*
{stuart.haber, bill.horne, tomas.sander}@hp.com, dyao@cs.brown.edu

November 10, 2006

Abstract

It is often desirable to be able to guarantee the integrity of historical data, ensuring that any subsequent modifications to the data can be detected. It would be especially convenient to extend such proofs of integrity to certain computations performed later using the historic data. We approach this question in the context of outsourced databases, where a data owner delegates the ability to answer users' queries to a service provider, and distrustful users may desire to verify the integrity of responses to their queries on the data. We present a solution for integrity verification of database aggregate queries, such as SUM and MAX. We design proofs of correctness and completeness of aggregate results. What makes the problem challenging is that individual data entries may be sensitive (e.g. as in medical databases), and should not be revealed to the user. We give cryptographic protocols to support verification of query results in a privacy-preserving fashion.

1 Introduction

For many applications, it is desirable to have *historical data integrity*, in which the integrity of some data is established at a specific point in time, and any subsequent modifications to that data can be detected. Of particular interest is the ability to establish the historical integrity of transactions and event logs, which are routinely collected in IT systems for a variety of applications such as intrusion detection, forensics, fraud detection, network monitoring and quality control. Recently, audit logs and IT auditing have become increasingly important as a means of assuring compliance with financial and legal regulations, such as the Sarbanes-Oxley Act (SOX) in the US and similar regulations worldwide.

Consider the following example. A corporation logs financial transactions into a general ledger. At periodic time intervals, a third-party audit is performed to verify that the corporation is following legally acceptable accounting practices. Although there are checks and balances in place, there is always a threat of fraud if an adversary is able to get access to the system and modify entries in the ledger. These threats are traditionally addressed using carefully managed access control systems and techniques such as segregation of duties. But this only indirectly protects the integrity of the data.

It is feasible to address this problem using cryptographic techniques such as message authentication codes or digital signatures to protect the integrity of the data. However, these techniques are not sufficient by themselves to protect against threats from a malicious adversary. A better approach might use a scheme guaranteeing "forward integrity" (e.g. [5, 6]), which makes it intractable

*Work performed during an internship at HP Labs, Princeton.

for an adversary to change previously collected data without access to some well-guarded offline secrets.

The adversary could be the corporation itself (or a small conspiracy of employees at that corporation) that modifies the entries in order to hide fraudulent activity from the third-party auditor. Here forward integrity doesn't help, because the corporation, which has access to the offline secrets, can always go back and reconstruct the records. One approach to this problem is to use third-party *time-stamping* (e.g. [21], in which the corporation commits to the entries in the ledger at a specific time in such a way that it is infeasible to modify the entries at a later time without being detected. Indeed, we wish to minimize the opportunity for fraud to occur wherever possible, so that we need only trust that the data was entered into the system correctly, without being concerned about its integrity thereafter.

It is convenient to frame these integrity issues as a three-party model in which a *data owner* first enters data into the system, and then turns that data over to a semi-trusted *service provider* who answers queries from *users* on behalf of the data owner. The service provider is semi-trusted in the sense that it is not expected to maintain the integrity of the data; instead, the integrity of its answers will be verified cryptographically by the users.

This model is closely related to the problem of outsourced databases in which a data owner stores its data at an external service provider that offers sufficient hardware, software, and network bandwidth to maintain the data and answer queries from users on behalf of the data owner. Outsourcing enables fast and fault-tolerant delivery of information. It relieves the data owner of the burden of maintaining the database servers and processing queries.

A substantial amount of research work has been done on how to verify outsourced data and computation [7, 15, 23, 22, 24, 30, 31, 32], including the verification of both correctness and completeness of relational database queries, such as SELECT, PROJECT, and UNION. There is one difference between the existing outsourced database models and the model that we study in this paper. In (most) existing models [24, 31], the client queries his or her *own* data hosted by a service provider (who is not trusted). Our model is more general: the data hosted by the service provider on behalf of the data owner can be queried by arbitrary parties. Under our more general model, the existing data encryption approach (e.g., aggregate queries over homomorphic encrypted data [24], pre-computed and encrypted aggregation [31]) does not apply, because the encrypted data hosted by a service provider can only be queried and decrypted by the data owner herself.

Once the integrity of the data is established, we still must be concerned about the parties to whom the data can be disclosed. For many applications, it is undesirable to disclose specific data elements, for example due to privacy concerns. However, it may be acceptable to disclose aggregate statistics about the data. This is a common problem that occurs in many areas such as censuses, medical research, and educational testing. For example, the static aggregates of confidential medical records of a group of patients can be accessible by the public; however, the medical record of an individual patient should be kept private. Several approaches to this problem have been proposeed, including certain methods for perturbing individual data elements (e.g. [3]), but none of them attempt to simultaneously guarantee the integrity of the underlying data.

1.1 Our contribution

We formalize the model and definitions for the properties integrity and privacy preserving aggregate queries on outsourced databases. We give a general model for querying outsourced data in a three-player setting (data owner, service provider, and user). The data owner delegates to a third-party service provider the task of answering queries from users.

We give protocols for privacy-preserving verification of aggregate queries including SUM, MAX,

MIN, COUNT, AVERAGE, and MEDIAN. The protocols allow a user to verify both correctness and completeness of aggregate results while the individual data values contributing to the results are kept secret from the user. The user interacts with the service provider to obtain aggregate results, and can verify whether or not the service provider returns the correct and complete results.

Our solutions for SUM-related aggregate queries are based on a homomorphic commitment scheme and make use of its linearity property. Our solutions for MIN and MAX queries are based on the proof of knowledge of a greater-than relation of two values. We also use Merkle hash trees for efficient authentication of commitment values. Our algorithms are efficient. Let n be the number of elements in the data set, and m be the number of elements to aggregate. The space complexity for the data owner and the service provider are $O(n)$ (per setup), and $O(m + \log n)$ for the user (per query). The time complexity is $O(n \log n)$ for the data owner (per setup), and $O(m + \log n)$ for both the service provider and the user (per query).

Our cryptographic approach is general enough for the verification of many other types of queries, including non-aggregate database queries [15, 33], historical persistency proofs [20], etc. Non-aggregate queries typically include SELECT, PROJECT, JOIN, SET UNION and INTERSECTION. Our protocol can be applied to prove the correctness and completeness of these queries without revealing unnecessary data entries. For example, for set intersection of set A and B, the verification of correctness of $A \cap B$ does not have to reveal to the user the data that is not in the result. Because the generalization of our aggregate protocols to non-aggregate ones is straightforward, we omit the definitions, protocol descriptions, and proofs of non-aggregate query verification.

In this paper, we do not address the problem of determining whether the queries themselves leak information. For example, it may clearly be a privacy violation if someone asks for an aggregate statistic for a data set that consists of just a single element. Similarly, we don't address the problem of *linkability*, where a user makes multiple queries and is able to infer some privacy sensitive data from the combined data sets. For example, one may make a series of queries that effectively amount to a binary search over some data element.

Outline: The paper is organized as follows. Our model and security definitions are given in §2. We describe our technical toos in §3. The protocols for verification of aggregate queries are presented in §4, in particular, correctness protocols are given in §4.1 and 4.2, and completeness protocol is described in §5. Security and efficiency are analyzed in §6. Related work is described in §7. Suggestions for further research are given in §8. The proof of security is given in the appendix.

2 Model and definitions

In this section, we present our model and security definitions for aggregate query verification, and introduce our approach with an example.

2.1 Trust model

There are three players in our model: the *data owner*, the *service provider*, and the *user*. The data owner is the originator or creator of a database. The data owner delegates to the service provider the ability to answer queries from users. The data owner gives the service provider a copy of the database, along with auxiliary information that enables the verification of query results. The user submits queries to the service provider, and verifies the correctness and the completeness of the results returned by the service provider.

Adapting the trust assumptions of the existing literature on outsourced databases [23, 22, 24, 30, 31, 32], we stipulate in our model that the user trusts the data owner, and in particular trusts any messages signed with respect to the data owner's public key. The user does not need to trust

the service provider, and only relies on responses from the service provider that have been verified as correct.

On the other hand, the service provider is semi-trusted by the data owner. Specifically, the data owner outsources its database to the service provider and trusts the service provider to keep the database secret, not releasing the plaintext data to anyone. But the data owner is *not* required to trust the service provider to answer queries correctly, since users are able to verify the correctness themselves.

In our model, there are three types of adversarial entities: a curious user who wants to infer the individual data entries from the response to an aggregate query, a compromised service provider who may provide untruthful aggregate results, and an adversary who may intercept and tamper with the communications between the user and the service provider. An adversary is allowed to modify query results, for example by inserting or deleting returned items, modifying the aggregate results, and modifying commitments.

2.2 Operations and properties

We assume that the attribute values to be aggregated are numeric values. At setup, the data owner takes as input a security parameter, computes a public-key/private-key pair (PK, SK) for a digital signature system, and public parameters *param*. The data owner keeps SK secret. We define an *aggregate query verification protocol* to have the following operations: **Commit**, **Query**, **Respond**, and **Verify**.

Commit: The data owner takes as input a data set $A = (a_1, \ldots, a_n)$. It generates auxiliary information *aux*, and computes a digital signature Sig, and sends (A, aux, Sig) to the service provider over a secure channel.

In our protocol, *aux* consists of a list of probabilistic commitments to the data items, along with the information necessary for opening the commitments.

Query: The user, who does not know its contents, sends to the service provider an aggregate query Q over a data set A.

Respond: The service provider takes as inputs $(param, A, aux, Sig, Q)$. It computes the aggregate answer *ans*, and prepares the proof *pf* for correctness and completeness. The service provider gives (ans, pf, Sig) to the user.

Verify: The user takes as inputs $(param, ans, pf, Sig)$. It verifies that the answer *ans* satisfies correctness and completeness properties with proofs *pf*, signature Sig, and the public key PK of the data owner (that is obtained from a trusted source).

We naturally define the correctness and completeness of an aggregate query verification protocol by requiring that when **Commit**, **Query**, and **Respond** are correctly computed, *ans* is the correct response to query Q.

For example, for a SUM query on a data set $A = (a_1, \ldots, a_n)$, we require that $ans = \sum_{i=1}^{i=n} a_i$. For a SELECT query on the same set A, for example for elements x satisfying $L < x < R$, we require that $ans = \{a_i \mid L < a_i < R\}$. The algorithms that we describe in this paper can only handle certain types of queries.

We also desire that our aggregate query verification scheme satisfy a *privacy* requirement. Intuitively, the user who receives *ans* as part of the response to query Q should learn no more about the data set A than is implied by (Q, *ans*). We make this precise in our formal definition of security, which is sketched in §6, and given in detail in the Appendix.

All of the algorithms discussed in this paper can be stated in terms of any sort of proofs of integrity that begin by hashing their inputs with a one-way hash function, including both digital signatures and time-stamp certificates. Since precise definitions of the security of time-stamping

schemes are not yet clear in the cryptographic literature (see [21, 11, 10]), we state all our security results only in terms of digital signatures.

2.3 Data structures

In order to illustrate the sorts of queries that we handle and how we are going to approach the problem, in this section we give a simple example.

The data owner (and the service provider) use an expanded table T for storing and maintaining data entries. The table not only stores the plaintext data entries, but also stores their sorting indices and commitments of values. For example, consider a regular database table that has l attributes, such as age, salary, and number of dependents. An expanded table T contains $3l$ columns: $o_1, \ldots, o_l, C(o_1), \ldots, C(o_l), \pi(o_1), \ldots, \pi(o_l)$, where o_i is the plaintext value of attribute i, $C(o_i)$ is the commitment of o_i, and $\pi(o_i)$ is the ordering index of o_i. See Table 1. In more detail:

- Plaintext attribute values $o_1, \ldots, o_l$ are stored in case they are insensitive and can be revealed. If they are sensitive and have to be kept secret from users, then they are redacted using existing digital redaction techniques such as in the sanitizable signature scheme [27, 19].
- Commitments values $C(o_1), \ldots, C(o_l)$ are used for proving the correctness and completeness of aggregate query results.
- Rankings $\pi(o_1), \ldots, \pi(o_l)$ are used for proving completeness, and are obtained by the data owner sorting the data according to each attribute.

Key	*Age*	*Salary*	*Num.*	*C(age)*	*C(sal)*	*C(Num)*	π*(age)*	π*(sal)*	π*(num)*
10001	25	\$65K	0	C(25)	C(65)	C(0)	1	2	1
10002	30	\$50K	2	C(30)	C(50)	C(2)	2	1	3
10003	35	\$70K	1	C(35)	C(70)	C(1)	3	3	2
10004	40	\$80K	3	C(40)	C(80)	C(3)	4	4	4

Table 1: An example of the expanded table maintained by the data owner and the service provider. *Key* can be a serial number of the row. *Num.* represents the number of dependents. $C(i)$ is the commitment of value i. The plaintext data is in columes *Age*, *Salary*, and *Num*, and their rankings are in columns π*(age)*, π*(sal)*, and π*(num)*.

In our proof protocols, the data owner constructs a Merkle hash tree whose leaves consist of the entries of the entire table, including plaintext data, their commitments, and their rankings. Each row of the table corresponds to a subtree whose leaves are cells of the row. An internal node contains the hash value of its child nodes. The root hash of the tree represents the digest of the entire table, and is signed by the data owner.

3 Preliminaries

We describe the basic building blocks that are used to construct our verification protocols, which include a Merkle hash tree, a homomorphic commitment scheme, zero-knowledge proofs of greater-than comparison.

3.1 Merkle hash tree

We use Merkle hash trees for authentication of commitments $C_1, \ldots, C_n$. A binary Merkle hash tree is a tree where an internal node h' is computed as the hash value $H(h_1, h_2)$ of two child nodes h_1 and h_2. In our construction, the order of inputs in the hash function matters and represents the node position in the tree, e.g., h_1 is the left node. The root hash y of the tree represents the

digest of all the values at the leaf nodes, which are commitments $C_1, \ldots, C_n$ in our construction. To authenticate that leaf C_i is in the hash tree, the proof is a sequence of hash values corresponding to the siblings of nodes that are on the path from C_i to the root. To verify the proof, anyone can recompute the root hash with C_i and the sequence of hash values in the proof.

3.2 Homomorphic commitment scheme

Let $\mathbb{G}$ be any group of large prime order q in which the computation of the discrete log is believed to be hard (for example, $q|p-1$ where p is a large prime). Let $g \in \mathbb{G}$ and $h \in \mathbb{G}$ be group elements of order q such that the discrete log $\log_g(h)$ is unknown. Let H denote a cryptographic hash function with domain $[0, q-1]$. A cryptographic commitment is a value that appears random yet binds to a unique input. A commonly-used commitment is the Pedersen commitment [35], where the commitment to x with randomness r is the group element $C_r(x) = g^x h^r$. $C_r(x)$ can be opened or de-committed by revealing r and x to a verifier. This commitment has properties of computationally binding and unconditionally hiding. We may say that a commitment $C_r(x)$ *corresponds* to x, since $C_r(x)$ can only feasibly be opened to value x. The Pedersen commitment has a homomorphic property such that $C_{r_1}(x_1)C_{r_2}(x_2) = C_{r_1+r_2}(x_1 + x_2)$.

3.3 Zero-knowledge proofs of knowledge

A zero-knowledge proof of knowledge allows a prover to demonstrate the knowledge of secret values or their relations (such as $\geq$) without revealing them. For example, a proof of knowledge of a Pedersen committed integer x demonstrates the knowledge of some x and r such that $C_r(x) = g^x h^r$. We use non-interactive proofs of knowledge where the proof is contained in a single set of data that can be verified later without the participation of the data owner or the service provider.

Suppose C_1 and C_2 are commitments of x_1 under random value r_1 and x_2 under random value r_2, respectively. Let $\phi(x_1, x_2)$ be the relation of x_1 and x_2 to be proved. We use the notation $POK(x_1, r_1, x_2, r_2 | C_1 = g^{x_1} h^{r_1}, C_2 = g^{x_2} h^{r_2}, \phi(x_1, x_2))$ to denote a zero-knowledge proof of knowledge of (x_1, r_1) and (x_2, r_2) satisfying all of $C_1 = g^{x_1} h^{r_1}$, $C_2 = g^{x_2} h^{r_2}$, and $\phi(x_1, x_2)$. This notation has been previously used [12]. The bit verification proof can be expressed as $POK(x, r | C = g^x h^r, x \in \{0, 1\})$.

Bit verification proof is used to prove that x is either 0 or 1 without revealing the actual value of x. The proof is a special case of OR proof [13, 38], and contains a five-item tuple (C, r_1, r_2, c_1, c_2), such that $c_1 + c_2 = H(y_1, y_2) \mod q$, where $y_1 = h^{r_1} C^{-c_1}$ and $y_2 = h^{r_2}(C/g)^{-c_2}$. The choice of c_1 and c_2 depends what is to be proved. If $x = 1$, then c_1 is random, otherwise, c_2 is random. We describe the completeness, zero-knowledge, and soundness properties of this protocol in the Appendix. The bit verification proof is used in the zero-knowledge proof of knowledge of greater-than relation of two values, which is described next.

Our protocols need proofs that two committed integers, x_1 and x_2, satisfy an inequality such as $x_1 \geq x_2$ [9, 13, 16, 28]. One approach is to show that $x_1 - x_2 \geq 0$. We review the greater-than proof by Durfee and Franklin that is based on the bit commitments of the difference $x_1 - x_2$, following their descriptions [16]: $POK(x_1, r_1, x_2, r_2 | C_1 = g^{x_1} h^{r_1}, C_2 = g^{x_2} h^{r_2}, x_1 - x_2 \geq 0)$.

The prover can compute the commitment $C' = C_{r_1 - r_2}(x_1 - x_2)$, and the verifier can compute this as $C' = C_1 / C_2$. Let $(\gamma_i)_{i=0}^{t-1}$ be the binary representation of $x_1 - x_2$, i.e., $x_1 - x_2 = \sum_{i=0}^{t-1} 2^i \gamma_i$, where t is the bit length of the difference. Choose random values $s_1, \ldots, s_{t-1}$ and set $s_0 = r_1 - r_2 - \sum_{i=1}^{t-1} 2^i s_i$. Let $\alpha_i = C_{\gamma_i}(s_i) = g^{\gamma_i} h^{s_i}$ for all $i \in [0, t-1]$. Suppose the verifier knows that the bound $x_1, x_2 \in [0, 2^t)$ holds. The prover provides the commitments $\alpha_0, \ldots, \alpha_{t-1}$ along with a proof that each γ_i is a bit (either 0 or 1). The verifier checks the proof that each bit committed by α_i is either 0 or 1 and confirms that equation $C_2 \prod \alpha_i^{2^i} = C_1$ holds. Suppose the verifier does not know the bound of x_1 or x_2. Then, a zero-knowledge proof that $x_1 \in [0, 2^t)$ and $x_2 \in [0, 2^t)$ can be constructed in a

similar fashion. In our protocol, we assume the bounds of x_1 and x_2 are known by the verifier (or the user), which is usually the case for most database entries such as zip code, salary, age, etc.

A special case of greater-than proof called interval proof. An interval proof proves that a committed integer satisfies an inequality such as $x \geq A$ or $y \leq B$, where A and B are constants: $POK(x_1, r_1 | C_1 = g^{x_1} h^{r_1}, x_1 - A \geq 0)$, $POK(x_2, r_2 | C_2 = g^{x_2} h^{r_2}, B - x_2 \geq 0)$.

4 Verification protocols for aggregate queries

We present our verification protocols for sum, max/min, count, and average queries. Our protocols can be generalized to answer combined aggregate queries, aggregate query with selection clause, generalized sum queries such as linear combination, generalized max queries such as median and top k-th. In this section, to clarify our explainations, we use a simple table with one attribute and no plaintext data. These building blocks can be easily expanded to include the general form of data structure as in §2.3.

We use the cryptographic tools described in §3. For each query-type, we present four operations: **Commit**, **Query**, **Respond**, and **Verify**. We present protocols for correctness verification first, and then give our solution for verification of completeness. The following protocols are run by the data owner, service provider, and the user to answer aggregate queries and verify the correctness of results. In the protocols described in S4.1 and §4.2, we assume that the query is over the complete set of unsorted data $(a_1, \ldots, a_n)$. To handle tables with multiple attributes, we generalize our protocol in §5.1.

General Settings: The data owner chooses a public/private key-pair (PK, SK) in a secure digital signature scheme. The data owner chooses a group G of order q, and two elements g and h in G such that the discrete log $\log_g(h)$ is unknown. The public parameters $param = (PK, C, H, S, POK)$, where C is a commitment scheme, H is a hash function, S is the signature scheme, and POK is a zero-knowledge proof protocol of greater-than.

Denote the set A of data by $(a_1, \ldots, a_n)$. Denote the Pedersen commitment of data a_i with random value r_i by C_i (see §3). The data owner has an unsorted set A of data $(a_1, \ldots, a_n)$. The data (in plaintext) is given to the service provider along with auxiliary information including commitments and a signature on the digest of commitments. The service provider answers aggregate queries on behalf of the data owner without revealing the data $(a_1, \ldots, a_n)$. Yet, the user is able to verify the result.

4.1 SUM queries

For sum query, the user obtains the sum s of set A from the service provider, and verifies the correctness of s with respect to the commitments $(C_1, \ldots, C_n)$ of the data along with data owner's signature on the root hash of commitments. The details are as follows.

Commit: The data owner with public/private key pair (PK, SK) and data $(a_1, \ldots, a_n)$ commits and signs the data as follows. Choose n random values $(r_1, \ldots, r_n)$. Compute the Pedersen commitment C_i of a_i with r_i as $C_i = g^{a_i} h^{r_i}$. Construct a Merkle hash tree with commitments $C_1, \ldots, C_n$ as leaf nodes of the tree, and denote the root hash of the tree by h_r. Sign the root hash h_r with the private key SK of the data owner, which gives a signature Sig. Send the following information to the service provider in a secure channel: $\{(a_1, \ldots, a_n), (r_1, \ldots, r_n), (C_1, \ldots, C_n), Sig)\}$. The random value r_is are for the service provider to open commitments of the sum (see operation **Respond**).

Query: The user queries for the sum of the data set A.

Respond: The service provider obtains from the data owner the following information: $\{(a_1, \ldots, a_n), (r_1, \ldots, r_n), (C_1, \ldots, C_n), Sig)\}$. It prepares the sum and its proofs as follows. Com-

pute the sum of data $s = \sum_{i=1}^{n} a_i$. Compute the sum of random values $r' = \sum_{i=1}^{n} r_i$. Send the following information to the user: $\{s, r', (C_1, \ldots, C_n), Sig\}$.

Verify: The user receives $\{s, r', (C_1, \ldots, C_n), Sig\}$ from the service provider. The user verifies the correctness of sum s as follows. Confirm the following equation holds: $g^s h^{r'} = \prod_{i=1}^{n} C_i$. Construct a Merkle hash tree with $(C_1, \ldots, C_n)$ as leaf nodes. Compute the root hash h_r and verify signature Sig of h_r with the public key PK of the data owner. We assume that the user obtains PK through a regular public key certificate process. Sum s is accepted if all verifications are successful, rejected otherwise.

The security and efficiency of the protocol in analyzed in §6. Our above protocol can be generalized to a query for any linear combination of sum without revealing the data values themselves. For example, a user can query for the sum of $3a_1 + 5a_2 + 12a_3 + \ldots$, which can be easily computed by the service provider who has a_i values. Our protocol can be easily modified to allow verification of the sum in a privacy-preserving fashion.

The verification protocols of count and average queries can be built based on the sum query. We consider aggregation over the entire set A. For verification of count result n, the user simply counts the number of commitments $(C_1, \ldots, C_n)$, and confirms that it is n. The user constructs the Merkle hash tree of commitments C_is and verifies the signature Sig of the root hash with the public key of the data owner as in sum protocol. The protocol for average query can be built by combining the sum and count verifications, and is not repeated here.

4.2 MAX/MIN queries

For a sorted list, the max/min query can be easily solved as follows. The data owner sorts and signs the root hash of the Merkle tree. The service provider returns the max or min element, and proves that the element is the last or the first element of the sorted list. The user trusts the data owner for sorting the list, and therefore the verification of correctness is equivalent to verifying the position of the result in the list.

We focus on how to verify the correctness of max/min query for unsorted data. We present our correctness verification protocol for max query, which can be easily modified to answer min query. The proofs generated by the service provider for max query are more complex than for sum query. We use the zero-knowledge proof of greater-than described in Section 3 for the proof of comparison result.

Commit: Same as in sum protocol.

Query: The user queries for the maximum element of set A.

Respond: The service provider computes the maximum element a_j of data set A which contains $(a_1, \ldots, a_n)$. For each data value $a_i \in A$ and $i \neq j$, prepare the zero-knowledge proof p_i for $a_j \geq a_i$:

$$p_i = POK(a_i, r_i, a_j, r_j | C_i = g^{a_i} h^{r_i}, C_j = g^{a_j} h^{r_j}, a_j - a_i \geq 0)$$

The service provider gives the following information to the user: $\{a_j, r_j, (C_1, \ldots, C_n), (p_1, \ldots, p_{j-1}, p_{j+1}, \ldots, p_n), Sig\}$. Note that all the data in A except the max is not revealed to the user.

Verify: The user obtains the following information from the service provider: $\{a_j, r_j, (C_1, \ldots, C_n), (p_1, \ldots, p_{j-1}, p_{j+1}, \ldots, p_n), Sig\}$, where p_i is the zero-knowledge proof of $a_j \geq a_i$. The user does: Open commitment C_j with a_j and r_j. Construct a Merkle hash tree with $(C_1, \ldots, C_n)$ as leaf nodes. Compute the root hash h_r and verify signature Sig of h_r with the public key PK of the data owner. We assume that the user obtains PK through a regular public key certificate process. Sum s is accepted if all verifications are successful, rejected otherwise.

4.3 Median or top k-th queries

A query for the median of a set of elements can be answered in a similar fashion as the max/min query. The main difference is the requirements on the comparison results. Namely, in **Verify**, the user verifies zero-knowledge proofs $\{p_i\}$ to ensure that there are $n/2$ number of items greater-than the median and $n/2$ number of items less-than the median. Our solution also applies to a more general top k-query, which is to return the top k-th element of a set. Top k-queries are important constructs in many data mining applications. To answer it, in **Verify**, the user verifies in a zero-knowledge fashion that there are $k-1$ items greater-than the result and the rest of items are less-than the result.

The above protocols assume that the queries are over the complete set of data. We generalize our solutions to handle tables with multiple attributes in §5.1.

4.4 Nested aggregate queries

Our correctness protocols can be composed and generalized to verify more complex aggregate queries, namely, nested aggregate queries. Nested aggregate queries are an important and expressive type of query in database systems. For example, a query asks for the max of the counts of numbers of cancer patients per year. The yearly cancer patient numbers are first counted, and then the maximum is found. Or, for example, a query asks for the max of the sums of revenues per quarters. The quarterly revenues are first summed up, and then the maximum is computed. Our previously presented protocols can be composed with an arbitrary depth to support nested aggregate queries. The integrity verification hides not only individual data entries but also intermediate values in nested aggregate queries.

To give a concrete example of protocol for nested aggregate query, we choose to present the verification protocol for max-sum nested pair next. There are m data sets: $A_1, \ldots, A_m$. The user wants the maximum sum of individual sets.

Commit: The data owner commits and signs elements in each set of $A_1, \ldots, A_m$, similar to the sum protocol. The data owner computes commitments of each data value in all sets, and signs the root hash of Merkle tree built over the commitments. Let $\{C\}$ represent all the commitments, $C_{i,k}$ be the commitment of k-th element in set A_i, and Sig be the signature.

Query: The user queries for the maximum number of the sums of individual set $A_1, \ldots, A_m$.

Respond: The service provider does: For each set A_i ($i \in [1, m]$), compute the sum s_i. Compute the commitment D_i of s_i by multiplying commitments of A_i's data in $\{C\}$: $D_i = \prod_{k=1}^{|A_i|} C_{i,k}$. Note that the service provider also has the random values to open D_is. Compute the maximum number of all sums, which is denoted by s_j ($j \in [1, m]$). For each sum s_i, prepare the zero-knowledge proof p_i for $s_j \geq s_i$:

$$p_i = POK(s_i, t_i, s_j, t_j | D_i = g^{s_i} h^{t_i}, D_j = g^{s_j} h^{t_j}, s_j - s_i \geq 0)$$

The above proofs are for the maximum computation. The service provider also needs to show proofs for summation computation using commitments $\{C\}$, and the authenticity of commitments $\{C\}$ is proved with signature Sig. The service provider gives the following information to the user: $\{s_j, \{C\}, (p_1, \ldots, p_{j-1}, p_{j+1}, \ldots, p_m), Sig\}$. Note that the intermediate sums are not revealed except the max.

Verify: The user obtains the following information from the service provider: $\{s_j, \{C\}, (p_1, \ldots, p_{j-1}, p_{j+1}, \ldots, p_m), Sig\}$, where p_i is the zero-knowledge proof of $s_j \geq s_i$. The user constructs a Merkle hash tree with $\{C\}$ as the leaf nodes. The user then computes the root hash and verifies signature Sig with the public key PK of the data owner. We assume that the user

obtains PK through a regular public key certificate process. She then computes the commitment D_i of intermediate sum for all $i \in [1, m]$ as $D_i = \prod_{k=1}^{|A_i|} C_{i,k}$. Finally, the user verifies p_i for all $i \in [1, m]$ and $i \neq j$ using commitments D_i. Max s_j is accepted if all verifications are successful, rejected otherwise.

5 Verification of completeness

The definition of the completeness of aggregate queries is directly based on the completeness of selection queries. The basic building block is the existing proof-of-knowledge protocol for proving greater-than relation of two values. One requirement of our solution is that the attributes used for selection need to be sorted by the data owner. For a relational database table, indices can be built for arbitrary attributes. For a table that has multiple attributes, the attribute used for selection can be different from the attribute for aggregation, for example, average blood pressure for patients older than 55. In this example, we require the table to be sorted under attribute age, but not under attribute blood pressure.

Our description of completeness verification proof requires a generalization of **Commit** operation in the previous section to support multiple attributes.

5.1 Support of multiple attributes

To support flexible aggregate with selection queries, we generalize our **Commit**, **Respond**, and **Verify** operations in the previous section to handle tables with multiple attributes. The main addition to **Commit** operation is that for a data entry with multiple attributes, each attribute value is committed and the hash value of concatenated commitments is used to build a Merkle hash tree. The commitments are also required for verification in **Verify** operation.

Let $(T_1, \ldots, T_l)$ be the attributes of a database table, and l is the number of attributes. Denote the value of attribute T_i by t_i (for $i \in [1, l]$). We assume that all the attributes are sensitive and cannot be revealed to users. If certain attributes are insensitive (and the problem becomes simpler), then the attribute values rather than their commitments are computed in the hash value.

In **Commit**, for each database table entry, the data owner commits to attribute value t_i for all $i \in [1, l]$, by computing $C_i = g^{t_i} h^{r_i}$, where r_i is chosen at random. The data owner computes the hash value of concatenated commitments: $h = H(C_1, \ldots, C_l)$, and constructs Merkle hash tree with all the hash values as leaf nodes. As before, commitments, database tables, random values, and the signature are given to the service provider.

Suppose a user submits an aggregate query for attribute T_1. In response, the service provider prepares correctness proofs for attribute T_1 as in previous protocols. The service provider also gives commitments $C_1, \ldots, C_l$ of all attributes $T_1, \ldots, T_l$ for each entry and proofs to the user. The user reconstructs the hash root and verifies the correctness of query result.

5.2 Completeness verification protocol

We present a completeness proof for aggregate queries with selection. Consider a table with l attributes $T_1, \ldots, T_l$. Our presentation of the protocol uses a range query $[a, b]$ for an attribute T_j, and aggregation is over attribute T_i. We augment the operations to support proof of completeness.

For simplicity, we assume that data entries on the Merkle hash tree are sorted under attribute T_j. That is, the left most entry on the tree has the smallest T_j value, and so on. Our protocol can be modified to allow arbitrary orderings without revealing unnecessary ranking information, which is discussed at the end of this section.

Commit: The data owner sorts data entries from small to large, based on one or more attributes that are used for selection, computes commitments as described in §5.1, constructs Merkle hash tree, and signs the root hash. Let the signature be Sig.

Query: Without loss of generality, let the user's query be an aggregation of attribute T_i ($i \in [1, l]$) over the selection over attribute T_j ($j \in [1, l]$) whose values lie between $[y_1, y_2]$.

Respond: To construct the proof for completeness, the service provider Selects the entries that lie in the selection range, which are denoted by A. Then computes the required aggregate (such as sum, max, etc) of attribute T_1.

- If the selected entries has two immediate neighboring entries, then the service provider constructs a zero-knowledge proof that the two entries are beyond the selection range $[y_1, y_2]$. Denote the zero-knowledge proofs as p_{left} and p_{right}. The ZK proof p_{left} shows that the entry immediately to the left [1] of the set of selected entries A has a T_j attribute value v_{left} smaller than y_1, i.e., $v_{left} < y_1$. The ZK proof p_{right} shows that the entry immediately to the right of the set of selected entries A has a T_j attribute value v_{right} larger than y_2, i.e., $v_{right} > y_2$. The proofs show that all of the entries satisfying the range are selected.
  Let r_{left} and r_{right} be the random values used by the data owner to compute the commitments of v_{left} and v_{right} in **Commit**, respectively. The proofs p_{left} and p_{right} are expressed below.

$$\begin{aligned} p_{left} &= POK(v_{left}, r_{left}, |C_{left} = g^{v_{left}} h^{r_{left}}, y_1 - v_{left} > 0) \\ p_{right} &= POK(v_{right}, r_{right}, |C_{right} = g^{v_{right}} h^{r_{right}}, v_{right} - y_2 > 0) \end{aligned}$$

  The service provider gives the following information to the user: commitments of selected entries denoted by C_A, commitments of neighbors C_{left} and C_{right}, proofs p_{left} and p_{right}, data owner's signature Sig, and companion hashes. Recall companion hashes are hash values at the roots of disjoint subtrees of the Merkle hash tree all of whose leaves correspond to commitments of unseleted data entries (i.e., not in set A).
- If the selected entries has one immediate neighboring entry, then the service provider constructs a zero-knowledge proof that the entry is beyond the selection range $[y_1, y_2]$ as above, i.e., either $v_{left} < y_1$ or $v_{right} > y_2$.
- If no element is out of the selection range, i.e., all entries are selected, the Merkle tree contruction implicitly proves the completeness. Hence, the service provider returns all the commitments and signature Sig.

Verify: The user verifies the completeness of results by computing the root hash of Merkle hash tree with commitments C_A, C_{left}, C_{right}, and companion hashes, verifying the signature Sig on the root hash with data owner's public key, and verifying the proofs p_{left} and p_{right}. The query is accepted if all verifications are successful, rejected otherwise.

The above protocol uses a range as the selection clause. For just $\geq$ or $\leq$ predicates, a simplied version of our protocol suffices, as the proof of only one neighbor is needed. For an equality predicate, a completeness ZK proof can be prepared in a similar fashion, showing immediate neighbors of selected entries are either larger or smaller than the predicate. Similarly, ZK proofs for inequality predicates can be computed. The solution presented supports the selection of one attribute. For more complex selections of multiple attributes, for example, age ≥ 30 and height ≥ 6', a multi-dimensional range tree [33] has to be constructed by the data owner.

5.3 Generalizing the Completeness Proof

Above we assumed that entries on the hash tree are sorted under attribute T_j, which is also used for the selection. In order to support general completeness proof, the data owner also sorts the

[1] Sorting in **Commit** is from small to large.

data under each attribute, and the indices or rankings are stored as part of the table, as shown, for example, in Table 1 in §2.3. To further hide the rankings, the rankings can also be committed and then folded into the hash tree, as described above. We demonstrate this using an example as follows.

Consider Table 1, instead of using ranking 3 for entry \$70K for attribute $\pi(sal)$ in the Merkle hash tree, the data owner computes a randomized commitment of 3, denoted by $C_\pi = g^3 h^r$ for some random r. Similarly, for entry \$65K, let $C'_\pi = g^2 h^{r'}$ be a commitment of ranking 2, for some random r'.

Suppose that the selection criteria of a user's query is for salary greater-than \$67K. This selects \$70K entry, but not \$65K entry. To prove that the selection is complete, the service provider shows that **(1)** \$65K $<$ \$67K, **(2)** ranking 3 is higher than ranking 2 by 1, and **(3)** \$65K has ranking 2 and \$70K has ranking 3. Requirement **(3)** is proved implicitly because in Merkle hash tree commitments of \$65K and ranking 2 are grouped and hashed together and similar for \$70K and ranking 3. Requirement **(1)** can be proved with zero-knowledge proofs without revealing \$65K. Finally, requirement **(2)** can be proved by showing that C_π / C'_π is a commitment of 1: $C_\pi / C'_\pi = g^{3-2} h^{r-r'} = g h^{r-r'}$. The user is given $r - r'$ to open the commitment of 1. Due to space limit, we omit the formal description of this generalized protocol in this version.

6 Security and efficiency

In this section, we analyze the adversarial model and prove the security of our protocols. We also give the complexity analysis of our verification protocols. We give formal security definitions in a random oracle model, using a game-based definition generalizing the one used to define the semantic security of an encryption scheme [17]. We allow an attacker to issue *commit queries* and *sign queries*, i.e., queries for commitments and signatures of data sets, respectively, and *aggregate queries*, i.e., queries for aggregate results and their proofs. Also, we allow the adversary to choose the data set on which it wishes to be challenged. Notice that an adversary may *choose* its targets adaptively. An adversary that chooses its targets adaptively first makes queries, and then chooses its targets based on the results of these queries. At the end of the query phase, the adversary outputs a guess aiming to break the correctness verification. In the statement of the following theorem, we define a protocol to be *secure* if no feasible adversary, issuing a polynomial number of queries in the game we define, achieves a non-negligible advantage in the game.

Theorem 1 *The verification protocols of aggregate query results on outsourced databases are secure.*

The proof of Theorem 1 is given in the appendix using security reduction. Our proof strategy is outlined here. Suppose there is an adaptive adversary Adv_a that has combined advantage ϵ against the verification protocol targeting one or more correctness, completeness, and privacy properties, and that makes a polynomial number of commit, sign, and aggregate queries. If the hash functions, H, used in answering sign queries is a random oracle, then there is an algorithm that breaks one or more of a collision-free hash function, a secure commitment scheme with binding and hiding properties, and a signature scheme secure against existential forgery with combined advantage $O(\epsilon)$ and running time $O(\text{time}(Adv_a))$.

6.1 Efficiency

We analyze the complexities of operations in our verification protocols, which is summarized in Table 2.

Our verification algorithms have cost linear in the number of data elements selected by a query. This is to be expected, given our approach, since in essence our procedures verify each of these data

	Commit	**Respond**	**Verify**	**Update**	**Storage**
Data owner	$O(n \log n)$	–	–	$O(k \log n)$	$O(n)$
Service provider	–	$O(m + \log n)$	–	$O(k \log n)$	$O(n)$
User	–	–	$O(m + \log n)$	–	$O(m + \log n)$

Table 2: Time and space complexities of the protocol. We consider the verification of both correctness and completeness. The analysis is independent of the specific type of aggregate query. n is the size of all data, and m is the size of data selected for query. k in **Update** is the number of data elements updated.

elements' contribution to the correct response. We leave to future work the problem of breaking this apparent lower bound.

Windowed aggregate: In certain applications (e.g., streaming databases, continuous queries [1]), the aggregate queries are pre-defined by users and known by the data owner in advance. The data comes from a dynamic data stream (e.g., stock quotes). The data owner can pre-compute, commit, and sign intermediate computation results, e.g., local max/min and sum. The intermediate values replace the original data and serve as basic data elements. Therefore, the computation leading to intermediate values does not need to be verified by the user, which saves communication and computation costs. Note that the verification time is still linear in the size of inputs, which are intermediate values.

7 Related work

With the increasing development of IT outsourcing, a substantial amount of research work has been done on how to verify outsourced data and computation [7, 15, 23, 22, 24, 30, 31, 32], including the verification of both correctness and completeness of relational database queries. Existing literatures on database query verification have focused on non-aggregate queries such as select, project, join, set union, and set intersect. Merkle hash trees have been used extensively for authentication of data elements [29]. Aggregate signatures are another approach for data authentication, where each data tuple is signed by the data owner [32]. Most recently, the privacy issue in verifying non-aggregate queries was first addressed by in [34], which gave an elegant solution using hashing for proving the completeness of selection queries without revealing neighboring entries. We provide an alternative solution for the privacy issue in completeness proof by utilizing zero-knowledge proofs and a commitment scheme.

The aggregate query verification problem has been studied in database-as-a-service (DAS) model [24, 31]. The DAS model [23, 31], is an instantiation of the computing model involving trusted clients, who store their data at an untrusted server that are administrated by the service provider. The challenge is to make it impossible for the system provider to correctly interpret the data. The data is owned by clients. The clients only have limited computational power and storage, and they rely on the server for the mass computational power and storage. The server exposes mechanisms for the clients to create and manage the client databases at the server. Data originates from the client. The recent paper by Hacigümüs, Iyer, and Mehrotra [24] addresses the execution of aggregate queries over encrypted data using homomorphic encryption scheme. Mykletun and Tsudik [31] proposed an alternative approach where the data owner pre-computes and encrypts the aggregate results and stores them in the service provider. This approach avoids the use of homomorphic encryption, which was found to have a security flaw when used for DAS [31]. The correctness and completeness definitions do not apply to these models as the user is also the data owner in DAS. Our model is different from DAS, and is suitable for a more general security

	Ours	NT [32]	DGMS [15]	PJRT [34]	HIM [24]/MT [31]
Aggregate Q.	Yes	No	No	No	Yes
Non-aggregate Q.	Yes	Yes	Yes	Yes	No
Correctness	Yes	N/A	N/A	N/A	N/A
Completeness	Yes	Yes	Yes	Yes	N/A
Authenticity	Yes	Yes	Yes	Yes	Yes
Privacy	Yes	No	No	Yes	Yes
Data Structure	Tree-based	Signature chain	Tree-based	Tree-based	N/A

Table 3: Comparisons of functionalities of our verification protocols with some of the existing approaches developed for outsourced systems.

setting, as the data does not have to be originated from the client. We compare major features of our work with existing solutions in Table 3.

Hohenberger and Lysyanskaya were the first to give formal security definitions for outsourced computation, and probabilistic solutions for checking failures in outsourced exponentiation and the Cramer-Shoup cryptosystem [26]. Their model has two parties: the data owner and the untrusted service provider. Our work studies a three-party model where the client who queries the service provider may not the same as the data owner.

Searchable symmetric-key encryption schemes for private-key storage outsourcing have been previously studied (e.g., [2, 39]). Most recently, improved security definitions and constructions are proposed by Curtmola, Garay, Kamara, and Ostrovsky [14]. Public-key systems have also been used to construct searchable encryption schemes [2, 8, 41], including a practical searchable and encrypted audit log system [41]. In general, symmetric key encryption is more efficient than public-key encryption. In the meantime, the symmetric key encryption typically requires live key updates, which incur communication costs. Our authentication protocol differs from the above work in that it focuses on the validation of query results, and supports data aggregate besides search (i.e., equality and comparison-based selection).

In data-mining literature [3, 37, 40, 42], an important approach to protect data privacy is to modify database tables such that an individual entry enjoys certain degree of anonymity. Our solutions differ from existing efforts in that we support authenticated ad hoc data analysis without releasing the microdata to the public. Because the aggregate is computed over exact data instead of generalized data, there is no loss of data accuracy in the aggregate results.

8 Future work

One interesting future direction is to develop unlinkable and verifiable data aggregation. The linkage problem occurs when a prover (database holder) answers several *different* queries from the verifier and returns the *same* set of commitments. Then, there is a possible leakage of information. For example, if one query asks how many people live in Springfield and another query asks how many are over forty years old, then by viewing the returned commitments the verifier could determine how many people over forty there are in Springfield, i.e. an information leakage occurred. One way to prevent this is to change commitments over time; for example, a prover could randomize commitments. This procedure should not require interaction from the prover with the proof preparer. In the meantime, a user should still be able to verify the randomized commitments are generated from authentic data.

The lower-bound given in Section 6 is proved based on input commitments whose size is linear in m, where m is the size of data selected for query. A possible future direction is to study whether

pre-computations by the data owner can reduce the input size of the **Verify** algorithm. The data owner might use auxiliary data structure (preferrably with a linear or polynomial complexity in the total size of data n). The auxiliary data structure is signed and given to the service provider, so that the user can verify the correctness of *any* aggregate query result with a number of inputs sublinear in m.

References

[1] D. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: A New Model and Architecture for Data Stream Management. *VLDB Journal Special Issue on Best Papers of VLDB 2002*, 12(2), 2003.

[2] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. M. Lee, G. Neven, P. Paillier, and H. Shi. Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions. In *In CRYPTO 2005*, volume 3621 of *LNCS*, pages 205–222. Springer, 2005.

[3] Rakesh Agrawal and Ramakrishnan Srikant. Privacy-preserving data mining. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, May 2000.

[4] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security (CCS)*, pages 62 – 73. ACM Press, 1993.

[5] M. Bellare and B. Yee. Forward integrity for secure audit logs. Technical report, University of California, San Diego, November 1997. Available at `ftp://www.cs.ucsd.edu/pub/bsy/pub/fi.ps`.

[6] M. Bellare and B. Yee. Forward security in private-key cryptography. In *CT-RSA*, volume 2612 of *LNCS*, pages 1–18. Springer-Verlag, 2003.

[7] Elisa Bertino, Beng Chin Ooi, Yanjiang Yang, and Robert H. Deng. Privacy and ownership preserving of outsourced medical data. In *Proceedings of the 21st International Conference on Data Engineering (ICDE)*, pages 521 – 532, 2005.

[8] D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. `http://crypto.stanford.edu/~dabo/papers/search.pdf`.

[9] F. Boudot. Efficient proofs that a committed number lies in an interval. In *Advances in Cryptology - EuroCrypt '00*, volume 1807 of *Lecture Notes in Computer Science*, pages 431 – 444. Springer-Verlag, 2000.

[10] A. Buldas and M. Saarepera. Do broken hash functions affect the security of time-stamping schemes? In *4th Internatioanl Conf. on Applied Cryptography and Network Security – ACNS '06*, volume 3989 of *Lecture Notes in Computer Science*, pages 50–65.

[11] A. Buldas and M. Saarepera. On provably secure time-stamping schemes. In *Advances in Cryptology — ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 500–514, October 2004.

[12] Jan Camenisch and Markus Michels. In *Advances in Cryptology - EUROCRYPT '99*, volume 1592 of *LNCS*, pages 106–121. Springer Verlag, 1999.

[13] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology - CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 174 – 187. Springer-Verlag, 1994.

[14] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In *Proceedings of the 13st ACM*

Conference on Computer and Communications Security (CCS), 2006.

[15] P. Devanbu, M. Gertz, C. Martel, and S. Stubblebine. Authentic third-party data publication. *Journal of Computer Security*, 11(3), 2003.

[16] Glenn Durfee and Matt Franklin. Distribution chain security. In *Proceedings of the 7th ACM Conference on Computer and Communications Security (CCS)*, pages 63–70, New York, NY, USA, 2000. ACM Press.

[17] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, April 1984.

[18] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptively chosen message attacks. *SIAM Journal on Computing*, 7(2):281–308, 1988.

[19] S. Haber, W. Horne, and T. Sander. Audit-log integrity using redactable signatures. Technical Report HPL-2006-64, 2006. HP Labs Technical Report.

[20] S. Haber and P. Kamat. A content integrity service for long-term digital archives. In *Proceedings of Archiving 2006*. Society for Imaging Science and Technology, 2006. Available at `http://www.hpl.hp.com/techreports/2006/HPL-2006-54.html`.

[21] S. Haber and W.S. Stornetta. Secure names for bit-strings. In *Proceedings of the 4th ACM Conference on Computer and Communication Security*, pages 28–35. ACM Press, April 1997.

[22] H. Hacigümüs, B. Iyer, C. Li, and S. Mehrotra. Executing SQL over encrypted data in the database-service provider model. In *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 216 – 227. ACM Press, June 2002.

[23] H. Hacigümüs, B. Iyer, and S. Mehrotra. Providing database as a service. In *Proceedings of International Conference on Data Engineering (ICDE)*, March 2002.

[24] H. Hacigümüs, B. Iyer, and S. Mehrotra. Efficient execution of aggregation queries over encrypted databases. In *Proceedings of International Conference on Database Systems for Advanced Applications (DASFAA)*, 2004.

[25] S. Halevi and S. Micali. Practical and provably-secure commitment schemes from collision-free hashing. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 201–215. Springer-Verlag, 1996.

[26] Susan Hohenberger and Anna Lysyanskaya. How to securely outsource cryptographic computations. In *Proceedings of the Second Theory of Cryptography Conference (TCC '05)*, pages 264–282, 2005.

[27] R. Johnson, D. Molnar, D. Song, and D. Wagner. Homomorphic signature schemes. In *Proceedings of the RSA Security Conference Cryptographers Track*, volume 2271 of *Lecture Notes in Computer Science*. Springer-Verlag, February 2002. Available at `http://www.ece.cmu.edu/~dawnsong/papers/hom-rsa02.pdf`.

[28] Wenbo Mao. Guaranteed correct sharing of integer factorization with off-line shareholders. In *Proceedings of the First International Workshop on Practice and Theory in Public Key Cryptograph (PKC '98)*, volume 1431 of *Lecture Notes in Computer Science*, pages 60–71. Springer, 1998.

[29] R. Merkle. Protocols for public key cryptosystems. In *Proceedings of the 1980 Symposium on Security and Privacy*, pages 122–133. IEEE Computer Society Press, 1980.

[30] E. Mykletun, M. Narasimha, and G. Tsudik. Authentication and integrity in outsourced databases. In *Proceedings of Symposium on Network and Distributed Systems Security (NDSS)*, February 2004.

[31] E. Mykletun and G. Tsudik. Aggregation queries in the database-as-a-service model. In *IFIP WG 11.3 Working Conference on Data and Applications Security (DBSec)*, July 2006.

[32] M. Narasimha and G. Tsudik. Authentication of outsourced databases using signature aggregation and chaining. In *International Conference on Database Systems for Advanced Applications (DASFAA)*, April 2006.

[33] Glen Nuckolls, Charles U. Martel, and Stuart G. Stubblebine. Certifying data from multiple sources. In *Data and Applications Security XVII: Status and Prospects, IFIP TC-11 WG 11.3 Seventeenth Annual Working Conference on Data and Application Security*, pages 47–60, 2003.

[34] HweeHwa Pang, Arpit Jain, Krithi Ramamritham, and Kian-Lee Tan. Verifying completeness of relational query results in data publishing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 407–418, 2005.

[35] T. P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract). In *Advances in Cryptology - EuroCrypt '91*, volume 547 of *Lecture Notes in Computer Science*, pages 522 – 526. Springer-Verlag, 1991.

[36] D. Pointcheval and J. Stern. Security proofs for signature schemes. In Ueli Maurer, editor, *Advances in cryptology – EuroCrypt '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 387 – 398. Springer-Verlag, 1996.

[37] Periangela Samarati. Protecting respondent's privacy in microdata release. *IEEE Transactions on Knowledge and Data Engineering*, 13(6):1010 – 1027, 2001.

[38] Claus P. Schnorr. Efficient identification and signatures for smart cards. In *Proceedings of Advances in cryptology – CRYPTO '89*, pages 239 – 252. Springer-Verlag, 1989.

[39] D. Song, D. Wagner, and A. Perrig. Practical techniques for searching on encrypted data. In *In Proceedings of 2000 IEEE Symposium on Security and Privacy*, pages 44 – 55, May 2000.

[40] Latanya Sweeney. k-Anonymity, a model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):557 – 570, 2002.

[41] Brent R. Waters, Dirk Balfanz, Glenn Durfee, and Diana K. Smetters. Building an encrypted and searchable audit log. In *Proceedings of Symposium on Network and Distributed Systems Security (NDSS '04)*, 2004.

[42] Xiaokui Xiao and Yufei Tao. Anatomy: Simple and effective privacy preservation. In *Proceedings of the 32nd Very Large Data Bases (VLDB)*, 2006.

A Zero-knowledge OR proof properties

For the completeness of our protocols, we review security properties of OR proof, which is the basis for bit verification proof used in our max/min proofs. The properties include completeness, zero-knowledge, and soundness. The non-interactive version of OR proof is called Sigma protocol [13], using the Fiat-Shamir transformation to replace the random challenge with a hash function H. The security of the non-interactive proof is in the random oracle model [4].

Completeness: Intuitively, completeness means that honest prover who knows the secret convinces the verifier with overwhelming probability. For commitments $C_r(0)$ and $C_r(1)$, such a proof can be efficiently computed as follows. If $x = 1$, thus commitment $C = g^1 h^r$, then let r_1, c_1, u_2 be chosen by the prover at random. The prover fakes (simulates) a proof for $x = 0$. Let $a_1 = h^{r_1} C^{-c_1} \mod p$, $a_2 = h^{u_2} \mod p$, $c = H(a_1, a_2)$, $c_2 = c - c_1$, and $r_2 = u_2 + c_2 r \mod q$. In the case where $x = 0$, thus commitment $C = g^0 h^r$, then let r_2, c_2, u_1 be chosen by the prover at random. The prover fakes a proof for $x = 1$. Let $a_2 = h^{r_2}(C/g)^{-c_2} \mod p$, $a_1 = h^{u_1} \mod p, c = H(a_1, a_2)$, $c_1 = c - c_2$, and $r_1 = u_1 + c_1 r \mod q$.

Soundness (being a proof of knowledge): Intuitively, soundness means that no one who does not know the secret can convince the verifier with non-negligible probability. Formally, this property shows that two acceptable protocol interactions $\{a_1, a_2, c, r_1, r_2, c_1, c_2\}$ and $\{a_1, a_2, c', r'_1, r'_2, c'_1, c'_2\}$ for a fixed commitment C with different challenges $\{c_1, c_2\} \neq \{c'_1, c'_2\}$ can be used to compute a witness pair (x, r) for $C = g^x h^r$. If $x = 0$, observe that $h^{(r_1 - r'_1)/(c'_1 - c_1)} = C$. And if $x = 1$, observe that $h^{(r_2 - r'_2)/(c'_2 - c_2)} = C/g$. Therefore, in either case, a value pair (x, r) that can open commitment $C = g^x h^r$ is found. The non-interactive version of the proof is sound by forking lemma [36] in the random oracle model.

Zero-Knowledge: Intuitively, the proof is zero-knowledge if what the verifier sees during the protocol (i.e., the transcript) can be simulated without knowing the secret. The OR proof is honest verifier zero knowledge, which means that if the verifier follows the protocol, the transcript can be simulated. For any commitment C and challenge c, a simulator chooses r_1, c_1, r_2, c_2 at random such that $c = c_1 + c_2$. The simulator who then computes $a_1 = h^{r_1} C_{-c_1}$ and $a_2 = h^{r_2} C / g^{-c_2}$, therefore, can simulate the interactions. The non-interactive version of the proof of OR protocol is zero-knowledge in the random oracle model [4].

B Adversarial model and security proofs

In this section we give our game-based definition of security, and then prove that our protocols satisfy this definition.

Definition 1 *A verification protocol for aggregate queries on outsourced databases is secure against adaptive correctness, completeness, and privacy attacks, if no polynomial time bounded adversary has a non-negligible advantage against the challenger in the game defined below.*

For simplicity, the game definition is for a single attribute table. It can be generalized to multiple attribute queries and is omitted.

Setup: The challenger takes a security parameter k, and generates public parameters *param*, which is given to the adversary. The challenger keeps the private key SK to itself.

Phase 1: The adversary issues queries $q_1, \ldots, q_m$, where q_i is one of the followings:

1. Commit query (A): The challenger computes commitments of data elements in set A. The commitments and random values used are given to the adversary.
2. Sign query (h_r): The challenger signs the root hash h_r with its private key.
3. Aggregate query (A, Q): The challenger runs the corresponding **Respond** algorithm to answer query Q of A. The resulting answer *ans*, correctness and completeness proofs *pf*, and the signature Sig of A are sent to the adversary.

These queries may be asked adaptively. Also, the queried set at each query may be distinct. Once the adversary decides that **Phase 1** is over, she chooses a challenge for attacking privacy. (No need of choosing challenge for attacking correctness and completeness.)

Privacy challenge: The adversary outputs two distinct equal size sets A_0 and A_1 and an aggregate query Q^* to be challenged, such that query Q^* on set A_0 and A_1 gives the same result – the adversary cannot tell them apart by just seeing the query result. The challenger picks a random bit $b \in \{0, 1\}$, computes the query results and proofs on set A_b by running **Respond** algorithm, which outputs (ans^*, pf^*, Sig^*). It sends (ans^*, pf^*, Sig^*) as a challenge to the adversary. The adversary needs to guess whether A_0 or A_1 is used to produce the aggregate result ans^*.

Correctness challenge: The adversary outputs a data set $\tilde{A} = (\tilde{a}_1, \ldots, \tilde{a}_n)$, commitments $\{\tilde{C}\} = (\tilde{C}_1, \ldots, \tilde{C}_n)$ of data elements, and random values $\tilde{r}_1, \ldots, \tilde{r}_n$ used in computing the commitments. The challenger opens the commitments by re-computing them with $\tilde{a}_i$ and $\tilde{r}_i$. If all the

commitments are verified successfully, the challenger constructs the Merkle hash tree and signs the root hash. The signature $\tilde{Sig}$ is given to the adversary.

Phase 2: The adversary issues more queries $q_{m+1}, \ldots, q_n$, where q_i is one of:

1. Commit query (A): The challenger responds as in **Phase 1**.
2. Sign query (h_r): The challenger responds as in **Phase 1**.
3. Aggregate query (A, Q): The challenger responds as in **Phase 1**.

Guess: The adversary outputs one or more of three guesses for attacking correctness, completeness, and privacy, respectively.

- **Privacy guess:** The adversary outputs a guess $b' \in \{0, 1\}$. The adversary wins the game if $b = b'$. We define its advantage in attacking the scheme to be $|\Pr[b = b'] - \frac{1}{2}|$.
- **Correctness guess:** The adversary outputs $(\tilde{Q}, \tilde{A}, \tilde{ans}, \tilde{pf}, \tilde{Sig}^*)$, such that $\tilde{ans}$ is *not* the correct result of query $\tilde{Q}$ over data set $\tilde{A}$, however $\tilde{pf}$ is an acceptable proof of correctness of result $\tilde{ans}$, and $\tilde{Sig}^*$ is an acceptable signature. We allow $\tilde{Sig}^*$ to be different from what is given in **Correctness challenge**. However, the additional constraint is that $\tilde{Sig}^*$ is a signature of a message (root hash) that has not been signed in **Phase 1** or **Phase 2**. In other words, the adversary can demonstrate that a wrong answer can pass the correctness verification. Note that the adversary needs to output the individual data values of $\tilde{A}$ in her attack.
- **Completeness guess:** The adversary outputs $(\tilde{Q}, \tilde{A}, \tilde{ans}, \tilde{pf}, \tilde{Sig})$, such that $\tilde{ans}$ is *not* the complete result of query $\tilde{Q}$ over the data set $\tilde{A}$, however $\tilde{pf}$ is an acceptable proof of completeness and $\tilde{Sig}$ is an acceptable signature of commitments of the data. In other words, the adversary can demonstrate that an incomplete answer can pass the completeness verification.

B.1 Building blocks in proofs

The security is proved based on the security of collision-free hash function, commitment scheme with hiding and binding properties, and signature scheme secure against existential forgery. We give simple definitions of security for all three commonly used primitives next. The game definitions of these primitives can be easily generalized and are omitted in this paper.

Definition 2 *Security of collision-free hash function H: a polynomial-time adversary has negligible probability of finding two different message $m_1 \neq m_2$ such that their hash values are the same, i.e., $H(m_1) = H(m_2)$.*

Definition 3 *Security of commitment scheme with binding and hiding properties: a polynomial-time adversary has negligible probability of breaking the hiding property by identifying from a commitment its corresponding message which is one of the two messages of her choice [25], and has negligible probability of breaking the binding property by finding a commitment that can be opened to two different messages.*

Definition 4 *Security of signature scheme with existential unforgeability [18]: a polynomial-time adversary has negligible probability of forging a valid signature S of a signer on a message m such that the signer has never signed m.*

Definition 5 *Security of a zero-knowledge proof of a greater-than relation has the following three properties.* Completeness*: the honest prover who knows the secrets a and b ($a \geq b$) convinces the verifier that a is greater-than-or-equal to b with overwhelming probability.* Soundness: *no one who does not know the secrets a and b ($a \geq b$) can convince the verifier with non-negligible probability.* Zero-Knowledge: *what the verifier sees during the protocol (i.e., the transcript) can be simulated without knowing the secret.*

Proof sketch of Theorem 1: For simplicity, our proof is for a single attribute table. The proof generalizes to multiple attribute queries naturally, which is omitted. Let Adv_a be the adversary that has advantage against our correctness or completeness verification protocol. Let us construct an adversary Adv_b that uses Adv_a to gain advantage against collision-free hash function, secure commitment scheme, secure signature scheme, or zero-knowledge proof of greater-than. The adversary Adv_b acts as the challenger for Adv_a and uses Adv_a's outputs as her own outputs. Adv_b answers Adv_a's queries as follows.

Setup: Adv_b's challenger chooses hash function H, commitment scheme C, signature scheme S, and the zero-knowledge proof P of greater-than for Adv_b to break. Adv_b's challenger gives Adv_b a public key PK of the signature scheme S. Adv_b then gives the adversary Adv_a the resulting public parameters $param = (PK, H, C, S, P)$. Note that Adv_b does not know the private key SK of signature scheme S.

Phase 1: For query q_i, Adv_b answers Adv_a's queries as follows. The queries may be asked adaptively. Also, the queried document at each query may be distinct.

1. Commit query (A): Adv_b runs the first several operations in **Commit** algorithm on A, including computing commitments, building hash tree over commitments, and gathering auxiliary information $Info$. The commitments and $Info$ are given to the user.
2. Sign query (h_r): Adv_b does not know how to sign the root hash, because he does not have the private key. Therefore, Adv_b submits a signing query on the root hash to his challenger (of the signature scheme to break), and obtains a signature Sig (the game definition for signature scheme is not given, please see [18]). Signature Sig is given to the user.
3. Aggregate query (A, Q): Adv_b runs a commit query and a sign query on A to obtain the signature Sig, commitments, and auxiliary information $Info$. Then Adv_b runs the corresponding **Respond** algorithm to compute query Q on A. The resulting answer ans, correctness and completeness proofs pf, and the signature Sig of data set A are sent to Adv_a.

Once Adv_a decides that **Phase 1** is over, she chooses a challenge for attacking privacy.

Privacy challenge: Adv_a outputs two distinct equal-size (n) data sets A_0 and A_1 and an aggregate query Q^* to be challenged, such that query Q^* on set A_0 and A_1 gives the same result (Adv_a should not be able to tell them apart by just seeing the query result or the size.) Adv_b chooses a random $i \in [1, n]$, such that the i^*-th elements of A_0 and A_1 are distinct. Denote the two elements by $a^0_{i^*}$ and $a^1_{i^*}$, respectively.

Adv_b needs to use Adv_a's advantage against the confidentiality to break the hiding property of commitment scheme. Adv_b needs to embed his commitment challenge in the challenge of Adv_a. Values $a^0_{i^*}$ and $a^1_{i^*}$ are Adv_b's two messages of choice for breaking the hiding property of commitment C. Adv_b's challenger generates a challenge for Adv_b as follows. Adv_b's challenger picks a random bit $b \in \{0, 1\}$, and computes a commitment of $a^b_{i^*}$. Denote this challenge as C^b_*.

Although Adv_b does not know b, Adv_b needs to compute commitments of elements in A_b ($b \in \{0, 1\}$) such that the correctness verification can pass. Adv_b first computes the aggregate result ans^* of query Q^*, then Adv_b chooses a random guess $b'' \in \{0, 1\}$. Note that, $b'' = b$ with probability $1/2$. For each j-th element in $A_{b''}$ for all $j \neq i^*$ and $j \neq 1$, Adv_b computes a commitment C_j.

Adv_b distinguishes two cases.

- For sum-related query Q^*, Adv_b chooses random r and computes a commitment for the sum: $C_s = g^{ans^*} h^r$. (The sum is the same for A_0 and A_1 as defined.) Adv_b then computes the commitment C_1 for the first element as $C_1 = C_s / (C^b_* \times \prod_{j=2, j\neq i^*}^{n} C_j)$. Now, Adv_b has embedded his commitment challenge at the i^*-th position of A's challenge. ans is the aggregate result. Random value r and commitments C_j ($j \neq i^*, j \in [1, n]$) and C^b_* are correctness proof

pf^* for summation. Adv_b also obtains a signature Sig^* as in sign query. (ans^*, pf^*, Sig^*) is given to Adv_a. Readers can verify that the correctness verification of (ans^*, pf^*, Sig^*) should be successful even though Adv_b does not know b.

- For max/min type of query Q^*, for the i^*-th position, Adv_b does not know $a^b_{i^*}$, therefore, he has to simulate the greater-than proof (i.e., transcript). The simulation can be done, because of the zero-knowledge property of greater-than protocol, which guarantees that the verifier sees during the protocol (i.e., the transcript) can be simulated by anyone without knowing the secret. Denote the simulated proof by p_{i^*}. One goal in simulating the proof is to prepare commitments $\alpha_0, \ldots, \alpha_{t-1}$, each corresponding to a random commitment of 0 or 1, such that $\prod_{i=0}^{t-1} \alpha_i^{2^i}$ is a commitment of $|a^b_{i^*} - ans^*|$, as in the zero-knowledge proof of greater-than. W.l.o.g., we assume that $ans^* > a^b_{i^*}$ and the query is max. Denote the commitments of $a^b_{i^*}$ and query result ans^* by C^b_* and C_{ans^*}, respectively. Note that Adv_b does not know $a^b_{i^*}$, thus does not know the bit presentation of $ans^* - a^b_{i^*}$. Adv_b first chooses random commitments for $\alpha_1, \ldots, \alpha_{t-1}$, and then computes $\alpha_0 = C_{ans^*}/(C^b_* \prod_{i=1}^{t-1} \alpha_i^{2^i})$.
  In addition, Adv_b has to simulate zero knowledge proofs to show that values committed by $\alpha_0, \ldots, \alpha_{t-1}$ are either 0 or 1. This can be done without Adv_b knowing the real values committed, because of the zero-knowledge property of the OR proof (see the previous section for the definition of zero-knowledge property). Therefore, Adv_b can simulate proof p_{i^*}. We omit further details of proof simulation and refer readers to literature for details [9, 13, 16, 28]. The correctness proof pf^* contains p_j ($j \in [1, n], j \neq i^*$) and p_{i^*}. Adv_b also obtains a signature Sig^* as in sign query. (ans, pf^*, Sig^*) is given to Adv_a.

Adv_b can also generate completeness proof in pf^* (if it applies). Because completeness proof would be the same for A_0 or A_1, it cannot be used to gain advantage for privacy attack, hence is omitted here.

Correctness challenge: Adversary Adv_a outputs a data set $\tilde{A} = (\tilde{a}_1, \ldots, \tilde{a}_n)$, commitments $\{\tilde{C}\} = (\tilde{C}_1, \ldots, \tilde{C}_n)$ of data elements, and random values $\tilde{r}_1, \ldots, \tilde{r}_n$ used in computing the commitments. Adversary Adv_b opens the commitments by re-computing them with $\tilde{a}_i$ and $\tilde{r}_i$. If all the commitments are verified successfully, Adv_b constructs the Merkle hash tree. Then, Adv_b asks its challenger to sign the root hash. The resulting signature $\tilde{Sig}$ is given to Adv_a.

Phase 2: The adversary issues more queries $q_{m+1}, \ldots, q_n$, where q_i is one of:

1. Commit query (A): The challenger responds as in **Phase 1**.
2. Sign query (h_r): The challenger responds as in **Phase 1**.
3. Aggregate query (A, Q): The challenger responds as in **Phase 1**.

Guess: Adversary Adv_a outputs one or more of three guesses for attacking correctness, completeness, and privacy, respectively.

- **Privacy guess:** Adversary Adv_a outputs a guess $b' \in \{0, 1\}$. Adv_b outputs b' as his guess for breaking the hiding property of the commitment scheme. If Adv_a has advantage ϵ_1 in breaking the confidentiality property, then Adv_b has advantage at least $\epsilon_1/2$ in breaking the commitment scheme. Recall that Adv_b does not know b and thus when computing commitments in the proofs for Adv_a, it guesses randomly whether to use elements from A_0 or A_1. For half of the time, Adv_a is given the right combination of committed values. Thus, Adv_b carries over advantage $\epsilon_1/2$ in breaking the hiding property of the commitment scheme.
  Adversary Adv_a may also try to gain advantage from proof information other than commitments, for example, from zero-knowledge proofs of great-than relation. Reduction can be directly constructed from advantages in such attacks to breaking the zero-knowledge property

of the greater-than proof protocol, and is omitted here.

- **Correctness guess:** Adversary Adv_a outputs $(\tilde{Q}, \tilde{A}, \tilde{ans}, \tilde{pf}, \tilde{Sig}^*)$, such that $\tilde{ans}$ is *not* the correct result of query $\tilde{Q}$ over data set $\tilde{A}$, however $\tilde{pf}$ is an acceptable proof of correctness, and $\tilde{Sig}^*$ is an acceptable signature of commitments of data.
  Adv_b's goal is to try to convert Adv_a's output into either breaking the binding property of commitment scheme or an existential signature forgery. We distinguish two cases.
  - $\tilde{Sig}^* \neq \tilde{Sig}$: $\tilde{Sig}^*$ is not the same as given in **Correctness challenge**. As defined, the constraint is that $\tilde{Sig}^*$ is a signature of a message (root hash) that has not been signed in **Phase 1** or **Phase 2**. This means that Adv_b obtains a signature that breaks the existential unforgeability property. Adv_b outputs $\tilde{Sig}^*$ and the corresponding message (which is the root hash of Merkle tree and can be easily obtained from the proof $\tilde{pf}$). If Adv_a has advantage ϵ_2 in this attack, then Adv_b has advantage ϵ_2 in breaking the existential unforgeability of the signature scheme.
  - $\tilde{Sig}^* = \tilde{Sig}$: Denote the commitments in $\tilde{pf}$ by $\{\tilde{C}^*\}$. We distinguish the following three cases.
    1. If the commitments $\{\tilde{C}^*\}$ are the same as the commitments $\{\tilde{C}\}$ (computed in **Correctness challenge**) and the query $\tilde{Q}$ is summation-based (e.g., sum, count, etc), then Adv_b can break the binding property of commitment scheme as follows. Adv_b computes the *correct* answer *ans* of set $\tilde{A}$ for query $\tilde{Q}$, and computes the commitment C_{ans} of *ans* based on the commitments $\tilde{C}_1, \ldots, \tilde{C}_n$: $C_{ans} = \prod_{i=1}^{n} \tilde{C}_i$. C_{ans} is also the commitment of the *incorrect* result $\tilde{ans}$, because $\prod_{i=1}^{n} \tilde{C}_i = \prod_{i=1}^{n} \tilde{C^*}_i$. Adv_b outputs C_{ans} as the commitments for both *ans* and $\tilde{ans}$ to its challenger. If Adv_a has advantage ϵ_3 in this attack, then Adv_b has advantage ϵ_3 in breaking the binding property of the commitment scheme. (Adv_b also knows how to open commitment C_{ans}.)
    2. If the commitments $\{\tilde{C}^*\}$ are the same as the commitments $\{\tilde{C}\}$ (computed in **Correctness challenge**) and the query $\tilde{Q}$ is comparison-based (e.g., min, max), then Adv_a can cheat on the greater-than protocol in the correctness proofs of max/min query. This means that Adv_b can break the soundness of zero-knowledge proof of greater-than. The analysis is similar to our completeness analysis (below), and is omitted here.
    3. If the commitments $\{\tilde{C}^*\}$ are different from the commitments $\{\tilde{C}\}$, then Adv_a has find a hash collision. That is, Adv_a has find at least a different message pair (commitments) giving the same hash value (and thus same signature). If Adv_a has advantage ϵ_4 in finding such a message-signature pair, then Adv_b has advantage ϵ_4 in breaking the collision-free hash function.
- **Completeness guess:** Adversary Adv_a outputs $(\tilde{Q}, \tilde{ans}, \tilde{A}, \tilde{pf}, \tilde{Sig})$, such that $\tilde{ans}$ is *not* the complete result of query $\tilde{Q}$ over a set of data, however $\tilde{pf}$ is an acceptable proof of completeness and $\tilde{Sig}$ is an acceptable signature of commitments of data. Let the selection range be $[x, y]$. This means that Adv_a cheats on the zero-knowledge greater-than proof in either one or both cases: (1) proving in zero-knowledge that $a_{left} < x$, however $a_{left} \geq x$; (2) proving in zero-knowledge that $a_{right} > y$, however $a_{right} \leq y$. If Adv_a achieves this, Adv_b can break the soundness of zero-knowledge proof of greater-than. Recall that soundness means that no one who does not know the secret can convince the verifier with non-negligible probability. In this proof protocol, it means that no one who does not know a secret satisfying the greater-than relation can convince the verifier with non-negligible probability. If Adv_a has advantage ϵ_5

in cheating the completeness proof, then Adv_b has advantage ϵ_5 in breaking the soundness of zero-knowledge proof of greater-than.

□