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 **Secure Outsourced Database** (SODB) 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 provide an in depth study of the problem of ensuring data integrity and suggest secure and practical schemes that help facilitate authentication of all query replies. Concretely, 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 mechanisms in this model has not received much attention.

In this paper, we focus on providing quick and efficient means of ensuring data integrity, while incurring minimal computational and bandwidth overhead. Concretely, we aim to provide schemes that help the client in verifying the integrity of the data returned by the service provider in response to a posed query. However, we do not verify the completeness of the results returned by the server, which could involve the server omitting certain data from the results.

Scope: In this paper, we only address queries testing equality and other logical comparison predicate clauses. In other words, we consider the standard SQL queries involving *SELECT* clauses which (usually) result in the selection of a set of records or fields that match a given predicate or a set thereof. We specifically do not address queries that involve data aggregation (exemplified by arithmetic operations, such as SUM or AVERAGE) which usually return a single value as the answer to the posed query.

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 aggrega-

tion 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 compare and contrast the relative performance of the above three schemes. Section 8 looks at relevant prior work that has been done in the field of database security as well as the field of batch cryptography and aggregated signatures. We finally conclude in section 9 by giving an outline for our future work.

2 System Model

The secure outsourced database (SODB) model is an example of the well-known Client-Server model. In this 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.

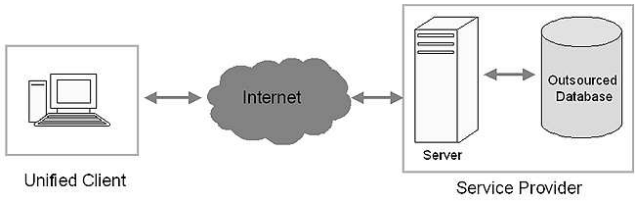


Figure 1. SODB – Unified Client Model Overview

In SODB, a 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 SODB 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), we distinguish among 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.

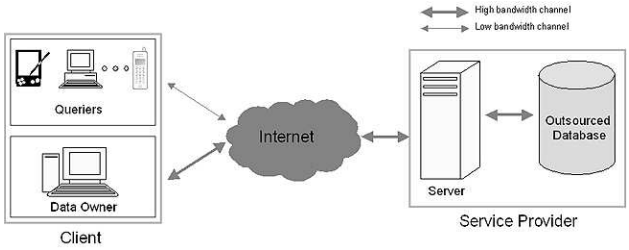


Figure 2. SODB – Multi-Querier Model Overview

In the third, most general, SODB model, a single database can have multiple owners (for a given client database). This is referred to as the *Multi-Owner Model* (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 straight-forward. 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.

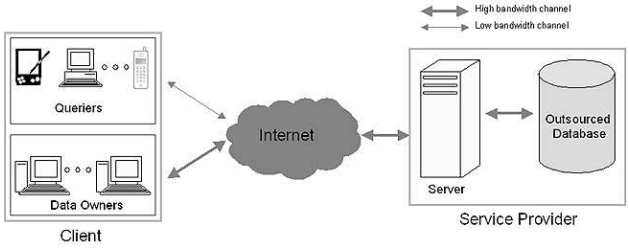


Figure 3. SODB – Multi-Owner Model Overview

3 Motivation

As mentioned above, the SODB 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 SODB model. We note that data privacy in SODB is orthogonal to data authentication and integrity. It is easy to envision an SODB scenario where privacy is important but integrity less so (e.g., because the server is somehow trusted with the latter but not the former). It is even more likely to encounter an SODB scenario where privacy is not a concern but data authentication and integrity are required. For example, a government funding agency (say, NSF) may choose to outsource its awards database and allow anyone to run queries over it. In this case, privacy is not an issue since the contents are essentially public knowledge, but their integrity and authenticity are very important.

From a more technical perspective, the motivation for this work is also fairly intuitive. When an SODB client queries its outsourced data, it expects in return a set of records (reply) satisfying the query 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 securely and efficiently facilitate authentication of all possible query replies?**

Of course, we can always assume that the communication channel between a client and a server is secure or, at least, authentic. However, this is insufficient to authenticate query replies since the server is NOT trusted with the integrity of the data, i.e., a malicious server may attempt to insert fake records into the database or modify existing records. More concretely, clients must be assured 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

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 a suitable form and has a mechanism for formulating queries (involving selection predicates) that can be executed *remotely* upon the stored data.

Granularity of Integrity: data integrity/authentication can be provided at different levels of granularity. In principle, we could compute integrity checks 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 value. Providing integrity checks at 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 huge amounts of data to the client. Hence, we do not consider this to be a viable solution. On the other hand, computing integrity checks at the level of individual 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 checks at the record level, i.e., for each individual record. This enables the server to return – in response to a query – any set of matching records along with their respective integrity checks. We note that computing integrity checks over the entire record, as opposed to individual fields, 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.

Signatures or MAC-s? One natural and intuitive solution to provide 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 quite low. This works well for the the *Unified Client* Model where the client and the querier is one and the same.

However, in the more general *M*ulti-Querier and *M*ulti-Owner models, for each client, we assume potentially many queriers. In these settings, MAC-s are not useful since they would require the MAC key to be shared among the owner(s) and all 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.[1]

[1] 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.

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 sources of overhead that we would like to minimize (listed in the decreasing order of perceived importance):

1. Querier computation: overhead incurred in verifying integrity of a query reply.
2. Querier bandwidth: overhead incurred in sending/receiving integrity data (this is in addition to the overhead incurred for sending/receiving the actual records in a reply).
3. Server computation: overhead incurred by server (for each query) due to potential manipulation of integrity information in the reply.
4. Owner computation: overhead incurred by the owner in computing integrity information to be stored in the outsourced database. (This is a longer-term overhead factor, independent of future queries.)
5. Server storage: overhead incurred by the server for storage of integrity information in an outsourced database. (This overhead is incurred only initially, i.e., it does not depend on queries.)

We claim that the first three are the most important overhead factors stemming from maintaining 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 because outsourced database records are placed onto a server much less frequently than client queries are run upon them. This observation favors signature schemes that are particularly efficient in signature verification. Efficiency in generating (computing) signatures is comparatively much 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 exist cryptographic methods that can be used to construct idealized or near-idealized solutions. Such methods are called *Aggregated Signature Schemes*. Some of these 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 for aggregated digital signatures. The first one is based on a simple extension of the well-known Batch RSA method, and the second is the recently proposed scheme due to Boneh, et al.[1].

6 Signature Schemes

As we noted in section 5, signature schemes that allow aggregation seem to be ideally suited for the SODB application. In this section, we examine some viable solutions and also point out their limitations. We start by presenting a simple single-signer RSA aggregation scheme. We, then, introduce an aggregate signature scheme due to Boneh et al. (henceforth referred to as the BGLS scheme), and conclude by presenting a discussion on the applicability of DLP based signature schemes in the SODB model.

6.1 Aggregated-RSA

RSA[4] exhibits multiplicative homomorphic property which renders it suitable for aggregating signatures generated by a single signer. If the aggregate signature verifies, it will signify to the verifier that each individual message contained in the signature was, indeed, signed by the actual signer. Aggregation of RSA signatures can be performed "incrementally" by any party who has access to the individual signatures.

RSA: We first describe the parameters involved in a standard RSA cryptosystem. A party has a public key $pk = (n, e)$ and a secret key $sk = (d)$, where n is a k-bit modulus being the product of two random $k/2$-bit primes p and q. $e, d \in Z_n^*$ and satisfy $ed \equiv 1 \mod \phi(n)$, where $\phi(n) = (p-1)(q-1)$. In today's literature, k is at least 1024. The security of RSA cryptosystem is based on the intractability of the integer factorization problem.

In practice, an RSA signature is computed on the hash of a message. Let $h()$ denote a cryptographically strong hash function, such as MD5 or SHA-1, which takes a variable length input and produces a fixed length output. Then for a message m, $h(m)$ gives the hash of m. A standard RSA signature on message m, is computed as: $\sigma = h(m)^d$

$(\mathrm{mod}\ n)$. Verification of the signature involves checking that $\sigma^e \equiv h(m) \mod n$. Both signature generation and verification involve computing one modular exponentiation each.

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

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

The resulting aggregated signature $\sigma_{1,t}$ is the same size as a standard RSA signature. When verifying the aggregated signature, the verifier needs to multiply together the hashes of the received messages and check the following:

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

Batch verification of RSA signatures: Aggregated-RSA verification is similar to *Batching* of RSA verifications. Batching, in general, helps reduce computational complexity in settings where many signature verifications (or other computationally intensive constructs) must be performed simultaneously. Batch RSA [5, 6, 7, 8] aims at speeding up the signing and verification process by reducing the total number of exponentiations needed. Given a batch instance of signatures $\{\sigma_1, ..., \sigma_t\}$ and distinct messages $\{m_1, ..., m_t\}$, a RSA batch verification (Fast screening [7]) consists of checking that:

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

Bellare et al. in [7] 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. Note that one of the main differences between Aggregated-RSA and Batch-RSA is that in the latter, the product of signatures is computed by the verifier. Consequently, the verifier has access to individual signatures. This in turn allows for auditing should the batch verification fail. That is, it is possible to check the validity of individual signatures to determine the faulty ones. On the other hand, in Aggregated-RSA, where a single aggregated signature is returned, this auditing is not possible. Therefore in aggregated-RSA, in the event of a verification failure, further steps (perhaps involving off-line mechanisms) become necessary.

Aggregated-RSA Security: We claim that *Aggregated-RSA is unforgeable against an adaptive chosen message attack*. In this section, we defend this claim by showing that if there is an adversary $\mathcal{A}$ who can break Aggregated-RSA, then using this adversary it is possible to create a forger $\mathcal{B}$ who can successfully create a batch instance that verifies the batch verification criterion without possessing the individual signatures on each of the messages in the instance.

Before proceeding further, we define what it means for an adversary $\mathcal{A}$ to *break* Aggregated-RSA. $\mathcal{A}$ succeeds in breaking Aggregated-RSA if it is able to produce a valid aggregated signature for messages $\{m_1, ..., m_t\}$ which satisfies equation 2 without possessing individual signatures for all t messages. This section describes the security of Aggregated-RSA by demonstrating that it is at least as secure as Batch RSA, which in turn was shown in [7] to be secure under the assumption that RSA is a collection of one-way functions. We point out that we are assuming the use of a full domain hash function (FDH) as described in [9]. FDH is a hash function $H_{FDH} : \{0,1\}^* \rightarrow Z_n^*$

Outline of Security: $\mathcal{A}$ 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 Aggregated-RSA by outputting a valid Aggregated-RSA signature $\sigma_{1,t}$. We now construct a forger $\mathcal{B}$ that breaks Batch-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 Aggregated-RSA signature returned by $\mathcal{A}$ be X
4. $\mathcal{B}$ 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 will satisfy the Batch-RSA 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 hence the Batch-RSA verification test succeeds.

Claim 2: If RSA is one way, then Aggregated-RSA is a secure scheme.
Batch-RSA was shown to be secure under the assumption that RSA is one way. Therefore, since Aggregated-RSA is at least as secure as Batch-RSA, it can be concluded that Aggregated-RSA is secure assuming RSA consists of a collection of one-way functions.

Overhead Costs: We now compare the querier computation and bandwidth costs incurred by Aggregated-RSA with those of standard RSA and Batch-RSA. We ignore the cost of hashing, as it is a relatively efficient cryptographic operation.

If standard RSA signatures are used, the querier would receive t signatures (the result set returned contains t records) and would have to perform t verifications, which involve t modular exponentiations. The bandwidth overhead, which is linear in the number of signatures, is $t * |n|$ bits. Batch-RSA would also require t individual signatures to be sent. The verification process would consist of computing the product of message hashes and the product of signatures, resulting in $2(t-1)$ multiplications, prior to performing one modular exponentiation. Once again, the bandwidth overhead is $t * |n|$ bits. Finally, if Aggregated-RSA is used, only one signature would be transmitted (resulting in a constant $|n|$ bits overhead), and verification would require the querier to perform $t-1$ multiplications, to compute the product of message hashes, and one exponentiation.

Aggregated-RSA in SODB application: Aggregated-RSA is clearly applicable to Unified-Client as well as Multi-Querier models, both of which assume 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, Aggregated-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 aggregate 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 [10]. 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: for any $p_1, p_2 \in G_1, q \in G_2, e(p_1 p_2, q) = e(p_1, q) \cdot e(p_2, q)$; and for any $p, q \in G_2, e(\psi(p), q) = e(\psi(q), p)$.

BGLS Signature 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 signature is σ. To verify a signature one needs to compute $h = h(m)$ and check that $e(\sigma, g2) = e(h, v)$.

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

Verification of an aggregate signature $\sigma_{1,t}$ involves computing the product of 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 computation cost of verifying an aggregated BGLS signature, we distinguish between the following two operations: multiplication and computation of the bilinear map. When there is a single signer and t messages, the verification consists of computing the product of hashes, requiring $t-1$ multiplications, before performing 2 bilinear mappings. When BGLS is utilized to aggregate signatures generated by k signers, where each have created t signatures, verification consists of $k*t-1$ multiplications of hashes as well as $k+1$ bilinear mappings.

BGLS in SODB application: Aggregated BGLS is applicable to all three models of SODB. Once again, the server executing a client query: selects records that match the query predicate; fetches the signatures corresponding to these records; aggregates them (as mentioned above) and sends back the single aggregated signature along with the records in the result set. Aggregated BGLS helps save both querier computation and querier bandwidth in case of Unified-Client as well as Multi-Querier models. However, in the Multi-Owner model, BGLS achieves bandwidth efficiency but may not provide much querier computation efficiency. More precisely, verifying an aggregated BGLS signature which is generated by aggregating t messages signed by k distinct signers involves computing $k+1$ bilinear mappings which are expensive operations (according to our measurements, computing a single bilinear mapping takes 31ms). We will discuss these costs in more detail in section 7.

6.3 Using DLP based signatures

Batch Verification for DSA signatures was introduced by Naccache et al. in [11]. Online generation of DSA or ElGamal-type signatures can be very efficient since they allow certain amount of precomputation. On the other hand, the verification is usually far less efficient (typically, a signature verification requires 2 modular 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 concerning DLP-based signature batch verification is due to Naccache et al.[11], Harn [12, 13], Yen and Laih [8] and Bellare et al. [7]. Note that the principle of batch verification, once again, is based on the multiplicative homomorphic property of these signatures.

DSA *signature aggregation* would have been useful in the SODB application especially in the *Multi-Owner Model*[2]. Unfortunately, there seems to be no secure way to *aggregate* DLP based signatures. The batching schemes which can be aggregated (for example, [13]) have been shown to be insecure by Boyd and Pavlovski in [14] who have demonstrated that it is possible for an adversary to easily introduce false signatures that would satisfy the batch verification criterion. The currently known methods for secure batch verification of DSA signatures involve techniques known as: *small exponent test* [11, 7] or *bucket test* [7] which require the verifier to perform operations on individual signatures (for e.g., in the small exponent test, as the name suggests, the verifier is required to carry out a modular exponentiation involving a small exponent on each individual signature before batching). Therefore, it seems impossible to send to the verifier a single aggregated signature. In other words, using a DLP based signature scheme along with a batch verification technique in SODB model will only help in reducing the querier computation costs but will not reduce the querier bandwidth requirements (Refer to [14] for details pertaining to secure batch-veifying of DSA signatures).

7 Discussion

In this section, we compare and contrast the signature schemes described above with respect to the various overhead factors outlined in section 5. We begin by counting the number of basic cryptographic operations (such as: modular multiplications, inverses, and exponentiations) required by Aggregated-RSA, Batch-DSA and Aggregated-BGLS schemes. Further, by finding out the time required for implementing each of these primitive operations, we show the actual overhead incurred by each of these schemes.

We utilized a P3-977Mhz machine running linux and used the OpenSSL library [15] to compute the individual costs associated with each of the primitive operations. We assumed 1024-bit moduli in RSA and DSA mechanisms: that is, 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 $|p|$ is 512-bits.

Cost Comparisons: We follow the notations summarized 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.
Table 2 outlines the overhead (computation, storage and bandwidth) associated with each of the signature schemes in terms of the number of required cryptographic operations. The table provides a breakdown of the total cost with respect to the overhead factors described in section 5. The overhead factors are arranged in the decreasing order of per-

[2]Note that in DSA, it is acceptable 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

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. Notations

ceived importance. Please recall that our main goal is to minimize the factors QC and QB.

Note that the number of signers $k = 1$ for both UC and MQ models. In the UC/MQ models, both Aggregated-RSA and BGLS have constant bandwidth requirement (independent of the number of component signatures that the aggregated signature is made up of). The querier computation (QC) overhead is linear in the number of signatures since both of these models involve t multiplications (for multiplying the hashes of the messages). Moreover, verifying an aggregated signature involves a single exponentiation in Aggregated-RSA but two bilinear mapping in BGLS, making Aggregated-RSA more efficient. In contrast, both QC and QB are linear in the number of signatures for DSA based scheme. This is due to the required small exponent test which involves one exponentiation (albeit, with a small exponent) per signature. However, in the DSA based scheme (since there is no aggregation involved), the server is not required to do any extra work in response to a query. In other words, SC is zero for batch-DSA whereas SC is linear in the number of individual signatures for both Aggregated-RSA and BGLS (since they both require the server to compute the product of individual signatures to generate the aggregated signature).

The querier computation (QC) overhead is linear in the number of signatures. In the Multi Owner model, BGLS still has constant QB since it is possible to aggregate signatures generated by distinct users. Whereas in Aggregated-RSA, the QB is now linear in the number of signers (k). In other words, the server can aggregate individual signatures generated by each signer in the query response and send them separately to the querier. In Batch-DSA, as before, there is no compression and therefore QB is linear in the number of signatures. In terms of QC overhead, once again Batch-DSA has very high overhead. Aggregated-RSA performs better than BGLS since computing a modular exponentiation is relatively much more efficient than computing a bilinear mapping and also computing modular multiplications in Z_n^* (where $|n|$ is 1024) turns out to be cheaper than scalar addition in Elliptic curves (in Field F_p where $|p|$ is 512 bits).

Table 3 gives the actual time required to generate a single signature and also the time required to verify a single signature, multiple signatures by a single signer, and multiple signatures by multiple signers in all three signature schemes. We set the public exponent e to 3 in RSA and used the Chinese Remainder Theorem to speed up signing. We would like to point out that we do not use any optimization technique in any of the verification operations.

8 Related Work

Our work pairs batch verification of signatures and aggregate signatures with security of outsourced databases in an attempt to provide fast and efficient mechanisms to ensure data integrity. In this section, we will briefly examine relevant prior work from these fields.

Database security has been studied extensively by database as well as cryptographic communities. Specifically, the problem of providing data secrecy has been investigated by many researchers. Hacigümüş et al. examine the various challenges associated with providing database as a service in [2]. In our work, we use a similar system model as outlined in this paper. The problem of Private Information Retrieval (PIR) [16, 17] deals with the exact matching problem and has been explored extensively in the cryptographic literature. The PIR problem is primarily concerned with retrieving *privately* parts of data that is 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 [16] of the data or using keywords [18]. However, most of the currently available PIR techniques aim for very strong security bounds and, consequently, are quite unsuitable for practical purposes. Concretely, the PIR schemes typically require either multiple non-colluding servers or multiple rounds of communication. Song et al. [19] propose a more efficient scheme to search on data encrypted using a secret symmetric key. Their scheme requires a single server and also has low computational complexity. However, all the above mentioned schemes only support exact match queries in which the server returns data that matches either the given address or the given keyword. Hacigümüş et al. in [3] explore how different types of 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.

Hacigümüş et al. study the problem of providing integrity in the secure outsourced database model in [20]. This work is most closely related to what we present in this paper. In their work, they use data encryption in combination with manipulation detection codes to maintain integrity. Here, we present an alternative solution which uses digital signatures for maintaining integrity and uses aggregation techniques to achieve efficiency. Digital signatures help us maintain integrity without requiring explicit data

	Aggregated-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 SODB model

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

Table 3. Cost comparison (time in msec): verification and signing.

encryption. Further, they provide authentication and non-repudiation which are important requirements in our model. Further, in order to facilitate efficient integrity assessment, we build a simple RSA [4] based aggregation technique which is essentially based on the technique of batch verification [7] of RSA signatures. The idea of batching to amortize the cost of private key operations (signing and decryption) was first introduced by Fiat in [5, 21]. We also compare this solution to aggregate signatures proposed by Boneh et al. [1]. The aggregate signature scheme proposed in [1] is based on bilinear maps and provides an elegant mechanism to aggregate n individual signatures on n messages into one single signature. Related to aggregate signature schemes are multi-party signatures [22] which aim at compressing multiple signatures from different signers on the same message.

9 Future Work and Conclusion

Careful readers may wonder why we have not considered Merkle trees [23] as a possible solution for the problem of providing integrity in SODB. Merkle trees are certainly efficient (since they involve computing simple hashes) and also provide authenticity (since the root of the tree is digitally signed) and hence do appear ideal for our problem. The reason for omitting this discussion in this paper is twofold: (i) We specifically wanted to study the applicability of aggregated signatures in this scenario and (ii) more importantly, we realized that a cursory examination was not sufficient to analyze the exact overhead introduced by Merkle trees. We justify the second reason using the following argument: Consider a Unified Client or a Multi Querier model of the SODB. Let us assume that the data owner uses a Merkle tree to protect the authenticity and integrity of the records. Therefore, the data owner generates a hash tree where the leaves correspond to hashes of individual data records and the root is digitally signed by the data owner. In this case, clearly, any modification (add, delete, update) to the database will require the data owner to have 2 extra rounds of communication with the server to make structural changes to the Merkle tree resulting in an increase in the Owner Computation overhead. Although querier computation overhead may be lesser since verification involves computing simple hashes, we believe that the querier bandwidth overhead will be considerably higher than both Aggregated-RSA and BGLS. As an example, consider a database with 2^h records. This will result in a Merkle tree of height h. Suppose a query results in the selection of t records where $t = 2^c$. Without loss of generality, we assume that the nodes corresponding to these records are randomly placed in the tree and therefore we assume that these t records have no common ancestors till the depth c from the root. Nodes from depth c and above are shared by these records and thus need to be transmitted only once. This implies that the bandwidth overhead is $(t*(h-c)+2^{c+1}-1)*hash$. As a concrete example: let h=30, t=1024 and hash = 160. Therefore c = 10 and the QB overhead in this case is 3604320 bits as opposed to 1024 bits in RSA and 512 bits in BGLS. However, we note that it might be possible to reduce this overhead through optimizations. As a part of our future work, we would like to explore this solution further and obtain better bounds on the overhead factors.

In conclusion, in this work, we looked at the problem of providing efficient data integrity mechanisms in the secure outsourced database model. We presented a secure and practical *Aggregated-RSA* scheme which performs well in the Unified Client as well as the Multi Querier models. However, since it does not provide a means to aggregate signatures from different signers, we saw that it is not optimal in the Multi Owner model. On the other hand, while the signature scheme proposed by Boneh et al. aggregates signatures from distinct users into one short signature, the

computational complexity of this scheme was shown to be quite high. Therefore, as a part of our future work, we want to focus on finding efficient and practical signature schemes for the Multi Owner model which can aggregate signatures generated by distinct users.

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] A. Fiat, "Batch RSA," in *Advances in Cryptology – CRYPTO '89* (G. Brassard, ed.), no. 435 in Lecture Notes in Computer Science, (Santa Barbara, CA, USA), pp. 175–185, International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 1990.

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

[7] 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.

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

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

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

[11] 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.

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

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

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

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

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

[17] 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.

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

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

[20] 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.

[21] A. Fiat, "Batch RSA," *Journal of Cryptology*, vol. 10, no. 2, pp. 75–88, 1997.

[22] S. Micali, K. Ohta, and L. Reyzin, "Accountable-subgroup multisignatures," 2001.

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