Authentication and Integrity in Outsourced Databases

EINAR MYKLETUN, MAITHILI NARASIMHA and GENE TSUDIK
Computer Science Department
School of Information and Computer Science
University of California, Irvine
{mykletun,mnarasim,gts}@ics.uci.edu

In the **Outsourced Database** (ODB) model, entities outsource their data management needs to third-party service providers. Such a service provider offers mechanisms for its clients to create, store, update and access (query) their databases. This work provides mechanisms to ensure data integrity and authenticity for outsourced databases. Specifically, this work provides mechanisms that assure the querier that the query results have not been tampered with and are authentic (with respect to the actual data owner). It investigates both security and efficiency aspects of the problem and constructs several secure and practical schemes that facilitate integrity and authenticity of query replies while incurring low computational and communication costs.

Categories and Subject Descriptors: H.2.0 [**Database Management**]: General—*Security*; *integrity*; *protection*

General Terms: Storage, Data Integrity and Authenticity
Additional Key Words and Phrases: Outsourced Databases, Authentication, Integrity, Signature Aggregation

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-as-a-Service (DAS) [Hacigümüş et al. 2002b] model 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' personal databases as well as mechanisms for the client to efficiently create, update and access their outsourced data.

Furthermore, the pervasive nature of the Internet brings with it the ability to

Portions of this paper appeared in [Mykletun et al. 2004a] and [Mykletun et al. 2004b].

reach a global audience and market. Consider, for example, an individual (referred to as a data owner) who is interested in selling or sharing her music, information or software etc., using the Intenet. Instead of investing in expensive back office infrastructure, the data owner can simply use an existing popular online publishing house to sell or distribute his data. The online publishers host the data owner's intellectual property, handle transactions with the customers and deal with various support issues among other things. In this scenario, the data owners are using the online third-party publishers to widely disseminate their databases over public networks. Note that this online-publishing setting is different from the DAS setting above in that the data owner is interested in distributing/selling his data to all interested parties whereas in the DAS setting, the outsourced data is essentially private in nature.

The database outsourcing paradigm poses numerous research challenges. One of the foremost challenges is the security of stored data. A *client* or *owner* outsources its data (which is a critical asset) to an external, potentially untrusted, *Service Provider*. It is, therefore, essential to provide adequate security measures to protect the stored data from both malicious outsider attacks and the Service Provider itself. Security, in most part, implies maintaining data integrity and guarding data privacy. In the two scenarios mentioned above, while it can be argued that data confidentiality is mainly relevant in the DAS setting, both DAS as well as online publishing models require that the service provider be able to meet the integrity, authenticity and the non-repudiation requirements of the clients or customers. Within the scope of the DAS model, some work [Hacigümüş et al. 2002] has been done to address the data confidentiality requirements. However, the problem of providing efficient integrity mechanisms in these models has not received much attention.

In this paper, we address this important security problem and provide efficient and secure means of ensuring data integrity and authenticity, while incurring minimal computational and bandwidth overhead. Concretely, we provide techniques based on digital signature aggregation that help the client in efficiently verifying the integrity and authenticity of the data (query reply) returned by the service provider.

Although our system model assumes that the database is essentially outsourced to external service providers, we would like to note that the techniques we propose in this work are equally applicable to maintain the integrity and authenticity in settings where the data is maintained in-house. Concretely, our techniques can be applied to databases with stringent reporting and auditing requirements as mandated by various acts of congress, such as Sarbanes-Oxley[United States Code 2002] and HIPAA[Law 1996].

Scope: The underlying data model in our approach is the relational data model. We assume that the data owner and the service provider manage the data using a typical relational database management system (RDBMS). The queries are formulated using SQL. In our work, we mainly address range selection queries, projections, joins and other set operation queries. Range selection queries involve 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.

Furthermore, in this work, we provide mechanisms for efficiently verifying the integrity and authenticity of the query replies returned by the service provider in the outsourced database (ODB) model. In other words, the primary concern here is to provide mechanisms to efficiently verify the *correctness* of the entire result set. A related concern involves ensuring the *completeness* of the result set which refers to assuring the querier that the service provider has, indeed, returned **all** tuples matching the query predicate. Providing completeness guarantees remains beyond the scope of this paper. However, interested readers may refer to [Narasimha and Tsudik 2005] for simple techniques that extend the current work to provide proof of completeness.

Organization: The rest of the paper is organized as follows: In Section 2, we provide an overview of our system model. Section 3 presents the motivation for our work. We consider the design choices and assumptions in Section 4, while Section 5 covers overhead factors and features associated with aggregate signature schemes. In Section 6, we discuss aggregation of RSA signatures and then examine an aggregate signature scheme proposed by Boneh, et al. in [Boneh et al. 2003] and finally discuss why aggregation of DLP-based signatures schemes is difficult. In Section 7, we compare the relative efficiency factors of the three signature schemes of Section 6. In Section 8, we discuss an interesting feature of some aggregated/condensed signatures which potentially results in a security loophole in certain ODB applications. Section 10 discusses some *immutability* extensions to the signatures proposed in Section 6 and Section 11 discusses the additional costs associated with these enhancements. In Section 12, we briefly discuss another viable solution for ensuring integrity in ODB model using the general approach of authenticated data structures. Section 13 considers other relevant prior work in the fields of database security, batch cryptography and aggregated signatures. The paper concludes in Section 14 with the discussion of future work.

2. SYSTEM MODEL

The outsourced database (ODB) model is an example of the well-known Client-Server paradigm. In ODB, the *Database Service Provider* (for brevity, referred to as simply **"Server"** from here on) has the infrastructure to host outsourced databases and provides efficient mechanisms for its 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.

In ODB, 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.

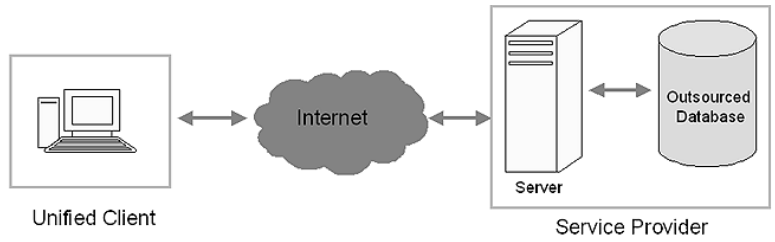

Fig. 1. ODB – Unified Client ODB

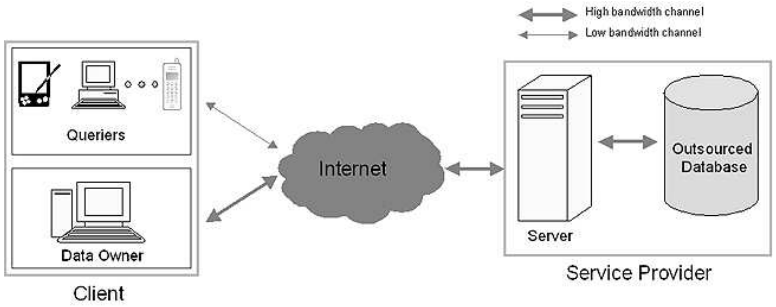

Fig. 2. ODB – Multi-Querier ODB

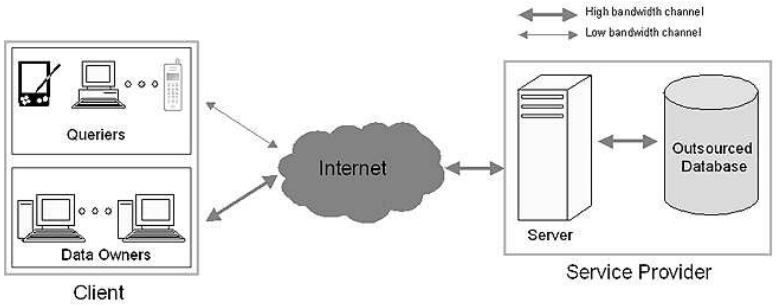

Fig. 3. ODB – Multi-Owner ODB

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), we distinguish among two types of clients: *owners* and *queriers*. The former is the actual data owner who adds, deletes and updates database tuples. Whereas, a querier is only allowed read access (i.e., query privileges) to the database or portions thereof. This distinction is both necessary and natural as it reflects many real-world database scenarios.

In practice, a querier may be a computationally weak and storage-challenged device, such as a cellphone or a wireless PDA. Moreover, the bandwidth available to

a querier may be severely limited due to the communication medium characteristics and/or the battery power consumed by receiving large amounts of data.

In the third, most general, ODB 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 can appear subtle. In the former, a single security principal creates and manipulates database tuples, whereas, in the latter, different tuples can be created by distinct security principals. However, in both models, there can be multiple queriers. The motivation for the Multi-Owner model is straight-forward: Consider an example of an outsourced customer/sales database. Each tuple in this database is created and maintained by the salesperson responsible for a particular customer. A salesperson then "owns" the tuples 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 [Hacigümüş et al. 2002b],[Hacigümüş et al. 2002] addressed this challenge by devising methods for running encrypted (or obfuscated) queries over (partially) encrypted databases.

This paper addresses the integrity of outsourced data in the ODB model. (We note that data secrecy in ODB is orthogonal to integrity.) Specifically, we focus on integrity-critical databases which are outsourced to untrusted servers and are accessed over insecure public networks. We assume that servers can be malicious and/or incompetent and, thus, might be processing and storing hosted data incorrectly. Furthermore, since it is difficult, in general, to guarantee absolute security of large on-line systems, we assume that the server can be compromised by an attack, such as a worm/virus or a system break-in. Therefore, we need efficient mechanisms to reduce the level of trust placed in the server and provide integrity guarantees to the clients.

From a more technical perspective, the motivation for this work is also fairly intuitive. When an ODB client queries its outsourced data, it expects in return a set of tuples (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 tuples in the database. In other words, a query reply can be any one of the 2^m tuple 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 tuples into the database or modify existing tuples. 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 work is not concerned with data confidentiality. Consequently, to keep the discussion general, we do not differentiate between the cases of the stored data being completely encrypted, partially encrypted or completely un-encrypted. 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 attribute values incurs a very large overhead which is expensive for the owner/querier (in terms of computation/verification) as well as for the the server (in terms of storage).

We believe that the optimal choice is to provide integrity checks at the tuple level, i.e., for each individual tuple. This enables the data owner to sign each tuple once before depositing it at the server and the server to return – in response to a query – any set of matching tuples along with their respective integrity checks. We note that computing integrity checks over the entire tuple, as opposed to individual fields, implies that the smallest unit of data returned as a query reply is an entire tuple, even when the querying client is only interested in a single field.

Signatures, MACs or Merkle Hash Trees? One natural and intuitive solution to provide tuple-level integrity is to use message authentication codes (MACs) as they tend to be small and efficient to compute and verify. A simple MAC-based scheme can be constructed whereby the the server stores each outsourced tuple along with a MAC (or keyed hash) of that tuple computed by the owner (client) 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 tuple-level MACs 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, recall that, in more general *Multi-Querier* and *Multi-Owner* models, we assume multiple queriers have read access to the data. In these settings, MACs are not useful since they would require the MAC key to be shared among all entities that make up a client: the owner(s) and the queriers. Unfortunately, non-repudiation for the queriers would not be achievable. Also, a rogue querier who knows the MAC key could collude with the server and insert fraudulent records with valid MACs. Therefore, the only viable choice is public key digital signatures. One natural way is to use signatures as tuple-level integrity checks. However, digital signatures introduce significant overhead in terms of storage, bandwidth and

computation. Another way is to generate a Merkle Hash Tree [Merkle 1980] with leaves representing individual hashes of all the tuples in the table and only the root signed using a digital signature scheme. Some relevant prior work [Devanbu et al. 2000],[Pang and Tan 2004] examined integrity issues in outsourced databases and suggested solutions based on Merkle Hash Trees. More precisely, these solutions use the general approach of *authenticated data structures* which are essentially derived from the simple hash tree.[1] In our work, we mainly concentrate on using tuple level signatures and present novel extensions that enable efficient verification of integrity and authentication of query replies.[2]

5. OVERHEAD FACTORS AND DESIRED FEATURES

The size of a typical digital signature ranges between 320 (e.g., for DSA) and 1024 (e.g., for RSA) bits. As discussed above, we are using tuple-level signatures. Now, considering that a query reply can potentially contain many thousands of tuples, receiving and verifying individual tuple signatures can be prohibitively expensive for a querier. Also, even if the storage on the server is not a constraint, recall that a 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 of the security services. To be more specific, 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 tuples 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.

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 tuples 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

[1] See sections 12 and 13 for the discussion of this and other related work.
[2] We observe that, even in the *Unified Client* model, there may be benefits to using signatures as opposed to MACs. The main incentive here is to protect the server from a malicious (or litigious) client who might fraudulently claim that the server mangled or faked tuples in the outsourced database. Record-level signatures would obviate this problem.

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 tuples in a query reply, the querier must at least recompute a hash or a similar function. Fortunately, there exist digital signature schemes that can be used to construct idealized or near-idealized solutions. Such signature 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 of these 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.

We propose two concrete techniques that support aggregation of multiple signatures. The first one is an extension of the well-known 'batch verification of RSA' method, and the second is the recently proposed scheme due to Boneh, et al.[Boneh et al. 2003].

6. SIGNATURE SCHEMES

Signature schemes that allow aggregation seem to be ideally suited for the ODB application. In this section, we examine some viable solutions and also point out their limitations. We first propose a novel scheme to aggregate RSA signatures generated by a single signer. We then introduce an aggregate signature scheme by Boneh et al. (hereafter referred to as the BGLS scheme) and conclude with the discussion of the applicability of DLP-based signature schemes.

6.1 Condensed-RSA

The well-known RSA [Rivest et al. 1978] signature, due to its multiplicatively homomorphic property, is suitable for combining signatures generated by a single signer into one *condensed* signature. As described below, a successful verification of a condensed RSA signature signifies (to the verifier) that each individual signature contained in the condensed signature has been, indeed, generated by the actual signer. Aggregation of RSA signatures can be performed "incrementally" by any party who has access to individual signatures.

RSA: We first describe the parameters involved in a standard RSA signature scheme. Each signer has a public key $pk = (n, e)$ and a secret key $sk = (d)$, where n is a k-bit modulus generated as 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 believed to be based on the intractability of the integer factorization problem.

An RSA signature is computed on the hash of the input message. Let $h()$ denote a cryptographically strong hash function (such as SHA-1) which takes a variable-length input and produces a fixed-length output. For an input message m, $h(m)$ denotes the hash of m and a standard RSA signature on message m is computed as: $\sigma = h(m)^d \pmod n$. (In practice, $h(m)$ is often padded before signing, following

the RSA-PSS/PKCS#1 format. However, we ignore the padding in the remainder of this paper.)

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.

Condensed-RSA Signature:

Given t different 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:

$$\sigma_{1,t} = \prod_{i=1}^{t} \sigma_i \pmod{n} \tag{1}$$

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:

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

Batch verification of RSA signatures:

Condensed-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 verification of RSA [Harn 1998b],[Bellare et al. 1998],[Yen and Laih 1995] aims at speeding up the 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 [Bellare et al. 1998]) consists of checking that:

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

Bellare et al. in [Bellare et al. 1998] prove that batch verification of RSA is secure under the assumption that RSA is a collection of one-way functions. They also 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 Condensed-RSA and Batch verification of RSA is that in the latter, the product of individual 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 Condensed-RSA, where a single aggregated signature is returned, this auditing is not possible. Therefore in Condensed-RSA, in the event of a verification failure, further steps (perhaps involving off-line mechanisms) become necessary.

Condensed-RSA Security:

We claim that *Condensed-RSA is unforgeable against an adaptive chosen message*

attack. In this section, we defend this claim by showing that if there is a probabilistic poly-time adversary $\mathcal{A}$ who can break Condensed-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* Condensed-RSA. $\mathcal{A}$ succeeds in breaking Condensed-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 Condensed-RSA by demonstrating that it is at least as secure as Batch verification of RSA, which in turn was shown in [Bellare et al. 1998] 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 [Bellare and Rogaway 1993]. FDH is a hash function $H_{FDH} : \{0,1\}^* \rightarrow Z_n^*$

Outline of Security: Adversary $\mathcal{A}$ 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) Generates $(t-1)$ random numbers $s_i \in_R Z_n^*$, $\forall i \in \{1, \ldots, t-1\}$.
(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) $\mathcal{B}$ 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 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 hence the Batch verification test succeeds.

Claim 2: If RSA is one way, then Condensed-RSA is a secure scheme.
Batch verification of RSA was shown to be secure under the assumption that RSA is one way. Therefore, since Condensed-RSA is at least as secure as Batch verification of RSA, it can be concluded that Condensed-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 Condensed-RSA with those of standard RSA and Batch verification of 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 tuples) 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 verification of 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 Condensed-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.

Condensed-RSA in ODB application: Condensed-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 tuples that match the query predicate; fetch the signatures corresponding to these tuples; aggregate them (by multiplying them, as mentioned above) and send back the single aggregated signature along with the tuples 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 linear in the number of signers. The client can verify these *partially* aggregated signatures by performing one verification per signer.

6.2 BGLS

Boneh, et al. in [Boneh et al. 2003] construct an interesting signature scheme that allows incremental aggregation of signatures generated by multiple signers on different messages into one short signature based on elliptic curves and bilinear maps. This scheme (BGLS) operates in a Gap Diffie-Hellman group (GDH) – a group where the Decisional Diffie-Hellman problem (DDH) is easy while the Computational Diffie-Hellman problem (CDH) is hard. The first instance of such a group was illustrated in [Joux and Nguyen 2001]. Before describing BGLS, we briefly overview the necessary parameters:

—G_1 is a cyclic additive group with generator g_1

—G_2 is a cyclic multiplicative group

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

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

(1) Bilinearity: $\forall P, Q \in G_1$ and $a, b \in \mathbb{Z}$, $e(aP, bQ) = e(P, Q)^{ab}$

(2) Non-degenerativity: $e(g_1, g_1) \neq 1$

These two properties imply that, for any $P_1, P_2, Q \in G_1, e(P_1 + P_2, Q) = e(P_1, Q) \cdot e(P_2, Q)$; and, for any $P, Q \in G_1, e(\psi(P), Q) = e(\psi(Q), P)$.

BGLS Signature Scheme: BGLS requires the use of a full-domain hash function $h() : \{0,1\}^* \rightarrow G_1$ that maps binary strings to non-zero points in G_1. Key generation involves picking a random $x \in \mathbb{Z}_p$, and computing $v = xg_1$. The public key is $v \in G_1$ 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 $\sigma = xH$. The signature is σ. To verify a signature one needs to compute $H = h(m)$ and check that $e(\sigma, g_1) = e(H, v)$.

BGLS Aggregated Signature Scheme To aggregate t BGLS signatures, one computes the point-addition operation (on the elliptic curve) of the individual signatures as follows: $\sigma_{1,t} = \sum_{i=1}^{t} \sigma_i$, where σ_i corresponds to the signature of message m_i. The aggregated signature $\sigma_{1,t}$ is of the same size as a single BGLS signature, i.e., $|p|$ bits. Similar to Condensed-RSA, the aggregation of signatures can be performed incrementally and by anyone.

Verification of an aggregate BGLS signature $\sigma_{1,t}$ involves computing the point-addition of all hashes and verifying that:

$$e(\sigma_{1,t}, g_1) = \prod_{i=1}^{t} e(H_i, v_i)$$

Due to the properties of the bilinear maps, we can expand the left hand side of the equation as follows:

$$e(\sigma_{1,t}, g_1) = e(\sum_{i=1}^{t} x_i H_i, g_1) = \prod_{i=1}^{t} e(H_i, g_1)^{x_i} = \prod_{i=1}^{t} e(H_i, x_i g_1) = \prod_{i=1}^{t} e(H_i, v_i)$$

BGLS Performance When analyzing the computation cost of verifying a BGLS signature, we consider the following operations: Map-To-Point operations which are $h() : \{0,1\}^* \rightarrow G_1$ that map binary strings to non-zero points in G_1, Scalar-Point multiplications and computation of the bilinear mappings. When there is a single signer and t messages, the verification consists of computing t map-to-point operations to obtain the message hashes, $t-1$ scalar-point multiplications to compute the product of the hashes, 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 $k*t$ map-to-point operations, $k*(t-1)$ scalar-point multiplications of hashes as well as $k+1$ bilinear mappings.

BGLS in ODB application: BGLS is applicable to all three models of ODB. The data owner(s) sign each record using BGLS signature scheme before depositing them at the server's site. Once again, the server executing a client query: selects tuples that match the query predicate; fetches the signatures corresponding to these tuples; aggregates them (as mentioned above) and sends back the single aggregated signature along with the tuples in the result set. BGLS achieves querier bandwidth efficiency by incurring constant overhead in all three flavors of ODB model. However, BGLS tends to be expensive in terms of computational costs. We 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 [Naccache et al. 1994]. Online generation of DSA or ElGamal-type signatures can be very efficient since they allow a 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.[Naccache

et al. 1994], Harn [Harn 1995],[Harn 1998a], Yen and Laih [Yen and Laih 1995] and Bellare et al. [Bellare et al. 1998]. 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 ODB application, especially in the *Multi-Owner Model*[3]. Unfortunately, there seems to be no secure way to fully *aggregate* DLP-based signatures. The batching schemes which can be aggregated (for example, [Harn 1998a]) have been shown to be insecure by Boyd and Pavlovski in [Boyd and Pavlovski 2000] 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* [Naccache et al. 1994],[Bellare et al. 1998] or *bucket test* [Bellare et al. 1998] which require the verifier to perform operations on individual signatures (e.g., in the small exponent test, as the name suggests, the verifier is required to perform a modular exponentiation involving a small exponent on each individual signature before batching). Therefore, it seems impossible to present the verifier with a single aggregated signature. In other words, using a DLP-based signature scheme along with a batch verification technique in ODB model helps only to reduce querier computation costs but does not reduce querier bandwidth requirements. (Refer to [Boyd and Pavlovski 2000] for details pertaining to secure batch-verifying of DSA signatures).

7. PERFORMANCE MEASUREMENTS

In this section, we compare and contrast the signature schemes described in section 6 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 Condensed-RSA, Batch-DSA and 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.

All computations were carried out on an Intel Pentium-3 800 MHz processor with 1GB memory running Linux and used the OpenSSL library [OpenSSL Project, available from lorg] 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. The results for BGLS are obtained by using the MIRACL library [mir] and the elliptic curve defined by the equation $y^2 = x^3 + 1$ over $\mathbb{F}_p$ where p is a 512 bit prime and q is a 160 bit prime factor of $p - 1$.

Cost Comparisons: We follow the notations summarized in Table I. We assume that the query result contains $k * t$ tuples where k denotes the number of signers (data owners) and t denotes the number of signatures (data tuples) generated by each signer.

Table II outlines the overhead (computation, storage and bandwidth) associated with each of the signature schemes in terms of the number of required cryptographic

[3]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

Table I. Notations

QC	Querier Computation
QB	Querier Bandwidth
SC	Server Computation
OC	Owner Computation
SS	Server Storage
$Mult^t(n)$	t modular multiplications with modulus of size $\|n\|$
$Exp_l^t(n)$	t modular exponentiations with modulus of size $\|n\|$ and exponent of size $\|l\|$
$Inv^t(n)$	t modular inverses with modulus of size $\|n\|$
$MTP(t)$	t Map-To-Point operations which are $h() : \{0,1\}^* \rightarrow G_1$ that map binary strings to non-zero points in G_1
$SPM(t)$	t Scalar-Point-Multiplications
$BM(t)$	t bilinear mappings

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

The OC costs are measured per signature. Note that the number of signers $k = 1$ for both Unified Client (UC) and Multi-Querier (MQ) models. In the UC/MQ models, both Condensed-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 Condensed-RSA but two bilinear mapping in BGLS, making Condensed-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 Condensed-RSA and BGLS (since they both require the server to combine individual signatures to generate the aggregated signature).

Table II. Cost comparison in the ODB model

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

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 Condensed-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, Condensed-RSA performs a lot 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 a lot cheaper than map-to-point operations and scalar additions in Elliptic curves (in Field F_p where $|p|$ is 512 bits).

Table III 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. In the table k denotes the total number of signers and t denotes the number of signatures generated by each signer. 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.

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

		Condensed-RSA	Batch-DSA	BGLS
Sign	1 signature	6.82	3.82	12.0
Verify	1 signature	0.16	8.52	77.4
	t = 1000, k = 1	44.12	1623.59	12085.4
	t = 100, k = 10	45.16	1655.86	12320.2

7.1 Discussion

From the formulas and results presented in Tables 2 and 3 above, it is clear that Condensed RSA is superior to both BGLS and Batch-DSA in terms of QC. This will indeed be the case for all possible combinations of number of signers and number of signatures per signer. Verifying BGLS aggregate signatures requires an additional expensive bilinear mapping operation (BM) for every signer involved, making its QC costly in the face of multiple signers. However, if we consider *query execution time (QE)* to be the total time it takes to transmit the aggregated signature from the server to the client and verification of the signature, then BGLS becomes the favored aggregate signature algorithm for certain scenarios. Observe from Table 2 that the querier bandwidth for BGLS remains constant (p bits), while for Condensed RSA and Batch-DSA it increases linearly with the number of signers. Figure 4 depicts QE for a varying number of signers (k) where every signer signs 3 records each and the assumed bandwidth between clients and the database server is 14.4 kbps. Although QC is cheaper for both Condensed RSA and Batch-DSA than for BGLS, the communication cost of the former two schemes begins to dominate the QE cost as the number of signers increases. Batch-DSA especially suffers from high transmission time because the size of its aggregate signature depends on the product of number of signers and number of signatures per signer. Therefore, for clients equipped with very low bandwidth, i.e. a 14.4 kbps GRPS connection, and

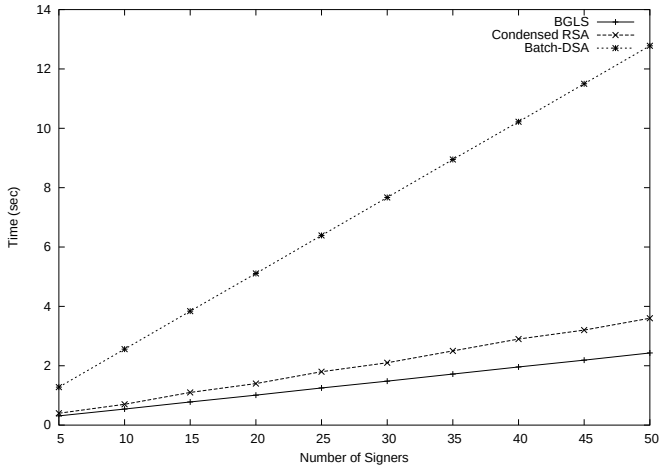

Fig. 4. Comparison of Total Query Execution Time (querier computation and cost of transmitting the aggregate signature)

with many signers and few signatures per signer, it may be favorable to use BGLS. For all other cases, Condensed RSA is the aggregate signature scheme of choice.

8. IMMUTABILITY EXTENSIONS

Although the Condensed RSA and BGLS signature techniques are fairly practical, each exhibits a potentially undesirable *mutability* feature. *Mutability* means that anyone in possession of multiple aggregated signatures can derive new and valid (authentic) aggregated signatures which may correspond to un-posed queries. For example, consider a database with two relations *employee* and *department* with the following respective schemas: *employee(empID, name, salary, deptID)* and *department(deptID, managerID)*. We now suppose that two SQL queries are posed, as shown in Figure 5. The first (Q1) asks for names and salaries of all managers with salary $> 100K$ and the second (Q2) asks for the same information for all managers with salary $\geq 140K$.

A querier who previously posed queries Q1 and Q2 and obtained corresponding replies along with their aggregated signatures S1 and S2, can compute, on her own, a valid new signature for the un-posed query Q3, as shown in Figure 5. In essence, the reply to Q3 is $(Q1 - Q2)$, i.e., information about all managers who earn between 100K and 140K. The specifics of computing a new signature from a set of existing signatures depend on the underlying aggregated signature scheme, as described in the next section.

We note that the above example is not specific to the use of aggregated signature schemes for integrity purposes in the ODB context. If, instead of aggregated signatures, plain tuple-level signatures were used (e.g., DSA or RSA), a single SELECT-style query would cause the server to compose a query reply containing a set of tuples matching the query predicate, each accompanied by its signature. The querier can then easily construct legitimate and authentic query replies for un-posed queries, since she is free to manipulate individual tuple-level signatures.

Furthermore, other methods, such as the constructs based on Merkle Hash Trees (MHTs) suggested by Devanbu, et al. [Devanbu et al. 2000], are equally susceptible to mutability of authentic query replies. (In [Devanbu et al. 2000], a querier obtains a set of tuples matching a posed query along with a set of of non-leaf nodes of an MHT. The exact composition of this set depends on the type of a query.) We now consider a few concrete scenarios where mutability is undesirable.

Q1.
```
SELECT e.name, e.salary
FROM employee e, department d
WHERE e.empID = d.managerID AND e.salary > 100000
```

Q2.
```
SELECT e.name, e.salary
FROM employee e, department d
WHERE e.empID = d.managerID AND e.salary ≥ 140000
```

Q3.
```
SELECT e.name, e.salary
FROM employee e, department d
WHERE e.empID = d.managerID AND
    e.salary BETWEEN 100000 AND 140000
```

Fig. 5. SQL Queries

Paid Database Services: Mutability is undesirable when the data owner wants to offer *paid database services* in association with the server. (This clearly applies only to the Multi-Querier and Multi-Owner ODB scenarios.) In essence, a server can be viewed as an *authorized re-distribution agent* for the information contained in, or derived from, the outsourced database. Consequently, one reason for avoiding mutability is to prevent *unauthorized splitting and re-distribution* of authentic query replies. For example, consider the case of data owner and/or server who wishes to charge a fee for each query over the outsourced database. Therefore, it might be important to prevent queriers from deriving new valid aggregated signatures from prior query reply sets and re-selling information that has not been paid for.

To make the example more specific, consider an on-line authorized music distributor with a large database of songs, each digitally signed by the artist. Suppose that the distributor only wishes to sell complete albums (compilations) and not individual songs. The distributor (server) can then simply aggregate the signatures of individual tracks to provide its clients a unified proof of authenticity and integrity for the entire album. In this case, signature aggregation gives the distributor the means to *mix and match* the songs to make various compilations.

One concern that would arise in this scenario (due to the mutability of the underlying aggregated signature scheme) is that clients could potentially start their own music distribution services with the goal of reselling individual songs at higher cost per song than that of the original distributor (who only sells complete albums).

Content Access Control: Consider the ODB scenario where the owner wants the server to enforce a certain content access control mechanism: for each client

(or a group of clients) access is restricted to a specific subset of the database. A client who poses a query only gets back the data that she is entitled to see based on her specific *privileges*. If the database is a collection of individually signed tuples, the server can aggregate individual tuple signatures to construct a single proof of authenticity and integrity for the entire query reply.

Two colluding clients (each with different access control privileges) can share their respective query replies. If the aggregated signature scheme is mutable, the two clients, by further aggregating the two query replies into a single quantity, can convince others that they have more privileges than they really do. In other words, a client can *combine* two aggregated signatures to produce a new and authentic aggregated signature that can act as proof that she has higher access privileges. This can have undesirable implications.

9. MUTABILITY OF PROPOSED SIGNATURE SCHEMES

This section demonstrates the mutability properties of both Condensed RSA as well as the BGLS signature schemes.

Mutability of Condensed RSA: Given two condensed signatures: $\sigma_{1,i}$ on messages $\{m_1, ..., m_i\}$ and $\sigma_{1,j}$ on messages $\{m_1, ..., m_j\}$ where $j < i$, it is possible to obtain a new condensed signature $\sigma_{j+1,i}$ on messages $\{m_{j+1}, ..., m_i\}$ by simply dividing $\sigma_{1,i}$ by $\sigma_{1,j}$ (modulo n).

$$(\sigma_{j+1,i}) \equiv (\sigma_{1,i})/(\sigma_{1,j}) \pmod{n}$$

Similarly, given two condensed signatures $\sigma_{1,i}$ on messages $\{m_1, ..., m_i\}$ and $\sigma_{i+1,j}$ on messages $\{m_{i+1}, ..., m_j\}$, anyone can obtain a new condensed signature $\sigma_{1,j}$ on messages $\{m_1, ..., m_i, m_{i+1}, ..., m_j\}$ (assuming all messages are distinct) by multiplying $\sigma_{1,i}$ and $\sigma_{i+1,j}$:

$$(\sigma_{1,j}) \equiv (\sigma_{1,i}) \times (\sigma_{i+1,j}) \pmod{n}$$

Mutability of Aggregated BGLS: Similar to Condensed-RSA, aggregated BGLS signatures can be manipulated to obtain new and valid signatures that correspond to un-posed query replies. Specifically, it is possible to either (or both) add and subtract available aggregated signatures to obtain new ones.

For example, given 2 aggregated BGLS signatures $\sigma_{1,i}$ on messages $\{m_1, ..., m_i\}$ and $\sigma_{i+1,j}$ on messages $\{m_{i+1}, ..., m_j\}$, if the messages $\{m_1, ..., m_i\}$ and $\{m_{i+1}, ..., m_j\}$ are all distinct (i.e.,the two queries do not overlap), the verifier can obtain a new BGLS signature $\sigma_{1,j}$ on messages $\{m_1, ..., m_i, m_{i+1}, ...m_j\}$ by adding $\sigma_{1,i}$ and $\sigma_{i+1,j}$.

$$(\sigma_{1,j}) \equiv (\sigma_{1,i}) + (\sigma_{i+1,j}) \pmod{p}$$

10. IMMUTABLE SIGNATURE SCHEMES

In this section, we propose extensions that strengthen previously described signature schemes and make them *immutable*.

10.1 Immutable Condensed RSA (IC-RSA)

To make Condensed-RSA signatures immutable, we use the technique of *proof of knowledge* of signatures. The server, instead of revealing the actual aggregated signature for a posed query, reveals only the proof of knowledge of that signature.

Since these proofs are essentially zero knowledge, they cannot be used or manipulated by the clients. In other words, the clients learn nothing about the signature (except its validity) from the server and further these proofs cannot be used to generate *new* proofs corresponding to unposed queries. We start our discussion below with the interactive variant that is based on the well-known Guillou-Quisquater identification scheme, and for non-interactive variant, we provide two protocols: one which is based on Guillou-Quisquater signature scheme and the other that is based on so-called "signatures of knowledge".

10.1.1 *GQ based Interactive Protocol .* This technique uses the well-known Guillou-Quisquater (GQ) identification scheme [Guillou and Quisquater 1988] which is among the most efficient follow-ons to the original Fiat-Shamir zero-knowledge identification Scheme [Amos Fiat and Adi Shamir 1987]. The version we present is an interactive protocol between the server (Prover) and the querier (Verifier) that provides the latter with a zero-knowledge proof that the Prover has a valid Condensed-RSA signature corresponding to the tuples in the query result set.

Basically, in response to a query, the server returns to the querier the result set along with a *witness*. The querier then sends a random *challenge* to which the server replies with a valid *response*. The *response* together with the *witness* convince the querier of server's knowledge of the Condensed-RSA signature, without revealing any knowledge about the Condensed-RSA signature itself. The actual protocol is shown in Figure 6. We use the terms *Prover* (P) and *Verifier* (V) instead of Server and Querier, respectively, since the protocol is not specific to the ODB setting [4]. Let $X = \sigma_{1,t} = \prod_{i=1}^{t} \sigma_i \pmod n$ be the Condensed-RSA signature computed as previously described. Recall that (e, n) is the public key of the original data-owner which all concerned parties are assumed to possess. Let $M \equiv \prod_{i=1}^{t} h(m_i) \pmod n$ and $X^e = (\sigma_{1,t})^e \equiv M \pmod n$.

In step 0, the querier poses a query (not shown in figure 6). In step 1, the server (prover) replies with the result set for that query as well as a commitment Y. Note that $Y = r^e \pmod n$ where r is a randomly chosen element in $\mathbb{Z}_n^*$ and n is the RSA modulus of the data owner who generated the individual RSA signatures corresponding to the tuples in the result set [5] and e, the corresponding public exponent. In step 2, the verifier (querier) sends back a challenge v that is chosen randomly from $\{0,1\}^{l(s)}$ where $l(s)$ is the bit-length of the public exponent e. In Step 3, server, upon receiving the challenge v, computes the response $z = rX^v \pmod n$ where X is the Condensed-RSA signature of the result set. In Step 4, the verifier accepts the proof if $z \neq 0$ and $z^e \equiv YM^v \pmod n$ where M is the product of (hashes of) all messages in the result set. Checking $z \neq 0$ precludes a malicious server from succeeding by choosing $r = 0$. Note that $z^e \equiv (rX^v)^e \equiv r^e X^{ev} \equiv Y(X^e)^v \equiv YM^v \pmod n$. Hence the protocol works.

Security Considerations: GQ is RSA-based; the interactive identity protocol

[4]The original GQ scheme proposed in [Guillou and Quisquater 1988] is *identity-based* since it is used by the Prover to prove his "identity" to the verifier. However, in the current scenario, we present a version that is not id-based and does not require a key generation phase since the server uses the public key of the data owner to prove knowledge of the Condensed-RSA signature by that data owner

[5]Recall that Condensed-RSA allows only single signer aggregation

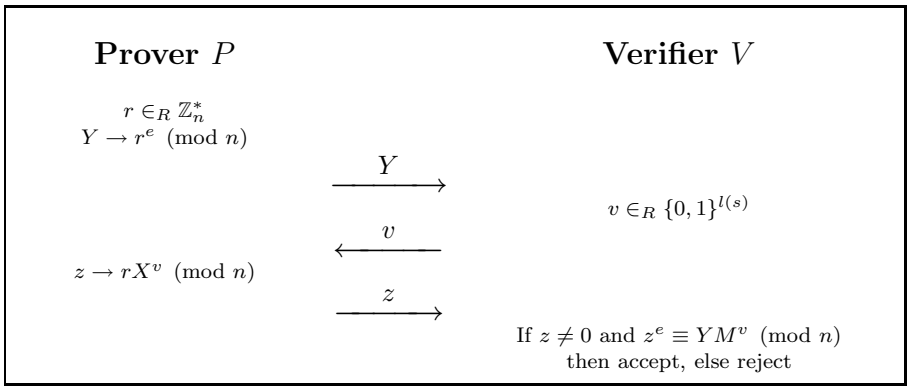

Fig. 6. IC-RSA GQ-based Interactive Technique

is known to be honest-verifier zero-knowledge and is secure against impersonation under passive attacks, assuming RSA is one-way [Guillou and Quisquater 1988]. Subsequently, it was also proven secure against impersonation under active attacks in [Bellare and Palacio 2002].

Forgery: The public exponent e defines the security level, i.e., a cheating prover can convince the verifier, and thus defeat the protocol with probability $1/e$, by correctly guessing the value of the *challenge* v *a priori*. Therefore, the bit-length of v (and, therefore, e since $v \in_R \{0,1\}^{l(s)}$) should be large enough. Note that the above protocol can be run multiple times for commensurably lower probability of successful forgery. In general, if it is run t times, the probability of forgery is e^{-t}.

Security Assumptions: The security of the protocol is based on the hardness of the RSA problem (i.e., computing e-th roots mod a composite integer n which is formed as a product of two large primes.)

10.1.2 *GQ based Non-interactive Protocol.* This technique uses the Guillou-Quisquater (GQ) signature scheme [Guillou and Quisquater 1988]. Below, we present a non-interactive protocol between the server (Prover) and the querier (Verifier) using GQ signature scheme that enables the Verifier to ensure that the Prover has a valid Condensed-RSA signature corresponding to the tuples in the query result set.

In this protocol, in response to a query, the server returns to the querier the result set along with a proof of knowledge of the signature. The proof, which is generated using the popular Guillou Quisquater signature scheme, essentially proves to the verifier (Querier) that the server knows the Condensed-RSA signature corresponding to the tuples in the result set. The actual protocol is shown in Figure 7. Essentially, this protocol is the same as the interactive zero-knowledge GQ identification protocol described above with the minor modification of the challenge being replaced with a cryptographically strong one-way hash function. The details are as follows: Let $X = \sigma_{1,t} = \prod_{i=1}^{t} \sigma_i \pmod n$ be the Condensed-RSA signature computed as shown previously. Recall that (e, n) is the public key of the original data-owner which all concerned parties are assumed to possess. Let $M \equiv \prod_{i=1}^{t} h(m_i) \pmod n$ and $X^e = (\sigma_{1,t})^e \equiv M \pmod n$.

In step 0, the querier poses a query (not shown in figure 7). In step 1, the server (prover) replies with the result set for that query as well as the GQ signature z. z is computed as $rX^H \pmod n$ where r is a randomly chosen element in $\mathbb{Z}_n^*$. H is a hash value and is computed as: $H = hash(M||Y||other\ info)$ where $Y = r^e \pmod n$ and *other info* includes values such as the posed query and a timestamp in order to guarantee uniqueness of the input to the hash function. In step 2, the verifier (querier) verifies the signature by checking that $z^e = YM^H \pmod n$. Note that $z^e \equiv (rX^H)^e \equiv r^e X^{eH} \equiv Y(X^e)^H \equiv YM^H \pmod n$. Hence the protocol works.

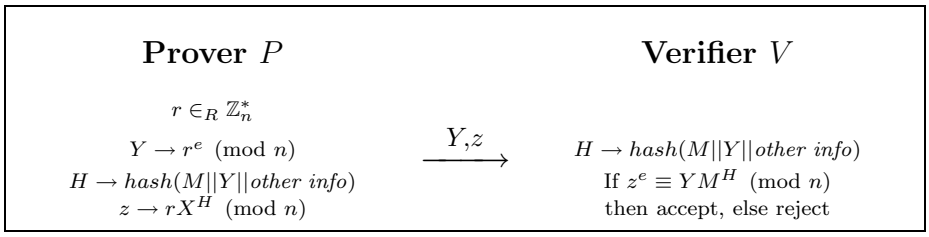

Fig. 7. IC-RSA GQ-based Non-Interactive Technique

Security Considerations: The GQ signature scheme is secure provided the public exponent e is sufficiently large. e must be large in order to exclude the possibility of forgery based on the birthday paradox [Menezes et al. 1997].

10.1.3 *Signatures of Knowledge based Non-interactive Protocol.* The second non-interactive variant uses the technique of *Signatures of Knowledge* first popularized by Camenisch and Stadler in [Camenisch and Stadler 1997]. Specifically, we use the so-called **SKROOTLOG** primitive which can be used to prove knowledge of an e-th root of the discrete logarithm of a value to a given base. Before presenting the details, we briefly describe how this technique is used in our scenario. Conceptually, the server reveals all tuples matching the query as well as a signature of knowledge for the actual Condensed-RSA signature corresponding to these tuples. A querier verifies by checking the SKROOTLOG proof. However, since the querier never actually gets the condensed signature, she can not exploit the mutability of Condensed-RSA to derive new signatures. In general, the querier can not derive proofs for any other queries by using proofs for any number of previously posed queries.

SKROOTLOG Details: Let $G =< g >$ be a cyclic group of order n. An e-th root of the discrete logarithm of $y \in G$ to the base g is an integer α satisfying $g^{(\alpha^e)} = y$ if such a α exists. If the factorization of n is unknown, for instance if n is an RSA modulus, computing e-th roots in $\mathbb{Z}_n^*$, is assumed to be infeasible. A signature of knowledge of an e-th root of the discrete logarithm of y to the base g is denoted $SKROOTLOG[\alpha : y = g^{\alpha^e}](m)$

Below, we briefly outline an efficient version of SKROOTLOG proposed in [Camenisch and Stadler 1997] which is applicable when the public exponent e is a small value (for instance, this efficient SKROOTLOG version is applicable when the value of e is set to 3).

DEFINITION 1. *If e is small, it is possible to show the proof of knowledge of the e-th root of the discrete log of $y = g^{\alpha^e}$ to the base g by computing the following $e-1$ values:*

$$y_1 = g^{\alpha}, y_2 = g^{\alpha^2}, \ldots, y_{e-1} = g^{\alpha^{e-1}}$$

and showing the signature of knowledge:

$$U = SKREP[\alpha : y_1 = g^{\alpha} \wedge y_2 = y_1^{\alpha} \wedge \ldots \wedge y = y_{e-1}^{\alpha}]$$

that the discrete logarithms between two subsequent values in the list g, y_1, ..., y_{e-1} are all equal (to α) and known (to the prover).

Below, we give the formal definition of SKREP as in [Camenisch and Stadler 1997]:

DEFINITION 2. *A signature of the knowledge of representations of $y_1, \ldots, y_w$ with respect to bases $g_1, \ldots, g_v$ on the message m is defined as follows:*

$$SKREP\left[(\alpha_1, \ldots, \alpha_u) : \left(y_1 = \prod_{j=1}^{l_1} g_{b_{1j}}^{\alpha_{e_{1j}}}\right) \wedge \ldots \wedge \left(y_w = \prod_{j=1}^{l_w} g_{b_{wj}}^{\alpha_{e_{wj}}}\right)\right](m)$$

where the indices $e_{ij} \in \{1, \ldots, u\}$ refer to the elements $\alpha_1, \ldots, \alpha_u$ and the indices $b_{ij} \in \{1, \ldots, v\}$ refer to the base elements $g_1, \ldots, g_v$.
The signature consists of an $(u+1)$ tuple $(c, s_1, \ldots, s_u) \in \{0,1\}^k \times \mathbb{Z}_n^*$ satisfying the equation

$$c = \mathcal{H}\left(m||y_1||\ldots||y_w||g_1||\ldots||g_v||\{\{e_{ij}, b_{ij}\}_{j=1}^{l_i}\}_{i=1}^{w}||y_1^c \prod_{j=1}^{l_1} g_{b_{1j}}^{s_{e_{1j}}}||\ldots||y_w^c \prod_{j=1}^{l_w} g_{b_{wj}}^{s_{e_{wj}}}\right)$$

SKREP can be computed easily if the u-tuple $(\alpha_1, \ldots, \alpha_u)$ is known. Prover first chooses $r_i \in_R \mathbb{Z}_n$ for $i = 1, \ldots, u$, computes c as

$$c = \mathcal{H}\left(m||y_1||\ldots||y_w||g_1||\ldots||g_v||\{\{e_{ij}, b_{ij}\}_{j=1}^{l_i}\}_{i=1}^{w}||\prod_{j=1}^{l_1} g_{b_{1j}}^{r_{e_{1j}}}||\ldots||\prod_{j=1}^{l_w} g_{b_{wj}}^{r_{e_{wj}}}\right)$$

and then sets $s_i = r_i - c\alpha_i \pmod n$ for $i = 1, \ldots, u$
SKROOTLOG Non-interactive IC-RSA: The server executing a client query is required to perform the following:

(1) select tuples that match the query predicate;

(2) fetch the signatures corresponding to these tuples;

(3) aggregate the signatures (by multiplying them modulo n, as mentioned above) to obtain the condensed RSA signature σ;

(4) send the individual tuples $\mathcal{M} = \{m_1, \ldots, m_t\}$ back to the querier along with a proof of knowledge of σ which is essentially a SKROOTLOG proof showing that the server knows the e-th root of g^{σ^e}. In other words, SKROOTLOG proof

shows that the server knows the e-th root of $g^{\prod m_i}$. In order to show this, the server sends g^{σ}, g^{σ^2} and[6] the SKREP proof computed as above.

Security Considerations: In practice, the efficient version of the SKROOTLOG proof which was described in the previous section cannot be used as is. This is because the values $y_1 \ldots y_{e-1}$ that are required for the SKREP proofs leak additional information about the secret. Hence a randomized version that is proposed in [Camenisch and Stadler 1997] needs to be used. The interactive protocol corresponding to the above definition of SKROOTLOG is proven honest-verifier zero-knowledge in [Camenisch 1998]. For brevity, we skip the details of this discussion and refer interested readers to [Camenisch and Stadler 1997]. However, we would like to note that the security of the SKROOTLOG protocol is based on the difficulty of the discrete logarithm problem and the RSA problem. In addition, SKREP is based on the security of Schnorr signature scheme. The Non-Interactive IC-RSA, which in essence is the SKROOTLOG primitive, is therefore honest-verifier zero-knowledge [Camenisch 1998]. This implies that the querier who is given only the proof of the condensed RSA signature, can not derive new signatures. In addition, the querier can not derive new proofs for any other queries by using proofs for any number of previously posed queries.

10.1.4 *Discussion.* In this section, we compare the three techniques presented above.

—**Initialization and Parameter Generation:** SKROOTLOG Non-interactive technique requires an elaborate parameter generation phase at the server. For each data owner whose RSA public key is (n, e), the server needs to generate a large prime $p' = j * n + 1$ (where n is the RSA modulus and j is some integer) and an element $g \in \mathbb{Z}^*_{p'}$ such that order of g is n. On the other hand, GQ based techniques require no additional parameter generation at the server since the server only requires to have knowledge of each data owner's RSA public key (n, e).

—**Verifiability:** In the GQ based non-Interactive as well as the SKROOTLOG techniques, the proof provided by the server is universally verifiable (or in other words, the proof is self authenticating and hence transferable). On the other hand, the GQ-based Interactive technique provides guarantees only to the interactive verifier who poses the challenge and the proof of knowledge in this case is non-transferrable.

—**Security Considerations:** The security of GQ based variants is based primarily on the hardness of the RSA problem (i.e., computing e-th roots mod a composite integer n which is formed as a product of two large primes.). Additionally, the security of the non-interactive variant relies on the collision resistance of the hash function which is used in place of a random challenge. On the other hand, the SKROOTLOG based technique mainly relies on the hardness of the discrete log problem (specifically, the SKREP primitive is based on the security of Schnorr

[6]Note that the server need not send $g^{\sigma^3} = g^{\prod m_i}$ explicitly since the querier can compute this value knowing g and the individual m_i-s

signature scheme). Thus, the two non-interactive techniques presented above are based on different security assumptions.

—**Communication Rounds:** Since SKROOTLOG non-interactive techniques require no interaction with the verifier for the proof, they require no additional rounds of communication. In other words, the server executes the query and returns the result set as well as the proof of knowledge of the corresponding unified Condensed-RSA signature. On the other hand, the GQ-based Interactive technique requires two additional rounds of communication with the verifier.

10.2 Immutable BGLS (iBGLS)

The extension to aggregated BGLS to achieve immutability is very simple: The server computes its own signature on the whole query reply and aggregates it with the aggregated BGLS signature of the owners. In other words, for a given query whose result includes messages $\{m_1, m_2, \ldots, m_k\}$, the server computes $\mathcal{H} = h(m_1||m_2...||m_k||$ other information[7]) where $||$ denotes concatenation, and signs this hash $\mathcal{H}$ using its own private key x_s to obtain $x_s\mathcal{H}$ and computes:

$$\sigma = \sigma_{1,t} + x_s\mathcal{H}$$

where $\sigma_{1,t}$ is the aggregated BGLS signature of t messages obtained as previously described.

Now, a valid and authentic query reply comprises of the tuples in the result set along with an authentic iBGLS signature on the entire result set. Due to this simple extension, it is no longer feasible for anybody to manipulate the existing iBGLS signatures to obtain new and authentic ones or get any information about individual component BGLS signatures. Verification of an iBGLS signature σ involves computing the individual hashes H_i-s of each message as well as computing the hash of the concatenation of all the messages $\mathcal{H}$ and verifying the following equality: $e(\sigma, g_1) = \prod_{i=1}^{t} e(H_i, v_i).e(\mathcal{H}, v_s)$ where v_s is the server's public key. Due to the properties of the bilinear mapping, we can expand the left hand side of the equation as follows:

$$e(\sigma, g_1) = e(\sum_{i=1}^{t} x_i H_i + x_s\mathcal{H}, g_1) = e(\sum_{i=1}^{t} x_i H_i, g_1).e(x_s\mathcal{H}, g_1) = \prod_{i=1}^{t} e(H_i, g_1)^{x_i}.e(\mathcal{H}, g_1)^{x_s} = \prod_{i=1}^{t} e(H_i, x_i g_1).e(\mathcal{H}, x_s g_1) = \prod_{i=1}^{t} e(H_i, v_i).e(\mathcal{H}, v_s)$$

Security Considerations: iBGLS is a direct application of the original aggregate BGLS signature scheme (see section 6.2). The security of BGLS relies upon a Gap-Diffie Hellman group setting and specifically requires that each message included in the aggregated signature be unique. Below we argue, informally, the security of our construction of iBGLS signatures.

[7]BGLS aggregation is secure only when the messages signed are all distinct. Therefore, if the server signed only the result set, it can potentially create a problem if the result set to a particular query contained a single tuple (message). Note that if the result set contained only one message then the owner's message as well as the server's message would be the same since server computes its signature on the entire result set. In order to avoid such situations, we require that the server add some additional information to the message that he signs. For example, the server may include query and/or querier specific information, timestamp etc.

Let r_i denote database tuple i. An iBGLS signature σ is then constructed as follows: $\sigma = \sum_{i=1}^{t+1} x_i h(m_i)$, where messages $m_1, m_2, ..., m_t$ correspond to the selected tuples $r_1, r_2, ..., r_t$ and $m_{t+1} = (r_1||r_2||...||r_t)$, i.e., the concatenation of the these tuples. $x_1, x_2, ..., x_t$ are the data owner's private keys and x_{t+1} is the server's key. We then claim that these $t+1$ messages are distinct. Each database tuple r_i contains a unique tuple identifier, resulting in messages $m_1, m_2, ..., m_t$ being distinct. m_{t+1} is unique for each tuple set returned, as it consists exactly of the selected tuples, and moreover, it is distinct from any m_j where $j < t+1$. Therefore, all $t+1$ messages are distinct.

The immutability property of the above scheme relies upon the inability of an adversary to forge the server's signature. This, in turn, implies that such an adversary cannot use an aggregate BGLS signature to generate a new iBGLS signature that verifies. In other words, the iBGLS signature construction is resistant to mutations assuming that BGLS signatures are unforgeable.

11. PERFORMANCE ANALYSIS

In this section, we present and discuss the experimental results for immutable signature schemes. We mainly consider the overheads introduced by the extensions we made to the Condensed-RSA and BGLS signature schemes to achieve immutability. We first provide estimates on the running costs by showing the number of additional basic cryptographic operations (such as modular exponentiations and multiplications) required by the extensions and also point out any additional communication overheads. We then present the actual cost (time) required to perform these additional operations.

Table IV enlists the computational as well as communication overheads associated with the various techniques we propose in Section 10. We use the same notation to describe the various basic cryptographic operations as described in Table I.

Table IV. Cost comparison of techniques for Immutability

	Computation		Communication
	at Client	at Server	
GQ Interactive	$Mult^1(n) + Exp_e^2(n)$	$Mult^1(n) + Exp_e^2(n)$	2
GQ Non-interactive	$Mult^1(n) + Exp_e^2(n)$	$Mult^1(n) + Exp_e^2(n)$	0
SKROOTLOG	$Exp_n^4(p') + Mult^3(p')$	$Exp_n^4(p') + Exp_2^1(n)$ $+Mult^1(n)$	0
iBGLS	$BM(1)$	$MTP(1) + SPM(1)$	0

Table V recaps 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 both condensed RSA and BGLS schemes. We set the RSA public exponent e to 3 for SKROOTLOG and set $e = (2^{30} + 1)$ for GQ based protocols. Note that it is essential to have a large e for GQ since e in this case is also the security parameter. Further, we have used a 1024 bit modulus n and the Chinese Remainder Theorem to speed up the signing procedure. All computations were done on an Intel Pentium-3 800 MHz processor with 1GB memory. The results for BGLS are obtained by using the MIRACL library [mir]

and the elliptic curve defined by the equation $y^2 = x^3 + 1$ over $\mathbb{F}_p$ where p is a 512 bit prime and q is a 160 prime factor of $p - 1$. In the table k denotes the total number of signers and t denotes the number of signatures generated by each signer.

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

		Condensed-RSA		BGLS
		$e = 3$	$e = 2^{30} + 1$	
Sign	1 signature	6.82	6.82	12.0
Verify	1 signature	0.14	0.56	77.4
	t = 1000, k = 1	44.12	45.531	12085.4
	t = 100, k = 10	45.16	50.31	12320.2

The next table (table VI) gives the time required to generate and verify an immutable signature under the different RSA-based techniques as well as the BGLS extension. In this table, we only measure the **overhead** associated with the immutability extensions. Therefore, the costs do not include the original Condensed-RSA or BGLS costs. In addition, we also do not count the communication delays introduced by the protocols, particularly in the case of interactive GQ which is a multi-round protocol. As a result, the cost overheads incurred for both the interactive and the non-interactive GQ extensions are the same.

Table VI. Cost comparison (time in msec): Immutable Signature Schemes

Technique Used	Computation at Client	Computation at Server	Total Overhead
GQ Interactive	0.309	0.309	0.618
GQ Non-interactive	0.309	0.309	0.618
SKROOTLOG	48.88	46.489	89.369
iBGLS	37.2	12.0	49.2

We also would like to mention at this point that SKROOTLOG, in addition to the above mentioned costs, also incurs additional setup costs. These costs are necessary to set the parameters prime p' and element g of order n. Further, we also note that since Condensed-RSA only enables single-signer aggregation, it is necessary for the server to set up multiple sets of parameters: one for each signer. (In other words, since moduli's n of distinct owners are different, it becomes necessary to find distinct pairs (p', g) for each n).

12. PRIOR WORK

In this section, we briefly explain the general approach of using authenticated data structures to provide authentication of query replies and then describe two closely related works that make use of this approach in the specific contexts of "Third party data publishing" and "Edge Computing".

The general approach of authenticated data structures builds upon the work by Merkle [Merkle 1980]. Merkle introduced a technique referred to as the Merkle Hash Trees which were initially used for the purpose of one-time signatures and in

Public Key Infrastructures to provide authentic and secure certificate revocation. Merkle tree construction and usage are as follows:

A Merkle hash tree (MHT) intended for authentic values $x_1, x_2, ..., x_n$ is constructed by building a tree in which the leaves correspond to the hashes of the values of the ordered elements in the set. Therefore, a leaf associated with element x_i contains the value $h(x_i)$, where $h()$ is a cryptographic one-way hash function, such as SHA-1. The values of the internal nodes correspond to the hash value of the concatenation of its children (maintaining their order). An internal node with children v_1 and v_2 therefore has the value $h(v_1||v_2)$. The root of the tree is digitally signed. A MHT can be used to prove the existence of an element in the set with the help of a *verification object*. A verification object essentially contains a set of hashes which helps the verifier in recomputing the hash of the root of the MHT and the verifier can then check the correctness of the root hash by verifying the signature on it. As an example, if we search for 5 in the MHT in figure 8, the verification object (VO) contains node values $5, h_1, h_{34}$, as well as the signature of the root. The verifier constructs $h'_2 = h(5)$, $h'_{12} = h(h_1||h'_2)$ and finally $h'_{1234} = h(h'_{12}||h_{34})$ and verifies the root by checking its signature.

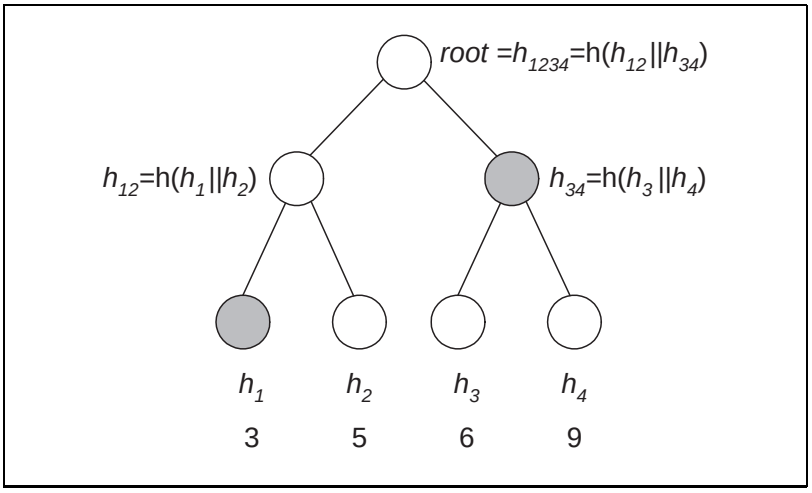

Fig. 8. Merkle Tree Example: shaded nodes form the VO for leaf value 5

12.1 Authentic Third Party Data Publication

In [Devanbu et al. 2000], Devanbu et al. mainly address the third party publication application which is essentially the multi-querier model of the ODB. Here, the (trusted) data owner produces integrity-critical content which is published by third party servers with an aim to widely disseminate the content. In their work, the authors essentially use the MHT approach to provide authentic third party data publication. The significant contributions of this work are as follows:

—Instead of using in-efficient binary search trees as authenticated dictionaries, the authors suggest and demonstrate the use of balanced and I/O efficient data structures such as B-trees.
—The authors suggest combining the authenticated data structures with existing index (if any) to reduce the overall query authentication time (and hence the overall query processing time)
—Authors suggest using more complex data structures such as: 1) multi-dimensional range trees in order to support conjunctive query predicates involving multiple dimensions (attributes), and 2) tries or suffix trees to support sub-string searches.
—Demonstrate how efficient and compact verification objects can be constructed if a pre-computed authenticated data structure for that type of query exists. Here, efficient and compact generally translate to logarithmic complexity in the size of the database.

One of the limitations of the above approach is the need to pre-compute and store a potentially large number of authenticated data structures. In other words, for efficient query verification, this method requires separate trees over different sort orders of the tuple sets. Without such pre-computed trees, this approach cannot provide small verification objects. To support simple range queries involving a single attribute, this approach requires pre-computed hash trees for all possible sort orders of the relation. This, in turn, introduces significant pre-computation costs and storage overheads at the server. Additionally, storing multiple trees for the same relation increases the cost of updates as well. In summary, this approach cannot efficiently support interactive, arbitrary querying. In other words, it can support applications where the querier applications commit *a priori* the queries that the applications wish to pose; the owner and the publisher then pre-compute the required data structures

In contrast to the approach presented in [Devanbu et al. 2000], our solution does not require pre-computation of data structures in order to answer specific queries. In fact, our technique of using tuple-level signatures is flexible in that VOs can be created for a wide range of SQL queries involving *SELECT* clauses. By not using additional data structures, we reduce the storage overhead at the server while also keeping updates efficient since only the affected tuples need to be resigned as against recomputing (and possibly rebalancing) all the data structures containing the affected tuples.

12.2 Authenticating Query Results in Edge Computing:

In [Pang and Tan 2004], Pang et al. mainly address edge computing. In this application, the (trusted) centralized server outsources (parts of) the database to proxy servers situated at the edge of the network in an attempt to reduce the network latency and thus improve the overall query execution time of the client's applications. In their work, the authors use a VB-tree which is a modified MHT built using a B-Tree where instead of signing only the root, all leaf nodes as well as all internal nodes are also signed. As a result, the VO is smaller and is linear only in the size of the result set and independent of the size of the database. (In comparison, the most efficient VO using Devambu's approach [Devanbu et al. 2000] is logarithmic in the size of the database in addition to being linear in the size of the result set.)

Furthermore, the authors suggest signing individual attributes and they make use of homomorphic hashing techniques and consequently, can execute projection queries at the servers' site as opposed to the approach outlined in [Devanbu et al. 2000] where projections need to be performed by the clients.

One of the main disadvantages of this method is clearly the high cost associated with the approach in terms of both computation and storage. It is necessary to generate signatures on individual attributes as well as entire tuples (as opposed to signing only the root in the previous approach) and the resultant tree has to also store all these additional signatures at all nodes of the tree. Our technique presented in this paper utilizes signatures at the tuple level and only creates one signature per tuple. We thereby avoid the two large costs associated with the technique in [Pang and Tan 2004], namely the pre-computation of signatures over all individual attribute fields of every tuple in the database, and the storage overhead resulting from the large number of signatures stored at the server (standard RSA signatures require 1024 bits each).

Furthermore, the approach in [Pang and Tan 2004] can be very expensive in terms of VO verification time for queriers. This is because the verification object includes signed digests for all the attributes that are filtered out as well as all the tuples that do not belong to the query result set but do fall inside the enveloping tree [8] for a given query. In order to authenticate the query results, the scheme requires the querier to verify the signatures of all these filtered attributes and tuples that are not part of the actual result set. Clearly, receiving (recall that a signature is at least 128 bytes long) and verifying (a single RSA signature verification takes 0.16 msec on P3-977 MHz machine) all these signatures can be computationally very expensive for the querier. In contrast, our VO does not contain any tuples that are not part of the actual result set.

13. 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 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 [Hacigümüş et al. 2002b]. In our work, we use a similar system model as outlined in this paper. The problem of Private Information Retrieval (PIR) [Chor et al. 1998],[Gertner et al. 1998] 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 [Chor et al. 1998] of the data or using keywords [Chor et al. 1997]. However, most of the currently available PIR techniques aim for very

[8] The enveloping tree is the smallest subtree within the VB-tree that envelops all the result tuples of the query

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. [Song et al. 2000] 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. In summary, searching on encrypted data is becoming an increasingly popular research topic with such recent interesting results as [Song et al. 2000],[Goh 2003]. However, all the above mentioned schemes only support exact match queries in which the server returns data that matches either the given physical address or the given keyword. Hacigümüş et al. in [Hacigümüş et al. 2002] 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 [Hacigümüş et al. 2002a]. 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 encryption. Further, they provide authentication and non-repudiation which are important requirements in our model. Also, in order to facilitate efficient integrity assessment, we build a simple RSA [Rivest et al. 1978] based aggregation technique (Condensed-RSA) which is essentially based on the technique of batch verification [Bellare et al. 1998] of RSA signatures. The idea of batching to amortize the cost of private key operations (signing and decryption) was first introduced by Fiat in [Fiat 1990],[Fiat 1997]. We also compare this solution to aggregate signatures proposed by Boneh et al. [Boneh et al. 2003]. The signature scheme proposed in [Boneh et al. 2003] is based on bilinear maps and provides an elegant mechanism to aggregate n individual signatures on n distinct messages generated by n signers into one single signature.

14. FUTURE WORK AND CONCLUSION

In conclusion, this work considered the problem of providing efficient data integrity mechanisms in the outsourced database model. We presented a novel approach based on signature aggregation techniques. Specifically, we proposed a secure and practical *Condensed-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

MIRACL Library. `http://indigo.ie/~mscott`.

Amos Fiat and Adi Shamir. 1987. How to prove yourself: practical solutions to identification and signature problems. In *Advances in Cryptology – CRYPTO '86*, A. M. Odlyzko, Ed. Number 263 in Lecture Notes in Computer Science. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, Santa Barbara, CA, USA, 186–194.

Bellare, M., Garay, J., and Rabin, T. 1998. Fast Batch Verification for Modular Exponentiation and Digital Signatures. In *Eurocrypt 1998*. Vol. 1403. 191–204.

Bellare, M. and Palacio, A. 2002. Gq and schnorr identification schemes: Proofs of security against impersonation under active and concurrent attacks. In *Advances in Cryptology – CRYPTO '02*, M. Yung, Ed. Number 2442 in Lecture Notes in Computer Science. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 162–177.

Bellare, M. and Rogaway, P. 1993. Random oracles are practical: a paradigm for designing efficient protocols. In *ACM Press*. 62–73.

Boneh, D., Gentry, C., Lynn, B., and Shacham, H. 2003. 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.

Boyd, C. and Pavlovski, C. 2000. Attacking and repairing batch verification schemes. In *Asiacrypt 2000*. 58–71.

Camenisch, J. 1998. Group Signature Schemes and Payment Systems Based on the Discrete Logarithm Problem. Vol. 2 of ETH-Series in Information Security and Cryptography. ISBN 3-89649-286-1, Hartung-Gorre Verlag, Konstanz.

Camenisch, J. and Stadler, M. 1997. Efficient Group Signature Schemes for Large Groups. In *Advances in Cryptology – CRYPTO '97*. Number 1294 in Lecture Notes in Computer Science. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 410–424.

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

Chor, B., Goldreich, O., Kushilevitz, E., and Sudan, M. 1998. Private Information Retrieval. *Journal of ACM 45,* 6 (Nov.), 965–981.

Devanbu, P., Gertz, M., Martel, C., and Stubblebine, S. G. 2000. Authentic third-party data publication. In *14th IFIP 11.3 Working Conference in Database Security*. 101–112.

Fiat, A. 1990. Batch RSA. In *Advances in Cryptology – CRYPTO '89*, G. Brassard, Ed. Number 435 in Lecture Notes in Computer Science. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, Santa Barbara, CA, USA, 175–185.

Fiat, A. 1997. Batch RSA. *Journal of Cryptology 10,* 2, 75–88.

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

Goh, E.-J. 2003. Secure indexes for efficient searching on encrypted compressed data. Cryptology ePrint Archive, Report 2003/216. `http://eprint.iacr.org/2003/216/`.

Guillou, L. and Quisquater, J. J. 1988. A "Paradoxical" Identity-Based Signature Scheme Resulting from Zero-Knowledge. In *Advances in Cryptology – CRYPTO '88*, S. Goldwasser, Ed. Number 403 in Lecture Notes in Computer Science. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, Santa Barbara, CA, USA.

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

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

Hacigümüş, H., Iyer, B., and Mehrotra, S. 2002b. Providing Database as a Service. In *International Conference on Data Engineering*.

Harn, L. 1995. DSA-type Secure Interactive Batch Verification Protocols. *Electronic Letters 31,* 4 (Feb.), 257–258.

Harn, L. 1998a. Batch Verifying Multiple DSA-type Digital Signatures. *Electronic Letters 34,* 9 (Apr.), 870–871.

Harn, L. 1998b. Batch verifying rsa signatures. *Electronic Letters 34,* 12 (Apr), 1219–1220.

Joux, A. and Nguyen, K. 2001. Separating Decision Diffie-Hellman from Diffie-Hellman in cryptographic groups. In *Cryptology ePring Archive*. Number Report 2001/003.

Law, P. 1996. The Health Insurance Portability and Accountability Act of 1996 (HIPAA). http://www.cms.hhs.gov/hipaa/. Accessed 12-May-2005.

Menezes, A. J., van Oorschot, P. C., and Vanstone, S. A. 1997. *Handbook of applied cryptography*. CRC Press series on discrete mathematics and its applications. CRC Press. ISBN 0-8493-8523-7.

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

Mykletun, E., Narasimha, M., and Tsudik, G. 2004a. Authentication and integrity in outsourced databases. In *Symposium on Network and Distributed Systems Security (NDSS'04)*.

Mykletun, E., Narasimha, M., and Tsudik, G. 2004b. Signature 'bouquets': Immutability of aggregated signatures. In *European Symposium on Research in Computer Security (ESORICS'04)*.

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

Narasimha, M. and Tsudik, G. 2005. DSAC: Integrity of Outsourced Databases with Signature Aggregation and Chaining. In *ACM Conference on Information and Knowledge Management*.

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

Pang, H. and Tan, K.-L. 2004. Authenticating Query Results in Edge Computing. In *International Conference on Data Engineering*. 560–571.

Rivest, R. L., Shamir, A., and Adleman, L. M. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM 21,* 2 (Feb.), 120–126.

Song, D., Wagner, D., and Perrig, A. 2000. Practical Techniques for Searches on Encrypted Data. In *2000 IEEE Symposium on Security and Privacy*.

United States Code. 2002. Sarbanes-Oxley Act of 2002, HR 3763, PL 107-204, 116 Stat 745. Codified in sections 11, 15, 18, 28, and 29 USC.

Yen, S. and Laih, C. 1995. Improved Digital Signature Suitable for Batch Verification. *IEEE Transactions on Computers 44,* 7 (July), 957–959.