Authentication and Integrity in Outsourced Databases

Einar Mykletun, Maithili Narasimha, Gene Tsudik
Computer Science Department
School of Information and Computer Science
University of California, Irvine
{mykletun,mnarasim,gts}@ics.uci.edu

Abstract

*In the **Outsourced Database** (ODB) model, organizations outsource their data management needs to an external service provider. The service provider hosts clients' databases and offers seamless mechanisms to create, store, update and access (query) their databases. This model introduces several research issues related to data security. One of the core security requirements is providing efficient mechanisms to ensure data integrity and authenticity while incurring minimal computation and bandwidth overhead. In this work, we investigate the problem of ensuring data integrity and suggest secure and practical schemes that help facilitate authentication of query replies. We explore the applicability of popular digital signature schemes (RSA and DSA) as well as a recently proposed scheme due to Boneh et al. [1] and present their performance measurements.*

1 Introduction

Continued growth of the Internet and advances in networking technology have fueled a trend toward outsourcing data management and information technology needs to external *Application Service Providers*. By outsourcing, organizations can concentrate on their core tasks and operate other business applications via the Internet, rather than incurring substantial hardware, software and personnel costs involved in maintaining applications *in house*. Database outsourcing [2] is a recent and important manifestation of this trend. In this model, the provider is responsible for offering adequate software, hardware and network resources to host the clients' databases as well as mechanisms for the client to efficiently create, update and access the outsourced data.

The database outsourcing paradigm poses numerous research challenges which influence the overall performance, usability and scalability. One of the foremost challenges is the security of stored data. A *client* stores its data (which is a critical asset) at an external, potentially untrusted, *Database Service Provider* site. It is essential to provide adequate security measures to protect the stored data from both malicious outsider attacks and the Database Service Provider itself. Security in most part, implies maintaining data integrity and guarding data privacy. Although some work [3] has been done to protect data confidentiality while guaranteeing its continued availability to authorized users, the problem of providing efficient integrity in this model has not received much attention.

In this paper, we focus on providing secure and effective means of ensuring data authentication and integrity, while incurring minimal computational and bandwidth overhead. In particular, we investigate techniques to help the ODB client authenticate the origin and verify the integrity of data returned by the service provider in response to a posed query.

Scope: In this paper, we consider only queries testing equality and other logical comparison predicate clauses. In other words, we consider the standard SQL queries involving *SELECT* clauses which typically result in selection of a set of records (or attributes) matching a given predicate or a set thereof. We specifically do not address queries that involve any kind of data aggregation (such as SUM or AVERAGE).

This paper focuses on providing mechanisms for efficient integrity and origin authenticity of query replies returned by the service provider in the ODB model. A related, and equally important, issue is the *completeness* of query replies. The term *completeness* refers to the correct execution of the query over the entire target domain, i.e., whether or not **all** records satisfying a query are returned. Although we consider it debatable whether *completeness* is a security concern, we acknowledge that it poses a challenge which needs to be addressed in the ODB model. However, it remains beyond the scope of this paper.

Organization: The rest of the paper is organized as follows: In section 2, we describe the details of our model and identify design requirements for the integrity mechanisms. Section 3 describes the motivation for studying aggregate signature schemes. We look at our design choices and our assumptions in section 4 while section 5 covers the overhead factors and design features that go along with aggregate signature schemes. In section 6, we discuss aggregation of RSA signatures, then describe an aggregate signature scheme proposed by Boneh et al. [1] and finally discuss why aggregation of DLP based signatures schemes is difficult. In section 7, we briefly discuss another viable solution for ensuring integrity in ODB model which uses *Merkle Hash Trees*. In section 8, we compare and contrast the relative performance of the three signature schemes described in section 6. Section 9 looks at relevant prior work in the fields of database security, batch cryptography and aggregated signatures. We finally conclude in section 10 by giving an outline for our future work.

2 System Model

The outsourced database (ODB) model is an example of the well-known Client-Server model. In the ODB model, the *Database Service Provider* (also referred to as the Server) has the infrastructure required to host outsourced databases and provides efficient mechanisms for clients to create, store, update and query the database. Before proceeding further, we need to clarify the meaning of the term "client". A client, in this context, is not necessarily a single entity, such as a user. Instead, a logical client can be thought of as an administrative entity, such as an organization or a set of authorized users within an organization.

In practice, an actual client may be a computationally weak and storage-challenged device, such as a cellphone or a wireless PDA. Moreover, the bandwidth available to a client may be severely limited due to the communication medium characteristics and/or the battery power consumed by receiving large amounts of data.

An ODB client is assumed to trust the server to faithfully maintain its data. Specifically, the server is relied upon for the replication, backup and availability of outsourced databases it stores. However, the server might not be trusted with the actual database contents and/or the integrity of these contents. This lack of trust is crucial as it opens up new security issues and serves as the chief motivation for our work.

We distinguish among three flavors of the ODB model: The most basic setting is where each outsourced database is used by a single entity, the client who creates, manipulates and queries the data. We refer to it as the *Unified Client Model* (Figure 1). In a more advanced, *Multi-Querier Model* (Figure 2), there are two types of clients: *owners* and *queriers*. The former is the actual data owner who adds, deletes and updates database records. Whereas, a querier is a client only allowed read access (i.e., query) to the database or portions thereof. This distinction is both necessary and natural as it reflects many real-world database scenarios. For example, an airline may allow its customers to query its database to check flight timings and availability or view specific information about itineraries and reservations. However, customers are, of course, not authorized to modify any flight-related data.

In the third, most general, ODB model, a single database can have multiple owners. This is referred to as the *Multi-Owner Model* (see Figure 3). The distinction between the Multi-Querier and Multi-Owner models might seem subtle. In the former, a single security principal creates and manipulates database records, whereas, in the latter, different records can be created by distinct security principals. However, in both models, there can be multiple queriers. The motivation for the Multi-Owner model is rather straightforward. Consider an example of an outsourced customer/sales database. Each record in this database is created and maintained by the salesperson responsible for a particular customer. A salesperson then "owns" the records that s/he creates.

3 Motivation

As mentioned above, the ODB model triggers some important security concerns. The more obvious concern is the privacy of outsourced data with respect to the server. The main challenge is how to reconcile the requirement for privacy with the need to outsource data. Some notable previous work [2, 3] addressed this challenge by devising methods for essentially running encrypted queries over encrypted databases. (Albeit, much work remains to be done to support aggregation-style queries.)

The focus of this paper is on data authentication and integrity in the ODB model. It is easy to see that data privacy in the ODB model is orthogonal to data authentication and integrity. We consider it unlikely that there are many realistic ODB settings where privacy is desired but integrity is not. On the other hand, one can easily envision ODB scenarios where privacy is not a concern but data authentication and integrity are required. For example, a government agency, e.g., the National Science Foundation (NSF), might outsource its awards database and allow anyone to run queries over it. A similar situation might arise if the US Patents and Trademarks Office (USPTO) were to outsource its issued patents and patent applications databases. In such cases, privacy is not an issue since the database contents (records) are public, yet their integrity and authenticity are very important.

From a more technical perspective, the motivation for

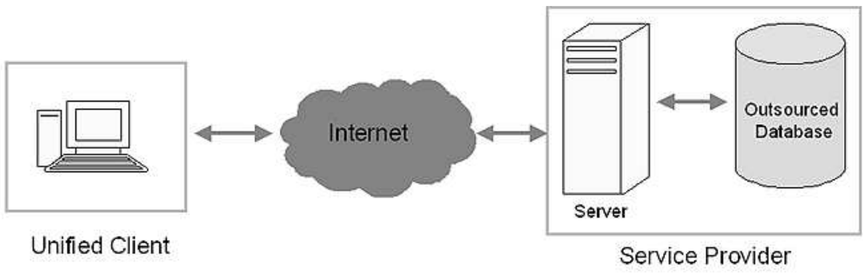

Figure 1. ODB – Unified Client Model Overview

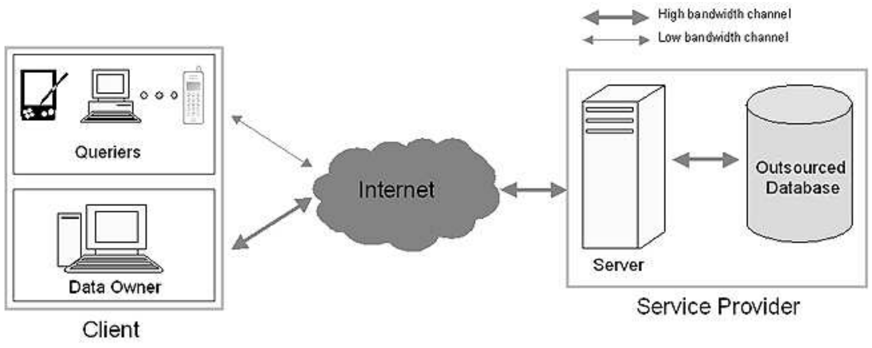

Figure 2. ODB – Multi-Querier Model Overview

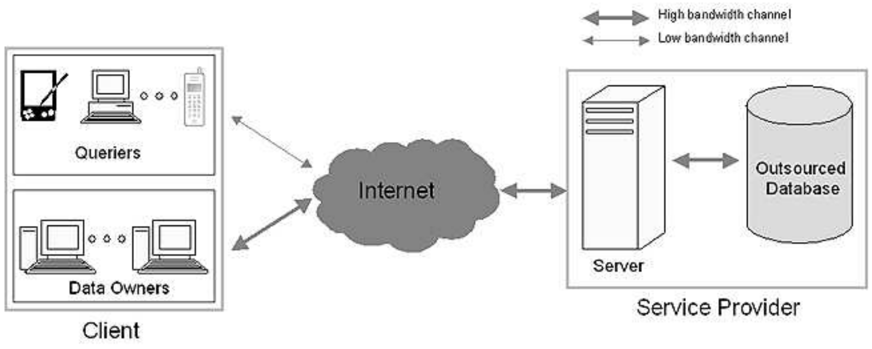

Figure 3. ODB – Multi-Owner Model Overview

our work is fairly intuitive. When an ODB client queries its outsourced data, it expects in return a set of records (query reply) satisfying the query's predicates. The size of the reply can vary, in principle, between zero and m, where m is the total number of records in the database. In other words, a query reply can be any one of the 2^m record subsets. Therefore, the main problem we face is: *how to facilitate* **secure** *and* **efficient** *authentication of all possible query replies?*

If we were concerned only with outside threats, i.e., adversaries tampering with the communication between queriers and servers, then standard tools for establishing secure channels (e.g., SSL or IPSec) might be appropriate. However, this is insufficient to authenticate query replies since, in our model, the server is NOT trusted with the integrity of the data. More concretely, the problem at hand is that a malicious server may attempt to insert fake records into the database or modify existing records. Consequently, our goal is to assure ODB clients that data they receive from the server has not been tampered with by either an external adversary or by the server itself.

4 Design Choices and Assumptions

In this section we discuss several assumptions and design choices.

Nature of Data: as mentioned earlier, this paper does not address the issue of data privacy. Consequently, to keep our discussion general, we do not differentiate between the cases of the stored data being completely encrypted, partially encrypted or completely unencrypted. We simply assume that the data owner, depending upon the nature of data, has stored it in some suitable form and has a mechanism for formulating queries (involving selection predicates) that can be executed *remotely* over the stored data.

Granularity of Integrity: data integrity/authentication can be provided at different levels of granularity. In principle, integrity checks can be at the level of a table (entire relation), a column (an attribute of the relation), a row (a record or a tuple of the table), or finally, an individual attribute value. Providing integrity checks at the table (or column) level implies that the entire data pertaining to that table (or column) should be returned in the query reply in order for the client to verify the integrity of the query response. This is clearly impractical as it requires transferring large amounts of data to the client. Hence, we do not consider this to be a viable approach. On the other hand, computing integrity checks at the level of individual attribute values yields a very large number of signatures which is very expensive for the signer (owner) in terms of computation as well as for the the server in terms of storage.

We believe that the optimal choice is to provide integrity at the record level. This enables the server to return – in response to a query – any set of matching records along with their respective integrity checks. Of course, computing integrity checks over the entire record, as opposed to individual attributes, implies that the smallest unit of data returned as a query reply is an entire record, even when the querying client is only interested in a single field.[1]

Signatures or MAC-s? One natural and intuitive solution for record-level integrity is to use message authentication codes (MAC-s) as they tend to be small and efficient to compute and verify. A simple MAC-based scheme can be constructed whereby the client asks the server to store each outsourced record along with a MAC (or keyed hash) of that record computed with some key known only to the client. Then, for any query reply, the server encloses a single integrity check computed as a hash of all record-level MAC-s in the query reply. The bandwidth overhead is minimal and the computation overhead at the client is low. This works well for the the *Unified Client* Model where the client and the querier are one and the same.

However, recall that in the more general *M*ulti-Querier and *M*ulti-Owner models, we assume potentially many queriers for each client. In these settings, MAC-s are not useful since they would require the MAC key to be shared among all owner(s) and all legitimate queriers. Obviously, non-repudiation for the queriers can not be achieved. Therefore, the only choice is use public key digital signatures. However, digital signatures introduce significant overhead in terms of storage, bandwidth and computation.[2]

5 Overhead Factors and Desired Features

A typical digital signature is between 320 (e.g., DSA) and 1024 (e.g., RSA) bits. As discussed above, we are using record-level signatures. Now, considering that a query reply can, potentially, contain many thousands of records, receiving and verifying individual record signatures can be prohibitively expensive for a querier. Also, even if the storage on the server is not a resource constraint, the querier can be a weak device with limited storage and computational resources. Therefore, it is essential to reduce the bandwidth as well as computational overhead introduced by the security services. To be more specific in our goals, we consider five overhead factors that we would like to minimize (listed in the decreasing order of perceived importance):

1. Querier computation: verifying integrity/authenticity of a set of records in a query reply.

2. Querier bandwidth: sending/receiving integrity data (in addition to the overhead incurred for sending/receiving actual records in a reply).

3. Server computation: server-side manipulation (if any) of integrity information in the query reply.

4. Owner computation: computing integrity information to be stored in the outsourced database.

5. Server storage: space required to store integrity information in an outsourced database.

We claim that the first three are the most important overhead factors stemming from the integrity of outsourced data. The last two are markedly less important. Server storage overhead is not much of a concern since servers are assumed to be interested in selling or renting more storage. Owner computation overhead is also comparatively less pressing since we assume that ODB records are manipulated (created, deleted, modified) much less frequently than client

[1]It is possible to compute integrity checks over hashes of record attributes rather than over actual attribute values. Even in this case, if the query requests only some attributes, hashes of other (un-requested) attributes must be returned.

[2]We observe that, even in the *Unified Client* model, there may be benefits to using signatures as opposed to MAC-s. The main incentive is to protect the server from a malicious (or litigious) client who might fraudulently claim that the server mangled or faked records in the outsourced database. Record-level signatures would obviate this problem.

queries are executed. This observation favors digital signature schemes that are particularly efficient in verification. Efficiency in generating (computing) signatures is, in comparison, less important.

Based on the above, we can outline an idealized solution. It would involve minimal querier computation overhead and constant (only in terms of integrity information) querier bandwidth overhead. It is easy to see that achieving constant querier computation overhead is impossible, since, for all records in a query reply, the querier must at least recompute a hash or a similar function. Fortunately, there are digital signature schemes that can be used to construct idealized or near-idealized solutions. Such schemes allow combining (or **aggregation**) of multiple individual signatures into one *unified* signature such that verification of the unified signature is *equivalent* to verifying individual component signatures. Some schemes only allow aggregation of a single signer's signatures, whereas, others allow signatures produced by multiple signers to be aggregated. For our purposes, the former can be used to support the *Unified Client* and *Multi-Querier* models, while the latter can additionally support the *Multi-Owner* model.

In the remainder of this paper we focus on two concrete techniques that support aggregation of multiple signatures. The first is based on a simple extension of the well-known "Batch verification of RSA" method, and the second is the recent scheme by Boneh, et al. [1].

6 Suitable Signature Schemes

As noted in section 5, signature schemes that allow aggregation seem to be ideally suited for ODB integrity. In this section, we examine some viable solutions and identify their limitations. We start by presenting a simple scheme for aggregating RSA signatures generated by a single signer. We then introduce an aggregated signature scheme due to Boneh et al. (hereafter referred to as the BGLS scheme), and conclude with the discussion of the applicability of DLP-based signature schemes in the ODB model. An alternative approach based on so-called *Merkle Hash Trees* is discussed in Section 7.

6.1 Condensed-RSA

The Condensed-RSA scheme discussed in this section is a simple extension of the standard RSA [4] scheme. One of the well-known features of RSA is its multiplicative homomorphic property. This feature makes RSA suitable for combining signatures generated by a single signer into a single "condensed" signature. Having successfully verified a condensed signature, a verifier can be assured that each individual message covered by the condensed signature was, indeed, properly signed by the purported signer. RSA signatures can be condensed "incrementally" by any party in possession of individual message signatures.

Standard RSA: We now briefly summarize the standard RSA scheme. An entity has a public key $pk = (n, e)$ and a secret key $sk = (d)$, where n is a k-bit modulus computed as the product of two random $k/2$-bit primes p and q. The respective public and secret exponents $e, d \in Z_n^*$ satisfy $ed \equiv 1 \mod \phi(n)$, where $\phi(n) = (p-1)(q-1)$. In today's cryptographic literature, k is assumed to be at least 1024 bits. The security of the RSA cryptosystem is widely believed to be based on the conjectured intractability of the integer factorization problem.

In practice, an RSA signature is computed over the hash of the input message. Let $h()$ denote a suitable cryptographic hash function (such as MD5 or SHA-1) which, upon a variable-length input m, produces a fixed-length output $h(m)$. A standard RSA signature on message m is computed as: $\sigma = h(m)^d \pmod n$. RSA signature verification involves checking that $\sigma^e \equiv h(m) \mod n$. Both signature generation and verification involve computing one exponentiation.

Condensed-RSA: Given t input messages $\{m_1, ..., m_t\}$ and their corresponding signatures $\{\sigma_1, ..., \sigma_t\}$ (generated by the same signer), a Condensed-RSA signature is given by the product of individual signatures:

$$\sigma_{1,t} = \prod_{i=1}^{t} \sigma_i \pmod n \qquad (1)$$

The resulting signature $\sigma_{1,t}$ has the same size as a standard RSA signature. When verifying a condensed signature, the verifier needs to multiply the hashes of all input messages and check that:

$$(\sigma_{1,t})^e \equiv \prod_{i=1}^{t} h(m_i) \pmod n \qquad (2)$$

Batch verification of RSA signatures: Condensed-RSA verification is very similar to *Batch* verification of RSA. Batching, in general, helps reduce computational complexity in settings where many signature verifications (or other computationally intensive tasks) must be performed simultaneously. Batch verification of RSA [5, 6, 7] aims at speeding up the verification process by reducing the total number of exponentiations. Given a batch instance of signatures $\{\sigma_1, ..., \sigma_t\}$ and distinct messages $\{m_1, ..., m_t\}$, an RSA batch verification (also called *fast screening* [6]) consists of checking that:

$$(\prod_{i=1}^{t} \sigma_i)^e \equiv \prod_{i=1}^{t} h(m_i) \pmod n \qquad (3)$$

Bellare et al. in [6] give the exact bounds for the probability that an adversary can successfully create a batch instance which satisfies the above equation without possessing individual signatures on each of the messages in the instance. The main difference between Condensed-RSA and Batch verification of RSA is that, in the latter, the product of individual signatures is computed by the verifier who has access to individual signatures. This allows for post-verification auditing if batch verification fails, i.e., it is possible to screen individual signatures to determine the faulty one(s). In contrast, Condensed-RSA has a single aggregated signature which makes auditing impossible. Therefore, in the event of verification failure, further steps (perhaps involving off-line mechanisms) become necessary.

Security of Condensed-RSA: We claim that *Condensed-RSA is unforgeable against adaptive chosen message attacks*. We support this claim by showing that, if an adversary $\mathcal{A}$ can break Condensed-RSA, then, using this adversary, we can construct a forger $\mathcal{B}$ to successfully create a batch instance that passes the batch verification test without the possession of valid individual signatures on each input message in the instance.

We begin by defining what it means for an adversary $\mathcal{A}$ to *break* Condensed-RSA. $\mathcal{A}$ succeeds in breaking Condensed-RSA if it produces a valid aggregated signature for messages $\{m_1, ..., m_t\}$ which satisfies equation 2 without possessing valid individual signatures for each of the t input messages. We argue the security of Condensed-RSA by demonstrating that it is at least as secure as Batch verification of RSA, which, in turn, was shown in [6] to be secure under the assumption that RSA is a collection of one-way functions. Note that we are assuming the use of a full-domain hash function (FDH), as described in [8]. FDH is a hash function $H_{FDH} : \{0,1\}^* \rightarrow Z_n^*$

Outline: $\mathcal{A}$ accepts as input $(m_1, ..., m_t)$ and $\theta = \{\sigma_1, ..., \sigma_w\}$ where $\sigma_i = H_{FDH}(m_i)^d \mod n$, and $w < t$. In other words, adversary $\mathcal{A}$ accepts as input t messages and a set θ which has valid signatures for w of these t messages $(w < t)$. $\mathcal{A}$, by our definition, breaks Condensed-RSA by outputting a valid Condensed-RSA signature $\sigma_{1,t}$. We now construct a forger $\mathcal{B}$ that breaks Batch verification of RSA.

Details: Forger $\mathcal{B}$, on input $(m_1, ..., m_t)$ and θ, outputs $(s_1, ..., s_t)$ such that $(\prod_{i=1}^{t} s_i)^e \equiv \prod_{i=1}^{t} H_{FDH}(m_i) \pmod n$. $\mathcal{B}$ proceeds in the following manner:

1. Creates random numbers $s_i \in_R Z_n^*$ where $1 \leq i \leq (t-1)$.
2. Transfers messages $(m_1, ..., m_t)$ and θ to $\mathcal{A}$
3. Let the forged Condensed-RSA signature returned by $\mathcal{A}$ be X
4. Computes $s_t = (\prod_{i=1}^{t-1} s_i)^{-1} * X \pmod n$
5. Outputs a batch instance $(s_1, ..., s_t)$ for messages $(m_1, ..., m_t)$.

Claim 1: Forger $\mathcal{B}$ produces a set of signatures that satisfy the Batch verification test.
Note that $\prod_{i=1}^{t} s_i \equiv \prod_{i=1}^{t} \sigma_i \pmod n$. Therefore, equation 3 is satisfied and the Batch verification test succeeds.

Claim 2: If RSA is one-way, then Condensed-RSA is a secure signature scheme.
Batch verification of RSA was shown secure under the assumption that RSA is one-way. Therefore, since Condensed-RSA is at least as secure as Batch verification of RSA, we conclude that Condensed-RSA is secure assuming RSA is a collection of one-way functions.

Overhead Costs: We now compare the costs of Condensed-RSA in the ODB model with those of standard RSA and Batch verification of RSA (also in ODB). The cost of hashing is ignored as it is negligible in comparison with that of modular arithmetic.

With standard RSA signatures, the querier would need to receive and process t signatures, one for each record in the query reply. To verify the signatures, it would have to perform t RSA verifications, i.e., t exponentiations. The bandwidth overhead would amount to $t * |n|$ bits. With Batch verification of RSA, the verification process would involve computing the product of all message hashes and the product of message signatures, resulting in $2(t-1)$ multiplications, followed by one exponentiation. The bandwidth overhead is the same as in standard RSA.

In contrast, the bandwidth overhead of Condensed-RSA is a single signature ($|n|$ bits), and verification costs would amount to $(t-1)$ multiplications, to compute the product of all message hashes, followed by one exponentiation. To summarize, Condensed-RSA saves $(t-1) * |n|$ bits of bandwidth and $(t-1)$ multiplications over Batch verification of RSA.

Condensed-RSA in ODB Setting: Condensed-RSA is clearly applicable to both Unified-Client and Multi-Querier models since each assumes a single data owner (signer). The server executing a client query is required to perform the following: select records that match the query predicate; fetch the signatures corresponding to these records; aggregate them (by multiplying them, as mentioned above) and send back the single aggregated signature along with the records in the result set. Note that, clearly, Condensed-RSA helps save both querier computation and querier bandwidth (Section 5). In case of Multi-Owner model, it is no longer possible to incur a constant bandwidth overhead. The server can aggregate signatures generated by each signer and send them separately which results in querier bandwidth overhead being linear in the number of signers. The client can verify these *partially* aggregated signatures by carrying out one verification per signer.

6.2 BGLS

Boneh, et al. in [1] describe an aggregated signature scheme that aggregates signatures generated by distinct signers on different messages into one short signature based on elliptic curves and bilinear mappings. Their scheme operates in a Gap Diffie-Hellman group (GDH) which is defined as a group where the Decisional Diffie-Hellman problem (DDH) is easy while the Computational Diffie-Hellman problem (CDH) is hard. The first such GDH group was described in [9]. Prior to describing their signature scheme we give a brief introduction to the parameters involved [1].

1. G_1 and G_2 are two (multiplicative) cyclic groups of prime order p;

2. g_1 is a generator of G_1 and g_2 is a generator of G_2;

3. ψ is a computable isomorphism from G_2 to G_1, with $\psi(g_2) = g_1$; and

4. e is a computable bilinear map $e : G_1 \times G_2 \rightarrow G_T$ as described below

A bilinear mapping $e : G_1 \times G_2 \rightarrow G_T$, where $|G_1| = |G_2| = |G_T|$, satisfies the following properties.

1. Bilinearity: $\forall p \in G_1,\ q \in G_2$ and $a, b \in \mathbb{Z}$, $e(p^a, q^b) = e(p, q)^{ab}$

2. Non-degenerate: $e(g_1, g_2) \neq 1$

These two properties imply that:
$\forall p_1, p_2 \in G_1, q \in G_2, e(p_1 p_2, q) = e(p_1, q) \cdot e(p_2, q) \text{and} \forall p, q \in G_2, e(\psi(p), q) = e(\psi(q), p)$

BGLS Scheme: BGLS uses a full-domain hash function $h() : \{0,1\}^* \rightarrow G_1$. Key generation involves picking a random $x \in \mathbb{Z}_p$, and computing $v = g_2^x$. The public key is $v \in G_2$ and the secret key is $x \in \mathbb{Z}_p$. Signing a message m involves computing $h = h(m)$, where $h \in G_1$ and $\sigma = h^x$ (the actual signature is σ). To verify a signature one computes $h = h(m)$ and checks that $e(\sigma, g2) = e(h, v)$.

BGLS Aggregation: To aggregate t BGLS signatures, one computes the product of individual signatures as follows: $\sigma_{1,t} = \prod_{i=1}^{t} \sigma_i$, where σ_i corresponds to the signature on message m_i. The aggregated signature $\sigma_{1,t}$ is of the same size as an individual BGLS signature, i.e., $|p|$ bits. Similar to Condensed-RSA, aggregation can be performed incrementally and by anyone.

Verification of an aggregated BGLS signature $\sigma_{1,t}$ involves computing the product of all message hashes and verifying the following equality: $e(\sigma_{1,t}, g_2) = \prod_{i=1}^{t} e(h_i, v_i)$. Due to the properties of the bilinear mapping, we can expand the left hand side of the equation as follows:
$e(\sigma_{1,t}, g_2) = e(\prod_{i=1}^{t} h_i^{x_i}, g_2) = \prod_{i=1}^{t} e(h_i, g_2)^{x_i} = \prod_{i=1}^{t} e(h_i, g_2^{x_i}) = \prod_{i=1}^{t} e(h_i, v_i)$

BGLS Performance When analyzing the cost of BGLS signature verification, we distinguish between the two operations: multiplication and computation of the bilinear map. For a single-signer BGLS signature and t input messages, verification costs amount to computing the product of message hashes ($t - 1$ multiplications), followed by two bilinear mappings. For a multiple-signer BGLS signature (with k signers and t signatures per signer), verification costs amount to $(k * t - 1)$ multiplications as well as $k + 1$ bilinear mappings.

BGLS in ODB Setting: BGLS is applicable to all three ODB models. The server executing a client's query by selecting records (along with their signatures) that match the query predicate; aggregates them (as described above) and sends back a single aggregated signature along with all records in the result set. BGLS saves both querier computation and querier bandwidth in case of Unified-Client as well as Multi-Querier models. However, although in the Multi-Owner model, BGLS provides bandwidth efficiency, it does not offer much savings in terms of querier computation. As mentioned above, verification of a BGLS signature which includes messages by k distinct signers involves computing $k + 1$ bilinear mappings, which is quite costly (see Section 8 below).

6.3 Whither DLP-based Signatures?

Batch verification of DSA signatures was introduced by Naccache et al. [10]. Online generation of DSA or ElGamal-type signatures can be made efficient due to some pre-computation techniques. Whereas, verification is usually far less efficient; a typical signature verification requires 2 exponentiations. Therefore, batching of multiple signature verifications, which is much more efficient than sequential verification of multiple signatures, becomes particularly attractive in this case. Some notable work on batch verification of DLP-based signatures is due to Naccache et al.[10], Harn [11, 12], Yen and Laih [7] and Bellare et al. [6]. The principle of DLP-based batch verification is, as in RSA, based on the multiplicative homomorphic property of these signatures.

Aggregation of DSA *signature aggregation* would be very useful in the ODB setting, especially, in the *Multi-Owner Model*[3]. Unfortunately, there seems to be no secure way to *aggregate* DLP-based signatures. The batching schemes which can be aggregated (for example, [12]) have been shown insecure by Boyd and Pavlovski in [13] who have demonstrated that an adversary can easily introduce false signatures that would pass the batch verification test. Current Batch-DSA methods involve techniques known as: *small exponent test* [10, 6] or *bucket test* [6] which require the verifier to perform operations on individual signatures. (For example, the small exponent test, as the name suggests, calls for the verifier to perform an exponentiation with a small exponent on each individual signature before batching). Therefore, it seems impossible to compose a single aggregated signature. In other words, using a DLP-based signature scheme along with a batch verification technique in ODB model would only help in reducing querier computation costs. and would not reduce querier bandwidth overhead. (See [13] for details on Batch-DSA).

7 Merkle Hash Trees

An alternative solution for providing record authentication and integrity in the ODB setting is to use Merkle Hash Trees (MHT-s) [14]. It is easy to construct an integrity architecture for an outsourced database using MHT-s: each leaf in a tree corresponds to a hash of a database record. In all other respects, the use of MHT-s is standard.

MHT is an appealing construct for our purposes: it is efficient, as most computation involves simple hashes, and provides authenticity, since the root of a tree is signed. However, upon a closer look, we are unsure whether MHT-s would compare favorably with pure signature-based techniques introduced earlier in this paper. In the following, we briefly sketch out potential applications of Merkle Trees.

The owner generates a hash tree with the leaves corresponding to hashes of individual data records and signs the root of tree using a standard signature technique. Any modification to the database (add, delete, update) requires the owner to conduct a multi-round protocol with the server to make structural changes to the MHT (which includes re-signing the root). This results in increased owner computation overhead.

Although querier computation overhead may be lower than in signature-based schemes, querier bandwidth overhead is considerably higher than in either Condensed-RSA or BGLS. In order to verify integrity of a single record, a set of intermediate tree nodes (corresponding to the co-path from the specific leaf up to the root) needs to be sent to the verifier. This translates into $(\log n)$ extra hashes. However, for a query reply containing multiple records, bandwidth overhead can be significantly reduced by eliminating common ancestors of returned records.

Despite the above shortcomings, MHT-s are a useful tool for ODB integrity. In a related paper [15], we explore their applications further. In particular, we investigate exact bounds on querier bandwidth overhead, discuss database modification issues and techniques for providing completeness of query replies efficiently.

[3]Note that, in DSA, it is customary for multiple users to share the same system parameters - primes p and q and generator g - which would have enabled aggregating signatures generated by distinct users

8 Discussion

In this section, we compare the signature schemes described in section 6 with respect to the overhead factors presented in section 5. We begin by assessing the cost in terms of basic cryptographic operations (e.g., multiplications, inverses, and exponentiations) in Condensed-RSA, Batch-DSA and BGLS schemes. Then, by plugging in the timings for each such operation, we show the actual overhead incurred by each scheme.

For the testing platform we used a P3-977Mhz Linux machine with the OpenSSL library [16] for computing the individual operations. We used 1024-bit moduli in RSA and DSA: 1024-bit n in RSA and 1024-bit p (along with 160-bit q) in DSA. For BGLS, we used a field F_p where $|p| = 512$.

Cost Comparison: We use the notation in Table 1. We assume that the query result contains $k * t$ records where k denotes the number of signers (data owners) and t denotes the number of signatures (data records) generated by each signer.

QC	Querier Computation
QB	Querier Bandwidth
SC	Server Computation
OC	Owner Computation
SS	Server Storage
$Mult^t(k)$	t modular multiplications with modulus of size $\|k\|$
$Exp_l^t(k)$	t modular exponentiations with modulus of size $\|k\|$ and exponent of size $\|l\|$
$Inv^t(k)$	t modular inverses with modulus of size $\|k\|$
$BM(t)$	t bilinear mappings

Table 1. Notation

Table 2 illustrates the overhead (computation, storage and bandwidth) associated with each scheme in terms of the number of cryptographic operations. The table provides a breakdown of the total cost with respect to overhead factors described in section 5. These factors are arranged in the decreasing order of perceived importance. (Our main goal is to minimize the cost of QC and QB.)

Note that the number of signers $k = 1$ for Unified Client and Multi-Querier models. In both models, Condensed-

RSA and BGLS have constant bandwidth requirement, independent of the number of component signatures. The querier computation (QC) overhead is linear in the number of signatures since both models involve t multiplications. Moreover, verifying an aggregated signature involves a single exponentiation in Condensed-RSA and two bilinear mapping in BGLS, making Condensed-RSA more efficient. In contrast, both QC and QB are linear in the number of signatures for Batch-DSA. This is because of the small exponent test. However, in Batch-DSA, since there no aggregation is performed, the server is not required to do any extra work.

The querier computation (QC) overhead is linear in the number of signatures. In the Multi-Owner model, BGLS has constant QB overhead, whereas, in Condensed-RSA, QB overhead is linear in the number of signers (k). In Batch-DSA, as before, QB is linear in the number of signatures.

Condensed-RSA performs better than BGLS since a modular exponentiation is much more efficient than a bilinear mapping and a modular multiplication in Z_n^* (where $|n| = 1024$) is cheaper than scalar addition in elliptic curves.

Table 3 shows the actual times required to generate and verify a single signature, multiple signatures by a single signer, and multiple signatures by multiple signers in each of the three schemes. We set the public exponent $e = 3$ in RSA and use the Chinese Remainder Theorem to speed up signing. Note that no optimization techniques were used in any of the verification operations.

		Condensed-RSA	Batch-DSA	BGLS
Sign	1 signature	6.82	3.82	3.54
Verify	1 signature	0.16	8.52	62
	t = 1000, k = 1	44.12	1623.59	184.88
	t = 100, k = 10	45.16	1655.86	463.88
	t = 1000, k = 10	441.1	16203.5	1570.8

Table 3. Cost comparison (in msecs): verification and signing. Notation: t – # signatures, k – # signers

9 Related Work

In this section, we briefly overview relevant prior work in database security.

Traditional database security has been studied extensively in the database as well as in cryptography and security communities. Database security in the ODB model is a more recent research topic. Hacigümüş, et al. examined various challenges associated with providing database as a service in [2]. In our work, we use a similar system model.

One seemingly related field is Private Information Retrieval (PIR) [17, 18] which has been explored extensively in the cryptographic literature. The PIR work is primarily concerned with *privately* retrieving parts of data stored at an external server such that no partial information about the query is leaked to the server. PIR techniques support searching based on either the physical location [17] of the data or keywords [19]. However, as most current PIR techniques aim for very strong security properties they are unsuitable for more practical purposes. Specifically, PIR schemes typically require either multiple non-colluding servers or multiple rounds of communication. Song et al. [20] propose a practical scheme to search encrypted data; it requires a single server and has low computational complexity. In general, searching encrypted data is becoming an increasingly popular research topic. (See, for example, [20, 21].) However, all current schemes only support exact match queries whereby the server returns data matching either a physical address or a keyword. Hacigümüş, et al. in [3] explore how SQL queries can be executed over encrypted data and provide details of query processing and optimization techniques. Specifically, they support *range* searches and *joins* in addition to exact match queries.

Another recent work [22] studies the problem of data integrity in the ODB model. This work is most closely related to this paper. It uses data encryption in tandem with manipulation detection codes to provide integrity; as such, it is applicable only to the Unified Client ODB model. In contrast, our solutions use digital signatures for providing integrity and aggregation techniques to achieve efficiency. Public key signatures allows us to provide integrity without requiring data encryption. Furthermore, they provide authentication and non-repudiation which are important requirements in Multi-Querier and Multi-Owner models.

10 Future Work and Conclusion

In conclusion, we investigated providing efficient data integrity mechanisms in the outsourced database (ODB) model. We presented a secure and practical *Condensed-RSA* scheme which performs well in Unified Client and Multi-Querier models. However, since it does not aggregate signatures by different signers, it is not well-suited for the Multi-Owner model. On the other hand, while the BGLS signature scheme aggregates signatures by distinct users into one short signature, the computational complexity is unfortunately quite high. Therefore, as part of future work, we plan to focus on finding efficient and practical signature schemes for the Multi-Owner model.

Acknowledgments

We thank NDSS'04 anonymous reviewers for their insightful reviews. Special thanks to Yongdae Kim for suggesting Merkle Hash Trees as a candidate solution and for providing the software for experimenting with BGLS signatures. We also thank Dan Boneh for his helpful comments.

	Condensed-RSA	Batch-DSA	BGLS
QC	$Mult^{k*(t-1)}(n) + Exp_e^k(n)$	$Mult^{4*k*t-1}(q) + Mult^{k*t}(p) + Exp_q^{k+1}(p)$ $+Exp_l^{k*t}(p) + Inv^{k*t}(q)$	$Mult^{k*t-1}(p) + BM(k+1)$
QB	$k*n$	$k*t*(p+q)$	p
SC	$Mult^{k*(t-1)}(n)$	0	$Mult^{k*t-1}(p)$
OC	$Exp_d^1(n)$	$Exp_q^1(p) + Inv^1(p) + Mult^2(p)$	$Exp_q^1(p)$
SS	$k*t*n$	$k*t*(p+q)$	$k*t*p$

Table 2. Cost comparison in the ODB model

References

[1] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and Verifiably Encrypted Signatures from Bilinear Maps," in *Advances in Cryptology – EUROCRYPT '2003* (E. Biham, ed.), Lecture Notes in Computer Science, International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 2003.

[2] H. Hacigümüş, B. Iyer, and S. Mehrotra, "Providing Database as a Service," in *International Conference on Data Engineering*, March 2002.

[3] H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra, "Executing SQL over Encrypted Data in the Database-Service-Provider Model," in *ACM SIGMOD Conference on Management of Data*, pp. 216–227, ACM Press, June 2002.

[4] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, pp. 120–126, Feb. 1978.

[5] L. Harn, "Batch verifying rsa signatures," *Electronic Letters*, vol. 34, pp. 1219–1220, Apr 1998.

[6] M. Bellare, J. Garay, and T. Rabin, "Fast batch verification for modular exponentiation and digital signatures," in *Eurocrypt 1998*, vol. 1403, pp. 191–204, 1998.

[7] S. Yen and C. Laih, "Improved Digital Signature Suitable for Batch Verification," *IEEE Transactions on Computers*, vol. 44, pp. 957–959, July 1995.

[8] M. Bellare and P. Rogaway, "Random oracles are practical: a paradigm for designing efficient protocols," in *ACM Press*, pp. 62–73, 1993.

[9] A. Joux and K. Nguyen, "Separating Decision Diffie-Hellman from Diffie-Hellman in cryptographic groups," in *Cryptology ePring Archive*, no. Report 2001/003, 2001.

[10] D. Naccache, D. M'Raïhi, D. Raphaeli, and S. Vaudenay, "Can DSA be improved: complexity trade-offs with the Digital Signature Standard," in *Advances in Cryptology – EUROCRYPT '94*, Lecture Notes in Computer Science, pp. 85–94, International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 1994.

[11] L. Harn, "DSA-type Secure Interactive Batch Verification Protocols," *Electronic Letters*, vol. 31, pp. 257–258, Feb. 1995.

[12] L. Harn, "Batch Verifying Multiple DSA-type Digital Signatures," *Electronic Letters*, vol. 34, pp. 870–871, Apr. 1998.

[13] C. Boyd and C. Pavlovski, "Attacking and repairing batch verification schemes," in *Asiacrypt 2000*, pp. 58–71, 2000.

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

[15] E. Mykletun, M. Narasimha, and G. Tsudik, "Providing Authentication and Integrity in Outsourced Databases using Merkle Hash Trees." UCI-SCONCE Technical Report, 2003. `http://sconce.ics.uci.edu/das/MerkleODB.pdf`.

[16] OpenSSL Project, available from, `http://www.openssl.org`.

[17] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private Information Retrieval," *Journal of ACM*, vol. 45, pp. 965–981, Nov. 1998.

[18] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin, "Protecting Data Privacy in Private Information Retrieval Schemes," in *30th Annual Symposium on Theory of Computing (STOC)*, (Dallas, TX, USA), ACM Press, 1998.

[19] B. Chor, N. Gilboa, and M. Naor, "Private Information Retrieval by Keywords," Tech. Rep. TR CS0917, Department of Computer Science, Technion, 1997.

[20] D. Song, D. Wagner, and A. Perrig, "Practical Techniques for Searches on Encrypted Data," in *2000 IEEE Symposium on Security and Privacy*, May 2000.

[21] E.-J. Goh, "Secure Indexes for Efficient Searching on Encrypted Compressed Data." Cryptology ePrint Archive, Report 2003/216, 2003. `http://eprint.iacr.org/2003/216/`.

[22] H. Hacigümüş, B. Iyer, and S. Mehrotra, "Encrypted Database Integrity in Database Service Provider Model," in *International Workshop on Certification and Security in E-Services (CSES'02 IFIP WCC)*, 2002.