Query Execution Assurance for Outsourced Databases

Radu Sion

Department of Computer Sciences
Stony Brook University
Stony Brook, NY 11794
sion@cs.sunysb.edu

Abstract

In this paper we propose and analyze a method for proofs of *actual* query execution in an outsourced database framework, in which a client outsources its data management needs to a specialized provider. The solution is not limited to simple selection predicate queries but handles arbitrary query types. While this work focuses mainly on read-only, *compute-intensive* (e.g. data-mining) queries, it also provides preliminary mechanisms for handling data updates (at additional costs). We introduce *query execution proofs*; for each executed batch of queries the database service provider is required to provide a strong cryptographic proof that provides assurance that the queries were actually executed correctly over their entire target data set. We implement a proof of concept and present experimental results in a real-world data mining application, proving the deployment feasibility of our solution. We analyze the solution and show that its overheads are reasonable and are far outweighed by the added security benefits. For example an assurance level of over 95% can be achieved with less than 25% execution time overhead.

1 Introduction

Outsourcing the "database as a service" [18] emerged as an affordable data management model for parties ("data owners") with limited abilities to host and support large in-house data centers of potentially significant resource footprint. In this model a *client* outsources its data management to a "database service provider" which provides online access mechanisms for querying and managing the hosted data sets. At the same time, most of the data server management and query execution load is incurred only by the service provider and not by the client.

This is intuitively advantageous and significantly more affordable for parties with less experience, resources or trained man-power (such as small companies and individuals). Compared with e.g., a small company, with likely a minimal expertize in data management, such a database service provider intuitively has the advantage of expertize consolidation. Moreover it is likely to be able to offer the service much cheaper, with increased service availability (e.g. uptime) guarantees.

Significant security issues are associated with such "outsourced database" frameworks, including communication-layer security and privacy, of both the data and associated access patterns. Of equal concern is the ability to provide assurances of service execution and correctness. For a batch of data queries, it is important to ensure result accuracy as well as authenticity of their input data sets; assurances are required for the fact that the serviced client queries were in fact executed *correctly* over the *entire* intended target data.

This is especially true in an online, incentive-driven, possibly hostile environment. Often the incentive model (e.g., data querying costs) of the service provider allows for a scenario in which query execution correctness could be questioned. A "lazy" or possibly malicious (e.g., with compromised security) service provider could avoid paying CPU and storage costs associated with query execution (e.g., maybe only for the more expensive queries) and reply with (cheaper) inexact or entirely incorrect results.

To enforce data outsourcing as a sound and truly viable alternative to in-house data management, it is essential to provide a solution to handle such scenarios. Existing research [28] deals with this issue in the context of proving authentication for results of "queries

Proceedings of the 31st VLDB Conference, Trondheim, Norway, 2005

testing equality and other logical comparison predicate clauses", queries that return a selection of a set of records matching a given simple predicate. This is an important first step in both identifying the issue and providing a solution for such (what we call) "identity queries" [1]. Ensuring *completeness* [2] and *authenticity* as well as handling complex (e.g., aggregate), arbitrary query types remain open hard issues.

In an outsourced database framework, developing a solution to this problem becomes especially challenging as the data is placed under the authority of an external party whose honest behavior is not guaranteed but rather needs to be ensured by this very solution. Additionally, there are clear cost-incentives for dishonest behavior. Not surprisingly, an intrinsic "unfriendly" behavior is to be anticipated and handled.

In this paper we propose a solution to provide such assurances of query execution correctness, for arbitrary queries, solution built around a mechanism of runtime query "proofs" in a challenge-response protocol. For each batch of client queries, the server is "challenged" to provide a proof of query execution which is then checked at the client site as a prerequisite to accepting the actual query results as accurate.

The execution proof mechanism is partially based on an extension to the *ringer* concept first introduced in [16]. Its core strength derives from the non-invertibility of cryptographic hash functions. In other words, a successful fake execution proof requires the inversion of a cryptographic hash or a lucky guess. The probability of the lucky guess is known, controllable and can be made arbitrary small. If, as part of the response to a query execution batch, the server includes a correct, verifiable query execution proof, the client is provided with a (tunable) high level of assurance that the queries in the batch were executed correctly. This constitutes a strong counter-incentive to "lazy", (e.g., cost-cutting) behavior. By construction, this provides a probabilistic solution (with arbitrary large confidence factors) to both issues of completeness and authenticity.

While on somewhat orthogonal dimensions, it might be worth noting that other important challenges are to be considered in the framework of database outsourcing. Transport layer security is important as eaves-dropping of data access primitives is unacceptable. This can be achieved by deploying existing traditional network security protocols such as IPSec/SSL. Additionally, because data resides on another party's server, there are likely scenarios in which privacy is a concern. Both the client/owner query access patterns and possibly the data itself are to be concealed. Significant progress has been made in this area [5] [6] [12] [13] [17]. Section 6 discusses related work.

The main contributions of this paper include: (i) the proposal and definition of the problem of query execution proofs for arbitrary *compute-intensive* query batch processing in an outsourced database model, (ii) a solution offering assurance of correct query execution, (iii) its analysis, (iv) a proof-of concept implementation and (v) the experimental evaluation thereof.

The paper is structured as follows. Section 2 introduces the main system, data and adversary models. Section 3 overviews, details and analyzes our solution. Section 4 discusses extensions and revisits some of the assumptions. Section 5 introduces our proof-of-concept implementation and provides an experimental analysis thereof. Section 6 surveys related work and Section 7 concludes.

2 Model and Tools

2.1 Data Outsourcing

We choose to keep the data outsourcing model concise yet representative. Data owned by a *data owner* (Alice) is placed on the database server situated at the site and under the control of a *database service provider* (Bob). From this moment on, Alice can access the outsourced data solely as a client through an online query interface exposed by Bob. The network layer is assumed to be secured by mechanisms such as SSL/IPSec. Here we discuss the scenario where all data access is initiated by or occurs through Alice only. This corresponds to a *unified client model* as defined in [14] and [28]. In Section 4 we discuss extensions and propose alternatives for more complex models.

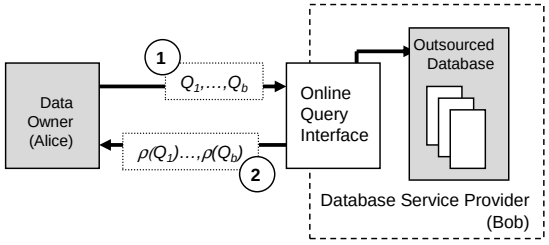

Figure 1: Database Outsourcing Overview

In a typical scenario Alice submits a batch of $\mathtt{b}$ queries $\mathbb{Q} = \{\mathtt{Q}_1, ..., \mathtt{Q}_\mathtt{b}\}$ for execution to Bob who executes them and then replies with the corresponding results $\{\rho(\mathtt{Q}_1),..., \rho(\mathtt{Q}_\mathtt{b})\}$.

There is a certain wisdom behind choosing a proper $\mathtt{b}$. For each batch of $\mathtt{b}$ queries, a query execution proof is required from Bob. If Bob does not execute any queries and instead attempts to "guess" a matching proof, its probability of success can be upper-bounded. More on this in Section 4.1.

[1] They return only "verbatim" (subsets of) the original data, no aggregation or other processing involved.

[2] "Correct execution over the entire data domain"[28].

Let the data be (conceptually) a collection of data segments $\mathbb{S} = \{\mathtt{S}_1, ..., \mathtt{S}_s\}$. This can be visualized for example by considering partitions of the data according to some attribute values (e.g., sales data by zip-code, or web-click data by source or date/time). Both Alice and Bob agree upon and are aware of this conceptual partitioning at the time the data is outsourced. Also, let us consider here the case where each batch of incoming queries target a particular data segment.

We call *identity query* any query that for a particular data set returns a un-aggregated "verbatim" subset of it (i.e., vertically partitioned), for example queries that return a selection of a set of records matching a given predicate (no aggregation or additional processing is performed in the data layer).

2.2 Cost Model

Let the server execution cost for a given read-only query $\mathtt{Q}$ over any segment of size σ be denoted by $\psi_{\mathtt{q}}(\mathtt{Q}, \sigma)$. Let the query execution cost for a read-only query $\mathtt{Q}$ at Alice's site (the server's client) be $\psi_{\mathtt{q}}^{\mathtt{owner}}(\mathtt{Q}, \sigma)$. Let $\psi_{\mathtt{i}}(\sigma) = \psi_{\mathtt{q}}(\mathtt{I}, \sigma) = \sigma$ be the cost of executing (linear pass) an identity query $\mathtt{I}$ over a data set of size σ. Let the cost of an update query $\mathtt{Q}$ over a data segment of size σ be $\psi_{\mathtt{u}}(\mathtt{Q}, \sigma)$ at the server site and $\psi_{\mathtt{u}}^{\mathtt{owner}}(\mathtt{Q}, \sigma)$ at the owner site respectively.

Due to their application specific nature, any generic assumptions about query result sizes are bound to be inaccurate. Let us consider query result sizes being proportional to the query inputs (i.e., $\gamma\sigma$). In reality it is often the case that the results are smaller. In the following, unless specified otherwise we are assuming the result output (for any query over a target segment of size σ) to be of size σ, i.e., $\gamma = 1$. Let the cost of transferring a certain amount of data of size σ over the network be $\psi_{\mathtt{net}}(\sigma) = \nu_{\mathtt{bw}}\sigma + \nu_{\mathtt{l}}$ (a combination of bandwidth and latency factors).

In this work we are considering workloads composed of compute intensive query batches, requiring both CPU and storage in comparable amounts. The natural cost model is such that Alice can at most afford to execute just a small percentage of the query load at its site and outsources the bulk of it to Bob (*significant cost* assumption):

> for a given query load and target data set, even if Alice would be in the possession of the entire data set, in terms of execution costs, it would still pay off to outsource the query load (or a large part of it) to Bob (instead of performing it locally).

Formally, we assume the cost model of the expected query executions is such that the amount of computation costs per query are of the same or greater order of magnitude with the data access costs: $\psi_{\mathtt{q}}(\mathtt{Q}, \sigma) > \psi_{\mathtt{q}}(\mathtt{I}, \sigma)$ and $\psi_{\mathtt{q}}^{\mathtt{owner}}(\mathtt{Q}, \sigma) > \psi_{\mathtt{q}}^{\mathtt{owner}}(\mathtt{I}, \sigma)$ for any query $\mathtt{Q} \neq \mathtt{I}$.

It is important to explore such compute-intensive queries as they (arguably) model more accurately real-world data processing scenarios in outsourced frameworks. This is due to both (i) the complexity of modern data processing jobs and the heterogeneity of associated data sources as well as (ii) the intrinsic nature of data outsourcing in itself. It is likely that the (likely resource-constrained) data clients will prefer to outsource as much as possible of their data access *and* processing needs rather than using the service provider as a simple storage device.

2.3 Adversary

A special type of adversarial behavior is of interest in such a framework. A malicious service provider Bob might have incentives to selectively avoid CPU and storage costs associated with query execution (e.g., maybe only for the more expensive queries) and reply with (cheaper) inexact or entirely incorrect results. In other words, Bob will not execute (some of) the queries submitted by its data clients and might execute the other ones only partially.

This is somewhat orthogonal to a malicious denial of service (DOS) behavior in which Bob might not deliver query results to the client even if it actually executed the associated queries. In other words, here we are mainly concerned with the core issues of *completeness* and *correctness* of query execution and do not directly tackle the orthogonal issue of DOS prevention. Deploying our solution however, has the potential to increase the cost of such DOS attacks, thus making them less likely. This is so because (as will be seen later on) to construct the required query execution proofs, a malicious Bob will have to actually correctly execute the query load in the first place.

2.4 Crypto-hashes

A special de-facto secure construct we are leveraging in our solution, is the *one-way cryptographic hash* (or simply "hash"). If $\mathtt{H}()$ is a cryptographic secure one-way hash, of interest are two of its properties: (i) it is computationally infeasible, for a given value $\mathtt{V}'$ to find a $\mathtt{V}$ such that $\mathtt{H}(\mathtt{V}) = \mathtt{V}'$ (one-wayness), and (ii) changing even one bit of the hash input causes random changes to the output bits (i.e., roughly half of them change even if one bit of the input is flipped). Examples of potential candidates for $\mathtt{H}()$ are the MD5 (fast) or the SHA class of hashes (more secure). The computation costs of computing most crypto hashes are linear in the size of the input. Let the cost of computing a crypto hash over a data set of size σ be $\psi_{\mathtt{h}}(\sigma) = \alpha_{\mathtt{h}}\sigma$ ($\alpha_{\mathtt{h}} \geq 1$). For more details see [34].

2.5 Extended Ringers

The *ringers* concept was first introduced in [16]. The main idea behind it is to provide computation proofs in a distributed computing market. In a basic version it works as follows: in a computation market composed of servers and clients, clients are willing to pay for computation services provided by servers. A client wishes to get one of these computations $\mathtt{h}()$ computed for a set of inputs, $\{\mathtt{x_1}, ..., \mathtt{x_a}\}$ by a service provider.

The initial assumption is that the *computations in the system are one-way non-invertible* functions. Under this assumption, to perform the computation a client first computes a challenge ("ringer") to be submitted along with the input data set to the service provider. This challenge is exactly the result of applying $\mathtt{h}$ to one of the inputs, $\mathtt{h}(\mathtt{x_t})$, where $\mathtt{t} \in [1, \mathtt{a}]$ is not known to the service provider. The implicit assumption here is that computing h for the entire input set is more expensive than for a single item in the input (e.g, if $\mathtt{i}$ is large enough).

The client then submits $\{\mathtt{x_1}, ..., \mathtt{x_a}\}$ *and* $\mathtt{h}(\mathtt{x_t})$ to the service provider. In addition to the normal computation results $\{\mathtt{h}(\mathtt{x_1}), ..., \mathtt{h}(\mathtt{x_a})\}$ the service provider is expected to return also (as a computation proof) the correct value for $\mathtt{t}$. Due to the assumed non-invertible nature of $\mathtt{h}$, a correct return provides a certain confidence of actual computation over the set of inputs.

The main power of the ringers lies in the assumed non-invertibility of the performed computations. To directly fake a proof (and produce a "valid" $\mathtt{t}$), the service provider would have to either: (i) act honest and perform $\mathtt{a}$ computations or (ii) cheat and perform only $\mathtt{0} \leq \mathtt{w} < \mathtt{a}$ computations hoping it finds the ringer in the process and if not simply guess as a last resort. The probability to succeed in such malicious behavior can be shown to be positively correlated to the amount of work performed. Over the course of multiple interactions it can be forced to arbitrary small values. For more details see Section 4.1.

Here we extend the ringers concept to (i) provide also proofs of actual *data access* (as opposed to computation only), (ii) work for arbitrary computations, by "wrapping" them in one-way cryptographic hash functions and (iii) avoid reuse of previously seen ringers. We lift the assumption of one-way non-invertibility for the computations in the system; $\mathtt{h}$ can be any function. The ringer challenge submitted to the service provider becomes now $\mathtt{H}(\mathtt{h}(\mathtt{x_t}))$ where $\mathtt{H}()$ is a one-way non-invertible cryptographic hash function. Thus, instead of the assumed one-wayness of computations, our extension puts the main power of ringers in the non-invertibility of the cryptographic hash deployed. We also extend the adversary model to consider "guessing" (see Section 3).

3 A Solution

An initial solution overview is illustrated in Figure 2. It proceeds as follows.

Data Preparation. Before outsourcing the data, in an initial step, for each data segment $\mathtt{S_i} \in \mathbb{S}$ the data owner computes and stores $\mathtt{H}(\mathtt{S_i})$, a segment "identity-hash", an authentication digest that can be later on used to authenticate identity query results for that particular segment. The main role of the identity-hashes is to provide authentication for identity queries in the later steps of the algorithm, not unlike the mechanisms deployed in [28].

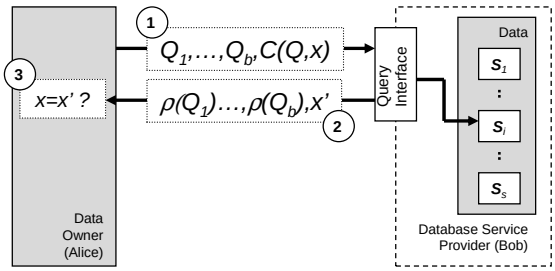

Figure 2: Solution Overview. A challenge token $\mathtt{C}(\mathbb{Q}, \mathtt{x}, \epsilon) = \{\mathtt{H}(\epsilon||\rho(\mathtt{Q_x})), \epsilon\}$ is sent together with the batch of queries. Upon return, query batch execution is proved if $\mathtt{x'} = \mathtt{x}$ holds.

Challenge-Tokens. For every batch $\mathbb{Q} = \{\mathtt{Q_1}, ..., \mathtt{Q_b}\}$ of queries to be executed over a data segment $\mathtt{S_i} \in \mathbb{S}$, the data owner picks a secret number $\mathtt{x} \in [1, \mathtt{b}]$ and a unique one-time use random nonce ϵ and computes a *query batch challenge token*,

$$\mathtt{C}(\mathbb{Q}, \mathtt{x}, \epsilon) = \{\mathtt{H}(\epsilon||\rho(\mathtt{Q_x})), \epsilon\}$$

where "||" denotes concatenation.

In other words Alice executes the corresponding query $\mathtt{Q_x}$ over the data segment $\mathtt{S_i}$ and computes a one-way cryptographic hash of the nonce concatenated with the query result. This token is then used in the next step to challenge the service provider to prove actual query execution. As will be detailed in the following, the purpose of the one-time use nonce ϵ is to avoid replays or "recordings" (by Bob) of previously used challenge tokens.

Let us also note here that, due to the ability to authenticate identity queries, Alice does not need to host a copy of the entire database to compute the challenge tokens. Instead she can do one of the following. (i) Periodically (e.g. when running "low" on tokens) retrieve a segment, generate a set of fake queries and compute and store the associated challenge tokens (for later use). This has the advantage of reducing communication costs. The drawback of such a scheme is that

it will incurr the execution of fake queries. This could be advantageous nevertheless, as Alice might want to do this at off-peak hours (e.g., nighttime) and precompute a large number of challenge tokens. The second option available is (ii) for each query batch to simply perform a identity query for the corresponding segment and the compute the challenge token at Alice's site. For cost evaluation purposes, as (i) is straight-forward, in the rest of the paper we chose to illustrate (ii) instead.

Query Submission. Next, the data owner submits the batch $\mathbb{Q} = \{\mathtt{Q}_1, ..., \mathtt{Q}_\mathtt{b}\}$ for execution to the service provider. It also submits its associated challenge token $\mathtt{C}(\mathbb{Q}, \mathtt{x}, \epsilon)$ (computed above). The service provider then executes the queries in the batch and also computes the value for $\mathtt{x}$, by identifying which of the queries maps to the received challenge token (it can do so by computing the associated challenge tokens for each executed query, for the given nonce ϵ). Let this value be $\mathtt{x}'$. This will constitute the main query execution proof.

Verification. The service provider then returns both the query execution proof $\mathtt{x}'$ and the query results for the batch, $\rho(\mathbb{Q}) = \{\rho(\mathtt{Q}_1), ..., \rho(\mathtt{Q}_\mathtt{b})\}$. The data owner verifies that indeed $(\mathtt{x}' = \mathtt{x})$; if true, it provides a statistical proof that the queries were performed correctly over their target data sets. This is so because, to answer correctly, the service provider has to be able to either compute the inverse of the one way crypto-hash used in the challenge token (impossible) or execute the query batch and find out which of the query results matches the given challenge token. In the following we explore this in more detail.

3.1 Unique Nonces

Let us first understand the need for one-time use nonces in the challenge token construct. This becomes clear if we consider a "smarter" Bob. In the absence of nonces Bob can apparently remember previously seen query batch challenge tokens. If enough storage capacity is available, as he performs queries and responds to challenges, he can "record" the associated tokens, for example by constructing a hashtable which maps (query,segment) pairs to challenge tokens. Eventually this will result in increasingly more tokens known to Bob. These can be used later on to respond to challenges (by a lookup in the hashtable) without the need to perform the associated work. In other words, if Alice happens to submit the same token twice, Bob can then return the answer directly without the requirement to do any actual work for the given query batch. Using an unique nonce for each challenge token guarantees that the computed token hash values become useless after their initial use, thus no recording at Bob's site is possible.

3.2 Cheating

To analyze the provided assurance levels, let us start by asking the following question: what are the chances of a malicious Bob to "get away" with cheating in the query execution phase? A "lazy" Bob will attempt to execute only $\mathtt{0} \leq \mathtt{w} < \mathtt{b}$ queries (less work) and try to find the query corresponding to the challenge token. If after $\mathtt{w}$ work, $\mathtt{Q}_\mathtt{x}$ is not found, simply guess $\mathtt{x}$ (among the remaining queries) and stop. The probability of success in returning the correct challenge response is

$$\mathtt{P}(\mathtt{w}, \mathtt{r}) = \frac{\mathtt{w}}{\mathtt{b}} + (\mathtt{1} - \frac{\mathtt{w}}{\mathtt{b}})\frac{\mathtt{1}}{\mathtt{b} - \mathtt{w}} = \frac{\mathtt{w} + \mathtt{1}}{\mathtt{b}}$$

This features (naturally) a linear behavior with respect to $\mathtt{w}$: the probability of cheating is naturally direct proportional to the amount of work performed. This is good news. Bob is forced into a position of requiring to do significant amounts of work to yield reasonable success. In the long term, over a large number of interaction instances (e.g. $\mathtt{v}$ of them), the chances of not getting caught (and thus the incentives to act maliciously) can be made arbitrary low.

3.3 Client-Side Checking

But why deploy query execution proofs when similar results can be achieved much simpler. In particular, let us consider the following alternate *client-side result checking* mechanism: for each batch of $\mathtt{b}$ queries, Alice simply verifies *one* of the query results. This will require the same type of setup (the ability to access original segment in an authenticated way) thus yield similar network costs but less server-side computation. Bob is now not required to compute challenge tokens anymore. The probability to get away with only $\mathtt{0} \leq \mathtt{w} < \mathtt{b}$ work done, will be similar to $\mathtt{P}(\mathtt{w}, \mathtt{r})$:

$$\mathtt{P}_\mathtt{c}(\mathtt{w}, \mathtt{r}) = \frac{\mathtt{w}}{\mathtt{b}}$$

The initial query execution proof construct introduced above thus proves to be less desirable and efficient than simple client-side result checking. But then why introduce execution proofs in the first place? In the following we show how the execution proof construct is but a first step towards a cheaper, more secure architecture. It can be built upon to actually provide much higher assurances than are possible with simple client-checking type of mechanisms.

But, before we do that, we need to extend the basic scheme to handle *multiple* challenge tokens. Instead of a single challenge token, Alice will now generate a *set* of tokens for Bob to reply to. An analysis of this mechanism is detailed in the following.

3.4 Multiple Challenge Tokens

If we assume now a number of $\mathtt{r} > \mathtt{1}$ challenge tokens per query batch, what is the success probability of lazy

behavior in this case?

Let us start by asking the question: For $\mathtt{r}$ challenge tokens, what is the probability of "finding" $\mathtt{x}$ of them by performing only $\mathtt{w} < \mathtt{b}$ work? In other words, what is the likelihood of simply finding them after doing less work than required. This can be modeled as a classical sampling experiment without replacement (retrieving $\mathtt{x}$ black balls out of $\mathtt{w}$ draws from a bowl of $(\mathtt{b}-\mathtt{r})$ white and $\mathtt{r}$ black balls):

$$\mathtt{P_0}(\mathtt{b},\mathtt{w},\mathtt{r},\mathtt{x}) = \frac{\binom{\mathtt{r}}{\mathtt{x}} \times \binom{\mathtt{b}-\mathtt{r}}{\mathtt{w}-\mathtt{x}}}{\binom{\mathtt{b}}{\mathtt{w}}}$$

where $\mathtt{x} \in [\mathtt{max}(0, \mathtt{w}+\mathtt{r}-\mathtt{b}), \mathtt{min}(\mathtt{r},\mathtt{w})]$. Additionally, we know the success probability of simple guessing of $\mathtt{r}$ challenge tokens without performing any work is (choosing $\mathtt{r}$ out of $\mathtt{b}$ items):

$$\mathtt{P_1}(\mathtt{b},\mathtt{r}) = \frac{1}{\binom{\mathtt{b}}{\mathtt{r}}}$$

A rational malicious Bob deploys the following strategy: do $\mathtt{w} < \mathtt{b}$ work (execute only $\mathtt{w}$ queries) and, if not all the tokens are discovered (possible if also $\mathtt{r} < \mathtt{w}$), simply guess the remaining ones. It can be shown that the success probability of such a strategy is

$$\begin{aligned}\mathtt{P}(\mathtt{w},\mathtt{r}) &= \sum_{\mathtt{i}=\mathtt{max}(0,\mathtt{w}+\mathtt{r}-\mathtt{b})}^{\mathtt{min}(\mathtt{r},\mathtt{w})} [\mathtt{P_0}(\mathtt{b},\mathtt{w},\mathtt{r},\mathtt{i}) \times \mathtt{P_1}(\mathtt{b}-\mathtt{w},\mathtt{r}-\mathtt{i})] \\ &= \frac{1}{\binom{\mathtt{b}}{\mathtt{r}}} \sum_{\mathtt{i}=\mathtt{max}(0,\mathtt{w}+\mathtt{r}-\mathtt{b})}^{\mathtt{min}(\mathtt{r},\mathtt{w})} \binom{\mathtt{w}}{\mathtt{i}}\end{aligned}$$

To better understand what this means we depicted the behavior of $\mathtt{P}(\mathtt{w},\mathtt{r})$ in Figures 3 and 4 for $\mathtt{b} = 20$. It can be seen that (e.g., for $\mathtt{r} = 5$) a significant amount of work (e.g., $\mathtt{w} > \frac{3}{4}\mathtt{b}$) needs to be performed to achieve even a 33% success probability. Figure 3 (b) illustrates

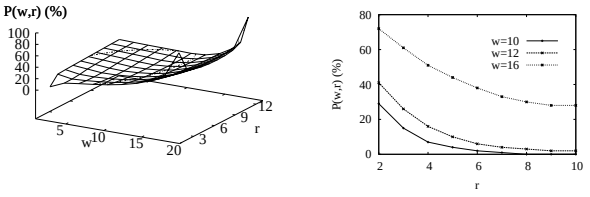

Figure 3: The behavior of $\mathtt{P}(\mathtt{w},\mathtt{r})$ (query batch size $\mathtt{b} = 20$). (a) 3-dimensional view, (b) inverse dependency of $\mathtt{r}$ to $\mathtt{w}$ (2-dimensional cut through (a)).

the inverse dependency on the number of challenge tokens $\mathtt{r}$ for specific values of performed work. The more challenges are presented to Alice, the less its probability of getting away with less work. Maybe more importantly, in Figure 4 the new behavior of $\mathtt{P}(\mathtt{w},\mathtt{r})$ is represented against the base case with a single token.

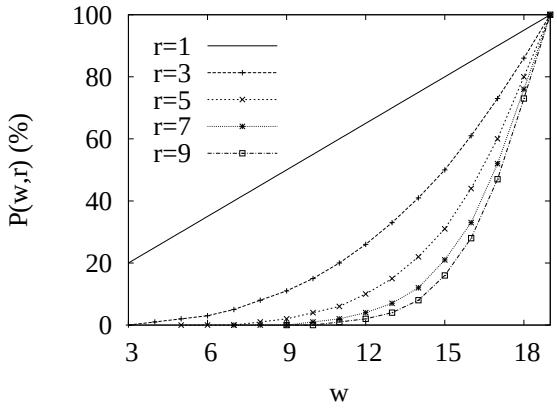

Figure 4: The behavior of $\mathtt{P}(\mathtt{w},\mathtt{r})$. A 2-dimensional cut through Figure 3 (a) showing the relationship between $\mathtt{P}(\mathtt{w},\mathtt{r})$ and the amount of performed work $\mathtt{w}$, plotted against the base case with one single challenge token ($\mathtt{r} = 1$).

It can be seen that, with more challenge tokens, (at the expense of performing more queries at the data owner's site), the shape of $\mathtt{P}(\mathtt{w},\mathtt{r})$ forces a malicious Bob to execute increasingly more queries to achieve the same success probability. For example, with 3 challenge tokens for the batch of 20 queries, to achieve a 40% probability, Bob now has to execute at least 14 queries in the query batch; in the single challenge token case, it could achieve the same probability by only executing 7 queries. Adding two challenge tokens doubled the amount of work required of Bob!

This is great news. Over multiple interactions (e.g. $\mathtt{v}$ of them), malicious behavior is not sustainable; the probability of getting caught increases exponentially and the cheating success probability converges to 0: $\prod_{\mathtt{i}=1}^{\mathtt{v}}(\mathtt{P}(\mathtt{w_i})) \to 0$, where $\mathtt{w_i} < \mathtt{a}$ is the work performed in each of the interactions.

While it is clear that additional tokens are aiding in decreasing the likelihood of undetected lazy behavior, in real life scenarios, the choice of $\mathtt{r}$ is also necessarily cost-driven. The trade-off between the additional cost of constructing challenge tokens at Alice's site and the guaranteed level of security (i.e. modeled ultimately by $(1 - \mathtt{P}(\mathtt{w},\mathtt{r}))$) needs to be custom tailored for each individual application.

But is this extension offering better assurances than the above client-side result checking method? After all, Alice could also simply *check* more results instead of computing more tokens. For $1 < \mathtt{r} \leq \mathtt{w}$ results checked by Alice [3], it can be shown that the probability of Bob getting away with only $\mathtt{w} < \mathtt{b}$ work is

$$\mathtt{P_c}(\mathtt{w},\mathtt{r}) = \frac{\binom{\mathtt{w}}{\mathtt{r}}}{\binom{\mathtt{b}}{\mathtt{r}}}$$

[3]It is obvious that if Alice checks more than $\mathtt{w}$ results she will find out if Bob cheated.

Because (it can be shown that) $\mathtt{P_c(w,r)} \leq \mathtt{P(w,r)}$ the answer to the above question is *no*. Multiple challenge tokens do not offer better assurances than simple client checking of multiple query results. The ability to handle multiple tokens is required however to build upon. In the following we show how this mechanism can be modified to yield much higher security assurances. We do so by introducing *fake challenge tokens*.

3.5 Fake Challenge Tokens

Before we proceed, let us first note an additional issue of concern with the above scheme. Because Bob knows the number of challenge tokens, once he finds all $\mathtt{r}$ of them it can simply stop working and directly reply with correct query execution proofs.

One simple yet effective solution to this problem is to add "fake" items to the set of challenge tokens In other words, at Alice's site, instead of executing $(\mathtt{r}+\mathtt{f})$ queries to construct $(\mathtt{r}+\mathtt{f})$ challenge tokens, execute just $\mathtt{r}$ queries and then simply generate $\mathtt{f} > 0$ random token-like items and add them to the set. This has the additional benefit of reducing Alice's query execution costs. Now Bob has to respond correctly only to the non-fake ones. Because he does not know which ones and how many of the challenges are fake, he is forced to execute all the queries to guarantee a correct answer (it cannot stop after it finds all the correct ones, as it doesn't know which ones and how many they are).

Introducing the fake challenge tokens solves the issue of Bob being able to simply stop after discovering all the tokens. It also offers higher security assurances. Let us explore why.

We start by looking at the impact of fake challenge tokens on the success probability of Bob's malicious behavior $\mathtt{P'(w,r,f)}$. To succeed, at each step, Bob needs now to first guess *exactly* what the value of $\mathtt{f}$ is. If he is off even by one and replies with a value to a fake token (instead of stating it is fake), Alice knows that Bob did not execute all the queries in the batch. It can be shown that:

$$\mathtt{P'(w,r,f)} = \frac{1}{\binom{b}{r}} \sum_{i=\max(0,w+r-b)}^{\min(r,w)} \left[\frac{\binom{w}{i}}{\min(b-w, \max(1, r+f-i))}\right]$$

where $\frac{1}{\min(b-w,\max(1,r+f-i))}$ is the probability of Bob guessing the value of $\mathtt{r}$ (and $\mathtt{f}$) after performing $\mathtt{w}$ work and discovering $\mathtt{i}$ correct challenge tokens. This is so because Bob knows that clearly $(\mathtt{r}-\mathtt{i}) \leq (\mathtt{b}-\mathtt{w})$ (number of remaining challenge tokens cannot exceed number of remaining un-executed queries). Then, there are $(\mathtt{r}-\mathtt{i})+\mathtt{f}$ remaining possible values for $\mathtt{f}$, only one of which is correct. The $\max()$ needs to be considered if $\mathtt{f}=0$ and Bob discovers all $\mathtt{r}$ tokens: it knows then that $\mathtt{f}=0$.

In Figure 5 we show the impact of deploying fake challenge tokens is by illustrating a scenario with 5 challenge tokens for both the normal case ($\mathtt{P(w,5)}$) and the case of 3 additional fake tokens ($\mathtt{P'(w,5,3)}$). Without fake challenge tokens, the cheating success probability climb towards 100% with increasing work is much faster. When using fake challenge tokens however, even after executing $\mathtt{w}=19$ queries (out of $\mathtt{b}=20$), Bob's uncertainty is still as high as 66% yielding an associated low success probability of only 33%.

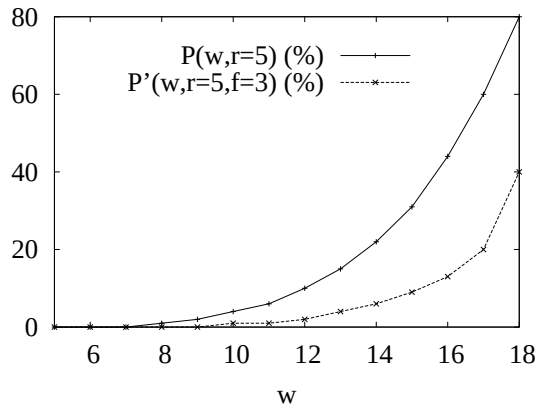

Figure 5: $\mathtt{P'(w,r,f)}$ and $\mathtt{P(w,r)}$ plotted for $\mathtt{r}=5$, $\mathtt{f}=3$.

To function properly, deploying fake challenge tokens requires the assumption that their number is random for every query batch and cannot be predicted by Bob. Alice has to make sure that throughout time, both $\mathtt{r}$ and $\mathtt{f}$ are randomly chosen and not correlated to each other or with their previous values.

Now let us see what the probability corresponding to the client-side result checking mechanism is. We discussed above that for $1 < \mathtt{r} \leq \mathtt{w}$ results checked by Alice, it can be shown that the probability of Bob getting away with only $\mathtt{w} < \mathtt{b}$ work is $\mathtt{P_c(w,r)} = \frac{\binom{w}{r}}{\binom{b}{r}}$. Because (it can be shown that) $\mathtt{P_c(w,r)} >> \mathtt{P'(w,r,f)}$ (for $\mathtt{f} > 1$), the level of assurance provided by query execution proofs is significantly higher than the levels that can be achieved using client-side result checking methods only. At the same time, while for each batch of queries the number of challenge tokens will be different, the average amount of work performed by Alice is going to be still $\mathtt{r}$ (where $1 < \mathtt{r} \leq \mathtt{w}$). With the same amount of work, query execution proofs offer much higher levels of assurance.

This result is illustrated in Figure 6 where the behavior of $\mathtt{P'(w,r,f)}$ is plotted against $\mathtt{P_c(w,r)}$ (client-side result checking mechanism) for $\mathtt{r}=2$ and $\mathtt{f}=2$. The addition of fake tokens decreases the ability to "get away" with less work. Because adding (as long as their number is random) more fake tokens comes at no cost, we also show in the same figure the case of $\mathtt{f}=4$. It can be seen that the probability of successful malicious behavior is further reduced to about half its

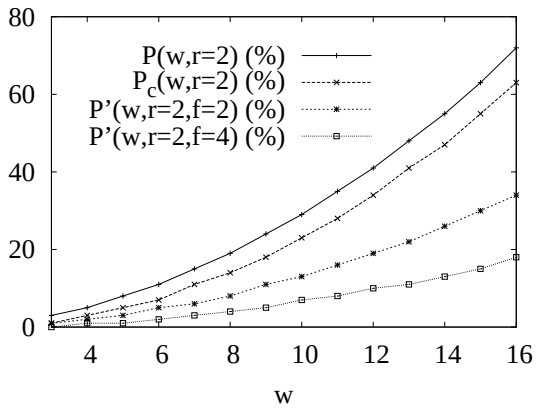

Figure 6: The behavior of $\mathtt{P}'(\mathtt{w},\mathtt{r},\mathtt{f})$ (fake tokens) plotted against $\mathtt{P_c}(\mathtt{w},\mathtt{r})$ (client-side result checking mechanism) showing that the query execution proof mechanism (with fake tokens) significantly decreases the ability to "get away" with less work.

value in the case of two fake tokens only ($\mathtt{f}=2$) and to about $\frac{1}{5}$th of the client-side result checking case.

In other words, by performing only 2 computations at the client ($\mathtt{r}=2$) and including 4 fake tokens in the request we achieve increased levels of assurance (roughly 5 times higher).

4 Discussion

4.0.1 Data Updates

The above solution does not consider the case of significant data updates. In the case of updates, additional issues need to be explored, mainly deriving from the fact that their presence will invalidate challenge tokens associated with the target data segments.

This is so because now Alice needs to make sure she can still authenticate segment identity queries (to construct challenge tokens). In a *unified client model*, where Alice can act as a "gateway", this can be naturally solved. As such it can continuously update its stored identity-hashes before it forwards the updates to Bob: before letting the data altering updates go through, Alice first issues a identity query (that can be verified by its current identity-hashes) and retrieves the target data segment. It then applies the updates to it, re-computes its corresponding identity-hash and also submits them to the service provider.

This however, will yield a set of additional costs. In Section 4.1 we analyze overheads in the case of at least one update per batch of read-only queries. Nevertheless, more efficient solutions to the issue of handling updates in such a secure outsourcing context should be devised. Existing efforts [14] [28] explore solutions for simple selection predicate query result authentication in a static read-only setting. To the best of our knowledge, the issue of handling data updates in authenticating predicate selection queries (which is required in our protocol as a subroutine for identity query authentication) has not been analyzed yet. Here we propose an initial step.

4.1 Analysis

In the following we explore query execution and communication costs. Let us consider a batch of $\mathtt{b}$ queries.

4.1.1 Query Execution

The query execution and computation costs include: (i) an identity query (Bob), (ii) $\mathtt{r}$ queries to construct the challenge tokens (Alice), (iii) the computation of $\mathtt{r}$ crypto-hashes over the $\mathtt{r}$ query results (Alice), (iv) executing the $\mathtt{b}$ queries (Bob) and, (v) the computation of $\mathtt{b}$ crypto-hashes to discover the ones corresponding to the challenge tokens (Bob).

Also, if we assume that for each query batch of $\mathtt{b}$ queries, on average at least one data-altering update will get executed, we incur these additional costs: (vi) an identity query to retrieve the segment corresponding to this update (Bob), (vii) applying the update over the retrieved segment (Alice), (viii) the computation of a crypto-hash to reconstruct its corresponding identity-hash (Alice), (ix) the actual execution of the update (Bob). This becomes:

$$\begin{aligned}\psi(\mathbb{Q}) = \; & \psi_{\mathtt{i}} + \mathtt{r}\psi_{\mathtt{q}}^{\mathtt{owner}} + \mathtt{r}\psi_{\mathtt{h}} + \mathtt{b}\psi_{\mathtt{q}} + \mathtt{b}\psi_{\mathtt{h}} \\ & + \psi_{\mathtt{i}} + \psi_{\mathtt{u}}^{\mathtt{owner}} + \psi_{\mathtt{h}} + \psi_{\mathtt{u}}\end{aligned}$$

In the traditional, non-secured case, for every $\mathtt{b}$ read-queries and 1 update query, the costs would have been composed of (iv) and (ix). The actual overhead incurred is:

$$\begin{aligned}\psi(\mathbb{Q}) = \; & \psi_{\mathtt{i}} + \mathtt{r}\psi_{\mathtt{q}}^{\mathtt{owner}} + \mathtt{r}\psi_{\mathtt{h}} + \mathtt{b}\psi_{\mathtt{h}} + \psi_{\mathtt{i}} + \psi_{\mathtt{u}}^{\mathtt{owner}} + \psi_{\mathtt{h}} \\ = \; & \sigma + \mathtt{r}\psi_{\mathtt{q}}^{\mathtt{owner}} + \mathtt{r}\alpha_{\mathtt{h}}\sigma + \mathtt{b}\alpha_{\mathtt{h}}\sigma \\ & + \sigma + \psi_{\mathtt{u}}^{\mathtt{owner}} + \alpha_{\mathtt{h}}\sigma \\ = \; & \sigma(2 + (\mathtt{r}+1)\alpha_{\mathtt{h}} + \mathtt{b}) + (\mathtt{r}\psi_{\mathtt{q}}^{\mathtt{owner}} + \psi_{\mathtt{u}}^{\mathtt{owner}})\end{aligned}$$

Because $\mathtt{r} < \mathtt{b}$, depending on the actual complexity of $\psi_{\mathtt{q}}^{\mathtt{owner}}$ and $\psi_{\mathtt{u}}^{\mathtt{owner}}$, the overhead can be dominated by either the first or the second term. If we normalize with respect to the identity query complexity and assume that the $\mathtt{r}$ queries and the update at Alice's site are *not* dominating [4] then the overheads are of order $\mathtt{O}(\mathtt{b})$ (mainly due to crypto-hashing).

4.1.2 Communication

Let us analyze the incurred communication costs. These are mainly composed of: (i) transferring an

[4] For example if they are $\mathtt{O}(\sigma)$ – linear in the size of the input, with possibly large constants.

identity query result (Bob to Alice) and (ii) transferring ($\mathtt{b} - \mathtt{r}$) of the batch query results [5] (Bob to Alice). As, for each query batch of $\mathtt{b}$ queries, we assume on average at least one data-altering update, we have the additional communication of (iii) transferring an identity query result (Bob to Alice):

$$\psi_{\mathtt{net}} = (\mathtt{b} - \mathtt{r} + 2)\psi_{\mathtt{net}}(\sigma)$$

In the traditional case the incurred costs would have consisted of transferring $\mathtt{b}$ batch query results from Bob to Alice. It can be seen that if $\mathtt{r} > 2$, our solution in fact reduces the communication costs, at the expense of additional load on Alice as discussed above. The incurred communication overheads are thus zero (or negative) in this case. This is due in no small part to the result to input size dependency factor assumed ($\gamma = 1$). If γ were just a factor let us see how this would impact the communication overheads:

$$\psi_{\mathtt{net}} = (2 + \gamma(\mathtt{b} - \mathtt{r}))\psi_{\mathtt{net}}(\sigma)$$

It can be derived from here that, for $\gamma > \frac{2}{\mathtt{r}}$, the overheads are still zero (in fact less communication happens with our solution). For $\gamma < \frac{2}{\mathtt{r}}$, the communication overheads are $(2 - \mathtt{r}\gamma)\ \psi_{\mathtt{net}}(\sigma)$. In Section 5 an experimental evaluation of these costs is shown.

5 Implementation and Results

We designed and built a proof of concept implementation of our solution (the *Secure Query Interface* briefly depicted in Figure 7). Its main components are (i) a client-side wrapper (**sqi.client.QueryClient**) that implements the client main steps outlined in Section 3 and, (ii) a server-side stub (**sqi.server.QueryServer**) representing the server-side of the challenge-token protocol, responding to challenge-tokens and executing queries by calling the underlying DBMS.

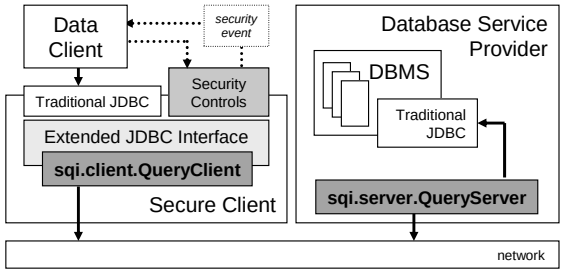

Figure 7: Proof-of-concept Overview: The QueryClient interfaces with the actual Service Provider. It exposes a security extension to a traditional query interface.

[5] There is no need to also transfer the results for the $\mathtt{r}$ queries that were used in computing the batch challenge tokens, as they are already known to Alice

To outside (legacy and security-aware) clients, the client-side wrapper exposes a traditional query submission interface. It accepts as input queries to be executed over a specific target data segment and returns the query execution results. If the query execution proof (challenge-token) protocol gets activated, an interface extension allows a client to determine the guaranteed correctness factor ($(1 - \mathtt{P}(\mathtt{w}, \mathtt{r} + \mathtt{f})$) see Section 4.1) for the returned query results. Additionally, a call-back mechanism is provided for the client to register to receive notifications of security events. One such event is the case of the service provider failing to correctly respond at any time in the challenge token protocol (security breach). Both the server and client side components are independent of and transparent to the underlying data layer and the data client. The challenge-token protocol runs behind the scenes, in parallel with any query submission events.

5.1 Experiments

To validate our solution and implementation we deployed it in the framework of a web transaction data mining scenario – constructing session profiles for individual web server sessions (with the ulterior purpose of predictive model scoring over the resulting data set). This scenario is important because often [10] (especially for large data sets) it makes sense to mediate (potentially public) access to this type of data through a specialized service provider.

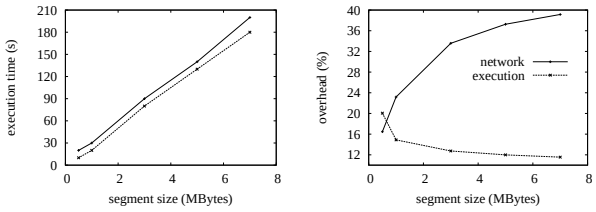

Figure 8: (a) Execution times behave naturally linear in the size of the input, (b) Execution time and network overheads behavior with increasing segment size ($\mathtt{r} = 1$).

Experiments were performed on general purpose 1Ghz+ linux boxes with approximately 512MB of RAM each in a local Ethernet network running at 100MBps during normal daily load [6]. A majority of code was written in Java. The deployed database was PostgreSQL version 7.4. We also confirmed that the code works with IBM DB2 8.2.

The data set used in the experiments was 42 days worth of web transaction data (log of our department's web server between 12/18/2003 and 01/27/2004), containing over 3.2+ million individual web hits (a tuple each). This data was vertically partitioned and

[6] In our experience, the application layer (TCP/IP) would not see more than 10 − 15MBps.

imported into the database as 42 (one per day) separate data tables, each indexed by the web transaction timestamp (augmented with a serial number to mediate collisions).

Each query execution over a given data set (table) involved also a CPU-intensive component: the invocation of a "web-session aggregation" UDF (DBMS user defined function) performing multiple read-only passes over the data, identifying an increasingly accurate set of web sessions (and associated web object accesses) corresponding to the given data set. The function returns a data set composed of multiple separate tuples for each identified web session (one tuple per session) containing additional aggregate statistics (e.g., average inter-click delay, data-transfer sizes).

Unless specified otherwise, for illustration purposes, in each of the experiments, the actual query load is a batch of $\mathtt{b} = 20$ queries (web session identification). The number of fake challenge tokens is $\mathtt{f} = 1$, as it does not have an impact in the actual costs. The web-click session discovery algorithm behaves linear (multiple passes) in the size of the input and can process roughly `80KBytes` of data (about 950 records) per second.

5.1.1 Overheads

To compute the incurred overheads we first submitted the same query load without any security controls. We then compared the resulting times with the ones obtained by running the queries through the secure interface (same client and server nodes).

In an initial experiment, we varied the amount of data used in each query computation over the data segments (of sizes over `7.5MBytes` each). Naturally, a virtually linear behavior can be observed (Figure 8 (a)), as both the crypto-hashing and the computation UDF behave linear in the size of the inputs.

For one challenge token ($\mathtt{r} = 1$), measured *execution time overheads* varied decreased slightly with increasing segment sizes as can be seen in Figure 8 (b). As segment sizes increase, the computation UDF execution times increase (linear), together with the hashing and identity query transfer times (see Figure 8 (a)). This yields a proportionally smaller actual overhead (as a fraction of total execution time). *Network overhead* on the other hand, increased with increasing segment size, mainly due to the identity query requirement (to construct the challenge token).

In the next experiment (Figure 9 (a)) we explored the *execution time overhead* behavior with increasing number of challenge tokens $\mathtt{r} > 1$ (for segment sizes of `1MByte`). It can be seen that increasing security assurances come naturally at the expense of additional computation. These overheads however are (arguably) quite reasonable considering the yielded security benefits. For example, it can be shown (see also Section 4.1) that for $\mathtt{r} = 4$ and a single fake extra token, a cheating Bob doing just half the work ($\mathtt{w} = \frac{\mathtt{b}}{2} = 10$) would only have a success probability of less than 5% (the incurred execution time overheads would be less than 25%).

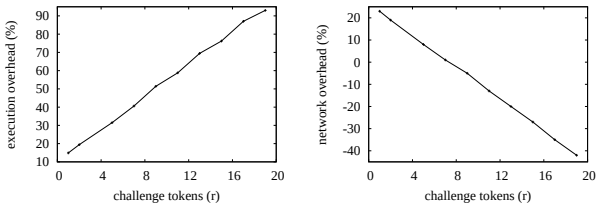

Figure 9: Overheads with increasing number of challenge tokens per batch: (a) execution overhead increases (b) network overhead decreases and eventually becomes negative.

On the other hand, in a multi-token scenario, *network overhead* behaves entirely different, as described in Section 4.1.2. Due to the proportionally less result data that needs to be transferred back to Alice, these overheads actually decrease with increasing number of challenge tokens. In Figure 9 (b) this virtually linear behavior is depicted. An actual reduction in network costs is observed as already predicted in Section 4.1.2.

Summary. Thus, there are two components of read query overheads, (i) additional network costs and (ii) cryptographic hashing costs. With increasing number of challenge tokens, network overheads decrease to zero. Cryptographic hashing costs bear thus much of the responsibility for the execution overhead. In our scenario these overheads were well within reasonable margins, e.g. 10% (for segments of 7 Mbytes and one challenge token). For a given data segment size, as the number of challenge tokens (and associated security guarantees) increases, so do execution overheads. As outlined above however, we believe the added security assurance is well worth it: e.g., cheating by only doing half of the work has a success probability of less than 5% with an overhead of under 25%.

6 Related Research

Extensive research has focused on various aspects of DBMS security, including access control techniques as well as general information security issues [7] [8] [13] [19] [20] [22] [23] [24] [29] [30] [33]. Additionally, increasing awareness of requirements for data storage security mechanisms and support can be found with DBMS vendors such as IBM [1] and Oracle [2]. In particular in [21] the authors focus on efficient mechanisms for secure storage and the inherent trade-off between an appropriate level of security and efficiency.

Additionally, (relatively recent) research efforts are to be found in the areas of statistical and "hippocratic" databases [5] and privacy preservation where issues have been identified early on [13] and more recently

also materialized, e.g., [6] [12].

The paradigm of providing a database as a service recently emerged [18] as a viable alternative, likely due in no small part to the dramatically increasing availability of fast, cheap networks. Given the global, networked, possibly hostile nature of the operation environments, security and integrity assurances have become paramount.

Few research efforts have been directly tackling the issues of authentication and completeness in a database outsourced model. A majority of existing efforts, focused on source data authentication and integrity. In [14] Merkle trees are deployed for the purpose of authenticating data published at a third party's site. In [26], the same authors explore a general model for authenticating data structures. In a publisher-subscriber model "hard to forge verification objects" ("search DAGs") are provided by publishers to prove authenticity and source guarantees of results.

In [17] a mechanism for querying encrypted data is discussed. This is an important component of privacy assurance in general (as the data can be stored encrypted at the server site). Queries are decomposed using operators that push as much as possible of the load to the server side without decrypting the actual data. The paper introduces a framework for query splitting to minimize client load.

In spirit, our work is also related to such efforts as SETI@Home [3] where the issue of computation correctness over outsourced data sets is of importance. There, the issue has been handled by majority voting over multiple instances of the same computation assigned randomly to computation nodes.

Finally, let us note that a (arguably more expensive) solution to the secure data outsourcing problem can be devised by deploying *Private Information Retrieval* (PIR) mechanisms [11] or similarly, *Byzantine Quorum Systems* [25]. Data could be split among multiple service providers (with associated assumptions of non-collusion of minimal subsets of them) and the query batch issued to all of the servers, followed for example by a quorum voting (or result aggregation) mechanism. Such a solution is explored by Aggarwal et. al. in [4]. The requirement for additional resources however, (e.g., at least a "few" additional independent non-colluding service providers) might sometimes prove impractical, as the main reason behind data outsourcing is often a limiting cost factor. It would be interesting to explore a "hybrid" solution, deploying query execution proofs in a PIR setting to provide both privacy and execution assurance.

Maybe the closest related effort is to be found in [28], which focuses on mechanisms for efficient integrity and origin authentication for simple selection predicate query results (specifically no aggregate queries or updates). Different signature schemes (DSA, RSA, Merkle trees [27] and BGLS [9]) are explored as potential alternatives for data authentication primitives (corresponding to our identity-hash mechanism). It is important to also note the fact that in this work the significance of guaranteeing query *completeness* ("the correct execution of the query over the *entire* target domain") is identified.

7 Conclusions

In this paper we proposed a solution for query execution assurance in outsourced database frameworks. It handles arbitrary query types and features reasonable overhead factors. While our solution mainly focuses on read-only queries, it also provides preliminary mechanisms for handling data updates (at additional costs). We introduced *query execution proofs*, a cryptographic proof mechanism that shows queries were actually executed. We implemented a proof of concept and performed experiments in a real-world mining application for web-click analysis.

More work is needed in understanding and dealing with complex query models (e.g., multi-querier and multi-owner) and more efficient mechanisms for handling update queries need to be designed. The relationship between query execution proofs and trust assessment mechanisms [32] should be explored. Additionally, deploying query execution proofs in private information retrieval (PIR) settings could yield an interesting "hybrid" mechanism for both privacy and execution assurance.

Additionally, an integration with security policy frameworks [15] [31] (e.g., also for querying possibly non-relational data) would allow for more complex constraint-specifications over the space of query proofs (e.g., which queries to include proofs for, what level of assurance is guaranteed or required by the client – maybe linked with a certain service level etc).

Ultimately, we believe this effort and others to bring the "database as a service" paradigm one step closer to being a viable *secure* alternative for data management.

References

[1] IBM Data Encryption for DB2. Online at `http://www.ibm.com/software/data/db2`.

[2] Oracle: Database Encryption in Oracle 10g. Online at `http://www.oracle.com/database`.

[3] SETI @ Home. Online at `http://setiathome.ssl.berkeley.edu`.

[4] Gagan Aggarwal, Mayank Bawa, Prasanna Ganesan, Hector Garcia-Molina, Krishnaram Kenthapadi, Rajeev Motwani, Utkarsh Srivastava, Dilys Thomas, and Ying Xu. Two can keep a secret: A distributed architecture for secure database services. In *Proceedings of the Second Conference on Innovative Data Systems Research*, 2005.

[5] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Hippocratic databases. In *Proceedings of the International Conference on Very Large Databases VLDB*, pages 143–154, 2002.

[6] Rakesh Agrawal and Ramakrishnan Srikant. Privacy-preserving data mining. In *Proceedings of the ACM SIGMOD*, pages 439–450, 2000.

[7] Elisa Bertino, M. Braun, Silvana Castano, Elena Ferrari, and Marco Mesiti. Author-X: A Java-Based System for XML Data Protection. In *IFIP Workshop on Database Security*, pages 15–26, 2000.

[8] Elisa Bertino, Sushil Jajodia, and Pierangela Samarati. A flexible authorization mechanism for relational data management systems. *ACM Transactions on Information Systems*, 17(2), 1999.

[9] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EuroCrypt*, 2003.

[10] Jim Challenger, Paul Dantzig, and Arun Iyengar. A scalable and highly available system for serving dynamic data at frequently accessed web sites. In *Proceedings of the 1998 High Performance Networking and Computing Conference*, Orlando, Fl, 1998.

[11] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *IEEE Symposium on Foundations of Computer Science*, pages 41–50, 1995.

[12] Chris Clifton, Murat Kantarcioglu, AnHai Doan, Gunther Schadow, Jaideep Vaidya, Ahmed Elmagarmid, and Dan Suciu. Privacy-preserving data integration and sharing. In *The 9th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pages 19–26. ACM Press, 2004.

[13] Chris Clifton and Don Marks. Security and privacy implications of data mining. In *Workshop on Data Mining and Knowledge Discovery*, pages 15–19, Montreal, Canada, 1996. Computer Sciences, University of British Columbia.

[14] Premkumar T. Devanbu, Michael Gertz, Chip Martel, and Stuart G. Stubblebine. Authentic third-party data publication. In *IFIP Workshop on Database Security*, pages 101–112, 2000.

[15] Irini Fundulaki and Maarten Marx. Specifying access control policies for xml documents with xpath. In *The ACM Symposium on Access Control Models and Technologies*, pages 61–69. ACM Press, 2004.

[16] Philippe Golle and Ilya Mironov. Uncheatable distributed computations. In *Proceedings of the 2001 Conference on Topics in Cryptology*, pages 425–440. Springer-Verlag, 2001.

[17] H. Hacigumus, B. Iyer, C. Li, and S. Mehrotra. Executing sql over encrypted data in the database-service-provider model. In *Proceedings of the ACM SIGMOD international conference on Management of data*, pages 216–227. ACM Press, 2002.

[18] H. Hacigumus, B. R. Iyer, and S. Mehrotra. Providing database as a service. In *IEEE International Conference on Data Engineering (ICDE)*, 2002.

[19] J. Hale, J. Threet, and S. Shenoi. A framework for high assurance security of distributed objects, 1997.

[20] E. Hildebrandt and G. Saake. User Authentication in Multidatabase Systems. In R. R. Wagner, editor, *Proceedings of the Ninth International Workshop on Database and Expert Systems Applications, August 26–28, 1998, Vienna, Austria*, pages 281–286, Los Alamitos, CA, 1998. IEEE Computer Society Press.

[21] B. Iyer, S. Mehrotra, E. Mykletun, G. Tsudik, and Y. Wu. A framework for efficient storage security in rdbms. 2003.

[22] S. Jajodia, P. Samarati, and V. S. Subrahmanian. A logical language for expressing authorizations. In *IEEE Symposium on Security and Privacy. Oakland, CA*, pages 31–42, 1997.

[23] S. Jajodia, P. Samarati, V. S. Subrahmanian, and E. Bertino. A unified framework for enforcing multiple access control policies. In *SIGMOD*, 1997.

[24] Li, Feigenbaum, and Grosof. A logic-based knowledge representation for authorization with delegation. In *PCSFW: Proceedings of the 12th Computer Security Foundations Workshop*, 1999.

[25] Dahlia Malkhi and Michael Reiter. Byzantine quorum systems. In *The ACM Symposium on Theory of Computing*, pages 569–578. ACM Press, 1997.

[26] C. Martel, G. Nuckolls, P. Devanbu, M. Gertz, A. Kwong, and S. Stubblebine. A general model for authenticated data structures. Technical report, 2001.

[27] R. Merkle. Protocols for public key cryptosystems. In *IEEE Symposium on Research in Security and Privacy*, 1980.

[28] E. Mykletun, M. Narasimha, and G. Tsudik. Authentication and integrity in outsourced databases. In *ISOC Symposium on Network and Distributed Systems Security NDSS*, 2004.

[29] M. Nyanchama and S. L. Osborn. Access rights administration in role-based security systems. In *Proceedings of the IFIP Workshop on Database Security*, pages 37–56, 1994.

[30] Sylvia L. Osborn. Database security integration using role-based access control. In *Proceedings of the IFIP Workshop on Database Security*, pages 245–258, 2000.

[31] Shariq Rizvi, Alberto Mendelzon, S. Sudarshan, and Prasan Roy. Extending query rewriting techniques for fine-grained access control. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 551–562. ACM Press, 2004.

[32] Staab S., Bhargava B., Lilien L., Rosenthal A., Winslett M., Sloman M., Dillon T.S., Chang E., Hussain F.K., Nejdl W., Olmedilla D., and Kashyap V. The pudding of trust. *IEEE Intelligent Systems*, 19(5):74–88, 2004.

[33] Ravi S. Sandhu. On five definitions of data integrity. In *Proceedings of the IFIP Workshop on Database Security*, pages 257–267, 1993.

[34] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms and Source Code in C*. Wiley & Sons, 1996.