Privacy-Preserving Complex Query Evaluation over Semantically Secure Encrypted Data

Bharath Kumar Samanthula[1], Wei Jiang[2], and Elisa Bertino[1]

[1] Department of Computer Science, Purdue University
305 N. University Street, West Lafayette, IN 47907
{bsamanth,bertino}@purdue.edu

[2] Department of Computer Science, Missouri S&T
500 W. 15th Street, Rolla, MO 65409
wjiang@mst.edu

Abstract. In the last decade, several techniques have been proposed to evaluate different types of queries (e.g., range and aggregate queries) over encrypted data in a privacy-preserving manner. However, solutions supporting the privacy-preserving evaluation of complex queries over encrypted data have been developed only recently. Such recent techniques, however, are either insecure or not feasible for practical applications. In this paper, we propose a novel privacy-preserving query processing framework that supports complex queries over encrypted data in the cloud computing environment and addresses the shortcomings of previous approaches. At a high level, our framework utilizes both homomorphic encryption and garbled circuit techniques at different stages in query processing to achieve the best performance, while at the same time protecting the confidentiality of data, privacy of the user's input query and hiding data access patterns. Also, as a part of query processing, we provide an efficient approach to systematically combine the predicate results (in encrypted form) of a query to derive the corresponding query evaluation result in a privacy-preserving manner. We theoretically and empirically analyze the performance of this approach and demonstrate its practical value over the current state-of-the-art techniques. Our proposed framework is very efficient from the user's perspective, thus allowing a user to issue queries even using a resource constrained device (e.g., PDAs and cell phones).

Keywords: Privacy, Complex Query, Encryption, Cloud Computing.

1 Introduction

In the past few years, there has been a significant growth in user's interest to outsource their data as well as operational services to the cloud. Along this direction, many small and medium size businesses have already outsourced their daily business processes to prominent cloud service providers such as Amazon, Google, and IBM. As privacy is a crucial requirement for many users, applications and organizations, data are usually encrypted before being uploaded to the cloud. By doing so, data confidentiality is still guaranteed even when a cloud server is compromised due to a hacking attack. However, the management of encrypted data poses several challenges, the most important of which is query processing. During query processing, we need to not only keep the data private from the cloud, but also the users' input queries. The question to ask is "how can the cloud perform searches over encrypted data without ever decrypting them or compromising the user's privacy". In the past decade, such question has resulted in a specific research area, known as *privacy-preserving query processing over encrypted data (PPQED)*.

As mentioned in [1], there are three different approaches to perform PPQED: (i) the query issuer downloads the entire encrypted database and performs a local search on the decrypted database, (ii) the cloud employs custom-designed cryptographic protocols to operate over encrypted data directly or indirectly, and (iii) the cloud deploys the tamper-proof trusted hardware (which is either trusted or certified by the clients) on the cloud-side that facilitates the cloud when operating over encrypted data inside a secure environment. The first approach, however, is not practical as it incurs heavy cost (both computation and communication) on the end-user (i.e., the query issuer). Techniques based on trusted hardware (e.g., [1]), such as the IBM 4764 or 4765 cryptographic co-processor, have gained significant attention in recent years. However, secure (or trusted) hardware is still very expensive and may not be suitable for cloud computing which is intended to use cheap commodity machines. Also, services based on secure hardware may not be affordable for some small businesses. Apart from those approaches, another widely investigated approach is based on the deployment of custom-designed cryptographic techniques by the cloud to operate over encrypted data.

Along this direction, researchers from both academia and industry have proposed several approaches (e.g., [2–8]). However, most of such approaches focus on privacy-preserving protocols for evaluating specific queries (e.g., range and aggregate) over encrypted data. That is, they are not directly useful to execute complex queries over encrypted data. As a result, privacy-preserving evaluation of complex and arbitrary queries over encrypted data is still an open and challenging problem for data outsourcing. Some recent approaches have addressed this problem to an extent. However, such approaches are either insecure or not feasible for practical applications (see Section 2 for more details). In particular, as highlighted in [9–11], data access pattern information can leak much valuable information (e.g., correspondence between plaintexts and ciphertexts) to the cloud. We believe that the data access patterns should be protected from the cloud which would otherwise compromise the semantic security [12] of encrypted data stored in the cloud. Unfortunately, most of the existing PPQED methods do not address access pattern issue (see Section 2.2 for more details).

Hence, the primary focus of this paper is to develop a secure cloud computing framework that can support the evaluation of complex queries and is also efficient from an end-user's perspective. To obtain the best performance, our framework switches between homomorphic encryption and garbled circuit techniques based on the underlying parametric values and the sub-task (part of query processing) at hand.

1.1 Problem Statement

In our problem setting, we consider three different parties: the data owner (also referred to as Alice), the cloud, and the data consumer (also referred to as Bob). Let T denote Alice's database with n records, denoted by $t_1, \ldots, t_n$, and m attributes. We assume that Alice initially encrypts T attribute-wise using her public key and outsources the encrypted database to a cloud. In this paper, we explicitly assume that Alice's secret key is generated using a semantically secure[1] and additive homomorphic encryption scheme (such as the Paillier cryptosystem [13]). First, it is worth pointing out that semantic security is necessary to ensure that the cloud cannot distinguish the encrypted data in the first place (i.e., ciphertexts should be computationally indistinguishable from the cloud's perspective). Second, by encrypting the data using an additive homomorphic encryption scheme, Alice makes it possible for the cloud to perform certain operations directly over encrypted data, such as operations that might also be useful for other data analytics tasks (e.g., secure clustering and classification). More details regarding the properties of the additive homomorphic encryption scheme used in our approach are provided in Section 3.2.

Let T' denote the encrypted database of Alice. Now consider an authorized user Bob (which would typically be authorized by Alice) who wants to securely retrieve data from T' in the cloud using his private (complex) query Q. In this paper, a complex query is defined as a query with arbitrary number of sub-queries where each sub-query can consist of conjunctions and/or disjunctions of arbitrary number of relational predicates. An example could be $Q = ((Age \geq 40) \vee ((Sex = M) \wedge (Marital\ Status = Married))) \wedge (Disease = Diabetes)$. We assume that Q is represented as a boolean function expressed in disjunctive normal form[2] (DNF) as follows.

$$Q : G_1 \vee G_2 \vee \ldots \vee G_{l-1} \vee G_l \rightarrow 0, 1$$

where the input to Q is a data record t_i. Here G_j denotes the j^{th} clause which is a conjunction of b_j predicates, for $1 \leq j \leq l$, and l denotes the number of clauses in Q. More specifically, $G_j = P_{j,1} \wedge P_{j,2} \wedge \ldots \wedge P_{j,b_j-1} \wedge P_{j,b_j}$ and each predicate $P_{j,k}$ is also a boolean function that returns either 0 or 1 depending on the underlying condition. In general, a predicate applies a relational operator (i.e., $>, \geq, <, \leq, =$) on specific attribute values and search input. For example, consider the predicate $P_{1,1} : AGE > 20$, where AGE is an attribute in T, and a record t_i from T. Then, $P_{1,1}(t_i) = 1$ iff the AGE attribute value in data record t_i is greater than 20. Otherwise, $P_{1,1}(t_i) = 0$.

Under the above system model, the goal of our approach is to facilitate Bob in efficiently retrieving the data records from T' (stored in the cloud) that satisfy Q in a privacy-preserving manner. We refer to such a process as privacy-preserving query processing over encrypted data (PPQED). More formally, we define a PPQED protocol as follows:

$$\text{PPQED}(T', Q) \rightarrow S$$

where $S \subseteq T$ denotes the output set of records that satisfy Q. That is, $\forall\, t' \in S$, $Q(t') = 1$. In general, a PPQED protocol should meet the following privacy requirements:

- **Data Confidentiality** - during the query processing phase, neither the contents of T nor of any intermediate results are disclosed to the cloud.

[1] Precisely, if the encryption scheme is semantically secure, then the ciphertexts are random numbers from the cloud's perspective.

[2] Note that any given boolean function can be represented in both DNF and conjunctive normal form (CNF). In this paper, we simply choose DNF to represent Q. However, our proposed protocol can be easily adopted to the later case upon simple modifications.

- **End-user's Privacy** - At any point of time, Bob's query Q should not be disclosed to the cloud and Alice.
- At the end of the PPQED protocol, S should be revealed only to Bob.
- $T - S$ (i.e., information other than the output) should never be disclosed to Bob.
- Data access patterns should never be disclosed to the cloud (as well as to Alice). That is, for any two queries Q_1 and Q_2, the corresponding output sets S_1 and S_2 should be computationally indistinguishable from the cloud's perspective.

In our proposed PPQED protocol, once Alice outsources her encrypted data to the cloud, she does not participate in the query processing task; therefore, no information is revealed to Alice. However, it may be required that in certain applications, Alice is able to validate Bob's query before forwarding it to the cloud. We claim that such extensions can be easily incorporated into the proposed PPQED protocol upon straightforward modifications. For simplicity, we do not consider such natural extensions to PPQED in the rest of this paper. Also, due to space limitations, we do not discuss how our solution can be extended to protect access pattern information in this paper. However, we refer the reader to our technical report [14] for a detailed discussion on extending our proposed solution to hide the data access pattern information.

1.2 Main Contributions

In this paper, we propose a new two-stage PPQED protocol under the cloud computing environment. At a high level, the main contributions of this paper are as follows.

(a). **Security:** The proposed PPQED protocol protects the confidentiality of data, privacy of the user's input query and also hides the data access patterns under the standard semi-honest model [15].
(b). **Efficiency:** Our proposed protocol incurs negligible computation cost on the end-user. Also, we propose an efficient mechanism that systematically combines the individual predicate results to compute the corresponding query evaluation result. Our theoretical analysis shows that the proposed solution improves the upper bound compared to the naive solution constructed from the existing techniques.
(c). **Flexibility:** Since the proposed PPQED protocol is a hybrid approach, in that it utilizes both homomorphic encryption and garbled circuits, it allows the developers to switch between the two depending on the application requirements, and thus enhancing flexibility. Specifically, our protocol can be used as a building block in larger privacy-preserving applications. E.g., the cloud can perform data analytics on different query results either using homomorphic encryption or garbled circuit techniques. More details on how to convert homomorphic values to garbled values and vice versa are presented in Section 4.

The rest of the paper is organized as follows. In Section 2, we review upon the existing work related to our problem domain. Section 3 introduces relevant background information on the threat model assumed in this paper and on additive homomorphic encryption scheme used in our approach. A set of security primitives that are utilized in the proposed PPQED protocol and their possible implementations are provided in Section 4. Also, the proposed PPQED protocol is explained in detail along with the security and complexity analysis in this section. Finally, we conclude the paper and highlight possible directions for future research in Section 5.

2 Related Work

2.1 Query Processing over Encrypted Data

In general, the computations involved in query processing depend on the query under consideration. Along this direction, several methods have been proposed to securely process range (e.g., [2, 4, 6–8, 16]) and aggregate (e.g., [3, 5, 17]) queries over encrypted data. It is worth noting that such methods are suitable for evaluating only specific queries; thus, they are not directly applicable to solve the PPQED problem (i.e., combination of multiple and different sub-queries) over encrypted data. Also, they leak different kinds of information for efficiency reasons. Due to space limitations, we refer the reader to our technical report [14] for more details regarding their disadvantages.

2.2 Existing PPQED Methods

Unfortunately, only a very few approaches have been proposed to address the PPQED problem. In what follows, we discuss the main differences of our work with approaches proposed along those directions. Table 1 highlights some of the key differences between the existing work and our solution.

Golle et al. [18] were the first to propose a protocol that can evaluate conjunctive equality queries on encrypted documents. However, their protocol supports neither disjunctive queries nor predicates with inequality conditions. As an improvement, Boneh and Waters [16] proposed a new searchable public-key system (referred to as hidden vector encryption) that supports comparison and general subset queries over encrypted data. We emphasize that their technique is very expensive and complex to implement. Also, their method is suitable for conjunctive queries, but not applicable to either disjunctive queries or combination of both. As an alternative approach, Popa et al. [19] proposed CryptDB, a system that executes SQL queries over encrypted data using a set of SQL-aware encryption schemes. At a high level, their system encrypts each data item using an onion of encryption schemes with the outermost layer providing maximum security, whereas the innermost layer provides more functionality and weak security. During the query processing stage, the cloud is given secret keys to decrypt the outer layers and perform the necessary operations over encrypted data at inner layers. However, CryptDB has some major drawbacks: (i) it uses a proxy which is a trusted third-party and thus makes it hard to use the system in practical applications, (ii) it reveals different types of information to the cloud server at different layers, and (iii) multiple onions may have to be generated for each data item which makes the approach very expensive. The actual security offered by an onion in CryptDB is the protection offered by its inner most layer. For example, consider an onion in CryptDB for comparison operations, it reveals the relative ordering among the attribute values to the cloud. Thus, CryptDB does not ensure data confidentiality in all cases. Also, none of the above PPQED methods addressed the access pattern issue which is a crucial privacy requirement [9–11].

In the past few years, researchers have also investigated secure query processing frameworks based on the use of tamper-proof trusted hardware on the cloud side. Along this direction, Bajaj and Sion [1] proposed TrustedDB, an outsourced database framework that allows a client to execute SQL queries by leveraging cloud-hosted tamper-proof trusted hardware in critical query processing stages. However, as mentioned in

Table 1. Comparison with the existing work

Method	Low Cost on Bob	Data Confidentiality	Query Privacy	Hide Data Access Patterns	CNF and DNF Query Support
Golle et al. [18]	✖	✔	✔	✖	✖
Boneh and Waters [16]	✖	✔	✔	✖	✖
Popa et al. [19]	✔	✖	✖	✖	✔
This paper	✔	✔	✔	✔	✔

Section 1, secure hardware is very expensive and may not be suitable for cloud computing which is intended to use cheap commodity machines. Also, services based on secure hardware may not be affordable for some small businesses. Another area of research is based on the use of Oblivious RAM (ORAM) techniques (e.g., [20]) to solve the PPQED problem. However, under ORAM techniques, the query issuer need to know the index structure before hand (which may not always be possible). In particular to the PPQED problem, each authorized user has to efficiently maintain multiple indexes to support complex queries. We believe that more research is needed to investigate the side effects of using secure processors and ORAM techniques to solve the PPQED problem and we leave this interesting open problem for future work.

We may ask whether fully homomorphic cryptosystems such as [21], which can perform arbitrary computations over encrypted data without ever decrypting them, are suitable to solve the PPQED problem. It is a known fact that fully homomorphic encryption schemes can compute any function over encrypted data [22]. However, such schemes are very expensive and their usage in practical applications has yet to be explored. For example, it was shown in [23] that even for weak security parameters one "bootstrapping" operation of the homomorphic operation would take at least 30 seconds on a high performance machine.

Based on the above discussions, it is clear that there is a strong need to develop an efficient PPQED protocol that can protect data confidentiality, privacy of the user's input query and data access patterns at all times.

3 Background

3.1 Adversarial Model

In this paper, privacy/security is closely related to the amount of information disclosed during the execution of a protocol. To maximize security guarantee, we adopt the commonly accepted security definitions and proof techniques in the literature of secure multi-party computation (SMC) to analyze the security of our proposed protocol. SMC was first introduced by the Yao's Millionaire problem [24, 25] under the two-party setting, and it was extended to multi-party computations by Goldreich et al. [26].

There are two common adversarial models under SMC: semi-honest and malicious. Due to space limitations, we refer the reader to [15] for more details regarding their security definitions and proof techniques. In this paper, to develop secure and efficient protocols, we assume that the participating parties are semi-honest. This implicitly assumes that there is no collusion between the parties. We emphasize that this assumption is not new and the existing PPQED methods discussed in Section 2.2 were also proposed under the semi-honest model. Indeed, it is worth noting that such an assumption makes sense especially under the cloud environment. This is because, since the current known cloud service providers are well established IT companies, it is hard to see the possibility for two companies, e.g., Google and Amazon, to collude to damage their reputations and consequently place negative impact on their revenues. Thus, in our problem domain, assuming that the participating parties are semi-honest is realistic. Note that, even under the malicious two-party setting, one has to assume that there is no collusion between the participating parties due to the theoretical limitation [27].

3.2 Paillier Cryptosystem

The Paillier cryptosystem is an additive homomorphic and probabilistic asymmetric encryption scheme [13]. Let E_{pk} be the encryption function with public key pk given by (N, g), where N is a product of two large primes and g is a generator in $\mathbb{Z}^*_{N^2}$. Also, let D_{sk} be the decryption function with secret key sk. Given two plaintexts $x, y \in \mathbb{Z}_N$, the Paillier encryption scheme exhibits the following properties.

a. **Homomorphic Addition:** $E_{pk}(x + y) \leftarrow E_{pk}(x) * E_{pk}(y) \bmod N^2$;
b. **Homomorphic Multiplication:** $E_{pk}(x * y) \leftarrow E_{pk}(x)^y \bmod N^2$;
c. **Semantic Security:** The encryption scheme is semantically secure [12]. Briefly, given a set of ciphertexts, an adversary cannot deduce any additional information about the plaintexts.

We emphasize that any other additive homomorphic encryption scheme (e.g., [28]) that satisfies the above properties can be utilized to implement our proposed framework. However, to be concrete and for efficiency reasons, this paper assumes that Alice encrypts her data using the Paillier cryptosystem before outsourcing them to a cloud.

4 The Proposed Framework

In this section, we first discuss a set of privacy-preserving primitives that will be later used in the proposed PPQED protocol as building blocks. Then, we demonstrate how to securely evaluate a predicate using homomorphic encryption and garbled circuits. Finally, we present our novel PPQED scheme that facilitates Bob in retrieving the data (that satisfy his query Q) from the cloud in a privacy-preserving manner.

In the proposed framework, we assume the existence of two non-colluding semi-honest cloud service providers, denoted by C_1 and C_2, which together form a federated cloud. We emphasize that such a setting is not new and has been commonly used in the recent related works (e.g., [29, 30]). Initially, as part of the key setup stage, the data owner Alice generates a pair of public/secret key pair (pk, sk) based on Paillier's scheme [13]. Suppose Alice outsources her encrypted database T' to C_1 and the secret key sk to C_2. That is, C_1 has $T'_{i,j} = E_{pk}(t_{i,j})$, for $1 \leq i \leq n$ and $1 \leq j \leq m$. In this paper, we explicitly assume that there exist secure communication channels (e.g., SSL) between each pair of participating parties. Note that other basic mechanisms, such as authentication and data replication, are well-studied problems under the cloud computing model; therefore, they are outside the scope of this paper.

Though we propose the PPQED protocol under the federated cloud model, we stress that it can also be implemented under the single cloud model with the same security guarantees. More specifically, under the single cloud setting, the role of the second cloud (i.e., C_2) can be played by Alice with her own private server holding the key sk. However, with limited computing resource and technical expertise, it is in the best interest of Alice to completely outsource its data management and operational tasks to a cloud. In general, whether Alice uses a private server or cloud service provider C_2 actually depends on her resources. In particular to our solution, after outsourcing encrypted data to C_1 and C_2, Alice does not participate in any future computations.

4.1 Basic Security Primitives

In this sub-section, we discuss three basic security primitives that will be later used in constructing our proposed PPQED protocol.

- Secure Multiplication (SMP): In this protocol, we assume that C_1 holds the private input $(E_{pk}(a), E_{pk}(b))$ and C_2 holds the secret key sk, where a and b are unknown to C_1 and C_2. The output of SMP is $E_{pk}(a * b)$ and revealed only to C_1. During this process, no information regarding a and b should be revealed to C_1 and C_2.
- Secure Bit-OR (SBOR): In this protocol, C_1 holds private input $(E_{pk}(o_1), E_{pk}(o_2))$ and C_2 holds sk. The goal of SBOR is to securely compute $E_{pk}(o_1 \vee o_2)$, where o_1 and o_2 are two bits. The output $E_{pk}(o_1 \vee o_2)$ should be known only to C_1.
- Secure Comparison (SC): In this protocol, C_1 holds private input $(E_{pk}(a), E_{pk}(b))$ and C_2 holds sk such that a and b are unknown to both parties, where $0 \leq a, b < 2^w$. Here w denotes the domain size (in bits) of a and b. The goal of the secure comparison (SC) protocol is to evaluate the condition $a > b$. At the end of the protocol, the output $E_{pk}(c)$ should be revealed only to C_1, where c denotes the comparison result. More specifically, $c = 1$ if $a > b$, and $c = 0$ otherwise. During this process, no information regarding a, b, and c is revealed to C_1 and C_2.

The efficient implementations of SMP and SBOR are given in [14]. On the other hand, though many SC protocols (under the two-party setting) have been proposed, we observe that they reveal the comparison result c to at least one of the participating parties. In this paper, we extend the SC protocol proposed in [31] to address our problem requirements. More details are given in the next sub-section.

4.2 Secure Evaluation of Individual Predicates (SEIP)

In this sub-section, we consider the scenario of evaluating a given predicate over T' stored in C_1. Without loss of generality, let $P : (k, \alpha, op)$ be a predicate, where α denotes the search input and k denotes the attribute index upon which the relational operator op has to be evaluated. More specifically, t_i satisfies the predicate P (i.e., $P(t_i) = 1$) iff the relational operation op on $t_{i,k}$ and α holds. In general, the possible set for op is $\{>, \geq, <, \geq, =\}$. It is important to note that the value of α should not be revealed to Alice, C_1, and C_2 for privacy reasons (note that Alice does not participate in query processing, so no information is revealed to her). To evaluate P, Bob first needs to send $E_{pk}(P) = (k, E_{pk}(\alpha), op)$ to C_1. However, if the number of predicates is large, Bob's computation cost for encryption can be high. E.g., if Q has 100 predicates, denoted by $P_1, \ldots, P_{100}$, then Bob has to compute $E_{pk}(P_1), \ldots, E_{pk}(P_{100})$.

We adopt the following simple strategy that incurs negligible computation cost on Bob and at the same time preserves the privacy of his predicate. Bob generates two random shares of α such that $\alpha_1 + \alpha_2 \mod N = \alpha$. A simple way to generate these shares is to set $\alpha_1 = N - r$ and $\alpha_2 = \alpha + r \mod N$, where r is a random number in $\mathbb{Z}_N$ known only to Bob. It is clear that $\alpha = \alpha_1 + \alpha_2 \bmod N$. After this, he sends $P^{\langle 1 \rangle}$ and $P^{\langle 2 \rangle}$ to C_1 and C_2, respectively, where $P^{\langle 1 \rangle} = (k, \alpha_1, op)$ and $P^{\langle 2 \rangle} = (\alpha_2, op)$. Here Bob needs to send the relational operator op to both C_1 and C_2 in order to evaluate P. Then, C_1 with input $\langle T'_i, P^{\langle 1 \rangle} \rangle$ and C_2 with input $P^{\langle 2 \rangle}$ need to securely verify whether the relational operation op holds between $t_{i,k}$ and α without revealing any information to C_1 and C_2, for $1 \leq i \leq n$. We refer to such a process as secure evaluation of individual predicates (SEIP).

For simplicity, let op be the greater than relational comparison operator (however, similar steps can be derived for other relational operators). Under this case, the goal is for C_1 with private input $\langle T'_i, (k, \alpha_1, >) \rangle$ and C_2 with $(\alpha_2, >)$ to securely determine whether $t_{i,k} > \alpha$, for $1 \leq i \leq n$. Let the evaluation result be c_i. Then the output should be $E_{pk}(c_i)$ such that $c_i = 1$ if $t_{i,k} > \alpha$, and $c_i = 0$ otherwise. At the end, the output $E_{pk}(c_i)$ should be revealed only to C_1. Also, the values of c_i and α should not be revealed to C_1 and C_2. In addition, during this process, no information regarding the contents of T should be revealed to C_1 and C_2.

At first sight, it is clear that the existing secure comparison (SC) protocols can be used to solve the SEIP problem (assuming greater than relational operator). Current SC protocols, under the two-party setting, are based on two techniques: (i) homomorphic encryption and (ii) garbled circuits. We now discuss how to solve the SEIP problem using SC with each of these two techniques.

SEIP Using Homomorphic Encryption. Given that C_1 holds $\langle T'_i, (k, \alpha_1, >) \rangle$ and C_2 holds $(\alpha_2, >)$, we aim to solve the SEIP problem using the homomorphic encryption based SC protocols (denoted by SEIP$_\text{h}$) as follows. To start with, C_2 initially sends $E_{pk}(\alpha_2)$ to C_1. Upon receiving, C_1 locally computes $E_{pk}(\alpha) = E_{pk}(\alpha_1) * E_{pk}(\alpha_2)$. Now, the goal is for C_1 and C_2 to securely evaluate the functionality $t_{i,k} > \alpha$ with $(E_{pk}(t_{i,k}), E_{pk}(\alpha))$ as input using the existing SC protocols. Remember that $(E_{pk}(t_{i,k}), E_{pk}(\alpha))$ is known only to C_1.

The existing SC protocols under homomorphic encryption strongly rely on encryptions of individual bits rather than on simple encrypted integers [31]. However, existing secure bit-decomposition (SBD) techniques can be utilized for converting an encrypted integer into encryptions of the corresponding individual bits. For example, consider two integers x and y such that $0 \leq x, y < 2^w$, where w denotes the domain size (in bits) of x and y. Let x_1 (resp., y_1) and x_w (resp., y_w) denote the most and least significant bits of x (resp., y), respectively. Given $E_{pk}(x)$ and $E_{pk}(y)$, C_1 and C_2 can securely convert them into $\langle E_{pk}(x_1), \ldots, E_{pk}(x_w)\rangle$ and $\langle E_{pk}(y_1), \ldots, E_{pk}(y_w)\rangle$ using the existing SBD techniques [32,33]. Note that the outputs are revealed only to C_1. Next, we detail the main steps involved in the SC protocol, proposed by Blake et al. [31], that takes the encrypted bit-wise vectors of x and y as input and outputs $c = 1$ if $x > y$, and 0 otherwise. To start with, for $1 \leq i \leq w$, C_1 performs the following operations:

- Compute an encryption of the difference between the i^{th} bits of x and y as $E_{pk}(d_i) = E_{pk}(x_i - y_i)$.
- Compute an encryption of the XOR between the i^{th} bits as $E_{pk}(z_i) = E_{pk}(x_i \oplus y_i)$. Note that $x_i \oplus y_i = x_i + y_i - 2x_i * y_i$. Therefore, this step requires an implicit secure multiplication (SMP) protocol as the building block to compute $E_{pk}(x_i * y_i)$.
- Generate an encrypted vector γ such that $\gamma_0 = 0$ and $\gamma_i = 2\gamma_{i-1} + z_i$.
- Generate an encrypted vector δ such that $\delta_i = d_i + r_i * (\gamma_i - 1)$, where r_i is a random number in $\mathbb{Z}_N$. The observation here is, if $\gamma_k = 1$ (denoting the first position at which the corresponding bits of x and y differ), then $\delta_k = d_k$. For all other indexes (i.e., $i \neq k$), δ_i is a random number in $\mathbb{Z}_N$.
- Let $\delta' = \langle E_{pk}(\delta_1), \ldots, E_{pk}(\delta_w)\rangle$. C_1 permutes δ' using a random permutation function π (known only to C_1) to get $\tau = \pi(\delta')$ and sends it to C_2.

Upon receiving, C_2 decrypts τ component-wise and checks for index k. If $D_{sk}(\tau_k) = 1$, then $x > y$. Similarly, if $D_{sk}(\tau_k) = -1$, then $y > x$. Note that $D_{sk}(\tau_j)$ always yields a random value in $\mathbb{Z}_N$, for $j \neq k$ and $1 \leq j \leq w$.

It is worth pointing out that we cannot directly use the SC protocol of [31] in SEIP$_\text{h}$ as it leaks the comparison result to C_2. Therefore, in order to use the method in [31], we need to somehow prevent this information leakage. Along this direction, with the goal of providing better security, we now provide a mechanism, as an extension to [31], that obliviously hides the comparison result from both C_1 and C_2. We denote the extended version of the SC protocol in [31] by SC$_\text{obv}$.

The main idea of SC$_\text{obv}$ is as follows. Instead of evaluating the greater than functionality directly, C_1 can randomly choose a functionality F (by flipping a coin), where F is either $x > y$ or $y \geq x$, and obliviously execute F with C_2. Since F is randomly chosen and known only to C_1, the comparison result is oblivious to C_2. Also, unlike [31], the output of SC$_\text{obv}$ is the encryption of comparison result (i.e., $E_{pk}(c)$) which will be known only to C_1. Note that the comparison result (i.e., c) should not be revealed to C_1 and C_2. The main steps involved in the SC$_\text{obv}$ protocol are as given below:

- Initially, C_1 chooses F randomly and proceeds as follows. If $F : x > y$, compute $E_{pk}(d_i) = E_{pk}(x_i - y_i)$. Else, compute $E_{pk}(d_i) = E_{pk}(y_i - x_i)$, for $1 \leq i \leq w$.
- C_1 computes the encrypted vector δ' using the similar steps (as discussed above) in the SC protocol of [31]. After this, C_1 sends $\tau = \pi(\delta')$ to C_2.
- Upon receiving, C_2 decrypts it component-wise and finds the index k. If $D_{sk}(\tau_k) = 1$, then compute $U = E_{pk}(1)$. Else, i.e., when $D_{sk}(\tau_k) = -1$, compute $U = E_{pk}(0)$. Then, C_2 sends U to C_1.
- Finally, C_1 computes the output $E_{pk}(c)$ as follows. If $F : x > y$, then $E_{pk}(c) = U$. Else, $E_{pk}(c) = E_{pk}(1) * U^{N-1}$.

It is important to note that, since U is in encrypted form, C_1 cannot deduce any information regarding the output c. In addition, as F is randomly chosen and known only to C_1, the output is oblivious to C_2. Hence, we claim that the comparison result c is protected from both C_1 and C_2. Note that $E_{pk}(c)$ is known only to C_1.

SEIP Using Garbled Circuits. In this sub-section, we discuss how to solve the SEIP problem using the garbled circuit technique (denoted by SEIP_g) [34]. For this purpose, we first need to convert the homomorphic value $E_{pk}(t_{i,k})$ into a garbled value. Also, a garbled value for α should be generated. To achieve this, we propose a simple solution which is as follows. Initially, C_1 generates random shares for $t_{i,k}$ using $E_{pk}(t_{i,k})$. That is, C_1 computes $E_{pk}(t_{i,k} + r)$, where r is a random value in $\mathbb{Z}_N$, and sends it to C_2. Upon receiving, C_2 decrypts it to get the random share $t_{i,k} + r \bmod N$. Also, C_1 sets his/her random share as $N - r$. Apart from this, remember that C_1 and C_2 have α_1 and α_2 (random shares of α), respectively. Also, C_1 picks a random number r' from $\mathbb{Z}_N$. Now, C_1 constructs a garbled circuit by extending the circuit corresponding to the SC protocol of [35] based on the following steps (assuming that C_2 is the circuit evaluator):

- Add the random shares of C_1 and C_2 (with an explicit modulo operation) to get $t_{i,k}$ and α as part of the circuit.
- Compare $t_{i,k}$ with α. It is important to note that the comparison result c is part of the circuit; therefore, not known to C_1 and C_2.
- Add r' to c (within the circuit followed by a modulo operation). The masked comparison result (i.e., $c + r' \bmod N$) is the final output of the circuit. Note that the circuit output should be known only to C_2 (i.e., the circuit evaluator).

After this, C_2 sends $E_{pk}(c + r')$ to C_1. Finally, C_1 removes the extra random factor using homomorphic operations to get $E_{pk}(c)$ locally.

In summary, given any predicate (where search input is randomly shared between C_1 and C_2) with relational operators $\{>, \geq, <, \leq\}$; C_1 and C_2 can securely compute the encryption of the predicate result on record t_i using either SEIP_h or SEIP_g, for $1 \leq i \leq n$. In general, which technique to use actually depends on the domain size of the attribute under consideration (more details are given in Sections 4.3 and 4.5). Similarly, C_1 and C_2 can securely evaluate the predicate with an equality operator. Once we know how to securely evaluate a given predicate, the next step is to securely combine the results of all predicates in Q and decide whether t_i satisfies Q. Along this direction, we next present a new two-stage protocol to solve the privacy-preserving complex query evaluation over encrypted data (PPQED) problem.

4.3 The Proposed PPQED Protocol

As mentioned in Section 1, this paper explicitly assumes that Bob's input query Q is represented in disjunctive normal form given by $G_1 \vee G_2 \vee \ldots \vee G_{l-1} \vee G_l$. Here G_j is a conjunction of b_j predicates given by $G_j = P_{j,1} \wedge P_{j,2} \wedge \ldots \wedge P_{j,b_j-1} \wedge P_{j,b_j}$.

We now propose a novel solution to the PPQED problem using Q as Bob's input query over encrypted data T' stored in C_1. At a high level, the proposed PPQED protocol consists of the following two stages:

- Stage 1 - Secure Evaluation of Predicates (SEP): In this stage, Bob initially sends his private query Q (using random shares) to C_1 and C_2. Then, C_1 and C_2 jointly

Algorithm 1. PPQED$(T', Q) \rightarrow S$

```
Require: $C_1$ has $T'$, $C_2$ has $sk$, and Bob has $Q$
1: Bob, for $1 \leq j \leq l$ do:
   (a). Send $P_j^{\langle 1 \rangle} = \{P_{j,1}^{\langle 1 \rangle}, \ldots, P_{j,b_j}^{\langle 1 \rangle}\}$ to $C_1$ and $P_j^{\langle 2 \rangle} = \{P_{j,1}^{\langle 2 \rangle}, \ldots, P_{j,b_j}^{\langle 2 \rangle}\}$ to $C_2$
2: for $1 \leq i \leq n$ do:
   (a). $C_1$ and $C_2$, for $1 \leq j \leq l$ do:
        – $L_{i,j}[h] \leftarrow \text{SEIP}\Big(\langle T_i', P_{j,h}^{\langle 1 \rangle}\rangle, P_{j,h}^{\langle 2 \rangle}\Big)$, where $\langle T_i', P_{j,h}^{\langle 1 \rangle}\rangle$ is the private input of $C_1$
          and $P_{j,h}^{\langle 2 \rangle}$ is the private input of $C_2$, for $1 \leq h \leq b_j$
   (b). $\text{SROD}_\text{s}(L_{i,1}, \ldots, L_{i,l})$, where $L_{i,j} = \langle L_{i,j}[1], \ldots, L_{i,j}[b_j]\rangle$ and $1 \leq j \leq l$
```

evaluate the predicates of each clause in Q using SEIP as a sub-routine. At the end of this stage, only C_1 knows the encryptions of the evaluation results of $P_{j,h}$'s on each data record t_i, i.e., $E_{pk}(P_{j,h}(t_i))$, for $1 \leq j \leq l$ and $1 \leq h \leq b_j$.
– Stage 2 - Secure Retrieval of Output Data (SROD): C_1 and C_2 compute $E_{pk}(Q(t_i))$ using the evaluation results on the individual predicates resulted from Stage 1. Then, Bob securely retrieves the output set S with the help of C_1 and C_2.

The main steps involved in the proposed PPQED protocol are given in Algorithm 1. Next, we discuss each stage of PPQED in detail.

Stage 1 - Secure Evaluation of Predicates (SEP). The key steps involved in Stage 1 are shown as steps 1 to 2(a) in Algorithm 1. To start with, as explained in the previous sub-section, Bob initially generates the random shares for each predicate in Q and sends them to C_1 and C_2. More specifically, given a predicate $P_{j,h}$, Bob sends $P_{j,h}^{\langle 1 \rangle} = (k_{j,h}, \alpha_{j,h}^{\langle 1 \rangle}, op_{j,h})$ and $P_{j,h}^{\langle 2 \rangle} = (\alpha_{j,h}^{\langle 2 \rangle}, op_{j,h})$ to C_1 and C_2, respectively, for $1 \leq j \leq l$ and $1 \leq h \leq b_j$. Here $\alpha_{j,h} = \alpha_{j,h}^{\langle 1 \rangle} + \alpha_{j,h}^{\langle 2 \rangle} \bmod N$ is the search input, $k_{j,h}$ is the attribute index to be searched, and $op_{j,h}$ is the relational operator of predicate $P_{j,h}$. Upon receiving the values, C_1 and C_2 jointly evaluate each predicate $P_{j,h}$ on T_i' using the SEIP solution discussed in the previous sub-section. Let the output be denoted by $L_{i,j}[h]$ which will be known only to C_1. Note that $L_{i,j}[h] = E_{pk}(P_{j,h}(t_i))$, where $P_{j,h}(t_i) = 1$ iff t_i satisfies $P_{j,h}$, and $P_{j,h}(t_i) = 0$ otherwise.

We emphasize that, depending on the domain size of the attribute in consideration, either SEIP_h or SEIP_g can be utilized in this step. As it will be clear in Section 4.5, for attributes with smaller domain size (e.g., Age attribute), SEIP_h gives better performance than SEIP_g. On the other hand, for attributes with larger domain sizes (e.g., Bank account numbers), SEIP_g is more efficient than SEIP_h. Hence, by conveniently choosing between homomorphic encryption (SEIP_h) and garbled circuit (SEIP_g) based solution depending on the underlying attribute domain size, our PPQED protocol takes advantage of both techniques and significantly improves the overall performance.

Stage 2 - Secure Retrieval of Output Data (SROD). Following from Stage 1, C_1 has the evaluation results (in encrypted form) for all the predicates in Q on each data record t_i. The goal of Stage 2 is to utilize these predicate results and compute the query evaluation result on t_i. Since $E_{pk}(P_{j,h}(t_i))$ is an encryption of either 0 or 1 and as Q is assumed to be in disjunctive normal form, a naive solution to compute $E_{pk}(Q(t_i))$ is by using secure multiplication (SMP) and secure bit-or (SBOR) protocols as sub-routines. More specifically, C_1 and C_2 can securely compute $E_{pk}(G_j(t_i))$ by applying the SMP protocol on $E_{pk}(P_{j,h}(t_i))$ as inputs, for $1 \leq j \leq l$ and $1 \leq h \leq b_j$. For example, consider the case of computing $E_{pk}(G_1(t_i))$. In this case, C_1 and C_2 initially compute $E_{pk}(P_{1,1}(t_i) \wedge P_{1,2}(t_i))$ by feeding $E_{pk}(P_{1,1}(t_i))$ and $E_{pk}(P_{1,2}(t_i))$ as inputs to the SMP protocol. The above result is fed as an input along with the next predicate result of G_1 to SMP and so on. At the end, C_1 has $E_{pk}(G_1(t_i)) = E_{pk}(P_{1,1}(t_i) \wedge \ldots \wedge P_{1,b_1}(t_i))$. After that, in a similar fashion, they compute $E_{pk}(Q(t_i))$ by applying the SBOR protocol on $E_{pk}(G_j(t_i))$ as inputs, for $1 \leq j \leq l$. We refer to the above basic solution as SROD_b. However, since its complexity grows linearly with the number of predicates in Q, we claim that SROD_b is not that efficient. More details regarding the complexities of SROD_b are given in Section 4.5.

To overcome this issue, we next propose an efficient approach to systematically aggregate predicate results (in encrypted form) to compute the corresponding query result on each data record t_i, where $1 \leq i \leq n$. We denote our approach by SROD_s (where the subscript 's' stands for summation). The main steps involved in SROD_s are shown in Algorithm 2. To start with, for each record T'_i, C_1 locally aggregates (in encrypted form) the evaluation results of predicates in each clause by computing $L'_{i,j} = \prod_{h=1}^{b_j} L_{i,j}[h] = E_{pk}\left(\sum_{h=1}^{b_j} P_{j,h}(t_i)\right)$, for $1 \leq j \leq l$.

Observation 1. *Since clause G_j is a conjunction of b_j predicates, a record t_i satisfies G_j, i.e., $G_j(t_i) = 1$, only if $P_{j,h}(t_i) = 1$, for $1 \leq h \leq b_j$. This further implies that $G_j(t_i) = 1$ only if $\sum_{h=1}^{b_j} P_{j,h}(t_i) = b_j$. In addition, if $\exists$ h such that $P_{j,h}(t_i) = 0$, then $G_j(t_i) = 0$ and $\sum_{h=1}^{b_j} P_{j,h}(t_i) < b_j$.*

Following from the above observation, in order to evaluate G_j, we need to securely check whether $\sum_{h=1}^{b_j} P_{j,h}(t_i)$ is equal to b_j or not. For this purpose, C_1 with input $L'_{i,j}$ and C_2 jointly involve in the SC_obv protocol (i.e., the extended version of the secure comparison protocol in [31] as discussed in Section 4.2). That is, C_1 and C_2 jointly check whether $\sum_{h=1}^{b_j} P_{j,h}(t_i)$ is greater than $b_j - 1$. If the comparison result is 1 (in encrypted form), then $\sum_{h=1}^{b_j} P_{j,h}(t_i) = b_j$. At the end of this step, the output $M_{i,j} = \text{SC}_\text{obv}(L'_{i,j}, b_j)$ will be known only to C_1. Remember that $M_{i,j} = E_{pk}(1)$ iff t_i satisfies G_j, and $E_{pk}(0)$ otherwise, for $1 \leq i \leq n$ and $1 \leq j \leq l$.

Once C_1 knows $E_{pk}(G_j(t_i))$, for $1 \leq i \leq n$ and $1 \leq j \leq l$, the goal is to compute the final evaluation result of Q on t_i (in encrypted form), i.e., $E_{pk}(Q(t_i))$. For this purpose, we use the following observation.

Observation 2. *Given any query Q which is a disjunction of l clauses, $Q(t_i) = 1$ only if $\exists$ j such that $G_j(t_i) = 1$. That is, if t_i satisfies at least one of the clauses in Q, then it also satisfies Q. This further implies that $Q(t_i) = 1$ only if $\sum_{j=1}^{l} G_j(t_i) > 0$. Furthermore, when $\sum_{j=1}^{l} G_j(t_i) = 0$, we have $Q(t_i) = 0$ (i.e., t_i does not satisfy Q).*

Algorithm 2. SROD$_s(L_{i,1}, \ldots, L_{i,l})$

```
Require: $C_1$ has $(L_{i,1}, \ldots, L_{i,l})$
 1: for $1 \le j \le l$ do:
      (a). $C_1$ compute $L'_{i,j} \leftarrow \prod_{h=1}^{b_j} L_{i,j}[h]$
      (b). $C_1$ and $C_2$: $M_{i,j} \leftarrow \text{SC}_{\text{obv}}(L'_{i,j}, b_j)$
 2: $C_1$ compute $M'_i \leftarrow \prod_{j=1}^{l} M_{i,j}$
 3: $C_1$ and $C_2$: $O_i \leftarrow \text{SC}_{\text{obv}}(M'_i, l)$
 4: $C_1$ send $O_i$ to $C_2$
 5: $C_2$ compute $x_i \leftarrow D_{sk}(O_i)$ and send $x_i$ to $C_1$
 6: $C_1$: if $x_i = 1$ then:
      (a). $V_{i,j} \leftarrow T'_{i,j} * E_{pk}(r'_{i,j})$, for $1 \le j \le m$, where $r'_{i,j}$ is a random number in $\mathbb{Z}_N$
      (b). Send $r'_{i,j}$ to Bob and $V_{i,j}$ to $C_2$
 7: $C_2$, foreach $V_i$ received do: $z_{i,j} \leftarrow D_{sk}(V_{i,j})$ and send $z_{i,j}$ to Bob
 8: Bob, foreach received entry pair $(z_i, r'_i)$ do:
      (a). $t'_j \leftarrow z_{i,j} - r'_{i,j} \bmod N$, for $1 \le j \le m$, and $S \leftarrow S \cup t'$
```

Based on the above observation, C_1 locally computes the encryption of sum of the evaluation results on l clauses in Q. That is, he/she computes $M'_i = \prod_{j=1}^{l} M_{i,j}$, for $1 \le i \le n$. It is important to observe that $M'_i = E_{pk}\left(\sum_{j=1}^{l} G_j(t_i)\right)$. After this, C_1 and C_2 securely verify whether the value of $\sum_{j=1}^{l} G_j(t_i)$ is greater than 0. For this purpose, they jointly involve in the SC$_{\text{obv}}$ protocol. Specifically, they together compute $O_i = \text{SC}_{\text{obv}}(M'_i, l) = E_{pk}(Q(t_i))$, for $1 \le i \le n$. Note that the output of SC$_{\text{obv}}$, i.e., O_i will be known only to C_1. Once C_1 computes the evaluation result of Q on a data record t_i (in encrypted form), the next step is for Bob to securely retrieve only those records that satisfy Q with the help of C_1 and C_2. We emphasize that there are many ways through which Bob can obliviously retrieve the output set of records from C_1. In this paper, we present a simple approach that is very efficient from the Bob's perspective.

For $1 \le i \le n$, C_1 sends O_i to C_2. Upon receiving, C_2 computes $x_i = D_{sk}(O'_i)$ and sends the result to C_1. After this, C_1 proceeds as follows:

- If $x_i = 1$, then $Q(t_i) = 1$. In this case, C_1 randomizes T'_i attribute-wise and sends $V_{i,j} = T'_{i,j} * E_{pk}(r'_{i,j})$ to C_2, where $r'_{i,j}$ is a random number in $\mathbb{Z}_N$, for $1 \le j \le m$. Here m denotes the number of attributes in T'. Also, C_1 sends $r'_{i,j}$ to Bob.
- Else, ignore the data record corresponding to the entry x_i.

Upon receiving the entry V_i, C_2 computes $z_{i,j} = D_{sk}(V_{i,j})$, for $1 \le j \le m$, and sends the result to Bob. Then, for each received entry pair (z_i, r'_i), Bob removes the extra random factors attribute-wise to get $t'_j = z_{i,j} - r'_{i,j} \bmod N$, for $1 \le j \le m$. Based on the above discussions, it is clear that t' will be a data record in T that satisfies the input query Q. Finally, Bob adds the data record t' to the output set: $S = S \cup t'$.

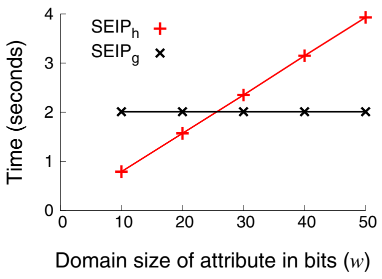

Fig. 1. Computation costs of SEIP$_h$ and SEIP$_g$ for encryption key size 1024 bits

4.4 Security Analysis of PPQED

First of all, since Bob's input query Q is randomly shared between C_1 and C_2, search input values in each predicate are never disclosed to C_1 and C_2. However, the PPQED protocol reveals the attribute index to be searched in each predicate to C_1 (but not C_2) for efficiency reasons. Nevertheless, we stress that such information will not be useful for C_1 to deduce any meaningful information about the attribute values.

We emphasize that the SC$_{obv}$ protocol which is used as a building block in PPQED is secure under the semi-honest model (security proof follows directly from [31]). As a result, the secure evaluation of individual predicates (SEIP) task at step 2(a) of Algorithm 1 do not reveal any information to C_1 and C_2. Thus, Stage 1 of PPQED is secure under the semi-honest model. During Stage 2, all the intermediate results revealed to C_2 are either random or pseudo-random. On the other hand, the intermediate results seen by C_1 are always in encrypted form (except at step 5 of Algorithm 2). Thus, whatever messages C_1 and C_2 receive in Stage 2 are computationally indistinguishable (assuming large key size, say 1024 bits) from random numbers in $\mathbb{Z}_N$. Therefore, Stage 2 is secure under the semi-honest model. Also, the outputs of Stage 1 which are passed as input to Stage 2 are also in encrypted format. According to the Composition Theorem [15], we claim that the sequential composition of Stages 1 and 2 leads to our PPQED protocol that is also secure under the semi-honest model. In short, the proposed PPQED protocol protects the confidentiality of the data as well as privacy of the user's input query. At the same time, it supports evaluation of complex queries over encrypted data.

In the proposed PPQED protocol, C_1 and C_2 know the value of x_i (at step 5 of Algorithm 2) that can reveal the data access pattern information (i.e., whether the data record t_i satisfies the input query) to C_1 and C_2. Nevertheless, our protocol can be easily extended to hide the data access patterns from both C_1 and C_2 at an additional cost. Due to space limitations, we do not go into any details regarding this extension. However, we refer the reader to our technical report [14] for more details on how to hide the access pattern information in the proposed protocol.

4.5 Complexity Analysis of PPQED

During Stage 1 of PPQED, the computation cost of the federated cloud (i.e., the combined cost of C_1 and C_2) is bounded by $O(n * l * s)$ instantiations of SEIP, where s

denotes the upper bound on the number of predicates in each clause. As mentioned earlier, one can use either SEIP$_\text{h}$ (i.e., SEIP under homomorphic encryption) or SEIP$_\text{g}$ (i.e., SEIP under garbled circuits) in Stage 1. To get more insights, we implemented both SEIP$_\text{h}$ and SEIP$_\text{g}$, and ran experiments on a Linux machine with an Intel® Xeon® Six-Core$^\text{TM}$ CPU 3.07 GHz processor and 12GB RAM running Ubuntu 10.04 LTS. We use the Paillier cryptosystem [13] and fix the encryption key size K to 1024 bits. In particular to SEIP$_\text{g}$, we constructed and evaluated the circuit under the FastGC [34] framework (the fastest known implementation for garbled circuits). We considered attributes of different domain sizes (in bits), denoted by w, and executed predicates at random using both SEIP$_\text{h}$ and SEIP$_\text{g}$. The results are shown in Figure 1. Note that, in PPQED, we consider predicates with relation operators $\{>, \geq, <, \leq\}$. Since the underlying operations are almost the same for all these four relational operations, the computation costs reported for SEIP$_\text{h}$ and SEIP$_\text{g}$ remain the same for any of these relational operators.

Following from Figure 1, the computation cost of SEIP$_\text{h}$ increases from 0.79 to 3.93 seconds when w varies from 10 to 50. On the other hand, the computation cost of SEIP$_\text{g}$ almost remains constant at 2.01 seconds. This is because the computation cost of SEIP$_\text{g}$ mainly depends on the addition circuits (operating over 1024 bits) whose costs remain the same for any fixed encryption key size. From the above results, we conclude that SEIP$_\text{h}$ is more efficient than SEIP$_\text{g}$ for attributes with domain size 2^{25} (i.e., $w = 25$). Note that the attribute domain size $[0, 2^{25})$ is realistic for most practical applications, e.g., the attribute domain size for Age, Annual Salary, and Temperature is less than 25 bits. Also, the categorical attributes usually take few values. However, if the domain size of an attribute is $\geq 2^{25}$, we would use SEIP$_\text{g}$ in PPQED for efficiency reasons. Our experimental results given in Figure 1 are based on one record and it is important to note that these results are independent of the record. Thus the reported costs remain the same for any given data record. As a result, the cloud providers can evaluate a predicate on multiple data records in parallel.

Next, for Stage 2, we analyze the costs associated with the query evaluation step (using the individual predicate results) in SROD$_\text{s}$ and compare its performance with the basic solution SROD$_\text{b}$. For any given data record t_i, the complexities of SROD$_\text{b}$ and SROD$_\text{s}$ are shown in Table 2. From Table 2, it is clear that our approach in SROD$_\text{s}$ outperforms (in terms of both computations and communications) SROD$_\text{b}$ if s is large. Also, the round complexity of both approaches is bounded by $O(\log_2 l + \log_2 s)$. However, if the round complexity is crucial in an application, one can replace SC$_\text{obv}$ in SROD$_\text{s}$ with the SC protocol based on garbled circuits [35] (which takes one round of communication to perform the secure comparison). However, for practical values of l and s, SC$_\text{obv}$ is more efficient than [35], thus providing a trade-off between efficiency and round complexity. Due to space limitations, we refer the reader to our technical report [14] for a more elaborated theoretical and empirical analysis of PPQED.

Nevertheless, the main advantage of the proposed PPQED protocol is that the computation cost on Bob is negligible. This is especially beneficial if Bob issues queries using a resource-constrained device (e.g., PDAs and cell phones).

Table 2. SROD_b vs. SROD_s for any given record t_i

Method	Computations	Communications
SROD_b	$O(l * s)$ encryptions	$O(K * l * s)$ bits
SROD_s	$O(l * \log_2 s)$ encryptions	$O(K * l * \log_2 s)$ bits

5 Conclusion and Future Work

In this paper, we proposed a novel protocol to securely evaluate complex queries over encrypted data in the cloud. The core of our protocol is based on a hybrid approach to evaluate the predicates in the user's query using both homomorphic encryption and garbled circuit techniques. Also, we developed an efficient approach to systematically combine the evaluation results of individual predicates to compute the corresponding query evaluation result. Our protocol protects data confidentiality, privacy of the user's input query and access patterns. Our empirical results show that techniques based on homomorphic encryption are efficient for attributes of smaller domain sizes. Also, we theoretically demonstrated the efficiency of our systematic approach to combine the predicate results.

As future work, we will implement and evaluate our framework using the MapReduce technique in a real federated cloud computing environment. We also plan to develop a sequence diagram for the proposed protocol in our future work. Another interesting direction is to extend our protocol to other adversary models, such as the malicious model, and evaluate the trade-offs between security and efficiency. Though our protocol concentrates on the relational operators, we believe that it can also support other SQL operations, such as JOIN and GROUP BY, as they are essentially based on the relational operations. We plan to investigate this problem in our future work.

Acknowledgements. The authors would like to thank the anonymous reviewers for their helpful comments. The work reported in this paper has been partially supported by the Purdue Cyber Center and by the National Science Foundation under grants CNS-1111512, CNS-1016722, and CNS-1011984.

References

1. Bajaj, S., Sion, R.: Trusteddb: a trusted hardware based database with privacy and data confidentiality. In: ACM SIGMOD, pp. 205–216 (2011)
2. Agrawal, R., Kiernan, J., Srikant, R., Xu, Y.: Order preserving encryption for numeric data. In: ACM SIGMOD, pp. 563–574 (2004)
3. Mykletun, E., Tsudik, G.: Aggregation queries in the database-as-a-service model. In: Damiani, E., Liu, P. (eds.) Data and Applications Security 2006. LNCS, vol. 4127, pp. 89–103. Springer, Heidelberg (2006)
4. Shi, E., Bethencourt, J., Chan, T.H.H., Song, D., Perrig, A.: Multi-dimensional range query over encrypted data. In: IEEE Security & Privacy, pp. 350–364. IEEE Computer Society (2007)

5. Chung, S., Ozsoyoglu, S., Anti-tamper, G.: Anti-tamper databases: Processing aggregate queries over encrypted databases. In: ICDE Workshops, p. 98 (2006)
6. Boldyreva, A., Chenette, N., Lee, Y., O'Neill, A.: Order-preserving symmetric encryption. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 224–241. Springer, Heidelberg (2009)
7. Hore, B., Mehrotra, S., Canim, M., Kantarcioglu, M.: Secure multidimensional range queries over outsourced data. The VLDB Journal 21(3), 333–358 (2012)
8. Samanthula, B.K., Jiang, W.: Efficient privacy-preserving range queries over encrypted data in cloud computing. In: IEEE CLOUD, pp. 51–58 (2013)
9. Williams, P., Sion, R., Carbunar, B.: Building castles out of mud: practical access pattern privacy and correctness on untrusted storage. In: CCS, pp. 139–148. ACM (2008)
10. De Capitani di Vimercati, S., Foresti, S., Samarati, P.: Managing and accessing data in the cloud: Privacy risks and approaches. In: 7th International Conference on Risk and Security of Internet and Systems, pp. 1–9 (2012)
11. Islam, M., Kuzu, M., Kantarcioglu, M.: Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In: NDSS (2012)
12. Goldreich, O.: Encryption Schemes. In: The Foundations of Cryptography, vol. 2, pp. 373–470. Cambridge University Press, Cambridge (2004)
13. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, p. 223. Springer, Heidelberg (1999)
14. Samanthula, B.K., Jiang, W., Bertino, E.: Privacy-preserving complex query evaluation over semantically secure encrypted data. Technical Report TR 2014-05, Dept. of Computer Science, Missouri S&T, Rolla (2014), `http://web.mst.edu/~wjiang/PPQED.pdf`
15. Goldreich, O.: General Cryptographic Protocols. In: The Foundations of Cryptography, vol. 2, pp. 599–746. Cambridge University Press, Cambridge (2004)
16. Boneh, D., Waters, B.: Conjunctive, subset, and range queries on encrypted data. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 535–554. Springer, Heidelberg (2007)
17. Hacıgümüş, H., Iyer, B., Mehrotra, S.: Efficient execution of aggregation queries over encrypted relational databases. In: Lee, Y., Li, J., Whang, K.-Y., Lee, D. (eds.) DASFAA 2004. LNCS, vol. 2973, pp. 125–136. Springer, Heidelberg (2004)
18. Golle, P., Staddon, J., Waters, B.: Secure conjunctive keyword search over encrypted data. In: Jakobsson, M., Yung, M., Zhou, J. (eds.) ACNS 2004. LNCS, vol. 3089, pp. 31–45. Springer, Heidelberg (2004)
19. Popa, R.A., Redfield, C.M.S., Zeldovich, N., Balakrishnan, H.: Cryptdb: Protecting confidentiality with encrypted query processing. In: SOSP, pp. 85–100. ACM (2011)
20. Shi, E., Chan, T.-H.H., Stefanov, E., Li, M.: Oblivious RAM with $o((\log n)^3)$ worst-case cost. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 197–214. Springer, Heidelberg (2011)
21. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC, pp. 169–178. ACM (2009)
22. Naehrig, M., Lauter, K., Vaikuntanathan, V.: Can homomorphic encryption be practical? In: The ACM Workshop on Cloud Computing Security, pp. 113–124. ACM (2011)
23. Gentry, C., Halevi, S.: Implementing gentry's fully-homomorphic encryption scheme. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 129–148. Springer, Heidelberg (2011)
24. Yao, A.C.: Protocols for secure computations. In: SFCS, pp. 160–164. IEEE Computer Society (1982)
25. Yao, A.C.: How to generate and exchange secrets. In: SFCS, pp. 162–167. IEEE Computer Society (1986)
26. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game - a completeness theorem for protocols with honest majority. In: STOC, pp. 218–229. ACM (1987)

27. Chaum, D., Crépeau, C., Damgard, I.: Multiparty unconditionally secure protocols. In: STOC, pp. 11–19. ACM (1988)
28. Damgard, I., Geisler, M., Kroigard, M.: Homomorphic encryption and secure comparison. International Journal of Applied Cryptography 1(1), 22–31 (2008)
29. Bugiel, S., Nürnberger, S., Sadeghi, A.R., Schneider, T.: Twin clouds: An architecture for secure cloud computing (extended abstract). In: Workshop on Cryptography and Security in Clouds (March 2011)
30. Wang, J., Ma, H., Tang, Q., Li, J., Zhu, H., Ma, S., Chen, X.: Efficient verifiable fuzzy keyword search over encrypted data in cloud computing. Computer Science and Information Systems 10(2), 667–684 (2013)
31. Blake, I.F., Kolesnikov, V.: One-round secure comparison of integers. Journal of Mathematical Cryptology 3(1), 37–68 (2009)
32. Schoenmakers, B., Tuyls, P.: Efficient binary conversion for paillier encrypted values. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 522–537. Springer, Heidelberg (2006)
33. Samanthula, B.K., Jiang, W.: An efficient and probabilistic secure bit-decomposition. In: ACM ASIACCS, pp. 541–546 (2013)
34. Huang, Y., Evans, D., Katz, J., Malka, L.: Faster secure two-party computation using garbled circuits. In: Proceedings of the 20th USENIX Conference on Security, pp. 35–35 (2011)
35. Kolesnikov, V., Sadeghi, A.-R., Schneider, T.: Improved garbled circuit building blocks and applications to auctions and computing minima. In: Garay, J.A., Miyaji, A., Otsuka, A. (eds.) CANS 2009. LNCS, vol. 5888, pp. 1–20. Springer, Heidelberg (2009)