Privacy-Preserving Computation and Verification of Aggregate Queries on Outsourced Databases *

Brian Thompson[1], Stuart Haber[2], William G. Horne[2], Tomas Sander[2], and Danfeng Yao[1]

[1] Department of Computer Science
Rutgers University
Piscataway, NJ 08854, USA
{bthom, danfeng}@cs.rutgers.edu

[2] Hewlett-Packard Labs
5 Vaughn Drive, Suite 301
Princeton, NJ 08540, USA
{stuart.haber, william.horne, tomas.sander}@hp.com

Abstract. Outsourced databases provide a solution for data owners who want to delegate the task of answering database queries to third-party service providers. However, distrustful users may desire a means of verifying the integrity of responses to their database queries. Simultaneously, for privacy or security reasons, the data owner may want to keep the database hidden from service providers. This security property is particularly relevant for aggregate databases, where data is sensitive, and results should only be revealed for queries that are aggregate in nature. In such a scenario, using simple signature schemes for verification does not suffice. We present a solution in which service providers can collaboratively compute aggregate queries without gaining knowledge of intermediate results, and users can verify the results of their queries, relying only on their trust of the data owner. Our protocols are secure under reasonable cryptographic assumptions, and are robust to collusion between k dishonest service providers.

Key words: Aggregate query, outsource, privacy, integrity, secret sharing, verification

* This work has been supported in part by NSF grant CCF-0728937, CNS-0831186, and the Rutgers University Computing Coordination Council Pervasive Computing Initiative Grant. This material is also based upon work supported by the U.S. Department of Homeland Security under grant number 2008-ST-104-000016. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the U.S. Department of Homeland Security.

1 Introduction

Privacy concerns are still a major obstacle that makes sensitive data inaccessible to data mining researchers and prevents collaborative data analysis and filtering among multiple organizations from becoming a reality. Many databases contain sensitive information, and the data owner may not want to share it in full with untrusted entities. Thus, the data owner may only want to allow queries of a statistical or aggregate nature. This privacy requirement has become a common issue for large collections of sensitive data with applications to census data, medical research, and educational testing [19]. For example, aggregate medical information about a group of patients may be accessible for research purposes. However, medical records of individual patients are confidential and should be kept hidden from all parties except for the hospital maintaining them [17].

An ever-increasing trend in today's organizational data management is data outsourcing and cloud computing. An owner may choose to outsource the data, that is, to allow the data to be hosted by third-party service providers. The data hosts would be given the ability to store full or partial information from the database, and the capability to answer queries of a certain type. Data outsourcing alleviates the workload of the data owner in answering queries by delegating the tasks to powerful third-party servers with large computational and network resources.

However, data outsourcing poses additional privacy risks to the sensitive contents. The outsourcing service providers may not be fully trusted by the data owner, or may be susceptible to attacks by malicious parties (both internal and external). Studies have shown that in an outsourced setting it is extremely easy for malicious employees at the service provider organization to access the passwords of business owners and thus their customer data [4]. Security breaches at providers caused by outside adversaries may expose sensitive hosted information.

However, existing Database-As-a-Service (DAS) models are unable to support sophisticated queries such as aggregation while maintaining the secrecy of microdata simultaneously. Existing approaches based on the encryption of outsourced contents [1, 33] apply to models where the user who queries the encrypted outsourced data is the data owner herself, i.e., the data owner and the query issuer are the same person. We consider a more general setting where the database can be queried by anyone. Thus, there is a gap between the security guarantees provided by existing data outsourcing systems and the privacy needs of the data owners. To protect sensitive data from these threats, it is desirable to outsource the data in such a way that *aggregate queries can be computed without revealing microdata (i.e., individual data entries) to service providers.* This paper presents a solution to realize this goal.

Cross-domain collaborative data analysis is another application that motivates our work. For example, multiple regional hospitals collaborate to discover the most frequently occurring flu strain of the season in that area. Existing solutions that support multi-party privacy-preserving data mining require either a trusted or semi-trusted third-party to moderate the computation [29] or the active online participation of players in order to complete the computation [5,

35]. Neither approach provides a practical solution that can be deployed and operated in a completely decentralized fashion. As it will soon become clear, we aim to realize a more practical model without any trusted party, where each data owner may pre-process their data once, independently, and then a (qualified) user can get aggregate query responses computed on the entire collection of data coming from the heterogeneous data sources. The privacy-preserving requirement specifies that neither the user nor the service providers learn any microdata – just the aggregation results.

Aside from the aforementioned privacy requirements, an outsourcing framework also has to address the clients' need for the *integrity of query results*. Here, clients are individual customers of the data owner who query the data. For example, a client may not trust a third-party service provider to accurately represent the data in the outsourced database. Suppose the client submits an aggregate query and gets an answer from the service provider. How can he be sure that the value was calculated correctly and completely without being permitted to see the individual data entries involved in the computation? Aggregate query integrity has been largely ignored in the current literature. In comparison, we present a comprehensive solution to the problem of securely computing and verifying aggregate queries on outsourced databases, which is described in more detail next.

Our Contributions In this paper, we formalize a model called PDAS (pronounced Pea-Das) for preserving privacy and integrity of aggregate query results. We describe a distributed architecture that supports querying outsourced data in a multi-player setting, in which a data owner delegates to third-party service providers the task of answering queries from users.

We construct lightweight cryptographic protocols for privacy-preserving computation and verification of the aggregate queries SUM and AVERAGE. We describe the handling of aggregate queries with SELECT clauses as an extension. Our protocols allow a user to verify correctness of aggregate results while the individual data values contributing to the results are kept secret from both the user and the service providers. The user interacts with a single service provider to obtain aggregate results, and can verify whether or not the service provider returns the correct results. Our solutions utilize simple cryptographic primitives such as a threshold secret-sharing scheme.

Our algorithms are efficient. Let n be the size of the data set, and let m be the number of service providers available to host the data. Let k be the threshold value, i.e. k data hosts must cooperate to compute a query. Then the setup cost is $O(nmk)$ time and $O(n)$ space for the data owner, plus a communication cost of $O(n)$ between the data owner and each of the m service providers. Each service provider requires $O(n)$ space but no additional setup time. The time complexity for computing a query over a subset of size s is only $O(s)$ for each of the k servers participating (done in parallel); the service provider responding to the query needs $O(k^2)$ time to compute the result, with a total communication complexity of $O(k)$ between the responder and the other servers. Verifying the result requires $O\left(\min(s \log n, n)\right)$ communication cost and run-time for the user.

Our contributions are summarized as follows:

- An efficient, distributed architecture for outsourcing databases
- A privacy-preserving protocol for computing aggregate queries that is resistant to the collusion of $k-1$ dishonest service providers
- A mechanism that allows users to verify the integrity and correctness of aggregate query responses

2 Preliminaries

In this section, we provide background on the cryptographic building blocks we use to construct our solution.

2.1 Shamir's Secret-Sharing Scheme

In a k-out-of-n secret-sharing scheme, the data owner distributes *shares*, or parts, of the secret to n servers in such a way that any k of them can cooperate and recover the entire secret, but any smaller group cannot. Secret sharing schemes [30] have been widely used in constructing fault-tolerant key management schemes [21, 31] or password recovery schemes [11].

Shamir's secret-sharing scheme [30] is based on polynomial interpolation. Suppose there are m participants, and any k of them should be able to recover the secret S. Let q be a large prime. The distributor chooses a random $(k-1)$-degree polynomial P over the field $\mathbf{F}_q$ such that $P(0) = S$. That is, he chooses $a_1, \ldots, a_{k-1}$ independently and uniformly at random from $[0, q-1]$, and lets $a_0 = S$, where S is interpreted as an element of $\mathbf{F}_q$. The corresponding polynomial will be

$$P = a_{k-1}x^{k-1} + \ldots + a_1 x + a_0.$$

The share for each participant is a distinct point on P, but obviously not $P(0)$. If any k participants share their knowledge, they collectively will have k distinct points on the curve, from which they can determine P using polynomial interpolation, and thus recover the secret $S = P(0)$. If only $k-1$ participants cooperate, however, they will be unable to recover the polynomial. Furthermore, each different value of S would yield a different polynomial that agrees with their $k-1$ points, so they have gained no knowledge about the secret S.

2.2 Pedersen's Commitment Scheme

A commitment scheme is a protocol for committing to a value without revealing it to observers, so that knowledge of the value may later be proven, but the value to which the commitment was made cannot be changed. There are many different schemes; here, we will use the Pedersen commitment scheme [26] because of its homomorphic properties.

There are two parties involved, a Prover and a Verifier. The Prover would like to commit to a value x, and only reveal it at a much later time. The Verifier wants to ensure that the Prover cannot modify the value of x during the

protocol. Both parties first agree on a group G_p of prime order p, and choose two generators g, h for which the discrete log problem is believed to be difficult and $\log_g h$ is unknown. The Prover generates a random number r, and publishes the commitment $c = C_r(x) = g^x h^r \pmod{\text{p}}$. Due to the use of randomness, the Verifier cannot determine anything about x. Later the Prover may prove his knowledge of x by revealing both x and r to the Verifier, who then checks that $g^x h^r \pmod{p} = c$. Because of the presumed difficulty of finding discrete logs, the Prover could not have changed his commitment to y because he would have had to find $r' = \log_h(c \cdot g^{-y})$. Thus the commitment scheme is computationally binding and unconditionally hiding.

The Pedersen scheme enjoys a convenient homomorphic property: Given commitments $c_i = C_{r_i}(x_i)$ for $i = 1, \ldots, m$, it is easy to compute a commitment to the sum of the unknown values $X = \sum_{i=1}^{m} x_i$ simply by computing the product of the individual commitments: $\prod_{i=1}^{m} c_i \pmod{p} = C_R(X) \in G_p$, where $R = \sum_{i=1}^{m} r_i$.

3 Models and Definitions

There are three types of players in our model. A *data owner* is the creator or maintainer of a database. The data owner delegates to a set of *service providers* the responsibility of answering queries. A service provider may be honest or dishonest, which we explain in more detail later. A *user* obtains query responses from a single service provider and does not interact directly with the data owner. The basic interaction model is as follows: The data owner gives the service providers partial information about each entry in the database, along with auxiliary information that enables the verification of query results. Upon receiving a query, a service provider seeks the cooperation of $k-1$ other service providers, who may then jointly reconstruct the result of the query. The result is then passed back to the user, along with sufficient information for the user to verify its correctness.

Here we describe our **trust model** among the players. Our model is similar to the trust assumptions of existing literature on outsourced databases [13, 22].

- *Between data owner and service provider:* The data owner trusts an honest service provider to follow the protocol. Honest service providers are expected not to disclose their data directly to others, but rather only to provide information as dictated by the protocol. Dishonest service providers may collude in order to attempt to reconstruct data entries in the database from their shares, or may not follow the protocol. For example, they may reveal their shares to others or replace their shares with arbitrary values when answering queries.
- *Between service provider and user:* The service provider is not necessarily trusted to answer queries correctly, since it may be malicious or compromised by outside attacks. Therefore, the user should be able to verify that responses from the service provider are correct and complete.

- *Between data owner and user:* The user must trust the data owner in the sense that the user trusts any messages signed with respect to the data owner's public key.

Adversarial Model There are three types of adversaries in our model.

- A curious player (user or service provider) 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 or not follow the protocol.
- An adversary who may intercept and tamper with the protocol communication, e.g., modifying query results, inserting or deleting messages.

Operations 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 *params*. The data owner keeps SK secret. We define the following operations: COMMIT, DISTRIBUTE, QUERY, RESPOND, and VERIFY.

COMMIT: The data owner takes as input a data set $D = (x_1, \ldots, x_n)$. It generates auxiliary information *aux*, computes a digital signature Sig, and publishes (aux, Sig) to m service providers.

DISTRIBUTE: The data owner splits database entries among the m service providers in such a way that it requires at least k providers to jointly retrieve the original values.

QUERY: The user sends to a service provider SP_j a request for an aggregate query Q over a selection of data set D.

RESPOND: SP_j and $k-1$ other service providers jointly compute the aggregate answer *ans*. SP_j prepares the correctness and integrity proofs *pf*, and returns the tuple (ans, pf, Sig) to the user.

VERIFY: The user takes as input $(params, ans, pf, Sig)$. It verifies that the answer *ans* satisfies correctness and integrity properties using proofs *pf*, signature Sig, and the public key PK of the data owner (that is obtained from a trusted source). The answer is accepted if the verification passes.

Security properties *Secrecy, correctness, integrity*, and *collusion-resistance* are the four required security properties in our protocol.

Intuitively, *secrecy* requires that no entity besides the data owner should learn more about the data set D than is implied by (Q, *ans*). *Correctness* requires that *ans* is the correct response to query Q. *Integrity* requires that *ans* is computed based on authentic (outsourced) data set D that has not been tampered with. *Collusion-resistance* requires that $k-1$ or less dishonest service providers cannot collude to break the secrecy requirement.

We address the property of correctness in Section 4, and the other three properties in our formal definition of security, which is given in Section 5.

D	A database
S	A subset of D
x	A database entry (or cell)
n	Number of rows
m	Number of service providers (SPs)
P	A polynomial
$P_i(j)$	SP_j's share of value x_i
k	Threshold for secret-sharing scheme
c_i	Commitment to x_i with random seed r_i
X	An aggregate query result

Table 1. Notation used in our protocol.

4 Our Protocol

For the simplicity of description, we consider a database D with n rows and one column, with each cell containing a positive integer. [3] All of our protocols can easily be generalized to accommodate multiple attributes (i.e., columns). Let $D = x_1, \ldots, x_n$. The data owner would like to outsource his database to m different servers, but with an important security requirement: *any k servers can cooperate to determine the answer to an aggregate query, but $k-1$ cooperating servers cannot.* To achieve this requirement, our approach is to have the data owner distribute the original entries among multiple service providers via a simple threshold secret-sharing scheme. In a naive solution, a service provider SP_j asks $k-1$ other providers for their secret shares of the data entries relevant to the query, and then combines the values to compute the aggregate for the user. Unfortunately, this naive approach fails because SP_j reconstructs the individual data entries as an intermediate result, which violates our privacy requirement.

To solve this problem, we leverage a nice and simple feature of polynomials that allows service providers to first aggregate or *blend* their shares associated with the distinct data entries, and then send the blended values to SP_j. The data leakage problem is eliminated as the service provider SP_j is unable to retrieve individual data shares. Yet, it can still interpolate the polynomial based on the blended shares to obtain the final aggregate result. A more detailed description is given next.

Furthermore, to verify the correctness of the aggregate computation, we use a special type of commitment scheme, namely a homomorphic commitment scheme, which allows anyone to verify the query result without knowing the data. PDAS also achieves the integrity requirement by cleverly utilizing existing authentication data structures over commitment values. As a result, the tampering of data entries during the computation process can be detected while the secrecy of data is safely protected.

[3] Our computation can also be applied to strings or multimedia data, which first need to be converted into numerical values using an encoding or transformation mechanism.

PDAS Protocol

The PDAS protocol is run between the data owner, the service providers, and the user. Let $N = \sum_{i=1}^{n} x_i$. For the setup, the data owner chooses a large prime $q >> N$. This will avoid potential problems with overflow later. The computation associated with aggregate queries is performed in the field $\mathbf{F}_q$. The computation with Pedersen's commitment is in group G_p. The operations in PDAS include COMMIT, DISTRIBUTE, QUERY, RESPOND, and VERIFY.

COMMIT The data owner chooses parameters (G_p, g, h) and computes commitments for the data entries using the Pedersen commitment scheme described in Section 2.2:

$$c_1 = C_{r_1}(x_1), \ldots, c_n = C_{r_n}(x_n).$$

The data owner then generates a Merkle hash tree [20] on the commitment values, and signs on the root hash of the tree.

DISTRIBUTE The data owner distributes each database entry x_i, along with its corresponding random seed r_i, according to Shamir's secret-sharing protocol (Section 2.1) as follows. He chooses random polynomials P_i and Q_i with $P_i(0) = x_i$ and $Q_i(0) = r_i$, and to service provider SP_j for $1 \leq j \leq m$, he gives the share $(j, P_i(j), Q_i(j))$. Thus, SP_j has the values $(j, P_1(j), Q_1(j)), \ldots, (j, P_n(j), Q_n(j))$. Shares $P_i(j)$ are for answering aggregate queries, and shares $Q_i(j)$ are for integrity and correctness verification. The data owner also gives both the entire Merkle hash tree and his signature on the root hash to all m service providers. The data owner can delete the intermediate values from storage.

QUERY A user submits an aggregate query to a service provider, say SP_1. Let us assume the query is for the SUM $X^S = \sum_{i \in S} x_i$ over the values in a subset $S \subseteq D$. (See Section 6 for more discussion on SELECT queries.) SP_1 then sends messages requesting cooperation from $k-1$ other service providers.

RESPOND The k service providers now jointly compute the aggregate query result X^S. Note that they simultaneously compute the corresponding value R^S for the purpose of verification.

1. Each of the k service providers computes its *share of the aggregate result* as follows. Provider SP_j computes $X_j^S = \sum_{i \in S} P_i(j)$, where $P_i(j)$ is the SP_j's share of value x_i. Similarly, SP_j calculates its *share of the random seed* $R_j^S = \sum_{i \in S} Q_i(j)$. Both X_j^S and R_j^S are returned to SP_1.
2. Provider SP_1 collects all $k-1$ shares (j, X_j^S), plus its own sum of relevant shares. Using polynomial interpolation, SP_1 determines the unique polynomial P of degree $k-1$ passing though these k coordinates. It computes $X = P(0)$ as the aggregate result.
3. Again using polynomial interpolation, SP_1 determines the unique polynomial Q of degree $k-1$ passing through the k points (j, R_j^S) from the $k-1$ assisting providers and itself, and computes $R = Q(0)$.
4. SP_1 finally sends the following information to the user: $(X, R, \{c_i\}_{i \in S}, Proof)$, where c_i is the commitment for value x_i, and the $Proof$ contains the values of all sibling nodes along paths from the commitments to the root in the Merkle hash tree, and the data owner's

signature on the root hash. The $Proof$ is provided to the user to verify the integrity of x_i and correctness of computation without revealing the microdata.

VERIFY Upon receiving response $(X, R, \{c_i\}_{i\in S}, Proof)$, the user verifies that the obtained sum X is *correctly computed on the original data*.

1. Using the publicly-known hash function, the user re-computes the root hash of the Merkle hash tree from the commitments $\{c_i\}_{i\in S}$ and their sibling values, which are in $Proof$. He verifies the signature of the root hash using the public key of the data owner, [4] and therefore knows that he has the *authentic* commitment values.
2. With value R and the public parameters g and h, the user calculates the corresponding commitment $C_R(X) = g^X h^R$.
3. The user checks whether the SUM is computed correctly by verifying that the obtained SUM is consistent with the individual commitments. If $\prod_{i\in S} c_i = C_R(X)$, then the delivered answer is accepted.

Correctness of PDAS

The correctness of our algorithm is based on the additive property of polynomials over a field F. If $P = P_1 + P_2$, then $P(x) = P_1(x) + P_2(x)$ for all $x \in F$. We state this claim concisely that, *the sum of the shares is a share of the sum*.

When the data owner distributes the data, he creates a polynomial P_i for each data item x_i. Each service provider SP_j gets a share that is the point $(j, P_i(j))$ along the curve. When a service provider receives a request for a SUM $X^S = \sum_{i\in S} x_i$ over the subset S, it returns the sum of its relevant shares, $X_j^S = \sum_{i\in S} P_i(j)$.

Consider the polynomial $\hat{P} = \sum_{i\in S} P_i$. The summed value returned by service provider SP_j in our protocol is $X_j^S = \sum_{i\in S} P_i(j) = \hat{P}(j)$. When the responding service provider interpolates the polynomial from these k values, it derives the summation polynomial $\hat{P}$. Therefore, the value $\hat{P}(0)$ returned to the user is equal to the desired aggregate value:

$$\hat{P}(0) = \sum_{i\in S} P_i(0) = \sum_{i\in S} x_i = X^S.$$

AVERAGE can be easily computed and verified by dividing SUM by the size of the subset $s = |S|$. Similarly, the above protocol can be generalized to compute any linear combination on the selected entries. Just as $(P+Q)(x) = P(x)+Q(x)$, we also have $(aP)(x) = a \cdot P(x)$, where a is an element of the field $\mathbf{F}_q$. For example, a user can query for the sum $3x_1 + 5x_2 + 12x_3 + \ldots$. To that end, each service provider simply needs to multiply their shares by the appropriate scalars.

[4] As in many security protocols, we assume that the user has an authenticated copy of the data owner's public key.

The above description completes the basic operations in our PDAS protocol. In Section 6, we describe several important extensions to PDAS, including how SELECT can be realized, support for multiple data owners, and how to accommodate dynamic databases.

5 Security and Efficiency

In this section, we analyze the adversary model and prove the security of PDAS. We also give the complexity analysis of our protocols. We provide security definitions, and prove that PDAS satisfies those security requirements.

We consider an attacker who can access *all commitment and signature values*, and can *adaptively choose a sequence of aggregate queries*, i.e., queries for aggregate results and their proofs. The adversary's goal is to have a non-negligible probability of success in violating one of the security properties of our protocol: *secrecy*, *integrity*, or *collusion-resistance*.

We consider three types of attacks: using intermediate or aggregate results to deduce sensitive information about individual entries (*inference attack*), computing a new incorrect query-response pair that passes the VERIFY algorithm (*spoofing attack*), or disrupting the computation of an aggregate query (*disruption attack*). We give security proofs for the inference and spoofing attacks, reducing the existence of a successful polynomially bounded adversary to the existence of an adversary that successfully breaks one or more of the signature scheme, the Pedersen commitment scheme, or the one-way hash function. We then explain how PDAS can easily deal with disruption attacks.

An adversary may act as a user, a service provider, or have a network of colluding service providers. Note that a SP can simulate a user's query request, and thus has at least as much discerning power as a user. Under our security assumptions, the value of k is chosen to be greater than the number of dishonest or compromised service providers. For the rest of this discussion, we assume the worst case: a *network adversary* that has a network of $k-1$ colluding service providers.

Theorem 1. *The PDAS protocol provides information theoretic security against an inference attack by a computationally unbounded adversary. No information is leaked beyond that which can be deduced from the aggregate query results alone.*

Proof Consider a network adversary who requests aggregate queries over l subsets of the data $S_1, \ldots, S_l$, yielding sums $X^{S_1}, \ldots, X^{S_l}$. To determine each sum X^{S_i}, he may request the corresponding shares from any of the m service providers: $X_1^{S_i}, \ldots, X_m^{S_i}$. In addition, he has access to all data shares from the $k-1$ service providers $SP_{\alpha_1}, \ldots, SP_{\alpha_{k-1}}$ in his adversarial network.

Suppose the adversary has an algorithm A that takes as input $\{X_1^{S_i}, \ldots, X_m^{S_i} | i = 1, \ldots, l\}$ and returns some sensitive information about the database (e.g., one of the individual data entries). Let O_A be an oracle for algorithm A. We construct an algorithm A^* that computes the same output as A but using only the l aggregate query results as input.

1. Input the aggregate query results $X^{S_1}, \ldots, X^{S_l}$.
2. Compute the aggregate shares $X^{S_1}_{\alpha_j}, \ldots, X^{S_l}_{\alpha_j}$ for each service provider SP_{α_j} in the adversarial network.
3. The aggregate query result X^{S_1} and the $k-1$ shares $X^{S_1}_{\alpha_1}, \ldots, X^{S_l}_{\alpha_{k-1}}$ are in total k points along the polynomial P_{S_1}; similarly for $P_{S_2}, \ldots, P_{S_l}$. Derive $P_{S_1}, \ldots, P_{S_l}$ using polynomial interpolation.
4. Query the oracle O_A using input $\{P_{S_i}(1), \ldots, P_{S_i}(m) | i = 1, \ldots, l\}$, and return the result.

Therefore no information about the data is leaked that cannot be gained from the query results alone, and so we have guaranteed the security properties of *secrecy* and *collusion-resistance*. The question of whether sensitive information can be inferred from the combination of multiple aggregate query results is an orthogonal issue known as *inference control* [8, 17], and privacy guarantees pertaining to that are beyond the scope of this paper.

Note that an adversary may also try to recover individual data entries using the published commitment values. However, a similar argument to that above shows that the individual random seeds involved in the commitment protocol are protected by the same means as for data entries. Since the random seeds are not disclosed, the Pedersen commitment scheme enjoys unconditional hiding of committed values (see Section 2.2).

Theorem 2. *The PDAS protocol is secure against a spoofing attack by a polynomially-bounded adversary. An adversary with access to the committed database values and signature on the root hash of the data owner's Merkle hash tree cannot spoof the data owner's signature on commitments to a set of incorrect data entries in polynomial time with non-negligible probability.*

Proof Suppose an adversary computes a new incorrect query-response pair that passes the VERIFY algorithm. Since the VERIFY algorithm checks the commitments against a signed root hash, the adversary must have achieved one of the following:

1. Generated a new pair (x'_i, r'_i) such that $C_{r'_i}(x'_i) = C_{r_i}(x_i)$.
2. Generated a commitment to a new value (x''_i, r''_i) such that $H\left(C_{r''_i}(x''_i)\right) = H\left(C_{r_i}(x_i)\right)$, where H is the collision-resistant hash function used in constructing the Merkle hash tree.
3. Forged the data owner's signature for the resulting new root hash.

In case (1), the adversary has broken the computationally binding property of the Pedersen commitment scheme. Case (2) is equivalent to finding a collision in the collision-resistant hash function. In case (3), the adversary has broken the signature scheme. By the respective security guarantees of these cryptographic tools, these tasks cannot be achieved with non-negligible probability in polynomial time. Thus we have preserved the *integrity* of the data.

Claim: *PDAS can effectively counter disruption attacks.*

	Commit	Respond	Verify	**Storage**
Data Owner	$O(nmk)$	——	——	$O(n)$
Primary SP	——	$O(k^2)$	——	$O(n)$
Helper SP	——	$O(s)$	——	
User	——	——	$O(\min(s \log n, n))$	——

Table 2. Summary of computation and space complexities for PDAS, not including communication costs. Distribute and Query are omitted because they incur no computation cost. Here, s is the size of the subset over which the query is performed.

	Distribute	Query	Respond
Data Owner	$O(nm)$	——	——
Primary SP	$O(n)$	$O(ks \log n)$	$O(k)$
Helper SP		$O(s \log n)$	$O(\min(s \log n, n))$
User	——	$O(s \log n)$	$O(\min(s \log n, n))$

Table 3. Communication complexity of operations in PDAS. There is no communication cost associated with the Commit or Verify operations.

Note that under our security assumptions, a service provider who gives an incorrect share value for a query can be detected. In this case, we would like to guarantee that the user can still retrieve the correct results of a query. This fault-tolerance property can be achieved using a publicly verifiable secret-sharing scheme [27], even if there are $k - 1$ dishonest service providers disrupting the procedure. Due to space limitations, discussion is omitted in this paper.

Furthermore, in the PDAS model, service providers can easily be held accountable for their actions. That is, if a service provider gives several faulty values, it can be reported to the data owner, who can then disregard the bad SP and redistribute the data to the remaining $m - 1$ service providers.

Efficiency of PDAS The run-time, space, and communication complexities of operations by each entity in PDAS are summarized in Tables 2 and 3. We note that the amount of storage required at the data owner is $O(n)$ instead of $O(nk)$, since the data owner does not need to store all the secret shares. The shares are erased by the data owner after they are distributed to service providers.

6 Extensions

Our PDAS protocol provides a general framework for managing the privacy and security of outsourced databases. In this section, we describe several important extensions of PDAS, including the handling of dynamic insertions and deletions, selection queries, and multiple data owners.

While some real databases remain largely unchanged over time, many other applications require a database system to allow for the addition or deletion of database entries. The first question to ask, then, is how well our infrastructure

User ID	age	state	[weight]
00159265	16	NJ	122
00173094	35	NJ	168
00298216	18	CA	145

```
SELECT AVG weight WHERE age > 15 AND age < 19 AND state = 'NJ'
```

Fig. 1. An aggregate query for the sensitive attribute 'weight', computed over a selection based on insensitive attributes.

for outsourced databases can deal with dynamic data. Similarly to before, we assume that honest service providers follow the protocol specification.

Additions When a new entry is added to the database, the data owner generates a random polynomial for the entry and distributes shares according to our protocol (Section 4). The shares of other database entries were independently generated and are not affected. The data owner must also update the Merkle hash tree and broadcast the update to all service providers. However, because of the tree structure, this only incurs an additional $O(\log n)$ cost.

Deletions In the case of a deletion, the data owner simply needs to broadcast to the service providers that they must delete their shares of that entry. There is no need to remove the commitment value from the Merkle hash tree - the value will just never be used again during verification.

Since all secret-sharing polynomials are independently and randomly chosen, these additional operations introduced to the PDAS protocol do not affect our security guarantees. In conclusion, our protocol can accommodate dynamic databases both efficiently and securely.

Our database system can also handle multi-attribute data and answer complex aggregate queries. Consider a database with one attribute containing sensitive data, and also several insensitive attributes, such as the example given in Figure 1. Values for the sensitive attribute will be distributed according to our secret-sharing protocol, whereas values for the insensitive attributes can be sent to the service providers in plaintext. When a user poses a complex query, the responding service provider first determines the subset over which to aggregate using the selection conditions on the insensitive data, and then computes the aggregate query according the PDAS protocol.

Consider an environment in which several data owners have disjoint sets of data with the same attributes, and are willing to collaborate to allow aggregate queries over the union of their data. However, they still want to protect the privacy of their own constituents, so are not willing to reveal their data to the other parties.

This scenario fits in perfectly with our PDAS protocol. The collaborating parties must first agree on choosing several parameters: the field $\mathbf{F}_q$, the m service providers, the security parameter k, and the parameters for the commitment scheme (G_p, g, h). From then on, their contributions are independent of

each other. Each data owner distributes his data and signs on the root hash of his Merkle hash tree. A service provider may then take aggregates over all the data regardless of its origin. When verifying a query result, the service provider simply sends the commitment values from the appropriate data owners.

7 Related Work

A substantial amount of research work has been done on how to verify outsourced data and computation [3, 6, 14, 13, 15, 22–24, 18], including the verification of both correctness and completeness of relational database queries. Existing literature on database query verification has 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 [20]. Aggregate signatures are another approach for data authentication, where each data tuple is signed by the data owner [24]. The privacy issue of verifying non-aggregate queries was addressed in [25], which gave an elegant solution using hashing for proving the completeness of selection queries without revealing neighboring entries.

The aggregate query verification problem has been studied in the DAS model [14, 15, 23], an instantiation of the computing model involving clients who store their data at an untrusted server, which is administrated by a third-party service provider. The clients have limited computational power and storage, and thus rely on the service provider for its large computational resources. The challenge is to make it impossible for the service provider to correctly interpret the data, but still allow it to compute and return aggregate queries. The data is owned by the clients, and only they are permitted to perform queries on the database.

The paper by Hacigümüs, Iyer, and Mehrotra [15] addresses the execution of aggregate queries over encrypted data using a homomorphic encryption scheme. Their model has two parties: the data owner and the untrusted service provider. Mykletun and Tsudik [23] propose an alternative approach where the data owner pre-computes and encrypts the aggregate results and stores them at the service provider [23]. 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 [16]. However, these solutions are only applicable under the two-party model, where the querier is also the data owner. In comparison, our PDAS protocol works in the more general three-party model, where the client who queries the service provider may not be the same as the data owner.

In data mining literature, one approach to protecting data privacy is to publish modified versions of database tables so that each individual entry enjoys a certain degree of anonymity [2, 28, 32, 34]. This imposes no restrictions on queries that may be performed on the data, but the anonymization process necessarily introduces some loss of integrity in the accuracy of the data. Our solutions differ from these existing efforts in that we support authenticated data analysis with-

out releasing any data to the public. Because the aggregate is computed over exact data instead of anonymized data, there is no loss of data accuracy in the aggregation results.

A new approach to providing anonymity when sharing data has appeared with the recent stream of research on *differential privacy* [7, 10, 12, 9], in which noise is added to query results to prevent the querier from inferring information about individuals. Our work, on the other hand, is concerned with adding proofs of integrity to exact responses to queries to the database, and so our protocols are vulnerable to the privacy attacks studied in the differential privacy literature – as are all protocols whose responses to queries are close to exact. It is a challenging open problem to design protocols that resist the differential-privacy attacks while still providing integrity guarantees for the protocol's responses compared to the original data.

8 Conclusions

In this paper, we proposed a simple privacy-preserving protocol PDAS for computing and verifying queries in outsourced databases. We focused on computing aggregate queries including SUM and AVERAGE with SELECT clauses. The main goal of PDAS is to prevent microdata (i.e., individual data entries) from being accessed by users or any of the third-party service providers who are delegated by the data owner to answer queries. Existing DAS models are unable to support sophisticated queries such as aggregation while maintaining secrecy of microdata simultaneously. We overcame this challenge and introduced two main techniques:

- **A distributed architecture** is introduced for outsourcing databases using multiple service providers. We extended threshold secret sharing schemes to support sophisticated aggregation operations by leveraging the additive property of polynomials over a field.
- **A verification protocol** is developed for the user to verify that the outsourced computation is indeed computed correctly, *without* leaking any microdata.

We provided security analysis that our protocol achieves secrecy, integrity, correctness, and collusion-resistance properties. We also discussed possible variants of our PDAS model, including handling of dynamic databases, multiple data owners, and inference control.

References

1. 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.

2. Rakesh Agrawal and Ramakrishnan Srikant. Privacy-preserving data mining. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, May 2000.
3. 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.
4. Tzvika Chumash and Danfeng Yao. Detection and prevention of insider threats in database driven web services. In *Proceedings of The Third IFIP WG 11.11 International Conference on Trust Management (IFIPTM)*, June 2009.
5. Isabel F. Cruz, Roberto Tamassia, and Danfeng Yao. Privacy-preserving schema matching using mutual information. In Steve Barker and Gail-Joon Ahn, editors, *DBSec*, volume 4602 of *Lecture Notes in Computer Science*, pages 93–94. Springer, 2007.
6. P. Devanbu, M. Gertz, C. Martel, and S. Stubblebine. Authentic third-party data publication. *Journal of Computer Security*, 11(3), 2003.
7. Irit Dinur and Kobbi Nissim. Revealing information while preserving privacy. In *PODS*, pages 202–210, 2003.
8. Josep Domingo-Ferrer, editor. *Inference Control in Statistical Databases, From Theory to Practice*, volume 2316 of *Lecture Notes in Computer Science*. Springer, 2002.
9. Cynthia Dwork. Differential privacy: A survey of results. In *TAMC*, pages 1–19, 2008.
10. Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, pages 265–284, 2006.
11. C. Ellison, C. Hall, R. Milbert, and B. Schneier. Protecting secret keys with personal entropy. *Journal of Future Generation Computer Systems*, 16(4):311318, Feb 2000.
12. Srivatsava Ranjit Ganta, Shiva Prasad Kasiviswanathan, and Adam Smith. Composition attacks and auxiliary information in data privacy. *CoRR*, abs/0803.0032, 2008.
13. 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.
14. H. Hacigümüs, B. Iyer, and S. Mehrotra. Providing database as a service. In *Proceedings of International Conference on Data Engineering (ICDE)*, March 2002.
15. 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.
16. 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.
17. Geetha Jagannathan and Rebecca N. Wright. Private inference control for aggregate database queries. In *ICDM Workshops*, pages 711–716. IEEE Computer Society, 2007.
18. Feifei Li, Marios Hadjieleftheriou, George Kollios, and Leonid Reyzin. Dynamic authenticated index structures for outsourced databases. In Surajit Chaudhuri, Vagelis Hristidis, and Neoklis Polyzotis, editors, *SIGMOD Conference*, pages 121–132. ACM, 2006.
19. Paul Massell, Laura Zayatz, and Jeremy Funk. Protecting the confidentiality of survey tabular data by adding noise to the underlying microdata: Application to

the commodity flow survey. In Josep Domingo-Ferrer and Luisa Franconi, editors, *Privacy in Statistical Databases*, volume 4302 of *Lecture Notes in Computer Science*, pages 304–317. Springer, 2006.
20. 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.
21. Sara Miner More, Michael Malkin, Jessica Staddon, and Dirk Balfanz. Sliding-window self-healing key distribution. In *Proceedings of the 2003 ACM workshop on Survivable and self-regenerative systems: in association with 10th ACM Conference on Computer and Communications Security*, pages 82–90, October 2003.
22. 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.
23. 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.
24. 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.
25. 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.
26. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991.
27. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO '91: Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, pages 129–140, London, UK, 1992. Springer-Verlag.
28. Periangela Samarati. Protecting respondent's privacy in microdata release. *IEEE Transactions on Knowledge and Data Engineering*, 13(6):1010 – 1027, 2001.
29. Monica Scannapieco, Ilya Figotin, Elisa Bertino, and Ahmed K. Elmagarmid. Privacy preserving schema and data matching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 653–664, 2007.
30. Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, November 1979.
31. Jessica Staddon, Sara Miner, Matt Franklin, Dirk Balfanz, Michael Malkin, and Drew Dean. Self-healing key distribution with revocation. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, May 2002.
32. Latanya Sweeney. k-Anonymity, a model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):557 – 570, 2002.
33. 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.
34. Xiaokui Xiao and Yufei Tao. Anatomy: Simple and effective privacy preservation. In *Proceedings of the 32nd Very Large Data Bases (VLDB)*, 2006.
35. Danfeng Yao, Keith B. Frikken, Mikhail J. Atallah, and Roberto Tamassia. Private information: To reveal or not to reveal. *ACM Trans. Inf. Syst. Secur.*, 12(1), 2008.