Two-Party Computation Model for Privacy-Preserving Queries over Distributed Databases

Sherman S.M. Chow Jie-Han Lee Lakshminarayanan Subramanian

Department of Computer Science
Courant Institute of Mathematical Sciences
New York University, NY 10012, USA
{schow, jhl406, lakshmi}@cs.nyu.edu

Abstract

Many existing privacy-preserving techniques for querying distributed databases of sensitive information do not scale for large databases due to the use of heavyweight cryptographic techniques. In addition, many of these protocols require several rounds of interactions between the participants which may be impractical in wide-area settings. At the other extreme, a trusted party based approach does provide scalability but it forces the individual databases to reveal private information to the central party.

This paper shows how to perform various privacy-preserving operations in a scalable manner under the honest-but-curious model. Our system provides the same level of scalability as a trusted central party based solution while providing privacy guarantees without the need for heavyweight cryptography. The key idea is to develop an alternative system model using a Two-Party Query Computation Model *comprising of a randomizer and a computing engine which do not reveal any information between themselves. We also show how one can replace the randomizer by a lightweight key-agreement protocol. We formally prove the privacy-preserving properties of our protocols and demonstrate the scalability and practicality of our system using a real-world implementation.*

1 Introduction

Over the past decade, there has been a growing need for large-scale privacy-preserving systems spanning several databases distributed over the Internet. One motivating example is the nation-wide electronic medical records (EMR) effort within the US which hopes to integrate the EMR of patients across a large number of hospitals while mandating stringent privacy requirements for patient records as specified in the HIPAA regulations [14].

Over the years, the research community has developed a wide range of privacy-preserving techniques for answering different types of queries [3, 4, 13, 23, 24, 27] without revealing information of any individual database which is irrelevant to the queries[1]. While many of these techniques offer strong privacy guarantees, they do not scale well for large databases and wide-area systems. The underlying reasons are three-fold. First, it is well known that to support privacy-preserving queries across different parties by *generic* secure multi-party computations (SMC) of any function, the function should be represented as a combinational circuit, which means it is impossible to perform arbitrary functions with linear communication overhead [18]. Second, other privacy-preserving techniques [23, 24, 27] which realize secure multi-party computations of some *specific* operation also rely on heavyweight cryptographic operations (e.g. zero-knowledge proof of knowledge and homomorphic encryption for each data value) to ensure malicious parties not to deviate from the protocol specification. Third, many techniques require [23, 24, 27] *multiple rounds of interactions* between the individual entities which is impractical in wide-area settings. The high wide-area latencies is a fundamental limiting factor that affects the scalability and the query processing time in wide-area distributed databases as observed in prior work such as Mariposa [31]. SMC protocols, and P2P approaches [13] which leverage secret sharing techniques, also suffer from the same multi-round interaction problem.

1.1 Motivation: The RHIO Initiative

In the nationwide EMR effort, the role of a Regional Health Information Organization (RHIO) is to integrate the

[1]In this paper, we do not consider protecting the privacy of the query from the databases, which can be achieved by private information retrieval.

databases of various hospitals in a geographic region in a privacy-preserving manner. Given the scalability concerns, many of the RHIOs have resorted to a simple trusted party based system for implementing electronic medical records. In the paradigm of trusted third party (TTP) computation service [5, 22], all data owners provide their data to a TTP to perform the computation. However, in reality, data owners are mutually distrustful and a TTP based solution is far from ideal since the trusted party has complete knowledge of the individual databases.

In our own experiences, our university medical center leads a large initiative called New York Clinical Information Exchange (NYCLIX) [28] to establish a RHIO across 15 leading hospitals within the New York region. While the primary developers of the RHIO system are well aware of different privacy-preserving technologies, they chose a TTP-based approach primarily due to the lack of other scalable alternatives. This partly motivated us to investigate the problem of performing scalable privacy-preserving operations in a distributed database environment.

1.2 Our Contributions

In this paper, we present a *Two-Party Query Computation model* for performing privacy-preserving operations in a distributed database environment assuming an honest-but-curious adversarial model. Many of the existing research on privacy-preserving techniques assume a more powerful adversarial model where nodes may arbitrarily deviate from the underlying protocol to gain as much advantage as possible to reveal other parties' data. As a result, the proposed protocols against powerful adversaries are often too inefficient to be used for very large databases. In reality, nodes that require privacy guarantees are *honest* (e.g. would not inject false values into the database) but at most *curious* about other nodes' data. We thus weaken the security requirements for practical efficiency. In this simplified adversarial model, we show that one can perform *scalable* privacy-preserving operations. Indeed, any real-world deployment which is willing to make a stronger assumption that a TTP exists is automatically viable in our model.

We show that our Two-Party Query Computation model provides the same level of scalability as a trusted party based system but with strong privacy guarantees. The two-party model in essence emulates a central party but with two important modifications. First, all the operations by the two parties are performed on short encoded data and not on real data. Second, the central party computational functionality is split across two entities. While the basic model assumes that these two parties cannot collude with each other, we show how to relax that assumption by using a key agreement protocol across the participating entities. We formally prove the privacy guarantees of our protocol in the random oracle model [7]. We also demonstrate the scalability of our techniques using a real-world implementation.

2 Query Computation Models

There has been a wide range of techniques for privacy-preserving queries in distributed database settings which offer different tradeoffs along the security and functionality axes. At a high-level, we classify these works into four categories: (a) Secure Multi-party Computation; (b) Deterministic Encryption; (c) Trusted Computing and (d) P2P Model. Table 1 compares our proposal with these models across six dimensions: (a) Need for encryption; (b) Need for decryption; (c) Need for dedicated hardware; (d) Network assumption; (e) Trust assumption and (f) Efficiency. The network assumption indicates if the solution requires some particular type of network, some kind of network hierarchy or information about the network. Similarly, we use the term trust assumption to indicate whether the security is relied on the assumption that a certain party is trusted not to *misbehave*. The meaning of misbehavior varies among different proposals; some proposals use a stronger assumption than others do. Computational requirements suggest whether encryption and decryption is needed for *each* record. Most efficiency comparisons in the table are self-evident. Note that in the trusted computing approach [3, 26], decryption is done by the secure co-processor but not the databases. We elaborate on some of the important related work in detail.

Secure Multi-party Computation. A recent trend to realize secure multi-party computation for specific privacy-preserving set operations [16, 23] can be viewed as a combination of two main ideas – polynomial representation of sets [16] and additively homomorphic cryptosystem [29].

Freedman *et al.* [16] proposed a protocol for two-party set-intersection. One party constructs a polynomial which has an irreducible polynomial factor and zeros at all data values. This polynomial is encrypted by encrypting its coefficients using homomorphic encryption. The other party makes use of these ciphertexts to perform an oblivious evaluation of this polynomial function at every data value he has, with some randomization. These results are sent back to the first party for decryptions. When decryptions give a zero, it can be concluded that their data sets intersect. They also extended their basic protocol for security against malicious adversaries, assuming a random oracle, but it is unclear how to extend it for more than two parties under the same adversary model [23]. Kissner-Song's multi-party protocol [23] proceeds by randomly selecting polynomials of the same degree as the polynomial representing the data values and encrypting some combinations of these polynomials to other data owners. At last, all parties jointly perform a decryption (i.e. all parties' secret keys are involved)

	Computation-Free		Assumption-Free			Efficiency
	Encryption	Decryption	Any Hardware	Any Network	No Trust	
Multi-party Computation	✘	✘	✓	✓	✓	Poor
Deterministic Encryption	✘	✓	✓	✓	✘	Moderate
Trusted Computing	✘	✘	✘	✓	✓	Moderate
P2P Model	✓	✓	✓	✘	✘	Good
Our Proposal	✓	✓	✓	✓	✘	Efficient

Table 1. Properties of related paradigms

to get the final result. The work of Kissner and Song [23] is not only about intersection but also other set operations.

Recently, Hazay and Lindell [21] proposed a new approach for privacy-preserving set-intersection. Instead of oblivious polynomial evaluation, oblivious pseudorandom function (PRF) evaluation [15] is used. The protocol proceeds as follow. One party who holds the set X chooses a secret PRF key k at random which defines a PRF $F_k(\cdot)$. Both engages in the oblivious PRF evaluation protocol such that the other party who holds the set Y gets the set $\{F_k(y)\}_{y \in Y}$ as the outcome of the protocol (and the first party learns nothing). Then, the first party locally computes $\{F_k(x)\}_{x \in X}$ and sends it to the second party. The second party can thus deduce the intersection by checking which elements appeared in both sets. Hazay-Lindell's protocol is more efficient than the polynomial based approach [23], but only provides security against malicious adversaries under a weaker definition ("one-sided simulatable"). The protocol can be modified to be fully simulatable against covert adversaries, which basically means a malicious adversary can cheat, but will be caught with good probability.

Weaknesses. For polynomial evaluation based approach, since the homomorphic property is crucial, public key encryption should be used. All tuples are needed to construct the polynomial, which requires a memory space in the order of the magnitude of the database size. To make the protocol secure against malicious adversaries, general (and hence inefficient) zero-knowledge proofs [19, 20] or inefficient cut-and-choose protocols (which run at bit-level) are needed. Hazay-Lindell's protocol also utilizes homomorphic encryption for oblivious transfer. The protocol requires $|Y|$ oblivious PRF evaluation, and each of them requires 4ℓ modular exponentiations, where ℓ is the bit-length of each element in the set. All these heavyweight operations are impractical for very large databases in wide-area settings.

Deterministic and Searchable Encryption. The idea of efficiently searchable encryption (ESE) was introduced recently in [6]. Efficiency here means an untrusted server can index, retrieve or update the encrypted data on request just as efficiently as if the data is unencrypted. ESE is a public key (asymmetric) encryption. Its basic idea is to use a deterministic encryption (with some additional property, refer to [6] for the technicalities) to encrypt the data, which makes a given plaintext always will be encrypted to the same ciphertext. If the ciphertext of distinct messages under a given public key rarely coincide, indexing the ciphertext is essentially the same as on unencrypted ones.

ESE is proposed for the outsourced database model, but not for our privacy-preserving operations. However, one may apply ESE in our scenario if a central party is assumed to join the databases faithfully based on the encrypted data.

Weaknesses. As noted in [6], a small plaintext space means offline dictionary attack is possible, one can test if a ciphertext corresponding to a given plaintext by a public key encryption, which can be done by anyone, including the central party. This weakness [11] also appears in some query processing systems over encrypted data [8]. So the security of ESE is based on an additional assumption that the plaintext space has high entropy, which is not (necessary) true in equijoin, where only primary keys are encrypted.

Commutative Encryption. The idea of using deterministic encryption also appear in prior work in privacy-preserving operations outside the outsourced database model (e.g. a protocol between two data owners), without the formal security treatment of the encryption as in [6]. Agrawal *et al.* [4] proposed an intersection protocol for two parties which uses commutative encryption ($E_{k_2}(E_{k_1}(m)) = E_{k_1}(E_{k_2}(m))$ where $E_k(m)$ is the encryption of the message m under the key k). The encryption scheme is deterministic and is realized by a modular exponentiation. Despite of the use of modular arithmetic, it is a symmetric encryption scheme. Again, due to the deterministic nature, multiple runs of the protocol may leak partial information about different queries.

Homomorphic Encryption for Addition/Multiplication. Another special class of encryption is homomorphic public key encryption. Consider an encryption function E under a certain public key, additive homomorphism means $E(m_1) + E(m_2) = E(m_1 + m_2)$. Paillier encryption [29] is a famous example. Homomorphic encryption schemes enable a simple privacy-preserving protocol for distributed addition. Different data sources can encrypt their values to be added under the public key of the one who made the query (the querier). Addition can be done by adding those ciphertexts, which can be done without the private key. The aggregated ciphertext is then sent to the querier. Roughly speaking, no one learns about the individual value if the individual ciphertexts are not leaked to the querier.

Weaknesses. The existence of both additive and multiplicative homomorphic encryption does not mean that we can do arbitrary arithmetic operations on the ciphertext. The first issue is about floating point division, which is not possible in group arithmetic. More importantly, devising a practical doubly homomorphic scheme, where one can both add and multiply ciphertexts, is still an open problem. Nevertheless, Boneh *et al.* [9] solved a special case of the problem. Their scheme uses bilinear groups of composite order to support quadratic multi-variate polynomials, i.e. ciphertexts can be multiplied once.

Trusted Computing. One recent trend is to use secure coprocessor (SC) to realize the functionalities provided by SMC. In the work by Agrawal *et al.* [3], privacy-preserving join operation is done with the help of a SC. Semantic-secure (probabilistic) encryption is done by the database on the data value. The SC decrypts them and performs the matching. With the original data recovered, the matching thus can be based on any arbitrary function other than a simple equality check or computation of degree-2 polynomials, in contrast with the many other approaches.

Weaknesses. SC is used to realize the assumption of faithful computation and the ideal functionalities of computing a function with the inputs and any intermediate values kept private from anyone. However, from a computational standpoint, it is much more expensive to implement cryptographic operations (e.g. decryption) in the SC in comparison to simple database indexing operations: for decrypting millions of records, the overall system performance will significantly reduce. In addition, the cost of tamper-proof memory also imposes a practical limitation on the type of algorithm that can be performed by the SC.

P2P Model. Emekci *et al.* [13] proposed a hash-based system which assumes the existence of a peer-to-peer (P2P) overlay network (e.g. Chord [30]) for computation of query results. They use a secret sharing protocol between the nodes to aid in query computation. Using a hashing-based query computation approach is definitely much less computationally intensive that other encryption-based approaches.

Weaknesses. However, their system does face certain security threats and scalability concerns. In a typical P2P network, it is almost certain that the number of online nodes is smaller than the size of typical cryptographic parameters, which makes their system exposed to two vulnerabilities. First, despite the countermeasures suggested in [13], the adversary can freely choose the helper nodes from the P2P network by being the last one in the protocol for selection. Second, the probability for false positive is non-negligible (with respect to the security parameter of the cryptographic algorithms) since the equation for matching just depends on the domain size of the node identifier.

From an efficiency standpoint, their protocol does require several rounds before computing a query output. Also, to do a single query computation in a P2P model would also involve shipping large volumes of data multiple times across the network. In addition, a query computation requires multiple agreement protocols on the coefficients in polynomial secret sharing among databases, which are neglected in [13] (otherwise the protocol will not be secure, again due to the small domain size of the coefficients), takes a significant amount of time. Finally, a P2P network is not available in many cases. We refer to this as the "network assumption" in Table 1. In critical applications like queries on medical data, nodes may not wish to participate in a P2P network to exchange encoded data streams with other unknown nodes.

3 Two-Party Query Computation Model

We outline the basic two-party query computation model in this section and later describe the types of privacy-preserving protocols that can be supported on top of this model in Section 4.

Figure 1 illustrates the basic two-party query computation model comprising of four different entities: the randomizer, the computing engine, the query front end engine and the individual databases. The two primary query computation entities in the system are the *randomizer* and the *computing engine*. The query front end engine which receives queries from different users forwards each query to the randomizer and an encoded version of the query (which contains the type of the query) to the computing engine which in turn coordinate with the individual databases to compute the query result. Our model assumes that all entities in the system require strong privacy guarantees but act in an *honest but curious* manner. In other words, every participating entity acts in an "honest" fashion and follows the protocol specification, but is "curious" to infer the entries of other participating databases.

Given this model, the basic steps in our query computation process are illustrated in Figure 1. The randomizer upon receiving a query, forwards the query to each individual database along with a set of *randomization parameters*. The randomizer also provides an essential set of the *derandomization parameters* to the query front end (which the querier may use to encode the query in case the selection predicate is based on a certain computation of the

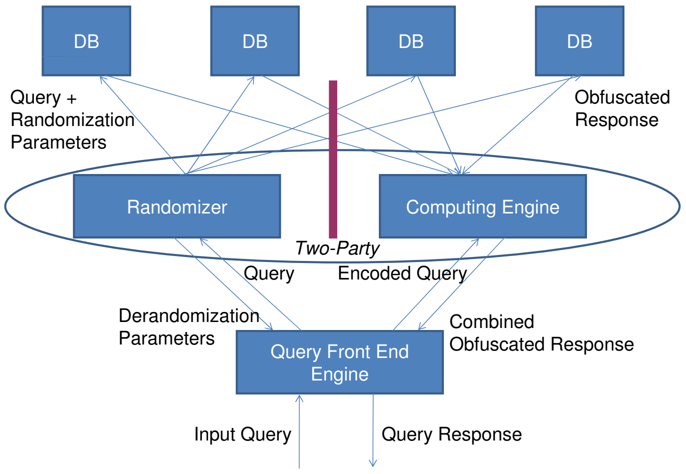


Figure 1. Our System Model

distributed data, and should be protected). Every database independently computes the local query response for the query and then *obfuscates* the query response using the randomization parameters. According to the query type, the computing engine performs the query computation by "combining" the individual obfuscated responses from the individual databases and produces an obfuscated response to the query. The query front end then makes use of the *derandomization parameters* to *deobfuscate* the query response from the computing engine.

We make a simplifying assumption that all the databases share the same schema which is also known to the querier; in practice, even if the individual schemas differ, the query processing engine at the individual databases can convert their individual results to a common schema.

3.1 Threat Model and Trust Assumptions

Our threat model assumes that all databases (DBs) do not trust each other and that no information from one set of data should be learned by another data owner. We also require that the computing engine cannot learn anything about the underlying data, the exact query (for examples, the selection conditions are protected but the computing engine must know the type of the queries to response accordingly) and its result. We make the following trust assumptions.

1. All DBs are honest but curious.
2. For correctness of the result, the computing engine will perform the algorithm faithfully.
3. For confidentiality of the intermediate result, the computing engine will not collude with any of DBs. Moreover, there is no communication between the randomizer and the computing engine.
4. The querier can encrypt to the randomizer, the randomizer can encrypt to all DBs (as a whole, not necessary individually), and all DBs can encrypt to the querier. These can be easily realized if the randomizer, the DBs, and the querier have authenticated public keys.
5. To avoid replay attack and the correctness of the end result received by the querier, these ciphertexts should contain a cryptographic checksum, e.g. by message authentication code, such that any outsider adversary cannot inject arbitrary messages to the protocol.
6. The querier is distinct from the computing engine, and it cannot collude with any of the DBs.[2] This can be realized by restricting the access to the querier terminal, which also conforms to regulations, e.g. HIPAA [14] requires that access to hardware and software should be limited to properly authorized individuals.

We acknowledge that the security of our scheme hinges on the primary requirement that the randomizer and the computing engine do not communicate. Later in Section 5, we describe cryptographic ways which will weaken such an assumption or even remove the need for the randomizer.

3.2 Comparison to Other Two-Party Models

The concept of utilizing two non-colluding parties to protect the privacy of the database is not entirely new. For

[2]Such collusion means one DB is allowed to query another DB, which contradicts our goal that one DB cannot learn any data from another DB.

secure database services which outsource the data management to untrusted server while preserving data privacy, Aggarwal *et al.* [1] proposed a two-party *storage* model. A data owner partitions the data into two shares according to the privacy constraint, and stores each share in different untrusted storages that cannot communicate with each other.

While both the two-party computation model and two-party storage model assume two non-colluding parties, there are differences in how the distributed data is generated, processed and stored. In our case, the data is generated by the respective data owner independently, except that they may share a similar schema (consider our motivating scenario, a patient may visit more than one hospital and hence has record stored in two databases). The data is still stored in the owners' database but may be sent to the two-party for coordinated processing. In two-party storage model, there is only one logical entity which created the data, then splits and distributes the data to different database systems for storage, i.e. the data is not stored with the data owner. All these shares of data are originated from a single entity, and their generations are coordinated.

Apart from the difference in who stores the data, who issues the query and who processes the query results from different data sources are also different. In our model, the querier is someone who does not own the database but nevertheless interested and entitled to extract some information from the distributed database (for examples, department of health and census bureau). The job of coordinating the responses from each database is performed by the two-party. In the two-party storage model, the querier is actually the data owner (who no longer stores the data), partial query results are given by the two-party after executed the query over their databases, and it is the querier who needs to coordinate the response from each database.

When the assumption breaks, says the two parties collude or one breaks into the other, the security implications of these models are also different. In the two-party storage model, since the whole database with privacy data can be recovered by integrating the two databases, the consequence may be as disastrous as causing the entire system broken. In our model, only the privacy related to the query issued during the breakdown is affected. For our intersection protocol, this only enables the adversary to launch an offline dictionary attack, but not a direct exposure of all related data.

3.3 Comparison to Other Computation Models

The primary motivation for the two-party query computation model is to develop a system that scales as well as the trusted central party system while providing privacy guarantees as SMC-based approach. In designing our query computation model, we show that it is possible to achieve this scalability objective if we explicitly assume a weaker adversary model where every database is honest-but-curious.

The underlying reason for the inefficiency of the SMC protocols is due to the security guarantees – strong security inherently comes with high complexities. In reality, we may be comfortable with some reasonable assumptions and are willing to sacrifice some theoretical security guarantees. For instance, the security model for SMC assumes the adversary even knows the private keys of all parties whom want to protect their data. This is actually desirable since many kinds of partial information about the private keys may leak during the execution of the protocol (e.g. the corresponding public key, the decryption of ciphertexts using the private key). However, it also means we cannot leverage the secrecy of private key to do something more efficient, e.g. using a pseudorandom function as a source of the randomness for privacy. Besides, we tend to assume that the private key cannot be recovered from the public key with "limited" computation power. Indeed, other approaches like Hazay-Lindell's protocol [21] and deterministic encryption can also be seen as attempts to weaken security requirements for better efficiency[3].

This honest-but-curious assumption holds in many real world settings especially if the underlying system is developed and operated under the governance of law. For instance, it is ideal in the RHIO setup where multiple hospitals wish to establish an EMR system between them. In this case, the hospitals are always "honest" and they would not randomly inject false data to the databases. Law enforcement agencies and the hospitals can together establish an institution which manages two different trusted parties to play the role of randomizer and computing engine. As part of the initial setup, a policy can be setup to make the two parties non-communicating with each other. Note that the need for two non-communicating trusted parties is not essential; in the absence of a randomizer, one can use key agreement protocol to play the role of the randomizer (as described in Section 5). It is also of the interest of the law enforcement agencies to preserve the privacy of all the databases, which gives no motivation of collusion with any database. We also note that our model suits for other deployments which have been designed using a TTP or a trusted computing base, simple due to the fact that they are deployed with a stronger assumption.

The privacy-preserving protocols that we describe based on this model provide several important benefits in comparison to other well-known models. First, we do not need any heavyweight cryptographic operations in contrast with SMC-based approaches. Second, unlike the case of SMC or P2P models, query computation in our model can be performed in a single round in a similar fashion as a trusted central party based system. Third, the mechanisms do not

[3]A semantically-secure encryption scheme must be probabilistic [18], so the conventional semantic security is sacrificed.

need any special hardware such as powerful secure co-processors in contrast with trusted computing approaches. Overall, we believe that the two-party computation model is a practical approach for achieving both strong privacy guarantees and scalability at the expense of a weaker honest-but-curious adversarial model.

4 Privacy-Preserving Operations

We first give a high-level description of our protocol. The querier initiates the protocol and sends the query in encrypted form to the randomizer, which is to be forwarded to different DBs. This also notifies the randomizer to generate a random nonce (number used only once) which will be sent to all DBs. This random nonce is kept secret from the computing engine (CE), but this nonce may be possibly sent to the querier. That concludes the preparation phase.

The query processing phase involves both the DBs and the CE. Each DB, after received the query and the random nonce from the randomizer, returns an encoded response ("obfuscated" by the random nonce) to the CE via a secure channel in which no other DB can learn the data being sent. The CE then makes use of these responses from different DBs to compute the query result. Since all values the CE got are just some encoded data, the CE does not hold the actual records to appear in the final result. Therefore, after the first round of processing there will be a second round where the CE tells each of DBs in the form of bit vectors indicating what records should appear in the query result. The DBs then send an encrypted form of all these records to the CE.

The last phase is the computation phase which is merely performed by the CE and the querier. After received data from different DBs, the CE rearranges them according to the bit vectors, and sends the rearranged data to the querier. The querier decrypts the encrypted records and gets the final result. In some cases, this step requires the random nonce received from the randomizer.

4.1 System Setup

The setup of our system is as follows. First, there is a secure channel between all the DBs and the querier; in particular, the DBs can encrypt their data to the querier such that the computing engine cannot read. Second, there are secure channels between the computing engine and each of the DBs, so that the answer from a certain DB cannot be read by any other DBs.

Now we describe the public system parameters. Let κ be the security parameter of the whole system, H be a cryptographic hash function (in particular, it should be collision resistant) taking bit-strings of arbitrary length as input.

For notational convenience, let ν be the maximum bit-length to represent the value stored in the database to be processed. This value does not need to be known in advance for the execution of the protocol.[4] In this case, the input of H is of limited length and H can be defined by $H : \{0,1\}^{\nu+\ell} \rightarrow \{0,1\}^{\rho}$, where ℓ and ρ are polynomial in κ. In particular, it can be $\ell = \rho = \kappa$. Also let p be a prime number such that $2^{\rho} < p < 2^{\rho+1}$,

The system parameter is $\{\kappa, p, \ell, \nu, \rho, H(\cdot)\}$ with the description of other cryptographic algorithms to be used (e.g. encryption for the secure channels). These cryptographic algorithms also employ the same security parameter κ.

H will be modeled as a random oracle [7] in our security analysis. In practice, H will be instantiated by off-the-self hash function like SHA-1 using the ways suggested in [7], in this case ρ is 160 bits.

We abuse the notation of H a little bit. Let $\mathcal{X}$ and $\mathcal{Y}$ be two sets, $\mathcal{Y}_r = H_r(\mathcal{X})$ denotes the results of hashing a concatenation of each element in the set $\mathcal{X}$ with the string r, i.e. if $\mathcal{X} = \{x_1, x_2, \cdots, x_m\}$, $\mathcal{Y}_r = \{H(x_1||r), H(x_2||r), \cdots, H(x_m||r)\}$.

4.2 Intersection and Union

The problem of intersection query processing across multiple private databases is defined as follows:

> Let $\mathcal{D} = \{D_1, \cdots, D_i, \cdots, D_n\}$ be a set of n data source, and $L_1, \cdots, L_i, \cdots, L_n$ be the lists containing the confidential primary key for the records in the respective databases. An intersection query $q = L_1 \cap L_2 \cap \cdots \cap L_n$ is posed by the querier $T \notin \mathcal{D}$. The problem is to obtain the answer to q with the help of a computing engine $\mathcal{CE} \notin \{T\} \cup \mathcal{D}$, without revealing any additional information to all entities except T (specifically, the computing engine $\mathcal{CE}$ and all data sources in $\mathcal{D}$) and by only providing the query result to the querier T.

For union query, the problem definition is similar but $q = L_1 \cup L_2 \cup \cdots \cup L_n$.

Below we describe the protocol to answer the intersection query, it is easy to see that a similar idea can be used to answer the union query.

4.2.1 Preparation Phase

1. The querier initiates the protocol by sending an encrypted query to the two-party.

2. The randomizer forwards the query to all DBs.

[4]Of course, an unreasonably large ν may make the addition/multiplication result overflows, see Section 6.1.

3. The randomizer picks a random ℓ-bit long string r and sends it to all DBs and the querier via confidential channels.

4.2.2 Query Processing Phase

1. Each DB $D_i, i \in \{1, \cdots, n\}$ runs the query to get the set of the values to be intersected (but not the other parts of the query result).

2. Each DB rearranges these values in lexicographic order to prevent any possible inference of the original set [4]. We call the resulting set $\mathcal{V}^{(i)} = \{v_{i,1}, \cdots, v_{i,j}, \cdots\}$.

3. Each DB maintains a 1-to-1 mapping of the original order and the lexicographic order.

4. Each DB sends $\mathcal{V}_r'^{(i)} = H_r(\mathcal{V}^{(i)})$ to the computing engine via a confidential channel in which no other DB can learn the data being sent.

4.2.3 Retrieval Phase

1. After received $\mathcal{V}_r'^{(i)}, i \in \{1, \cdots, n\}$ from each DB, the computing engine performs the intersection matching among them, and sends bit vectors B_i to the i-th DB denoting which elements from the i-th DB appear in all other databases.

   Formally, write B_i as a series of bits $b_{i,1}b_{i,2}\cdots$, $b_{i,k}$ = 1 if $\forall j \in \{1, \cdots, n\}, \exists k_1', k_2', \cdots$ such that $H(v_{i,k}||r)$ = $H(v_{j,k_j'}||r)$.

2. According to the bit vector and the 1-to-1 mapping previous maintained, each DB then sends an encrypted form of the required records to the computing engine.

4.2.4 Final Computation Phase

The final phase is simple. The computing engine sends the final result to the querier after a rearrangement of all these encrypted records. The querier just decrypts the encrypted records and gets the final result. Note that the computing engine does not need to know anything about the query, says the field being intersected, except the obvious fact that it is an intersection query.

4.3 Addition and Multiplication

Addition (multiplication) query processing across multiple private databases is defined as follows:

Let $\mathcal{D} = \{D_1, \cdots, D_i, \cdots, D_n\}$ be a set of n data source, and $L_1, \cdots, L_i, \cdots, L_n$ be the equal-sized lists containing the confidential numerical values which are the result of a certain query in the respective databases, in the form of $L_i = \{v_{i,1}, \cdots, v_{i,j}, \cdots, v_{i,m}\}$. An addition (or a multiplication) query posed by a querier $T \notin \mathcal{D}$ is to compute the sums $\sum_{i=1}^{n}\{v_{i,1}, \cdots, v_{i,m}\}$ (or the products $\prod_{i=1}^{n}\{v_{i,1}, \cdots, v_{i,m}\}$) with the help of a computing engine $\mathcal{CE} \notin \{T\} \cup \mathcal{D}$. The problem is to obtain the answer to the query without revealing any additional information to all entities except T (specifically, the computing engine $\mathcal{CE}$ and all data sources in $\mathcal{D}$) and by only providing the query result to the querier T.

Note that we assume the lists from different databases are of equal sizes, and are all arranged in a order such that a correct set of values from different databases are aggregated. These assumptions are reasonable, e.g. when the aggregation operation is done after the intersection query, and hence the correspondences of values to be aggregated are known from the previous query.

Traditional approach for privacy-preserving addition and multiplication uses homomorphic encryption, as described in Section 2. With the help of the computing engine, we can do the same job without using public key operation in the following way. The description here is for addition but it is trivial to see multiplication can be done in the same way.

4.3.1 Preparation Phase

This is essentially the same as the preparation phase in the intersection protocol.

1. The querier initiates the protocol by sending an encrypted query to the two-party.

2. The randomizer forwards the query to all DBs.

3. The randomizer picks a random ℓ-bit long string r and sends it to all DBs and the querier via confidential channels.

4.3.2 Query Processing Phase

1. Each DB $D_i, i \in \{1, \cdots, n\}$ runs the query to get the set of the values to be added. Again, this set does not include any data which is not an operand of addition.

2. For each $j \in \{1, \cdots, n\}$, the i-th DB computes $R_{i,j} = H(\hat{i}||\hat{j}||r)$ where $\hat{i}, \hat{j}$ are $\nu/2$-bit representations of i, j and $R_{i,j}$ is parsed as an element in $\mathbb{Z}_p$. The additions below are defined as those in modulo-p arithmetic.

3. The i-th DB computes and sends $V_{i,j}' = V_{i,j} + R_{i,j}$ for each j to the computing engine via a confidential channel between the i-th DB and the computing engine.

4.3.3 Final Computation Phase

1. The computing engine computes $V'_j = \sum_i V'_{i,j}$, and sends the result to the querier.

2. For each j, the querier computes $V_j = V'_j - \sum_i H(i||j||r)$.

4.4 Selection based on Addition Result

The above description only just suggests how the addition result can be computed. To make use of the result to perform query, or in other words, to support queries with a selection predicate depends on an addition result, we require the querier to compute and send to the computing engine an "encoded predicate" at the end of the preparation phase (and do nothing in the final computation phase). For example, consider a predicate which requires the summation of two values from two different databases to be 9, the querier should send $U'_1 = H(\hat{1}||\hat{1}||r) + H(\hat{2}||\hat{1}||r) + 9 \mod p$ to the computing engine after received r. Under our assumption of "isolated" computing engine, i.e. it would not be compromised by either the querier or any of the databases, the bit-string $H(\hat{1}||\hat{1}||r) + H(\hat{2}||\hat{1}||r)$ acts as a one-time pad and hence the selection criteria is perfectly protected, which also means the computing engine does not learn anything extra except the fact that the querier is looking for a value which will be encoded as U'_1.

Fetching the records passed this encoded predicate, can be easily done in the manner as described in Section 4.2, i.e. the computing engine returns a bit vector to the databases to indicate which record should be passed to the querier.

4.5 Applicability and Discussions of Limitations

With our proposed protocols, we can support a wide range of queries including selection, equijoin, addition, multiplication and combinations of them across databases. For examples, suppose the tables 'patientFile' are distributed across databases (vertical fragmentation), the table 'medHis1' only exists in DB$_1$ and the table 'medHis2' only exists in DB$_2$ (horizontal fragmentation), the following SQL queries can be supported.

```
SELECT icd9 FROM patientFile WHERE date
BETWEEN 2009/01/01 AND 2009/12/31
GROUP BY icd9;
```

```
SELECT name FROM medHis1 JOIN medHis2
ON medHis1.patientID = medHis2.patientID
WHERE medHis1.visit + medHis2.visit = 9;
```

4.5.1 Comparison across Databases

While our model supports equality checking in a simple and efficient manner, it does not naturally support comparison across databases in general (but local comparison can be supported, such as the BETWEEN predicate above.) To see this, note that the computing engine is not supposed to know any *a prior* information about the distribution of the values to be compared. At the same time, the encoding done by the DBs are not coordinated except by the randomizer, this means that the same value should be mapped to the same or somehow "similar" encoding, otherwise the computing engine will treat the same values from different DBs as different. For one-to-one mapping, all DBs should use the same random factor to encode the same value. On the other hand, the random factors encoding different values of the DBs should make the encoded values preserve the same ordering, unless the computing engine is given "some knowledge" about the secret random factors (which possibly relies on some nice algebraic structures for correctness and number-theoretic assumptions for security). The order-preserving property implies there are some "relationships" between the random factors used to encode different values. If the algorithm executed at the DBs side are not mutating across different search queries, the computing engine can possibly deduce the relationship between two different values by exploiting the relationship of the random factors, e.g. a known function of the difference between two values, which is something more than the fact about which number is larger, and should not be leaked.

The above heuristic arguments also suggest a few directions for future research – 1) devise "efficient" public key cryptographic mechanism, 2) assign dynamic yet somehow coherent behaviour to the DBs according to the randomness, 3) weaken the security guarantee, says allow the leakage of some partial information about the data distribution and 4) weaken the correctness guarantee, says values which are "closed" to each other are mapped to the same encoding such that only values differ by more than a certain threshold can be distinguished.

A large body of work has been devoted to creating solution for comparing among values from different parties in a private manner, which also shown to us the difficulty of tackling this problem (e.g. many solutions operate in bit-level for comparing among only two integers). Among the paradigms reviewed in Section 2, only SMC and trusted computing can answer these queries. We remark that there are specific solutions for comparison queries, says top-k queries [2, 32]. However, they are inefficient for very large database and more research is needed before they can be considered to be practical.

4.5.2 Nested Queries

Another limitation of our protocols is the handling of *all kinds of* nested queries without leakage of partial information, says executing our first sample query over the result of our second one (assuming 'patientID' is a primary key which also presents in the relation 'patientFile') instead of merely from 'patientFile'. Different from the case for a predicate based on an addition result in Section 4.4, it is not always possible to enable the querier to help the computing engine in processing the next step of a nested query, without revealing unnecessary partial information to the querier.

Similarly, while it is possible to perform nested computation (instead of a single addition/multiplication) in several rounds, partial information will be leaked. However, complicated formula is not supported by the traditional approach based on homomorphic encryption either simply due to the restricted homomorphism of the current schemes. Finally, both approaches do not naturally support floating point arithmetic due to the use of cryptographic groups.

5 Removing the Trusted Parties

5.1 Reducing the Trust on the Randomizer

Our solution places trust on the pseudorandomness of the random number generation at the randomizer. This gives a target of attack, says one may try to introduce a virus to make "random number generation" follows a non-uniform distribution. One possible way to thwart this threat is to employ a low-cost secure co-processor which is only required to perform pseudorandom number generation.

5.1.1 Key Agreement Protocol

Yet there are various cryptographic means to solve this problem. One way is to carry out key agreement protocol (KA) [10] among the databases. KA gives a random session key that is only known to the involving parties but no one else, even there are no previously established confidential channels among them. The security property of KA further guarantees that no collusion of participants can bias its randomness. The session key established by KA possesses exactly the properties we expect from the random number generated by the randomizer. The KA protocol should also be authenticated to avoid man-in-the-middle attack.

Multi-party (authenticated) KA (also known as conference key protocols) can be obtained by generalising the two-party KA protocol. Each party first picks a random number locally, computes an one-way function of it and either sends or broadcasts it to all other parties else. This may take multiple rounds of communication. But note that this is not invoked per every database tuple. It also seems necessary since there is no "dealer" that all of these parties can trust in our model, in contrast with the TTP paradigm.

Indeed, KA is needed once for all instead of per query. The trick is to generate new random nonces by maintaining state and using the key established by KA as an initial seed of an one-way function. More concretely, suppose the session key established by a single invocation of KA is K, one can derive many numbers by $H(K), H(H(K)), \cdots$, which are all pseudorandom. Indeed, one can use any one-way function H and extract hardcore bits [17] out of H's output to get forward-secure pseudo random number generation – all the previously output bits remain pseudorandom even if the current state is exposed [17]. On the down side, this method requires another multi-party KA protocol invocation if some databases join or leave the system.

5.1.2 Verifiable Random Function

The use of KA removes the randomizer, there is yet another cryptographic primitive that can help reducing the trust on the randomizer. Verifiable random function (VRF), introduced in [25], is a pseudorandom function [17] that provides a non-interactively verifiable proof for the correctness of the random output.

The entity responsible for the random number generation has a pair of private and public keys. It takes a private key and a certain input value x to produce a pseudorandom number y and the associated proof π. With the help of public key and the proof π, any third party can check whether y is really the output values corresponding to x produced by the private key. On the other hand, only the one who has the private key can calculate the output value y, any party which only hold the public key and the input value x but without the corresponding proof cannot compute the output.

Apart from the verifiability, VRF possesses uniqueness property, which guarantees that it is computationally difficult to find y_1, y_2, π_1, π_2 such that π_1 can prove y_1 is the output value corresponding to x while π_2 can prove y_2 is the output value corresponding to the same x.

Here we review a simple and efficient construction of VRF [12]. For a public key $PK = g^{SK}$ where $SK \in_R \mathbb{Z}_p$, the VRF output is $y = \hat{e}(g, g)^{1/(x+SK)}$ and the corresponding proof is $\pi = g^{1/(x+SK)}$. Verification can be done by the bilinearity of the mapping $\hat{e}$. If both $\hat{e}(g^x \cdot PK, \pi) = \hat{e}(g, g)$ and $y = \hat{e}(g, \pi)$ hold, y is legitimate. As described, both the proof and the public key are of constant-size so it is bandwidth-efficient. Furthermore, the secret key involved can be distributed to multiple parties [12] such that the randomizer will not be the single point of failure.

In our scenario, using VRF in the randomizer means all DBs can verify the random coin flipping process, without the fear that the generation follows a non-uniform distribution. The security is guaranteed as long as the private key for generating the secret is not leaked.

5.2 Secure Processor for Weaker Assumption

The encoded values from the databases are sent to the computing engine via confidential channels. The only other potential vulnerability is in the computing engine itself. This makes it a target to attack. On the other hand, there is risk of collusion between the computing engine and the randomizer.

We can use a secure co-processor to ensure the integrity of the code resided at the computing engine and the confidentiality of the data resided at the computing engine's memory. The requirement for code-integrity in our case is *much less* than that in [3], where public key operations are to be performed by the co-processor. Timing figures in Section 7 also supports that no expensive operations are needed.

6 Security Analysis

6.1 Soundness

We first consider the soundness requirement, which means the end result obtained by the querier is sound with respect to the query and the records in the databases. We prove that our protocols are sound, an aspect that is often neglected (and may go wrong [13], see Section 2).

Definition 1 *A system for processing intersection query across distributed databases is said to be sound if it is of overwhelming probability that the final result is $L_1 \cap L_2 \cap \cdots \cap L_n$, nothing more or less.*

Since we assume the computing engine performs the protocol faithfully and the randomizer generates a fresh random value each time, our system is sound as long as hash collision, i.e. $\exists v, v'$ such that $H(v||r) = H(v'||r)$ but $v \neq v'$, does not occur. If there is an adversary breaking the soundness property, we find a collision pair $(v||r)$ and $(v'||r)$ which breaks the collision resistance of the hash function.

Our basic system can be easily extended to enable the querier to detect if such a collision occurred with a higher probability. Instead of a single random nonce, the randomizer sends two random nonce r and r' to the databases. The second one is only used in the final computation phase to give the querier some clue in collision detection. For the privacy of $v_{i,j}$ in the case that it is not useful for the querier (e.g. as an internal primary key value), instead of sending $v_{i,j}$ in clear, only the hash value $H(v_{i,j}||r')$ is sent. Ignoring the negligible probability that collision occurs for two independent nonces, i.e. $H(v_{i,j}||r) = H(v'||r)$ and $H(v_{i,j}||r') = H(v'||r')$, a difference in the second hash values indicates a collision has occurred and the record concerned should be dropped from the final result.

For the addition protocol, all $R_{i,j}$ terms introduced at first are removed later and thus it is sound as long as the intermediate computations do not overflow, which never happen if the computation are done in a cyclic group. In the rare case that the final result is a huge number that is larger than $2^{\rho+1}$ (or p to be accurate), if the querier had domain-specific knowledge, the correct value may be inferred. For multiplication, a similar argument also applies. The soundness follows directly from the fact that multiplicative inverse exists for any non-zero element in $\mathbb{Z}_p$.

6.2 Adversary Model

We consider the following two kinds of adversaries.

1. Databases: An adversary $\mathcal{A}_{DB}$ models a coalition of all but one databases, who aims to break the privacy of the remaining honest database. $\mathcal{A}_{DB}$ is equipped with the knowledge of the nonce from the randomizer, which models an "insider" adversary.

2. Computing Engine: An adversary $\mathcal{A}_{CE}$ models a curious computing engine, who aims to break the privacy of any honest database. This models an "outsider" attack since the nonce from the randomizer is kept secret from $\mathcal{A}_{CE}$.

We do not consider security against a malicious querier since it receives nothing about the database from the protocol other than the query result (recall our assumption that the querier is authenticated and cannot be any of the data owner). In particular, the querier does not have access of any intermediate values sent from the databases to the computing engine. On the other hand, we cannot afford a stronger adversary which models a collusion of the databases with the computing engine. By the correctness of the protocol, such a collusion can deduce any information of the honest database.

This separation suggests why our systems achieve privacy against $\mathcal{A}_{DB}$. For each query, a honest database only sends out a single message to the computing engine; As long as the computing engine does not leak this message to $\mathcal{A}_{DB}$, $\mathcal{A}_{DB}$ has completely no idea about anything about the honest database.

6.3 Privacy Against Computing Engine

The notion of privacy deserves more explanations. Our formal definition is based on the common security formulation in which the adversary plays a two-phased game against a challenger. In the first phase, the adversary picks two elements v_0^*, v_1^* of its choice. The challenger provides *a priori* knowledge about the distribution of the data to the adversary except the existence of v_0^*, v_1^*. In the second phase, the challenger makes a random decision which of v_0^*, v_1^* is put to a database. The challenger, acts as the databases, then "interacts" with the adversary, as a computing engine. The interaction gives the transcripts of communication between each database and the computing engine computed according to the protocol specification. After many interactions, the adversary is expected to guess what is the random decision made by the challenger. Privacy against computing engine means that the adversary cannot decide whether v_0^* or v_1^* presents in a database better than a wild guessing. Our formulation models the situation that even the computing engine has some *a priori* knowledge about the database, the thing it is uncertain remains uncertain even after it saw the transcript of our protocol. We will not allow the adversary to know the existence of v_0^*, v_1^* in all databases, which matches our model that database and the computing engine will not collude to compromise the privacy of another database.

Our model does not only equip the adversary with the knowledge of the data distribution, but actually let it decide most of the distribution even in an adversarial way. It is entitled with some power to query the databases too. This models a "limited collusion" of the computing engine with some of the databases and the querier. Finally, we emphasise that only the adversarial behaviour of computing engine is formally defined since it is the only kind of adversary that is (intellectually-)interesting to deal with. It does not mean that our protocol is insecure against other kinds of adversary including database, randomizer or querier. The attack mode of $\mathcal{A}_{CE}$ is formally defined below.

Definition 2 *A system for processing intersection query across distributed databases is (x, n, κ)-secure against a curious computing engine if any probabilistic polynomial time adversary $\mathcal{A}_{CE}$ has negligible advantage (in the security parameter κ) in winning the following game, where x is a positive integer smaller than the size of the set of possible data values to be processed and n is the number of databases.*

Below describes a game played between the adversary $\mathcal{A}_{CE}$ and a challenger $\mathcal{C}$. The set of possible data values to be processed is represented by $\{0,1\}^\nu$ and the size of this set is 2^ν. In other words, each data value can be uniquely represented by a ν-bit number. Without loss of generality, we assume that $\mathcal{A}_{CE}$ is interested in breaking the privacy of the first database D_1. The hash function H is modeled as a random oracle. We grant the adversary a polynomial number of random oracle accesses. Each time H is queried to give $H(x)$ for a "new" x that never appears in any of the previous H queries, a random value from $\{0,1\}^\rho$ is chosen and assigned as the value of $H(x)$.

1. (Setup Phase:) $\mathcal{A}_{CE}$ picks two elements $v_0^*, v_1^* \in \{0,1\}^\nu$.
2. $\mathcal{A}_{CE}$ also outputs n vectors of 2^ν-bit long $(b_1, \cdots, b_n)$, where the i-th bit of b_j is on if $\mathcal{A}_{CE}$ wants the record indexed by i to appear in the j-th database. Otherwise, the corresponding record should not appear. There are two constraints in choosing the bit vectors:
   (a) the v_0^*-th and v_1^*-th bit of all vectors should be 0
   (b) the maximum number of bits which are 1 in all vector is x
3. (Challenge Phase:) The challenger picks a random bit c. The data value v_c is put into database D_1, while the presences of v_0^* and v_1^* in other databases are completely arbitrary, but still kept secret from $\mathcal{A}_{CE}$.
4. The challenger runs our protocol to answer intersection queries from $\mathcal{A}_{CE}$ for all the records across all n databases, according to the setup requested by $\mathcal{A}_{CE}$ and the random bit chosen by $\mathcal{C}$. To prevent $\mathcal{A}_{CE}$ from winning trivially, it can only issue query which contains either both or none of v_0^* and v_1^*.
5. $\mathcal{A}_{CE}$ outputs a bit c'.

$\mathcal{A}_{CE}$ is considered to be won this game if $c' = c$. Its advantage in winning the game is defined as $|\Pr[c' = c] - \frac{1}{2}|$. We defined the privacy in terms of a single interaction, but a simple hybrid argument can show that this definition implies privacy over many sequential instances of the protocol.

6.3.1 Security of Our Intersection/Union Protocol

The privacy guarantee of our proposed protocol for intersection is summarized by the theorem below.

Theorem 1 *In the random oracle model, the intersection query protocol is $(2^\nu - 2, n, \kappa)$-secure.*

Proof 1 *Firstly, we bound the probability for $\mathcal{A}_{CE}$ to learn the random nonce r. Assume $\mathcal{A}_{CE}$ made q_H queries to the random oracle H, including those implicitly made by seeing the communication transcripts of the protocol; in the random oracle model, $\mathcal{A}_{CE}$ can only learn r by finding a collision among the responses of the random oracle queries it made. The probability that q_H hash values have at least one collision equals to $1 - \prod_{i=1}^{q_H-1} \frac{2^\rho - i}{2^\rho} \approx 1 - \exp(\frac{-q_H(q_H-1)}{2^{\rho+1}})$. Since both ρ and q_H are polynomial in the security parameter κ, this probability is negligible in κ.*

We complete the proof by showing that the distribution of the view of $\mathcal{A}_{CE}$ remains the same no matter what the bit c is. Note that $\mathcal{A}_{CE}$ can control the appearances of at most $2^\nu - 2$ data values. For $c = 1$, $H(v_1^*||r)$ appears in the transcript sent to $\mathcal{A}_{CE}$. For $c = 0$, the simulator implicitly puts v_0^* into the database by including $H(v_0^*||r)$ in the transcript to $\mathcal{A}_{CE}$. Due to the property of the random oracle, the distributions of $\mathcal{A}_{CE}$'s view with $H(v_0^*||r)$ and its view with $H(v_1^*||r)$ are the same, hence the view of $\mathcal{A}_{CE}$ for $c = 0$ and $c = 1$ are simply indistinguishable for a polynomial time adversary $\mathcal{A}_{CE}$.

Our intersection/union protocol achieves *optimal* privacy in our formulation – up to $2^\nu - 2$ records in each of the databases can be adversarially controlled, where the total number of possible records is 2^ν.

We note that our model does not consider the size of the intersection, and our protocol does leak it to the computing engine. A general solution is to have data padding. We borrow the first bit of the random nonce for this purpose (which reduces the size of the nonce space by a factor of two). All real data values existing in the database are appended with bit 0. Each database can then introduce random bogus data by appending bit 1. When the computing engine returns the bit vector to the databases, each database identifies the bogus data in the intersection results by this special bit and returns garbage records accordingly. Finally, the querier simply discards these garbage records and gets the final correct result. This method partially hides the size of the intersection. A lower bound of the size is still leaked.

It is possible to formulate a security model to ensure the computing engine cannot learn the query from the response of databases. But this is not our major concern and we omit this due to page limitation.

6.3.2 Security of Our Addition Protocol

Similar to the game for the privacy of intersection, we want to ensure the security of our addition protocol even if the computing engine adversary can adversarially choose the data values to be added, except one of them. The adversary only knows this special value comes from one of the two possibilities previously chosen by the adversary itself. Privacy guarantee comes from the fact that the adversary cannot decide which of them is the real case better than a wild guessing.

Definition 3 *A system for processing addition query across distributed databases is said to be (m, n, κ)-secure against a curious computing engine if any probabilistic polynomial time adversary $\mathcal{A}_{CE}$ has negligible advantage (in the security parameter κ) in winning the following game, where n is the number of databases and each database has m records to be added.*

Without loss of generality, we assume that the set of possible data values to be added are in the range of $\mathcal{R} = [0, 2^{\nu-1}]$ and $\mathcal{A}_{CE}$ is interested in breaking the confidentiality of the first element of the database D_1.

1. (Setup Phase:) $\mathcal{A}_{CE}$ picks two values v_0^*, v_1^* from the range $\mathcal{R}$. $\mathcal{A}_{CE}$ also outputs n lists of numerical values in the form of $L_i = \{v_{i,1}, \cdots, v_{i,j}, \cdots, v_{i,m}\}$, where $L_i \in \mathcal{R}^m$, for $i \in \{1, \cdots, n\}$. $\mathcal{A}_{CE}$ is constrained to set $v_{1,1} = 0$.

2. (Challenge Phase:) The challenger picks a random bit c and set $v_{1,1} = v_c^*$.

3. The challenger runs our protocol to issue an addition query across all n databases, according to the setup requested by $\mathcal{A}_{CE}$ and the random bit chosen by $\mathcal{C}$. $\mathcal{C}$ then forwards the transcripts of communication between each database and the computing engine to $\mathcal{A}_{CE}$.

4. $\mathcal{A}_{CE}$ outputs a bit c'.

$\mathcal{A}_{CE}$ is considered to be won this game if $c' = c$. Its advantage in winning is defined as $|\Pr[c' = c] - \frac{1}{2}|$.

We have the following theorem for the privacy of our addition protocol.

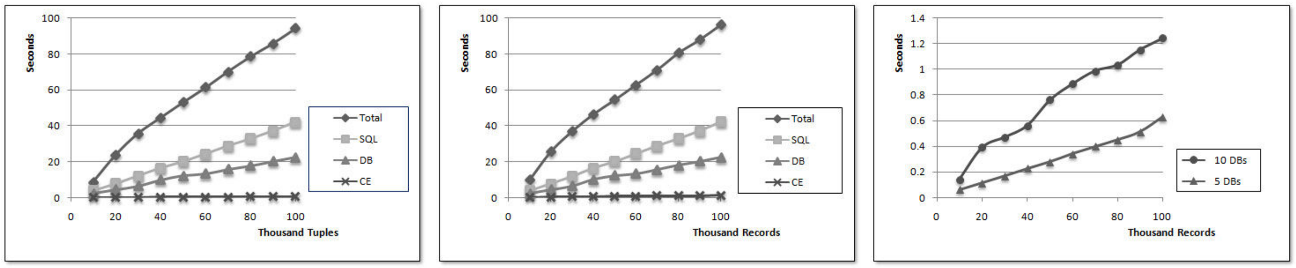
Theorem 2 *In the random oracle model, the addition query protocol is $(m = poly(\kappa), n, \kappa)$-secure.*

Proof 2 *The proof is similar to that for the Theorem 1. The probability for $\mathcal{A}_{CE}$ to learn the random nonce r is bounded by a quantity negligible in κ. Conditioned on the event that r is not leaked, we are now going to show the view of $\mathcal{A}_{CE}$ remains computationally indistinguishable no matter what the bit c is. Let $\hat{1}$ be the $\nu/2$-bit representation of the integer 1. Note that $H(\hat{1}||\hat{1}||r)$ does not appear in anywhere other than $V'_{1,1} = V_{1,1} + H(\hat{1}||\hat{1}||r)$. Due to the property of the random oracle, the distribution of $V'_{1,1}$ remains the same no matter $V_{1,1} = v_0^*$ or $V_{1,1} = v_1^*$. Hence, any polynomial time adversary $\mathcal{A}_{CE}$ has only negligible advantage in deciding which is the real case.*

7 Evaluation

We implemented a prototype of the proposed privacy-preserving operations in Java with JCE for cryptographic operations, without using any native code. We modified hsqlDB (`http://www.hsqldb.org`) to implement the individual database servers. For the choices of cryptographic algorithms, we instantiate the cryptographic hash function by SHA-1, and use AES for realizing the confidential channels. In our prototype, the secret keys for AES are pre-established and shared among different entities. For the choices of parameters, the key-length we used for AES is 128 bits. SHA-1 produces a hash value that is 160 bits long, i.e. $\rho = 160$. The random nonce r is generated as a 1024-bit number using a pseudorandom number generator.

For our experiments, we used 10 distinct Sun-Fire-280R, each with 900MHz UltraSparcIII CPU and 4 GB RAM, running on Solaris SunOS 5.10, as database servers. For the computing engine, we used a Linux machine with dual 3.06GHz Intel Xeon CPU and 2 GB RAM running Red Hat Linux 3.4.6.3. In our prototype, we deployed the randomizer and querier on the same machine with dual 3.06GHz



(a) (b) (c)

Figure 2. Timing figures of our intersection protocol: (a) Time for different computations for 5 databases (b) Computation times for 10 databases (c) Computing engine processing time

Intel Xeon CPU and 2 GB RAM running on Red Hat Linux 3.4.6. Because of this simplification, the timing figures for the addition protocol below omitted the communication time for transferring the nonce from the randomizer to the querier. Since, all our protocols require only one round of communication (other than the actual transfer of the records depending on the result of the intersection/addition, which must requires another round of communication for any protocol that tries to minimize the bandwidth), we ignore wide-area propagation delays across nodes.

Regarding the data generation method, our dataset is synthetically generated. The data size (total records in DB) of each DB is fixed to $100,000$ records with the schema partially derived from a RHIO setting. Experiments were repeated 100 times using the same data over which timings were averaged.

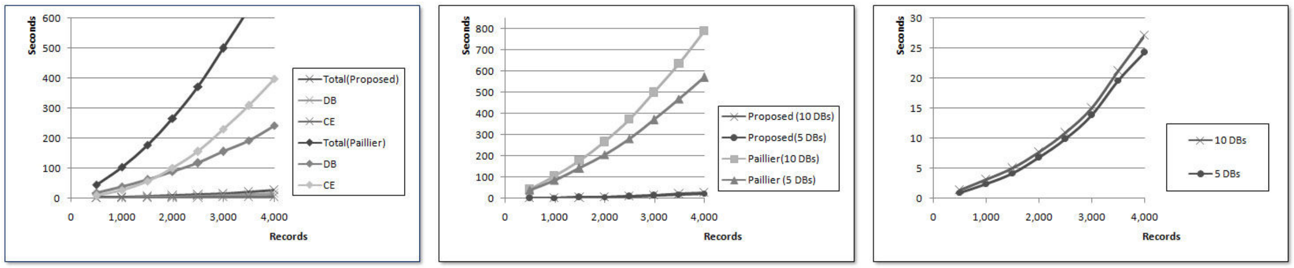
Figures 2(a) and 2(b) illustrate the scalability characteristics of our intersection protocol. SQL refers to the time consumed by each DB to answer a SQL query, CE refers to the time taken in the computation engine, DB refers to the additional time taken by the agent resided in each DB. We make the following observations. First, the total running time is dominated by the response time of a SQL query that is unavoidable. The time consumed by our agent is only approximately $1/5$ of the total time for $100,000$ records. This included the time for encoding the query results, a logical step which is necessary for all privacy-preserving systems. Second, the running time of the computing engine is insignificant when compared with the other timing figures. Finally, the relationship of the total running time and the number of records is roughly linear. These results show that our system scales even in the face of millions of records. For the scalability issue related to the number of databases, increasing the number of DBs only increases the time required by the computing engine to perform the set operations, as inferred from Figures 2(a) and 2(b). Furthermore, this is a very small fraction of the total running time as shown in Figure 2(c). These results came from the fact that the computational task for each database is parallelized in our system: this gives us confidence that our system will scale in wide-area settings.

Figures 3(a) and 3(b) show the experiment results of our addition protocol in comparison to the Paillier encryption based protocol (512-bit key), which is the standard primitive being used whenever additive homomorphic property is expected. Numbers of records in the figures here refer to the total number of records being processed. For example, in the 10 databases cases, $5,000$ records means each database owns 500 of them and 500 10-record-additions are done across the databases. Each set of experiments include executions for 500 to $4,000$ records. The numbers are small but are large enough to contrast the efficiency of our proposal from the encryption-based approach.

We make the following observations. First, as expected, the encryption based approaches are much slower than our protocol. In fact for the encryption-based approach, the increase in the time required changes at a higher rate than the increase in the number of records. Second, the aggregation step of encryption-based approach also takes a significant portion of the total time. Finally, for our proposed protocol, increasing the number of DBs only slightly increases the total time required. All these in essence demonstrate the scalability of our addition protocol.

8 Conclusions

Distributed database systems such as hospital information systems that operate in the real world need to be scalable in terms of the number of databases in the system and the sizes of the individual databases. Despite the abundance of privacy-preserving techniques that have been proposed in the research literature, many of the existing implementations have adopted the simple central party based computation model for query processing. While this places significant trust on the central party and individual entities, and in essence reveals significant information to the central trusted



(a) (b) (c)

Figure 3. Timing figures of our addition protocol: (a) Time for different computations for 10 databases (b) Total time comparison for 5 and 10 databases (c) Total time for our proposed protocol

party; this approach has gained acceptance due to its simplicity and scalability. In this paper, we have proposed an alternative *Two-Party Query Computation Model* which operates under the honest-but-curious adversarial assumption. Under this assumption (which is realistic in many real world settings), we show how one can perform different types of privacy-preserving queries in a scalable manner while still not revealing any additional information. The trust assumption can be weakened by cryptographic means. In particular, it is possible to remove the randomizers from the system. We have formulated a security model and formally proven the security of our schemes. We have built a prototype to validate the scalability experimentally and we plan to build a large-scale privacy-preserving system in the future for the specific case of electronic medical records.

Acknowledgement

The authors would like to thank Eric Lo, Man-Lung Yiu and the anonymous referees for their helpful comments.

References

[1] G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, R. Motwani, U. Srivastava, D. Thomas, and Y. Xu. Two Can Keep A Secret: A Distributed Architecture for Secure Database Services. In *Conference on Innovative Data Systems Research (CIDR 2005)*, pages 186–199, 2005.

[2] G. Aggarwal, N. Mishra, and B. Pinkas. Secure Computation of the kth-Ranked Element. In *EUROCRYPT 2004*, LNCS 3027, pages 40–55. Springer, 2004.

[3] R. Agrawal, D. Asonov, M. Kantarcioglu, and Y. Li. Sovereign Joins. In *ICDE 2006*, page 26. IEEE Computer Society, 2006.

[4] R. Agrawal, A. V. Evfimievski, and R. Srikant. Information Sharing Across Private Databases. In *SIGMOD 2003*, pages 86–97. ACM, 2003.

[5] S. Ajmani, R. Morris, and B. Liskov. A Trusted Third-Party Computation Service. Technical Report MIT-LCS-TR-847, MIT, May 2001.

[6] M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and Efficiently Searchable Encryption. In *CRYPTO*, LNCS 4622, pages 535–552. Springer, 2007.

[7] M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *CCS 1993*, pages 62–73, 1993.

[8] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. Public Key Encryption with Keyword Search. In *EUROCRYPT 2004*, LNCS 3027, pages 506–522. Springer, 2004.

[9] D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-DNF Formulas on Ciphertexts. In *TCC 2005*, pages 325–341, 2005.

[10] C. Boyd and A. Maturia. *Protocols for Authentication and Key Establishment*. Springer-Verlag, 2003.

[11] J. W. Byun, H. S. Rhee, H.-A. Park, and D. H. Lee. Off-Line Keyword Guessing Attacks on Recent Keyword Search Schemes over Encrypted Data. In *Secure Data Management*, LNCS 4165, pages 75–83. Springer, 2006.

[12] Y. Dodis and A. Yampolskiy. A Verifiable Random Function with Short Proofs and Keys. In *PKC 2005*, LNCS 3386, pages 416–431. Springer, 2005.

[13] F. Emekçi, D. Agrawal, A. E. Abbadi, and A. Gulbeden. Privacy Preserving Query Processing Using

Third Parties. In *ICDE 2006*, page 27. IEEE Computer Society, 2006.

[14] Centers for Medicare & Medicaid Services. The Health Insurance Portability and Accountability Act of 1996 (HIPAA). `http://www.cms.hhs.gov/hipaaGenInfo`.

[15] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword Search and Oblivious Pseudorandom Functions. In *TCC 2005*, pages 303–324, 2005.

[16] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient Private Matching and Set Intersection. In *EUROCRYPT 2004*, LNCS 3027, pages 1–19. Springer, 2004.

[17] O. Goldreich. *Foundations of Cryptography*, volume 1. Cambridge University Press, 2001.

[18] O. Goldreich. *Foundations of Cryptography*, volume 2. Cambridge University Press, 2004.

[19] O. Goldreich, S. Micali, and A. Wigderson. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *STOC 1987*, pages 218–229. ACM, 1987.

[20] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[21] C. Hazay and Y. Lindell. Efficient Protocols for Set Intersection and Pattern Matching with Security Against Malicious and Covert Adversaries. In *TCC 2008*, pages 155–175, 2008.

[22] N. Jefferies, C. J. Mitchell, and M. Walker. A Proposed Architecture for Trusted Third Party Services. In *Cryptography: Policy and Algorithms*, LNCS 1029, pages 98–104. Springer, 1995.

[23] L. Kissner and D. X. Song. Privacy-Preserving Set Operations. In *CRYPTO 2005*, LNCS 3621, pages 241–257. Springer, 2005.

[24] Y. Lindell and B. Pinkas. Privacy Preserving Data Mining. *J. Cryptology*, 15(3):177–206, 2002.

[25] S. Micali, M. O. Rabin, and S. P. Vadhan. Verifiable Random Functions. In *FOCS 1999*, pages 120–130, 1999.

[26] E. Mykletun and G. Tsudik. On Security of Sovereign Joins. Cryptology ePrint Archive, Report 2006/380, 2006.

[27] M. Naor, B. Pinkas, and R. Sumner. Privacy Preserving Auctions and Mechanism Design. In *Electronic Commerce 1999*, pages 129–139. ACM, 1999.

[28] NYCLIX. New York Clinical Information Exchange. `http://www.nyclix.org`.

[29] P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT 99*, LNCS 1592, pages 223–238. Springer, 1999.

[30] I. Stoica, R. Morris, D. R. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *SIGCOMM 2001*, pages 149–160, 2001.

[31] M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu. Mariposa: A Wide-Area Distributed Database System. *VLDB J.*, 5(1):48–63, 1996.

[32] J. Vaidya and C. Clifton. Privacy-Preserving Top-K Queries. In *ICDE 2005*, pages 545–546. IEEE Computer Society, 2005.