Secure Data Management in the Cloud

Divyakant Agrawal, Amr El Abbadi, and Shiyuan Wang

Department of Computer Science, University of California at Santa Barbara
{agrawal,amr,sywang}@cs.ucsb.edu

Abstract. As the cloud paradigm becomes prevalent for hosting various applications and services, the security of the data stored in the public cloud remains a big concern that blocks the widespread use of the cloud for relational data management. Data confidentiality, integrity and availability are the three main features that are desired while providing data management and query processing functionality in the cloud. We specifically discuss achieving data confidentiality while preserving practical query performance in this paper. Data confidentiality needs to be provided in both data storage and at query access. As a result, we need to consider practical query processing on confidential data and protecting data access privacy. This paper analyzes recent techniques towards a practical comprehensive framework for supporting processing of common database queries on confidential data while maintaining access privacy.

1 Introduction

Recent advances in computing technology have resulted in the proliferation of transformative architectural, infrastructural, and application trends which can potentially revolutionize the future of information technology. *Cloud Computing* is one such paradigm that is likely to radically change the deployment of computing and storage infrastructures of both large and small enterprises. Major enabling features of the cloud computing infrastructure include *pay per use* and hence *no up-front cost for deployment*, *perception of infinite scalability*, and *elasticity of resources*. As a result, cloud computing has been widely perceived to be the "dream come true" with the potential to transform and revolutionize the IT industry [1]. The Software as a Service (SaaS) paradigm, such as web-based emails and online financial management, has been popular for almost a decade. But the launch of Amazon Web Services (AWS) in the second half of 2006, followed by a plethora of similar offerings such as Google AppEngine, Microsoft Azure, etc., have popularized the model of "utility computing" for other levels of the computing substrates such as Infrastructure as a Service (IaaS) and Platform as a Service (PaaS) models. The widespread popularity of these models is evident from the tens of cloud based solution providers [2] and hundreds of corporations hosting their critical business infrastructure in the cloud [3]. Recent reports show that many startups leverage the cloud to quickly launch their businesses applications [4], and over quarter of small and medium-sized businesses (SMBs) today rely on or plan to adopt cloud computing services [5].

S. Kikuchi et al. (Eds.): DNIS 2011, LNCS 7108, pp. 1–15, 2011.

With all the benefits of storing and processing data in the cloud, the security of data in the public cloud is still a big concern [6] that blocks the wide adoption of the cloud for data rich applications and data management services. In most cases and especially with Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS), users cannot control and audit their own data stored in the cloud by themselves. As the cloud hosts vast amount of valuable data and large numbers of services, it is a popular target for attacks. At the network level, there are threats of IP reuse, DNS attacks, Denial-of-Service (DoS) and Distributed Denial-of-Service (DDoS) attacks, etc [7]. At the host level, vulnerabilities in the virtualization stack may be exploited for attack. Resource sharing through virtualization also gives rise to side channel attacks. For example, a recent vulnerability found in Amazon EC2 [8] makes it possible to cross virtual machine boundary and gain access to another tenant's data co-located on the same physical machine [9]. At application level, vulnerabilities in access control could let unauthorized users access sensitive data [7]. Even if the data is encrypted, partial information about the data may be inferred by monitoring clients' query access patterns and analyzing clients' accessed positions on the encrypted data. The above threats could compromise *data confidentiality*, *data integrity*, and *data availability*.

To protect the confidentiality of sensitive data stored in the cloud, encryption is the widely accepted technique [10]. To protect the confidentiality of the data being accessed by queries, *Private Information Retrieval* (PIR) [11] can completely hide the query intents. To protect data integrity, Message Authentication Codes (MAC) [12], unforgeable signatures [13] or Merkle hash trees can validate the data returned by the cloud. To protect data availability and data integrity in case of partial data corruption, both replication and error-correcting mechanisms [14, 15, 16] are the potential solutions. Replication, however, potentially offers attackers multiple entry points for unauthorized access to the entire data. In contrast, error-correcting mechanisms that split data into pieces and distribute them in different places [17, 18, 19, 15, 16] enhance data security in addition to data availability. These techniques have been implemented in a recently released commercial product of cloud storage [20] as well as in Google Apps Service for the City of Los Angeles [21].

Integrating the above techniques, however, cannot deliver a *practical* secure relational data management service in the cloud. For data confidentiality specifically, practical query processing on encrypted data remains a big challenge. Although a number of proposals have explored query processing on encrypted data, many of them are designed for processing one specific query (e.g. range query) and are not flexible to support another kind of query (e.g. data updates), yet some other approaches lose balance between query functionality and data confidentiality. In Section 2, we discuss the relevant techniques and present a framework based on secure index that targets to support multiple common database queries and strikes a good balance between functionality and confidentiality. As for data confidentiality at query access, PIR provides complete query privacy but is too expensive in terms of computation and communication.

As a result, alternative techniques for protecting query privacy are explored in Section 3. The ultimate goal of the proposed research is to push forward the frontier on designing practical and secure relational data management services in the cloud.

2 Processing Database Queries on Encrypted Data

Data confidentiality is one of the biggest challenges in designing a practical secure data management service in the cloud. Although encryption can provide confidentiality for sensitive data, it complicates query processing on the data. A big challenge to enable efficient query processing on encrypted data is to be able to *selectively retrieve data* instead of downloading the entire data, decoding and processing them on the client side. Adding to this challenge are the individual filtering needs of different queries and operations, and thus a lack of a consistent mechanism to support them. This section first reviews related work on query processing on encrypted data, and then presents a secure index based framework that can support efficient processing of multiple database queries.

2.1 Related Work

To support queries on encrypted relational data, one class of solutions proposed processing encrypted data directly, yet most of them cannot achieve strong data confidentiality and query efficiency simultaneously for supporting common relational database queries (i.e., range queries and aggregation queries) and database updates (i.e., data insertion and deletion). The study of encrypted data processing originally focused on keyword search on encrypted documents [22, 23]. Although recent work can efficiently process queries with equality conditions on relational data without compromising data confidentiality [24], they cannot offer the same levels of efficiency and confidentiality for processing other common database queries such as range queries and aggregation queries. Some proposals trade off partial data confidentiality to gain query efficiency. For example, the methods that attach range labels to bucketized encrypted data [25, 26] reveal the underlying data distributions. Methods relying on order preserving encryption [27, 28] reveal the data order. These methods cannot overcome attacks based on statistical analysis on encrypted data. Other proposals sacrifice query efficiency for strong data confidentiality. One example is homomorphic encryption, which enables secure calculation on encrypted data [29, 30], but requires expensive computation and thus is not yet practical [31]. Predicate encryption can solve polynomial equations on encrypted data [32], but it uses public key cryptographic system which is much more expensive than symmetric encryption used above.

Instead of processing encrypted data directly, an alternative is to use an encrypted index which allows the client to traverse the index and to locate the data of interest in a small number of rounds of retrieval and decryption [33, 34, 35, 36]. In that way, both confidentiality and functionality can be preserved. The other alternative approach that preserves both confidentiality and functionality is to use

a secure co-processor on the cloud server side and to put a database engine and all sensitive data processing inside the secure co-processor [37]. That apparently requires all the clients to trust the secure co-processor with their sensitive data, and it is not clear that how the co-processor handles large numbers of clients and large amount of data. In contrast, a secure index based approach [33, 34, 35, 36] does not have to rely on any parties other than the clients, and thus we believe that it is promising to be a practical and secure framework. In the following, we discuss our recent work [36] on using secure index for processing various database queries.

2.2 Secure Index Based Framework

Let I be a B+-tree [38] index built on a relational data table T. Each tuple t has d attributes, $A_1, A_2, ..., A_d$. Assume each attribute value (and each index key) can be mapped to an integer value taken from a certain range $[1, ..., MAX]$. Each leaf node of I maintains the pointers to the tuple units where the tuples with the keys in this leaf node are stored. The data tuples of T and indexes I are encoded under different secrets C, which are then used for decoding the data tuples and indexes respectively. Each tree node of the index and a fixed number of tuples are single units of encoding. We require that these units have fixed sizes to ensure that the encoded pieces have fixed sizes. The encoded pieces are then distributed on servers hosted by external cloud storage providers such as Amazon EC2 [8]. Queries and operations on the index key attribute can be efficiently processed by locating the leaf nodes of I that store the requested keys and then processing the corresponding tuple units pointed by these leaf nodes.

Fig. 1 demonstrates the high-level idea of our proposed framework. The data table T is organized into a tuple matrix TD. The index I is organized into an index matrix ID. Each column of TD or ID is an encoding unit. ID is encoded into IE and TD is encoded into TE. Then IE and TE are distributed in the cloud.

Encoding Choices. Symmetric key encryption such as AES can be used for encoding [33, 34], as symmetric key encryption is much more efficient than asymmetric key encryption. Here we consider using Information Dispersal Algorithm (IDA) [17] for encoding, as IDA naturally provides data availability and some degrees of confidentiality.

Using IDA, we encode and split data into multiple uninterpretable pieces. IDA encodes an $m \times w$ data matrix D by multiplying an $n \times m$ ($m < n$) secret dispersal matrix C to D in Galois filed, i.e. $E = C \cdot D$. The resulting $n \times w$ encoded matrix E is distributed onto n servers by dispersing each row onto one server. To reconstruct D, only m correct rows are required. Let these m rows form an $m \times w$ sub-matrix E^* and the corresponding m rows of C form an $m \times m$ sub-matrix C^*, $D = {C^*}^{-1} \cdot E^*$. In such a way, data is intermingled and dispersed, so that it is difficult for an attacker to gather the data and apply inference analysis. To validate the authenticity and correctness of a dispersed piece we apply the *Message Authentication Code* (MAC) [12] on each dispersed piece.

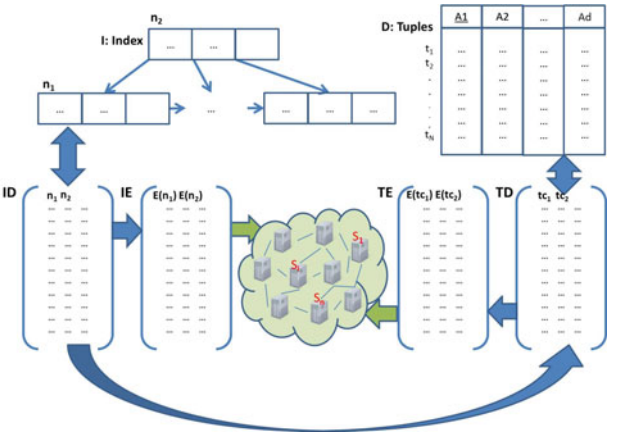


Fig. 1. Secure Cloud Data Access Framework

Since IDA is not proved to be theoretically secure [17], to prevent attackers' direct inference or statistical analysis on encoded data, we propose to add *salt* in the encoding process [39] so as to randomize the encoded data. In addition to the secret keys C for encoding and decoding, a client maintains a secret seed ss and a deterministic function fs for producing random factors based on ss and input data. Function fs can be based on pseudorandom number generator or secret hashing. The generated random values are added to the data values before encoding, and they can only be reconstructed and subtracted from the decoded values by the client.

Encoding Units of Index. Let the branching factor of the B+-tree index I be b. Then every internal node of I has $[\lceil b/2 \rceil, b]$ children, and every node of I has $[\lceil (b-1)/2 \rceil, b-1]$ keys. To accommodate the maximum number of children pointers and keys, we fix the size of a tree node to $2b+1$, and let the column size of the index matrix ID, m be $2b+1$ for simplicity. We assign each tree node an integer column address denoting its column in ID according to the order it is inserted into the tree. Similarly, we assign a data tuple column of TD an integer column address according to the order its tuples are added into TD.

A tree node of I, *node*, or the corresponding column in ID, $ID_{:,g}$, can be represented as

$$(isLeaf, col_0, col_1, key_1, col_2, key_2, ..., col_{b-1}, key_{b-1}, col_b) \tag{1}$$

where $isLeaf$ indicates if *node* is an internal node ($isLeaf = 0$), or a leaf node ($isLeaf = 1$). key_i is an index key, or 0 if *node* has less than i keys. For an internal node, $col_0 = 0$, $col_i\,(1 \leq i \leq b)$ is the column address of the ith child node of *node* if key_{i-1} exists, otherwise $col_i = 0$. For existing keys and children, (a key in child column col_i) $< key_i \leq$ (a key in child column col_{i+1}) $< key_{i+1}$. For a leaf node, col_0 and col_b are the column addresses of the predecessor/successor

leaf nodes respectively, and $col_i (1 \leq i \leq b-1)$ is the column address of the tuple with key_i.

We use an *Employee* table shown in Fig. 2 as an example. Fig. 3(a) gives an example of an index built on *Perm No* of the *Employee* table (the upper part) and the corresponding index matrix ID (the lower part). In the figure, the branching factor of the B+-tree $b = 4$, and the column size of the index matrix $m = 9$. The keys are inserted into the tree in ascending order 10001, 10002, ... 10007. The numbers shown on top of the tree nodes are the column addresses of these nodes. The numbers pointed to by arrows below the keys of the leaf nodes are the column addresses of the data tuples with those keys.

	Perm No	Salary	Age
t_1	10001	4000	25
t_2	10002	5000	28
t_3	10003	4000	25
t_4	10004	4000	26
t_5	10005	6000	30
t_6	10006	5500	28
t_7	10007	6000	31

Fig. 2. An Employee Table

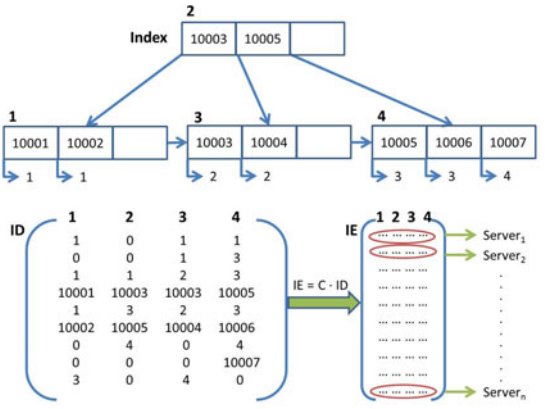
Encoding Units of Data Tuples. Let the column size of the tuple matrix TD also be m. To organize the existing d-dimensional tuples of D into TD initially, we sort all the data tuples in ascending order of their keys, and then pack every p tuples in a column of TD such that $p \cdot d \leq m$ and $(p+1) \cdot d > m$. The columns of TD are assigned addresses of increasing integer values. The p tuples in the same column have the same column address, which are stored in the leaf nodes of the index that have their keys. Fig. 3(b) gives an example of organizing tuples in *Employee* table into a tuple matrix TD, in which two tuples are packed in each column.

Selective Data Access. To enable selective access to small amount of data, the cloud data service provides two primitive operations to clients, i.e. storing and retrieving fixed sizes of encoding units. Since each encoding unit or each column of ID or TD has an integer address, we denote these two operations as store_unit(D, i) and retrieve_unit(E, i), in which i is the address of the unit. store_unit(D, i) encodes data unit i, adds salt into it on the client side and then stores it in the cloud. retrieve_unit(E, i) retrieves the encoded data unit i from the cloud, and then decodes the data unit and subtracts salt on the client side.

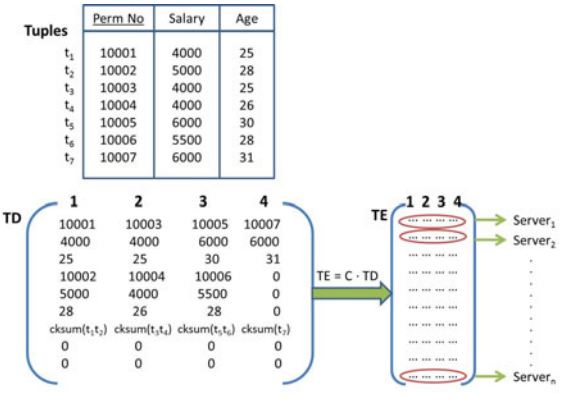
2.3 Query Processing

We assume that the root node of the secure index is always cached on the client side. The above secure index based framework is able to support exact, range and aggregation queries involving index key attributes, as well as data updates, inserts and deletes efficiently. These common queries form the basis for general purpose relational data processing.

Exact Queries. Performing an exact query via the secure B+-tree index is similar to performing the same query on a plaintext B+-tree index. The query is processed by traversing the index downwards from the root, and locating the keys of interests in leaf nodes. However, each node retrieval calls retrieve_unit(IE, i)



(a) Index Matrix of Employee Table



(b) Tuple Matrix of Employee Table

Fig. 3. Encoding of Index and Data Tuples of Employee Table

and the result tuple retrieval is through retrieve_unit(TE, i). Fig. 4 illustrates the recursive procedure for processing an exact query at a tree node. When an exact query for key x is issued, the exact query procedure on the root node, $ID_{:,\mathbf{root}}$, is called first. At each *node*, the client locates the position i with the smallest key that is equal to or larger than x (Line 1), or the rightmost non-empty position i if x is larger than all keys in *node* (Line 2-4).

Range Queries. To find the tuples whose index keys fall in a range $[x_l, x_r]$, we locate all qualified keys in the leaf nodes, get the addresses of the tuple matrix columns associated with these keys, and then retrieve the answer tuples from these tuple matrix columns. The qualified keys can be located by performing an exact query on either x_l or x_r, and then following the successor links or predecessor links at the leaf nodes. Note that since tuples can be dynamically inserted and deleted, the tuple matrix columns may not be ordered by index

Define: x, search key.
Define: t, the tuple with index key x.
Define: $nkeys(node)$, number of keys in $node$.
Find the smallest i s.t. $x \leq node.key_i \ \wedge \ i \leq nkeys(node)$
if not exist i **then**
  $i \leftarrow nkeys(node) + 1$
if $node.isLeaf = 0$ **then**
  $ID_{:,node.col_{i+1}} \leftarrow$ retrieve_unit$(IE, node.col_{i+1})$
  exact_query$(ID_{:,node.col_{i+1}}, x, process_on_not_found)$
else
  if $i < nkeys(node) \ \wedge \ x = node.key_i$ **then**
    $TD_{:,node.col_i} \leftarrow$ retrieve_unit$(TE, node.col_i)$
    Locate t with key x in $TD_{:,node.col_i}$
    Return $(t, node.col_i, node)$.
  else
    t does not exist

Fig. 4. Algorithm exact_query$(node, x)$

keys, thus we cannot directly retrieve the tuple matrix columns in between the tuple matrix columns corresponding to x_l and x_r.

Aggregation Queries. An aggregation query involving selection on index key attributes can be processed by first performing a range query on the index key attributes and then performing aggregation on the result tuples of the range query on the client side. Some aggregation queries on index key attributes can be directly done on the index on the server side, such as finding the tuples with MAX, MIN keys in a range $[x_l, x_r]$.

Data Updates, Insertion and Deletion. Data update without change on index keys can be easily done by an exact query to locate the unit that has the previous values of the tuple, a local change and a call of store_unit(TD, i) to store the updated unit. Data update with change on index keys is similar to data insertion, which is discussed below.

Data insertion is done in two steps: tuple insertion and index key insertion. Data deletion follows a similar process, with the exception that the tuple to delete is first located via an exact query of the tuple's key. Note that the order that the tuple unit is updated before the index unit is important, since the address of the tuple unit is the link between the two and needs to be recorded in the index node.

We allow flexible insertion and deletion of data tuples. An inserted tuple is appended to the last column or added to a new last column in TD regardless of the order of its key. A deleted tuple is removed from the corresponding column by leaving the d entries it occupied previously empty. Index key insertion and deletion are always done on the leaf nodes, but node splits (correspondingly adding an index unit for the new node and updating an index unit for the split node) or merges (correspondingly deleting a tuple unit for the deleted node and updating an index unit for the node to merge with) may happen to maintain a proper B+-tree.

Boosting Performance at Accesses by Caching Index Nodes on Client. The above query processing relies heavily on index traversals, which means that the index nodes are frequently retrieved from servers and then decoded on the client, resulting in a lot of communication and computation overhead. Query performance can be improved by caching some of the most frequently accessed index nodes in clear on the client. Top level nodes in the index are more likely to be cached.

3 Protecting Access Privacy

In a secure data management framework in the cloud, even if the data is encrypted, adversaries may still be able to infer partial information about the data by monitoring clients' query access patterns and analyzing clients' accessed positions on the encrypted data. Protecting query access privacy to hide the real query intents is therefore needed for ensuring data confidentiality in addition to encryption. One of the biggest challenge in protecting access privacy is to strike a good balance between privacy and practical functionality. Private Information Retrieval (PIR) [11] seems a right fit for protecting access privacy, but the popular PIR protocols relying on expensive cryptographic operations are not yet practical. On the other hand, some lightweight techniques such as routing query accesses through trusted proxies [36] or mixing real queries with noisy queries [40] have been proposed, but they cannot quantify and guarantee the privacy levels that they provide. In this section, we first review relevant work on protecting access privacy, and then discuss hybrid solutions that combine expensive cryptographic protocols with lightweight techniques.

3.1 Related Work

The previous work on protecting access privacy can be categorized as Private Information Retrieval and query anonymization or obfuscation using noisy data or noisy queries.

Private Information Retrieval (PIR) models the private retrieval of public data as a theoretical problem: Given a server which stores a binary string $x = x_1...x_n$ of length n, a client wants to retrieve x_i privately such that the server does not learn i. Chor et al. [11] introduced the PIR problem and proposed solutions for multiple servers. Kushilevitz and Ostrovsky followed by proposing a single server, computational PIR solution [41] which is usually referred to as *c*PIR. Although it has been shown that multi-server PIR solutions are more efficient than single-server PIR solutions [42], multi-server PIR does not allow communication among all the servers, thus making it unsuitable to use in the cloud. On the other hand, *c*PIR and its follow-up single-server PIR proposals [43], however, are criticized as impractical because of their expensive computation costs [44]. Two alternatives were later proposed to make single-server PIR practical. One uses oblivious RAM, and it only applies to a specific setting where a client retrieves its own data outsourced on the server [45, 46], which can be applied in the

cloud. The other bases the foundation of its PIR protocol based on linear algebra [47] instead of the number theory which previous single-server PIR solutions base on. Unfortunately, the latter lattice based PIR scheme cannot guarantee that its security is as strong as previous PIR solutions, and it incurs a lot more communication costs.

Query anonymization is often used in privacy-preserving location based services [48], which is implemented by replacing a user's query point with an enclosing region containing $k-1$ noisy points of other users. A similar anonymization technique which generates additional noisy queries is employed in a private web search tool called TrackMeNot [40]. The privacy in TrackMeNot, however, is broken by query classification [49], which suggests that randomly extracted noise alone does not protect a query from identification.

To generate meaningful and disguising noise words in private text search, a technique called *Plausibly Deniable Search* (PDS) is proposed in [50, 51]. PDS employs a topic model or an existing taxonomy to build a static clustering of cover word sets. The words in each cluster belong to different topics but have similar specificity to their respective topics, thus are used to cover each other in a query.

3.2 Hybrid Query Obfuscation

It is hard to quantify privacy provided in a query anonymization approach. Since the actual query data and noisy data are all in plaintext, the risk of identifying the actual query data could still be high. k-Anonymity in particular has been criticized as a weak privacy definition [52], because it does not consider the data semantic. A group of k plaintext data items may be semantically close, or could be semantically diverse. In contrast, traditional PIR solutions can provide complete privacy and confidentiality. We hence consider hybrid solutions that combine query anonymization and PIR/cryptographic solutions.

A hybrid query obfuscation solution can provide access privacy, data confidentiality and practical performance. PIR/cryptographic protocols ensure access privacy and data confidentiality, while query anonymization upon these protocols reduce computation and communication overheads, thus achieving practical performance. Such hybrid query obfuscation solutions have been used in preserving location privacy in location-based services [53, 54] and in our earlier work on protecting access privacy in simple selection queries [55].

Bounding-Box PIR. Our work is built upon single-server *c*PIR protocol [41]. It is a generalized private retrieval approach called *Bounding-Box* PIR (*bb*PIR). We describe how *bb*PIR works using a database / data table as illustration. For protecting access privacy in the framework given in the last section, we can consider an index nodes, an index / tuple column as a data item and treat the collection of them as a virtual database for access.

*c*PIR works by privately retrieving an item from a data matrix for a given matrix address [41]. So we consider a (key, address, value) data store, where each value is a b-bit data item. The database of size n is organized in an $s \times t$ matrix

M ($s = t = \lceil \sqrt{n} \rceil$ by default). Each data item x has a numeric key KA that determines the two dimensional address of x in M. For example, the column address of an index / tuple column can be the key for identifying the index / tuple column.

A client can specify her privacy requirement and desired charge budget (ρ, μ), where ρ is a privacy breach limit (the upper bound probability that a requested item can be identified by the server), and μ is a server charge limit (the upper bound of the number of items that are exposed to the client for one requested tuple). The basic idea of *bb*PIR is to use a bounding box BB (an $r \times c$ rectangle corresponding to a sub-matrix of M) as an anonymized range around the address of item x requested by the client, and then apply *c*PIR on the bounding box. *bb*PIR finds an appropriately sized bounding box that satisfies the privacy request ρ, and achieves overall good performance in terms of communication and computation costs without exceeding the server charge limit μ for each retrieved item. The area of the bounding box determines the level of privacy that can be achieved, the larger the area, the higher the privacy, but with higher computation and communication costs.

The above scheme retrieves data by the exact address of the data. To enable natural retrieval by the key of data, we simply let the server publish a one-dimensional histogram, H, on the key field KA and the dimensions of the database matrix M, s and t. The histogram is only published to authorized clients. The publishing process, which occurs infrequently, is encrypted for security. When a client issues a query, she calculates an address range for the queried entry by searching the bin of H where the query data falls. In this way, she translates a retrieval by key to a limited number of retrievals by addresses, while the latter multiple retrievals can be actually implemented in one retrieval if they all request the same column addresses of the matrix.

Further Consideration on Selecting Anonymization Ranges. In current *bb*PIR, we only require that an anonymization range bounding box encloses the requested data, and although the dimensions of the bounding box are fixed, the position of the bounding box can be random around the requested data. In real applications, the position of the bounding box could also be important to protecting access privacy. Some positions may be more frequently accessed by other clients and less sensitive, while some positions may be rarely accessed by other clients and easier to be identified as unique access patterns. These information, if incorporated into the privacy quantification, should result in a bounding box that provides better privacy protection under the constraints of the requested data and the dimensions. One idea is to incorporate access frequency in privacy probability, but we should be cautious that a bounding box cannot include all frequent accessed data but the requested data, since in this case the requested data may be also easily filtered out.

4 Concluding Remarks

The security of the data stored in the public cloud is one of the biggest concerns that blocks the realization of data management services in the cloud, especially for sensitive enterprise data. Although numerous techniques have been proposed for providing data confidentiality, integrity and availability in the context and for processing queries on encrypted data, it is very challenging to integrate them into a practical secure data management service that works for most database queries. This paper has reviewed these relevant techniques, presented a framework based on secure index for practical secure data management and query processing, and also discussed how to enhance data confidentiality by providing practical access privacy for data in the cloud. We contend that the balance between security and practical functionality is crucial for the future realization of practical secure data management services in the cloud.

Acknowledgement. This work is partly funded by NSF grant CNS 1053594 and an Amazon Web Services research award. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

References

[1] Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: Above the Clouds: A Berkeley View of Cloud Computing. Technical Report 2009-28, UC Berkeley (2009)

[2] Amazon: AWS Solution Providers (2009), http://aws.amazon.com/solutions/solution-providers/

[3] Amazon: AWS Case Studies (2009), http://aws.amazon.com/solutions/case-studies/

[4] Li, P.: Cloud computing is powering innovation in the silicon valley (2010), http://www.huffingtonpost.com/ping-li/cloud-computing-is-poweri_b_570422.html

[5] Business Review USA: Small, medium-sized companies adopt cloud computing (2010), http://www.businessreviewusa.com/news/cloud-computing/small-medium-sized-companies-adopt-cloud-computing

[6] InfoWorld: Gartner: Seven cloud-computing security risks (2008), http://www.infoworld.com/d/security-central/gartner-seven-cloud-computing-security-risks-853?page=0,1

[7] Mather, T., Kumaraswamy, S., Latif, S.: Cloud Security and Privacy. O'Reilly Media, Inc., Sebastopol (2009)

[8] Amazon: Amazon elastic compute cloud (amazon ec2), http://aws.amazon.com/ec2/

[9] Ristenpart, T., Tromer, E., Shacham, H., Savage, S.: Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In: ACM Conference on Computer and Communications Security, pp. 199–212 (2009)

[10] NIST: Fips publications, http://csrc.nist.gov/publications/PubsFIPS.html

[11] Chor, B., Kushilevitz, E., Goldreich, O., Sudan, M.: Private information retrieval. J. ACM 45(6), 965–981 (1998)
[12] Bellare, M., Canetti, R., Krawczyk, H.: Keying Hash Functions for Message Authentication. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 1–15. Springer, Heidelberg (1996)
[13] Agrawal, R., Haas, P.J., Kiernan, J.: A system for watermarking relational databases. In: Proc. of the 2003 ACM SIGMOD International Conference on Management of Data, pp. 674–674 (2003)
[14] Plank, J.S., Ding, Y.: Note: Correction to the 1997 tutorial on reed-solomon coding. Softw. Pract. Exper. 35(2), 189–194 (2005)
[15] Bowers, K.D., Juels, A., Oprea, A.: Hail: a high-availability and integrity layer for cloud storage. In: CCS 2009: Proceedings of the 16th ACM Conference on Computer and Communications Security, pp. 187–198 (2009)
[16] Abu-Libdeh, H., Princehouse, L., Weatherspoon, H.: Racs: a case for cloud storage diversity. In: SoCC 2010: Proceedings of the 1st ACM Symposium on Cloud Computing, pp. 229–240 (2010)
[17] Rabin, M.O.: Efficient dispersal of information for security, load balancing, and fault tolerance. J. ACM 36(2), 335–348 (1989)
[18] Shamir, A.: How to share a secret. Commun. ACM 22(11), 612–613 (1979)
[19] Agrawal, D., Abbadi, A.E.: Quorum consensus algorithms for secure and reliable data. In: Proceedings of the Sixth IEEE Symposium on Reliable Distributed Systems, pp. 44–53 (1988)
[20] CleverSafe: Cleversafe responds to cloud security challenges with cleversafe 2.0 software release (2010), http://www.cleversafe.com/news-reviews/press-releases/press-release-14
[21] InfoLawGroup: Cloud providers competing on data security & privacy contract terms (2010), http://www.infolawgroup.com/2010/04/articles/cloud-computing-1/cloud-providers-competing-on-data-security-privacy-contract-terms
[22] Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: SP 2000: Proceedings of the 2000 IEEE Symposium on Security and Privacy, pp. 44–55 (2000)
[23] Chang, Y.-C., Mitzenmacher, M.: Privacy Preserving Keyword Searches on Remote Encrypted Data. In: Ioannidis, J., Keromytis, A.D., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 442–455. Springer, Heidelberg (2005)
[24] Yang, Z., Zhong, S., Wright, R.N.: Privacy-Preserving Queries on Encrypted Data. In: Gollmann, D., Meier, J., Sabelfeld, A. (eds.) ESORICS 2006. LNCS, vol. 4189, pp. 479–495. Springer, Heidelberg (2006)
[25] Hacigumus, H., Iyer, B.R., Li, C., Mehrotra, S.: Executing SQL over encrypted data in the database service provider model. In: SIGMOD Conference (2002)
[26] Hore, B., Mehrotra, S., Tsudik, G.: A privacy-preserving index for range queries. In: Proc. of the 30th Int'l Conference on Very Large Databases VLDB, pp. 720–731 (2004)
[27] Agrawal, R., Kiernan, J., Srikant, R., Xu, Y.: Order preserving encryption for numeric data. In: SIGMOD 2004: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, pp. 563–574 (2004)
[28] Emekci, F., Agrawal, D., Abbadi, A.E., Gulbeden, A.: Privacy preserving query processing using third parties. In: ICDE (2006)
[29] Ge, T., Zdonik, S.B.: Answering aggregation queries in a secure system model. In: Proceedings of the 33rd International Conference on Very Large Data Bases, pp. 519–530 (2007)

[30] Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC 2009: Proceedings of the 41st Annual ACM Symposium on Theory of Computing, pp. 169–178 (2009)
[31] Schneier, B.: Homomorphic encryption breakthrough (2009), http://www.schneier.com/blog/archives/2009/07/homomorphic_enc.html
[32] Katz, J., Sahai, A., Waters, B.: Predicate Encryption Supporting Disjunctions, Polynomial Equations, and Inner Products. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 146–162. Springer, Heidelberg (2008)
[33] Damiani, E., di Vimercati, S.D.C., Jajodia, S., Paraboschi, S., Samarati, P.: Balancing confidentiality and efficiency in untrusted relational dbmss. In: ACM Conference on Computer and Communications Security, pp. 93–102 (2003)
[34] Shmueli, E., Waisenberg, R., Elovici, Y., Gudes, E.: Designing secure indexes for encrypted databases. In: Proceedings of the IFIP Conference on Database and Applications Security (2005)
[35] Ge, T., Zdonik, S.B.: Fast, secure encryption for indexing in a column-oriented dbms. In: ICDE, pp. 676–685 (2007)
[36] Wang, S., Agrawal, D., Abbadi, A.E.: A Comprehensive Framework for Secure Query Processing on Relational Data in the Cloud. In: Jonker, W., Petković, M. (eds.) SDM 2011. LNCS, vol. 6933, pp. 52–69. Springer, Heidelberg (2011)
[37] Bajaj, S., Sion, R.: Trusteddb: a trusted hardware based database with privacy and data confidentiality. In: Proceedings of the 2011 International Conference on Management of Data, SIGMOD 2011, pp. 205–216 (2011)
[38] Comer, D.: Ubiquitous b-tree. ACM Comput. Surv. 11(2), 121–137 (1979)
[39] Robling Denning, D.E.: Cryptography and data security. Addison-Wesley Longman Publishing Co., Inc., Boston (1982)
[40] Howe, D.C., Nissenbaum, H.: TrackMeNot: Resisting surveillance in web search. In: Lessons from the Identity Trail: Anonymity, Privacy, and Identity in a Networked Society, pp. 417–436. Oxford University Press (2009)
[41] Kushilevitz, E., Ostrovsky, R.: Replication is not needed: Single database, computationally-private information retrieval. In: FOCS, pp. 364–373 (1997)
[42] Olumofin, F.G., Goldberg, I.: Revisiting the computational practicality of private information retrieval. In: Financial Cryptography (2011)
[43] Gentry, C., Ramzan, Z.: Single-database private information retrieval with constant communication rate. In: Proceedings of the 32nd International Colloquium on Automata, Languages and Programming, pp. 803–815 (2005)
[44] Sion, R., Carbunar, B.: On the computational practicality of private information retrieval. In: Network and Distributed System Security Symposium (2007)
[45] Williams, P., Sion, R.: Usable private information retrieval. In: Network and Distributed System Security Symposium (2008)
[46] Williams, P., Sion, R., Carbunar, B.: Building castles out of mud: practical access pattern privacy and correctness on untrusted storage. In: ACM Conference on Computer and Communications Security, pp. 139–148 (2008)
[47] Melchor, C.A., Gaborit, P.: A fast private information retrieval protocol. In: IEEE Internal Symposium on Information Theory, pp. 1848–1852 (2008)
[48] Mokbel, M.F., Chow, C.Y., Aref, W.G.: The new casper: A privacy-aware location-based database server. In: ICDE, pp. 1499–1500 (2007)
[49] Peddinti, S.T., Saxena, N.: On the Privacy of Web Search Based on Query Obfuscation: A Case Study of Trackmenot. In: Atallah, M.J., Hopper, N.J. (eds.) PETS 2010. LNCS, vol. 6205, pp. 19–37. Springer, Heidelberg (2010)
[50] Murugesan, M., Clifton, C.: Providing privacy through plausibly deniable search. In: SDM, pp. 768–779 (2009)

[51] Pang, H., Ding, X., Xiao, X.: Embellishing text search queries to protect user privacy. PVLDB 3(1), 598–607 (2010)
[52] Dwork, C., McSherry, F., Nissim, K., Smith, A.: Calibrating Noise to Sensitivity in Private Data Analysis. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 265–284. Springer, Heidelberg (2006)
[53] Olumofin, F.G., Tysowski, P.K., Goldberg, I., Hengartner, U.: Achieving Efficient Query Privacy for Location Based Services. In: Atallah, M.J., Hopper, N.J. (eds.) PETS 2010. LNCS, vol. 6205, pp. 93–110. Springer, Heidelberg (2010)
[54] Ghinita, G., Kalnis, P., Kantarcioglu, M., Bertino, E.: A Hybrid Technique for Private Location-Based Queries with Database Protection. In: Mamoulis, N., Seidl, T., Pedersen, T.B., Torp, K., Assent, I. (eds.) SSTD 2009. LNCS, vol. 5644, pp. 98–116. Springer, Heidelberg (2009)
[55] Wang, S., Agrawal, D., El Abbadi, A.: Generalizing PIR for Practical Private Retrieval of Public Data. In: Foresti, S., Jajodia, S. (eds.) Data and Applications Security and Privacy XXIV. LNCS, vol. 6166, pp. 1–16. Springer, Heidelberg (2010)