Is Homomorphic Encryption the Holy Grail for Database Queries on Encrypted Data?

Shiyuan Wang, Divyakant Agrawal, and Amr El Abbadi

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

Abstract. Homomorphic encryption has been used for supporting simple aggregations, numeric calculations on encrypted data as well as for private information retrieval. Recently, theoretical breakthroughs on homomorphic encryption resulted in *fully homomorphic encryption*, which is able to compute arbitrary functions on encrypted data. As a result, homomorphic encryption is generally believed to be the holy grail for solving database queries on encrypted data. However, there has not been a systematic study that analyzes the use of fully homomorphic encryption for solving database queries beyond simple aggregations and numeric calculations, such as selection, range and join queries. Our paper fills this gap by identifying what fully homomorphic encryption can do and what it cannot do well for supporting general database queries at a conceptual level. We show that using a fully homomorphic encryption scheme that supports addition, multiplication, AND and XOR on ciphertexts, it is possible to process a complex selection, range, join or aggregation query on encrypted data on the server side, and to return the encrypted matching answers in a result buffer. For queries without fixed answer sizes, it is however *not guaranteed* all matching answers will be correctly constructed from the result buffer, instead the answers can be constructed from the result buffer with overwhelming probability.

1 Introduction

The outsourcing of data storage and computation has been popularized with recent advances in computing technologies represented by cloud computing. Despite the desirable features of no up-front cost for deployment, pay-per-use, scalability and elasticity of resources in the cloud, concerns about the security of sensitive data stored in the cloud and over the privacy of accessing data via the cloud remain the roadblocks to the widespread adoption of the cloud for data management and query processing tasks. Encrypting the data in the cloud seems like an obvious solution.

Nevertheless, making use of the encrypted data in the cloud, i.e. processing database queries on encrypted data, while maintaining complete

data confidentiality and access privacy, is a difficult task. The techniques enabling keyword search on encrypted data [1, 2] preserve complete data confidentiality, but reveal the positions that match the search words. A large body of previous proposals reveal partial information about the data to allow the cloud to evaluate complex predicates on encrypted data, i.e. for supporting range queries [3, 4, 5, 6]. Some other encrypted index based proposals offload predicates evaluation to the client side so as to not lose data confidentiality in the cloud [7, 8, 9], but they still need cautious protection for the privacy of index accesses [10]. In addition, it is not clear how many of the above approaches can support complex database queries involving multiple tables and attributes.

Recently, *homomorphic encryption* has gained increasing attention because of the theoretical breakthrough in constructing *fully homomorphic encryption* that is able to compute arbitrary functions over encrypted data [11, 12] without the need for decryption. Despite the fact that the performance of homomorphic encryption needs to be improved for it to be practically useful, it is generally believed that fully homomorphic encryption can solve the problem of querying encrypted data and the dream of data security in the cloud will become reality [13, 14]. Homomorphic encryption has been used for calculating simple aggregations [15], statistical functions on encrypted data [16] as well as for preserving the privacy of data access and retrieval [17, 18]. It is however not clear how fully homomorphic encryption can solve general database queries such as selection, range and join queries beyond simple aggregations and numerical calculations.

The goal of this paper is to analyze what fully homomorphic encryption can do and what it cannot do well for supporting general database queries, i.e. selection, range, join and complex aggregation queries, on encrypted data. Our analysis is based on the theoretical properties of (fully) homomorphic encryption that allow addition, multiplication, XOR, and AND operations on ciphertexts which have the same effect as applying these operations to the underlying plaintext data [11, 12, 16]. We show that by representing a predicate condition statement as a circuit consisting of these operations, *it is possible to directly evaluate the predicate on encrypted data in the cloud, thus enabling selection, range, join and group-by queries on encrypted data.* For the query initiator client to successfully retrieve the query answers (especially a set of answers), however, the client is required to estimate a number of answers for the cloud to allocate a fixed size result buffer. When the result buffer is returned to the client, most of the query answers will be correctly recovered from

the result buffer with overwhelming probability, while it is *not guaranteed* that all the query answers can be recovered. The query processing achieves *complete data confidentiality* and *strong access privacy*. The only information revealed are a rough result size, the types of queries and the involved attributes. The actual query contents, such as the predicate conditions and the matching answers are not revealed at all.

Our contribution is to show the conceptual level usability of homomorphic encryption in outsourcing database services in a secure and private manner. We do not target to solve the practicality issue of homomorphic encryption, which may need a few years to solve. However, knowing that homomorphic encryption can be used for secure processing of general queries in outsourcing database services allows us to put efforts on improving its performance in the future research.

In the rest of the paper, we review relevant work of query processing on encrypted data in Section 2, and describe homomorphic encryption in details in Section 3. We then construct a basic framework for query predicate evaluation and result retrieval in Section 4 for supporting general database queries on encrypted data in Section 5. Towards the end in Section 6, we also discuss the limitations of homomorphic encryption for practical database applications.

2 Related Work

There are multiple issues to consider when using a technique for processing database queries on encrypted data stored on cloud servers, such as functionality, performance, degrees of data confidentiality and data access privacy provided. We focus on functionality, data confidentiality and access privacy in the paper, because the performance of a technique may be improved with the relevant theoretical and technology advances, e.g. achieving smaller ciphertext sizes for homomorphic encryption [19].

The study of encrypted data processing originally focused on keyword search on encrypted texts [1, 2]. The proposed techniques rely on symmetric encryption and stream ciphers to provide complete data confidentiality, but reveal positions of matching answers when scanning the encrypted words and the indices built on the encrypted words. Compared to keyword search and equality condition queries, range queries on encrypted data are more difficult, and many techniques even have to compromise partial data confidentiality in order to evaluate range predicates on the server side. For example, the methods that attach range labels to bucketized encrypted data [3, 4] reveal the underlying data distributions. Methods relying on

order preserving encryption [5, 20] reveal the data order. These techniques are vulnerable to attacks based on statistical analysis on encrypted data. A recent work, CryptDB [6], processes different types of database queries using layers of different encryption schemes, and removes layers of encryption by decrypting to an appropriate layer for solving a specific query, i.e. deterministic encryption for equality condition queries, order-preserving encryption for range queries, and homomorphic encryption for aggregation queries. In such a way, CryptDB does not support the same levels of data confidentiality during query processing, and in the long run it downgrades to the lowest level of data confidentiality provided by the weakest encryption scheme. Instead of processing encrypted data directly, one alternative is to use an encrypted index on the cloud server and retrieve a small number of index nodes onto the client for decryption and predicate evaluation [7, 8, 9]. The index accesses, however, should be protected, as otherwise they might reveal the underlying data distribution [21]. Query processing, however, still must be done at the client side, thus eventually leveraging the cloud for storage but rendering it useless for computation. Moreover, it is still not clear how to use the index approach to support complex queries involving multiple predicates on different attributes or range aggregations, group-by queries while preserving strong privacy. Another alternative is to use a secure co-processor on the cloud server side and to put the database engine and all sensitive data processing inside the secure co-processor [22]. That apparently requires all the clients to trust the secure co-processor with their sensitive data.

Homomorphic encryption has been used for evaluating simple aggregations such as SUM, AVG [15], and for evaluating other numerical functions such as statistical functions on encrypted data [16]. Its ability to directly perform addition and multiplication on ciphertexts without the need for an index or other helper information allows it to achieve complete data confidentiality. Even for predicates evaluation of other queries which are shown later in the paper, as long as the query processing touches the entire data, strong access privacy is also achieved. These nice properties make homomorphic encryption of recent research interest with the hope of solving arbitraries queries on encrypted data.

3 Preliminaries

3.1 Homomorphic Encryption

In this section, we describe (fully) homomorphic encryption at an abstract level instead of based on a concrete realization. The concrete realization

of different (fully) homomorphic encryption schemes can be found in [11, 12, 16].

We consider a homomorphic encryption scheme as an asymmetric encryption scheme. It has the following four algorithms:

- $KeyGen(\lambda)$. Given a security parameter λ, generates a public encryption key pk which is available to the servers in the cloud settings, and a secret decryption key sk which is only known to the client or the data owner.
- $Encrypt(pk, m)$, abbreviated as $Enc(m)$. $c \leftarrow Enc(m)$, encrypts a plaintext message m into a ciphertext c. m is an integer in a finite field.
- $Decrypt(sk, c)$, abbreviated as $Dec(c)$. $m \leftarrow Dec(c)$, decrypts a ciphertext c into a plaintext m.
- $Evaluate(pk, f, c_1, ..., c_t)$, abbreviated as $f^*(c_1, ..., c_t)$. For ciphertext message $c_i \leftarrow Enc(m_i)$, $c \leftarrow f^*(c_1, ..., c_t)$ such that $Dec(c) = f(m_1, ..., m_t)$, evaluates function f on ciphertexts $c_1, ..., c_t$ without the need for decrypting each individual ciphertext c_i and then evaluating f on plaintexts m_i. f^* can be the same as f, e.g. in the ElGamal cryptosystem [23], f^* is also a multiplication function when f represents a multiplication function. f^* can also be different from f, e.g. in the Paillier cryptosystem [24], f^* is a multiplication function when f represents an addition function. For simplicity of the discussion in the paper, we assume f^* is the same as f.

Before Gentry's fully homomorphic encryption proposal [11], homomorphic encryption schemes could not support both arbitrary additions and multiplications on ciphertexts, e.g. the Paillier cryptosystem [24] cannot perform multiplication in ciphertexts, and Boneh's scheme [25] supports only one multiplication.

Definition 3.11 *A homomorphic encryption scheme is fully homomorphic if it can handle all functions, represented by addition and multiplication in finite fields.*

Addition and multiplication are basic functions based on which other functions can be constructed. For example, subtraction can be easily represented using addition and multiplication. In modulo 2 field, addition and subtraction are XOR, and multiplication is AND. For $a, b \in \{0, 1\}$, omitting mod 2 on the right side of equations without loss of generality, we have $a \oplus b = a + b = a - b$, $a \wedge b = ab$, $\overline{a} = 1 \oplus a$, $a \vee b = \overline{\overline{a} \wedge \overline{b}}$, which means that f can be any circuit constructed from XOR, AND, NOT, and OR

gates in the binary field. Since homomorphic encryption is probabilistic, operations on ciphertexts would result in noise. We need to pay attention to make sure that the noise introduced by functions like multiplications does not increase beyond an error bound for successful decryption [12], e.g. by controlling the number of consecutive multiplications.

There are two constraints with a homomorphic encryption scheme which will also affect our use of homomorphic encryption for database queries. First, as we consider f as a circuit, its output is usually fixed in advance. However for many database queries such as range and join queries, the result sets are not of fixed sizes and could be very large. We solve the difficulty of representing and retrieving such query results in Section 4. Second, f needs to scan all the input ciphertexts $c_1, ..., c_t$ for complete privacy, otherwise revealing that only some ciphertexts, i.e. the ciphertexts accessed, are relevant to the evaluation of f. Similarly for a database search with complete access privacy, the evaluation on ciphertext data needs to scan all encrypted database records. This is unavoidable, although for performance consideration, we could scan partial records for partial access privacy [18].

3.2 Data Model

Consider a data table DT with d attributes, $A_1, ..., A_d$. For simplicity of discussion, assume that the values of every attribute A_i are mapped to a finite integer field and they are encrypted by a fully homomorphic encryption scheme. An attribute value $A_i.val$, or val when A_i is clear in the context, has its binary representation as an $(n+1)$-bit string $A_i.bval$ or $bval$ with the first (leftmost) bit as the sign bit (0 when $A_i.val$ is positive and 1 when $A_i.val$ is negative). Correspondingly, each value of attribute A_i has two encrypted values stored on cloud servers, $Enc(val)$ and $Enc(bval)$, in which $Enc(bval)$ is bit-wise encryption of $bval$ and consists of $(n+1)$ ciphertexts $[Enc(bval)]_j$ corresponding to bit $[bval]_j$ $(1 \leq j \leq n+1)$. Let the number of tuples in DT be N. Then an attribute A_i of DT has $(Enc(val_1), Enc(bval_1)), ..., (Enc(val_N), Enc(bval_N))$ consecutively stored (not ordered) in the cloud. According to [16], we assume the availability of a technique to pack the bit-wise encryption $(n+1)$ ciphertexts $Enc(bval)$ into a single integer ciphertext $Enc(val)$, $Enc(val) \leftarrow pack(Enc(bval))$.

We are not concerned about data access control for different user roles in this paper, although we admit that this is an important issue for databases and should be paid special attention along with the advances of fully homomorphic encryption. We assume that all user clients of DT

have the same complete accesses to the data in DT, and they all have the same secret decryption key sk.

3.3 Adversary Model

As it is common for secure query processing on outsourced data [3, 6], we assume that the cloud is *honest but curious*. The cloud follows the query processing procedures and does not alter query inputs nor answers, but the cloud or any adversaries who can observe data and query execution in the cloud try all means to infer the encrypted data, the query contents, and users' access patterns. However, we assume that the adversaries (including the cloud) do not possess prior knowledge about the data nor about the user interests on the data.

4 Framework for Selection Queries under Homomorphic Encryption

Unconditioned aggregations and statistical functions on encrypted data of an attribute A_i can be realized by applying addition, subtraction and multiplication to all the encrypted data values $Enc(val_1), ..., Enc(val_N)$, and then a single or a few ciphertext results (to avoid overflow of a ciphertext) are returned to the client for decryption [15]. Selection and range queries, however, need to evaluate predicates on each encrypted data value $Enc(val_j)\,(1 \leq j \leq N)$, and ideally only retrieve the encrypted values that satisfy the predicates. This evaluation and processing should be performed on the cloud servers. Returning to the client for evaluation of predicates for each encrypted value is no better than returning all the encrypted values for decryption and processing on the client side, and both are not practical solutions.

This section considers using fully homomorphic encryption to solve simple equality conditions and range queries on a single attribute, i.e. for an attribute A_i and a given pivot value x, find all values $\{val_j \,|\, val_j = x\}$, $\{val_j \,|\, val_j > x\}$ or $\{val_j \,|\, val_j < x\}$. We propose solutions for blind predicate evaluation and retrieving matching answers below.

4.1 Blind Predicate Evaluation

Based on our discussion in Section 3.1, we assume that the following properties hold for a fully homomorphic encryption scheme for any x, y in a finite integer field and their binary representations bx, by. The functions

such as addition, subtraction and multiplication in properties P1-P6 are concrete instances of f and f^* in *Evaluate* algorithm in Section 3.1. The realization of these properties for a specific homomorphic encryption scheme, i.e. f and f^*, could be in different but similar formats.

- P1. *Integer Addition.* $Enc(x) + Enc(y) = Enc(x + y)$.
- P2. *Integer Multiplication.* $Enc(x) * Enc(y) = Enc(x * y)$.
- P3.A. *Integer Subtraction.* $Enc(x) - Enc(y) = Enc(x - y)$.
- P3.B. *Binary Subtraction.* $Enc(bx) - Enc(by) = Enc(bx - by)$.
- P4. *Bitwise AND.* $Enc(bx) \wedge Enc(by) = Enc(bx \wedge by)$.
- P5. *Bitwise OR.* $Enc(bx) \vee Enc(by) = Enc(bx \vee by)$.
- P6. *Bitwise NOT.* $Enc(1) \oplus [Enc(bx)]_i = Enc(\overline{[bx]_i})$.

The above properties allow us to directly perform calculation on ciphertexts, resulting in a ciphertext that after decryption is the same as the result of calculation on plaintext data, e.g. in P1, adding $Enc(x)$ and $Enc(y)$ together and then decrypting gives us $(x + y)$. In P3.B-P5, the operations are performed on bit-wise encrypted ciphertexts, e.g. P3.B actually means $[Enc(bx)]_i - [Enc(by)]_i = [Enc(bx - by)]_i$ $(1 \leq i \leq n+1)$. P6 uses the homomorphic property of XOR, $Enc(1) \oplus [Enc(bx)]_i = Enc(1 \oplus [bx]_i)$, and $\overline{[bx]_i} = 1 \oplus [bx]_i$.

Based on the properties P1-P6, we design a solution for blind predicate evaluation under homomorphic encryption. Assume the cloud creates a sufficiently large result buffer, and only puts the encrypted values that satisfy a predicate in the buffer. Given a set of values $val_j (1 \leq j \leq N)$, we are trying to get all values such that $(val_j = x)$. This can be achieved by initializing all the entries in the buffer as $Enc(0)$, and multiplying available $Enc(0)$ entries with $Enc(val_j)$ such that $val_j = x$, which is $Enc(val_j) * ((Enc(val_j) - Enc(x) == Enc(0))\,?\,Enc(1) : Enc(0))$ applying property P3.A. After the multiplication, an $Enc(0)$ entry becomes $Enc(val_j)$ for val_j such that $val_j = x$, but is still $Enc(0)$ for val_j such that $val_j \neq x$, so the values that do not satisfy the predicate do not materialize in the underlying plaintext answers of the result buffer.

The problem is now evaluating $((Enc(val_j) - Enc(x) == Enc(0))\,?\,Enc(1) : Enc(0))$ on cloud servers. With the encrypted binary representations $Enc(bval_j)$ and $Enc(bx)$ for $Enc(val_j)$ and $Enc(x)$ respectively, evaluating $(Enc(val_j) - Enc(x) == Enc(0))$ is the same as evaluating $(Enc(bval_j) - Enc(bx) == Enc(0))$. Applying property P3.B, we know that after performing $Enc(bval_j) - Enc(bx)$ in the latter evaluation, we just need to check if the second to the $(n + 1)$th bit-wise encryption ciphertexts of $(Enc(bval_j) - Enc(bx))$ are all $Enc(0)$, since the first bit

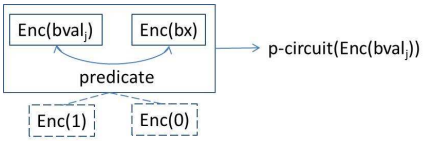

Fig. 1. Evaluation of Predicate as Circuit

is a sign bit and it does not need to be evaluated. Thus $((Enc(bval_j) - Enc(bx) == Enc(0))\,?\,Enc(1) : Enc(0))$ can be represented as a binary circuit $\wedge_{i=2}^{n+1} Enc(1) \oplus ([Enc(bval_j)]_i - [Enc(bx)]_i)$ The binary circuit for the condition statement $((Enc(val_j) - Enc(x) == Enc(0))\,?\,Enc(1) : Enc(0))$ is then

$$\wedge_{i=2}^{n+1}(Enc(1) \oplus ([Enc(bval_j)]_i - [Enc(bx)]_i)) \tag{1}$$

Similarly, for evaluating a typical range query condition $(val_j < x)$, we construct the following circuit for the condition statement $((Enc(val_j) - Enc(x) < Enc(0))\,?\,Enc(1) : Enc(0))$,

$$[Enc(bval_j)]_1 - [Enc(bx)]_1 \tag{2}$$

If $(val_j < x)$ holds, the value $(val_j - x)$ should be negative and the sign bit (first bit) of its binary representation should be 1, thus $[Enc(bval_j)]_1 - [Enc(bx)]_1$ should be $Enc(1)$. For evaluating another typical range query condition $(val_j > x)$, we construct the following circuit for the condition statement $((Enc(val_j) - Enc(x) > Enc(0))\,?\,Enc(1) : Enc(0))$,

$$Enc(1) \oplus ([Enc(bval_j)]_1 - [Enc(bx)]_1) \tag{3}$$

If $(val_j > x)$ holds, the value $(val_j - x)$ should be positive and the sign bit of its binary representation should be 0, thus $[Enc(bval_j)]_1 - [Enc(bx)]_1$ should be $Enc(0)$ and $\overline{[Enc(bval_j)]_1 - [Enc(bx)]_1}$ should be $Enc(1)$.

A similar circuit can be constructed on the binary representation of an encrypted value $Enc(val_j)$, $Enc(bval_j)$, for blindly evaluating a predicate p on $Enc(val_j)$. Let such a circuit for predicate p be *p-circuit* and the result of applying the circuit function to $Enc(bval_j)$ be $p\text{-}circuit(Enc(bval_j))$, as shown in Fig. 1. To be multiplied with $Enc(val_j)$, the ciphertext result of such a binary circuit has to be packed to a single ciphertext in the same finite field as $Enc(val_j)$. The blind evaluation of a predicate p for each encrypted value $Enc(val_j)$ is therefore

$$Enc(val_j) * pack(p\text{-}circuit(Enc(bval_j))) \tag{4}$$

4.2 Retrieving Matching Answers

From the above, we know that the cloud is able to record the ciphertext of a value val_j that satisfies a predicate p by multiplying $Enc(val_j) * pack(p\text{-}circuit(Enc(bval_j)))$ to an available $Enc(0)$ entry in the result buffer. However, the cloud does not know which entries are $Enc(0)$. The cloud cannot decide the size of the result buffer, or the entries to multiply $Enc(val_j) * pack(p\text{-}circuit(Enc(bval_j)))$. A naive way to retrieve matching answers for a selection or range query is to create a result buffer which is as big as the number of attribute values N, to initialize every entry corresponding to each attribute value val_j to $Enc(0)$, and to multiply $Enc(val_j) * pack(p\text{-}circuit(Enc(bval_j)))$ to each corresponding entry at the jth position. Then the communication and client computation costs would be $O(N)$, which are very expensive and are waste of resources when the number of actual answers $M \ll N$. We do not attempt to reduce server computation cost to ensure strong privacy for users, but our goal, however, is to keep communication cost and computation cost at the client small. Basically, we want to realize the benefit of cloud computing where a user query initiated at a client is processed in the cloud and the client receives the final answer without significant post-processing work.

We thus turn to a probabilistic solution, i.e. to create a fixed size result buffer B, and to save matching answers with overwhelming probability. We require that a client provides an estimated upper bound of the number of matching answers M', and a parameter to control search quality, γ. The cloud creates a ciphertext result buffer B whose size is proportional to $\gamma M'$, initializes each entry of B to $Enc(0)$. For each encrypted attribute value $Enc(val_j)\,(1 \leq j \leq N)$, the cloud multiplies $Enc(val_j) * pack(p\text{-}circuit(Enc(bval_j)))$ to γ entries at random into the buffer, as shown in Fig. 2. In this way, non-matching answers would be saved in the buffer B with probability 0, because $Enc(val_j) * pack(p\text{-}circuit(Enc(bval_j))) = Enc(val_j) * Enc(0) = Enc(0)$. For matching answers, if only one matching answer $Enc(val_j)$ is at one entry, the client can decrypt the ciphertext at the entry and recover the matching answer val_j, which we call a *survival*. However if more than one matching answers are multiplied to one entry, which we call a *collision*, none of them can be recovered at the client. Therefore, a matching answer can be successfully retrieved by the client if its ciphertext survives (not in collision) at one of the γ entries it is multiplied to. The server computation cost of query processing using this solution is still $O(N)$, i.e. the server needs to process each encrypted attribute value $Enc(val_j)\,(1 \leq j \leq N)$,

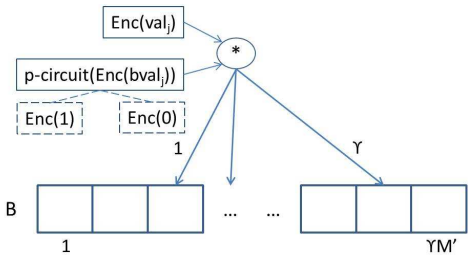

Fig. 2. Probabilistic Solution to Store Answers

but its communication cost and client computation cost are only $O(\gamma M')$.

The above probabilistic solution is similar to the private filter construction in private stream search [17]. Since the correctness of the solution is proved in [17], we just present the correctness statement without proving it again. Here $neg(\gamma)$ means a fractional number which is in $(0, 1)$ and negligible in γ.

Theorem 4.21 *If the number of matching answers is less than M', the probability that all matching answers are saved in B is larger than $1 - neg(\gamma)$. If the number of matching answers is larger than M', the probability that a subset of matching answers are saved in B is larger than $1 - neg(\gamma)$.*

To differentiate a valid entry which stores a recoverable matching answer from a collision entry, we use a probabilistic method from [17]. Append to each encrypted value $Enc(val_j)$ h bits which are partitioned into $h/3$ triples of bits, and randomly assign one bit in each triple to 1 and the other two bits as 0. A buffer entry is valid if exactly one bit in each triple of appended bits is 1.

4.3 Discussion

Sofar we see that we can construct circuits to perform blind predicate evaluation directly in the cloud, but cannot provide the guarantee to recover all matching answers when the answer size for a database query such as a range query varies.

Because a homomorphic encryption is semantically secure, we ensure complete data confidentiality. Because we perform all predicate evaluations blindly on all the encrypted values, we ensure strong access privacy for querying clients. The only information revealed are an estimated upper bound number of matching answers, M', the attribute with the query

predicate, A_i, and the type of the query predicate (indicated by the circuit). However, this information does not give any details about the actual contents of a query, i.e. x, or about the underlying data encrypted.

5 Supporting General Database Queries

Using the basic framework presented in Section 4, we now explore if this framework can be extended to support a variety of database queries on encrypted data rather than simple equality condition and range queries. For processing each type of query, we look at (1) how to represent its predicates into binary circuits that can be evaluated on the ciphertext values, and (2) how to store the projection attribute values in the result buffer (or how to update the aggregates). We do not repeat below but from Section 4 we know that for queries whose result sizes are not fixed, the matching answers can be recovered from the result buffer only with high probability.

5.1 Complex Selection and Range Queries

In Section 4 we process simple selection and range queries with predicate and projection on only one attribute. Now we consider how to solve the following general selection and range queries with predicates and projections on multiple attributes.

$$\begin{array}{rll} \text{SELECT } A_{k1}, ..., A_{kg} & & \\ \text{FROM} & DT & \\ \text{WHERE} & A_{i1} & [= \mid > \mid <]\ x_1 \\ [\text{AND}|\text{OR}] & ..... & \\ [\text{AND}|\text{OR}] & A_{il} & [= \mid > \mid <]\ x_l \end{array}$$

Predicate Evaluation. Following equations (1), (2) and (3) in Section 4, the predicates on attributes $A_{i1}, ...A_{il}$ can be represented as binary circuits on the ciphertexts of binary representation of the attributes' values, $p\text{-}circuit(Enc(A_{i1}.bval_j))$, ..., $p\text{-}circuit(Enc(A_{il}.bval_j))$ $(1 \leq j \leq N)$. Evaluating these predicates together on the jth tuple, we get circuit $p\text{-}circuit(Enc(A_{i1}.bval_j))$ $[\wedge|\vee]$ ... $[\wedge|\vee]$ $p\text{-}circuit(Enc(A_{il}.bval_j))$.

Results Storage and Retrieval. The ciphertext values of projection attributes $A_{k1}, ..., A_{kg}$ for which the corresponding tuples satisfy the predicates should be put in the result buffer. This can be achieved by allocating a result buffer whose size is proportional to $g\gamma M'$ and which

consists of $O(\gamma M')$ groups of g consecutive entries with each group being initialized as g $Enc(0)$s, and then for each tuple at the jth position, multiplying $Enc(A_{k1}.val_j) * pack(p\text{-}circuit(Enc(A_{i1}.bval_j))$ $[\wedge|\vee]$ ... $[\wedge|\vee]$ $p\text{-}circuit(Enc(A_{il}.bval_j)))$, ..., $Enc(A_{kl}.val_j) * pack(p\text{-}circuit(Enc(A_{i1}.bval_j))$ $[\wedge|\vee]$ ... $[\wedge|\vee]$ $p\text{-}circuit(Enc(A_{il}.bval_j)))$ to the g entries of randomly selected γ groups into the buffer. Upon receiving the result buffer, a client decrypts a set of g group entries from groups that are not collisions or $Enc(0)$s to retrieve the attribute values of matching tuples to the query.

During the above query processing, the only information revealed are an estimated upper bound number of matching answers M', the projection attributes $A_{k1}, ..., A_{kg}$, the attributes with query predicates, $A_{i1}, ..., A_{il}$ and their (AND, OR) relationships. This information gives few details about the actual query, i.e. $x_1, ..., x_l$, or about the underlying data.

5.2 Joins

Without loss of generality, consider the following two table join without additional selection and range predicates.

$$\begin{array}{rc} \text{SELECT } DT_1.A_{k1}, ..., DT_1.A_{kg}, DT_2.A_{q1}, ..., DT_2.A_{ql} \\ \text{FROM} \qquad DT_1, DT_2 \\ \text{WHERE} \qquad DT_1.A_{i1} = DT_2.A_{i2} \end{array}$$

Join query processing is easier if the join keys of the two tables, $DT_1.A_{i1}$ and $DT_2.A_{i2}$, are ordered (indexed) or hashed. However, that approach bears the same vulnerabilities of order-preserving encryption [5], i.e. revealing the data order, and of data bucketization [3, 4], i.e. revealing the data distribution. We therefore only consider using block nested loop to solve join queries on encrypted data.

Predicate Evaluation. Specifically, the cloud needs to perform $N_1 \times N_2$ blind evaluations to check if $DT_1.A_{i1}.val_{j1} - DT_2.A_{i2}.val_{j2} == 0$ $(1 \leq j1 \leq N_1, 1 \leq j2 \leq N_2)$, where N_1, N_2 are the number of tuples in the two tables DT_1 and DT_2 respectively. This can be achieved by constructing an equality condition circuit following equation (1), and multiplying the ciphertext result $Enc(1)$ (matching) or $Enc(0)$ (not matching) to the ciphertexts of projection attribute values $Enc(DT_1.A_{k1}), ..., Enc(DT_1.A_{kg})$, $Enc(DT_2.A_{q1}), ..., Enc(DT_2.A_{ql})$ respectively.

Results Storage and Retrieval. Allocating a result buffer whose size is proportional to $(g + l)\gamma M'$ and which consists of $O(\gamma M')$ groups of $(g + l)$ consecutive entries with each group being initialized as $(g + l)$

$Enc(0)$s, the $(g+l)$ multiplication results for one pair of DT_1 and DT_2 tuples are multiplied to the $(g+l)$ entries of randomly selected γ groups into the buffer. Upon receiving the result buffer, a client decrypts a set of $(g+l)$ group entries from groups that are not collisions or $Enc(0)$s to retrieve the attribute values of joining tuples.

During the above query processing, the only information revealed are an estimated upper bound number of matching answers M', the projection attributes $DT_1.A_{k1}, ..., DT_1.A_{kg}, DT_2.A_{q1}, ..., DT_2.A_{ql}$ and the join keys $DT_1.A_{i1}, DT_2.A_{i2}$. This information gives few details about the query, i.e. which pair of tuples can be joined, or about the underlying data.

5.3 Range Aggregations

Without loss of generality, consider the following simple range aggregation which only returns a single aggregate value per attribute.

SELECT	$AVG(A_{k1}), ..., SUM(A_{kg})$	
FROM	DT	
WHERE	A_{i1}	$[= \mid > \mid <]\ x_1$
[AND|OR]	.....	
[AND|OR]	A_{il}	$[= \mid > \mid <]\ x_l$

From the above processing of complex selection and range queries, we know that the range predicates can be represented as a circuit to be evaluated on each tuple (say the jth tuple), $p\text{-}circuit(Enc(A_{i1}.bval_j))$ $[\wedge|\vee]$ ... $[\wedge|\vee]$ $p\text{-}circuit(Enc(A_{il}.bval_j))$. Then a range aggregation query such as a SUM can be processed by initializing a ciphertext *enc-sum* as $Enc(0)$ for each attribute to aggregate on, and then keeping adding the multiplication result of the encrypted attribute value with the circuit evaluation result.

For example to obtain $SUM(A_{kg})$, at each tuple (the jth tuple), the cloud performs multiplication $Enc(A_{kg}.val_j) * pack(p\text{-}circuit(Enc(A_{i1}.bval_j))$ $[\wedge|\vee]$ ... $[\wedge|\vee]$ $p\text{-}circuit(Enc(A_{il}.bval_j)))$, and adds the ciphertext multiplication result to *enc-sum*. Note that *enc-sum* may overflow if the number of attribute values satisfying the predicates is too large. This problem can be solved by breaking *enc-sum* into several *enc-sum*s as in [15], which we do not repeat here. For AVG aggregations such as $AVG(A_{k1})$, the cloud needs to maintain a ciphertext counter *enc-count* in addition to *enc-sum*, and after the summation on all the attribute values has been

performed, to perform a division *enc-sum/enc-count*. Division on ciphertexts can be implemented as in [12]. Initialize *enc-count* as $Enc(0)$, and for each tuple (say jth tuple), the cloud performs multiplication $Enc(1) * pack(p\text{-}circuit(Enc(A_{i1}.bval_j))\ [\wedge|\vee]\ ...\ [\wedge|\vee]\ p\text{-}circuit(Enc(A_{il}.bval_j)))$, and adds the ciphertext multiplication result to *enc-count*.

During the above query processing, the only information revealed are the attributes to aggregate on, $A_{k1}, ..., A_{kg}$, their aggregation types, the attributes with query predicates, $A_{i1}, ..., A_{il}$ and their (AND, OR) relationships. These information gives few details about the query, i.e. $x_1, ..., x_l$, or about the underlying data.

5.4 Group-By Queries

Group-by queries are complicated to implement given that the encrypted attribute values are not ordered or hashed. Consider a typical group-by query as follows.

$$\begin{aligned} \text{SELECT } & A_{k1}, ..., SUM(A_{kg}) \\ \text{FROM } & DT \\ \text{GROUP BY } & A_{i1}, ..., A_{il} \end{aligned}$$

By replicating table DT as DT' and joining DT and DT' on attributes $A_{i1}, ..., A_{il}$, pairs of tuples with equal $A_{i1}, ..., A_{il}$ values can be found, but the tuples of equal $A_{i1}, ..., A_{il}$ values still need to be grouped together. We solve this problem by performing $(N-1)$ rounds of range and aggregation queries. At the jth round $(1 \leq j < N)$, the cloud chooses the jth tuple as the pivot tuple, compares all $(j+1)$th to Nth tuples with this pivot tuple on $A_{i1}, ..., A_{il}$, keeps and aggregates on $A_{k1}, ..., A_{kg}$ values for the tuples which satisfy the equal condition with the pivot tuple values. The query for each round is like below.

$$\begin{aligned} \text{SELECT } & A_{k1}, ..., SUM(A_{kg}) & & \\ \text{FROM } & DT & & \\ \text{WHERE } & A_{i1} & = & A_{i1}.val_j \\ \text{AND } & ..... & & \\ \text{AND } & A_{il} & = & A_{il}.val_j \end{aligned}$$

The client needs to decrypt results from each round and to merge the results together to form the final answers to the group-by query.

During the above query processing, the only information revealed are the projection attributes and the attributes to aggregate on, $A_{k1}, ..., A_{kg}$,

their aggregation types, the group attributes $A_{i1}, ..., A_{il}$. These information gives few details about the query, i.e. the group information, or about the underlying data.

5.5 Discussion on General Database Queries

A real database query is just a composition of the above simplified queries, and can be solved based on the processing of the above queries. As we see from the above, because the circuits constructed and the tables, attributes involved for different types of queries are different, although the query evaluation in the cloud is totally blind, it is not avoidable to reveal some information about the types of queries being processed. However, the actual query contents, i.e. predicates, or the answers are not revealed at all. In that sense, our query processing using homomorphic encryption achieves strong privacy for processing different types of queries compared to previous proposals for processing (usually limited) queries on encrypted data [3, 4, 5, 6, 21].

6 Conclusion

With the recent development of fully homomorphic encryption, homomorphic encryption has received great interests and expectations for solving arbitrary queries on encrypted data. However from the existing work it is not clear how homomorphic encryption processes complex database queries such as range and join queries beyond simple aggregations and statistical function calculations. To the best of our knowledge, this paper has conducted a first systematic study on how to use the powerful primitives of fully homomorphic encryption to solve general database queries on encrypted data while preserving *complete data confidentiality* and *strong access privacy*. We have shown that it is possible to perform blind evaluations on encrypted data for a variety of database queries, while for queries without fixed answer sizes, such as range and join queries, *most but not all* answers are guaranteed to be successfully retrieved by the querying client.

Despite demonstrating the ability of fully homomorphic encryption for solving different types of database queries on encrypted data, we are also aware of the limitations of the current fully homomorphic encryption schemes, which need to be solved before these schemes can be used in practical database applications. Performance of homomorphic encryption operations is one of the biggest issues. Lack of schemes to support different data access control needs is another issue. To enable arbitrary query

operations on encrypted data, all the data have to be encrypted using the same public key, making it inconvenient to enforce different levels of data access control. In our future work, we will explore using parallel computing and hardware acceleration to optimize the performance of homomorphic encryption operations, and design alternative schemes to solve the access control problem of homomorphic encryption.

References

[1] Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: SP '00: Proceedings of the 2000 IEEE Symposium on Security and Privacy. (2000) 44–55

[2] Chang, Y.C., Mitzenmacher, M.: Privacy preserving keyword searches on remote encrypted data. In: ACNS. (2005) 442–455

[3] Hacigumus, H., Iyer, B.R., Li, C., Mehrotra, S.: Executing SQL over encrypted data in the database service provider model. In: SIGMOD Conference. (2002)

[4] 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. (2004) 720–731

[5] Agrawal, R., Kiernan, J., Srikant, R., Xu, Y.: Order preserving encryption for numeric data. In: SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data. (2004) 563–574

[6] Popa, R.A., Redfield, C.M.S., Zeldovich, N., Balakrishnan, H.: Cryptdb: protecting confidentiality with encrypted query processing. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. SOSP '11 (2011) 85–100

[7] 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. (2003) 93–102

[8] 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)

[9] Wang, S., Agrawal, D., El Abbadi, A.: A comprehensive framework for secure query processing on relational data in the cloud. In: Proceedings of the 8th VLDB international conference on Secure data management. SDM'11 (2011) 52–69

[10] De Capitani di Vimercati, S., Foresti, S., Paraboschi, S., Pelosi, G., Samarati, P.: Efficient and private access to outsourced data. In: Proc. of the 31st International Conference on Distributed Computing Systems (ICDCS 2011). (June 2011)

[11] Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC '09: Proceedings of the 41st annual ACM symposium on Theory of computing. (2009) 169–178

[12] Gentry, C.: Computing arbitrary functions of encrypted data. Commun. ACM **53** (2010) 97–105

[13] Johnson, R.C.: Ibm encryption breakthrough could secure cloud computing. http://www.smartertechnology.com/c/a/Technology-For-Change/IBM-Encryption-Breakthrough-Could-Secure-Cloud-Computing (2009)

[14] Simonite, T.: A cloud that can't leak. http://www.technologyreview.com/computing/38239 (2011)

[15] 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. (2007) 519–530
[16] Naehrig, M., Lauter, K., Vaikuntanathan, V.: Can homomorphic encryption be practical? In: Proceedings of the 3rd ACM workshop on Cloud computing security workshop. CCSW '11, New York, NY, USA, ACM (2011) 113–124
[17] Ostrovsky, R., Skeith, W.E.: Private searching on streaming data. Journal of Cryptology **20** (2007) 397–430
[18] Wang, S., Agrawal, D., Abbadi, A.E.: Towards practical private processing of database queries over public data with homomorphic encryption. Technical Report 2011-06, Department of Computer Science, University of California at Santa Barbara (2011)
[19] Smart, N., Vercauteren, F.: Fully homomorphic encryption with relatively small key and ciphertext sizes. In Nguyen, P., Pointcheval, D., eds.: Public Key Cryptography C PKC 2010. Volume 6056 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2010) 420–443
[20] Emekci, F., Agrawal, D., Abbadi, A.E., Gulbeden, A.: Privacy preserving query processing using third parties. In: ICDE. (2006)
[21] De Capitani di Vimercati, S., Foresti, S., Paraboschi, S., Pelosi, G., Samarati, P.: Efficient and private access to outsourced data. In: Proc. of the 31st International Conference on Distributed Computing Systems (ICDCS 2011). (2011)
[22] 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 '11 (2011) 205–216
[23] El Gamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Proceedings of CRYPTO 84 on Advances in cryptology, New York, NY, USA, Springer-Verlag New York, Inc. (1985) 10–18
[24] Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Advances in Cryptology - EUROCRYPT '99. Volume 1592 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (1999) 223–238
[25] Boneh, D., Goh, E.J., Nissim, K.: Evaluating 2-dnf formulas on ciphertexts. In: Proceedings of Theory of Cryptography Conference 2005. Volume 3378 of LNCS., Springer (2005) 325–342