Practical Private Information Retrieval from a Time-Varying, Multi-attribute, and Multiple-Occurrence Database

Giovanni Di Crescenzo, Debra Cook, Allen Mcintosh, Euthimios Panagos

Practical Private Information Retrieval from a Time-Varying, Multi-Attribute, and Multiple-Occurrence Database

Giovanni Di Crescenzo, Debra Cook, Allen McIntosh, Euthimios Panagos

Applied Communication Sciences, NJ, USA.
E-mail: {gdicrescenzo,dcook,amcintosh,epanagos}@appcomsci.com

Abstract. We study the problem of privately performing database queries (i.e., keyword searches and conjunctions over them), where a server provides its own database for client query-based access. We propose a cryptographic model for the study of such protocols, by expanding previous well-studied models of keyword search and private information retrieval to incorporate a more practical data model: a time-varying, multi-attribute and multiple-occurrence database table.
Our first result is a 2-party private database retrieval protocol. Like all previous work in private information retrieval and keyword search, this protocol still satisfies server time complexity linear in the database size.
Our main result is a private database retrieval protocol in a 3-party model where encrypted data is outsourced to a third party (i.e., a cloud server), satisfying highly desirable privacy and efficiency properties; most notably: (1) *no unintended information* is leaked to clients or servers, and only minimal 'access pattern' information is leaked to the third party; (2) for each query, all parties run in time *only logarithmic* in the number of database records; (3) the protocol's runtime is practical for real-life applications, as shown in our implementation where we achieve response time that is *only a small constant* slower than commercial non-private protocols like MySQL.

1 Introduction

As reinforced by current technology trends (e.g., 'Big Data', 'Cloud-based Data Retrieval'), in today's computer system there is critical need to efficiently store, access and manage massive amounts of data. While data management and storage systems evolve to take these new trends into account, privacy considerations, already of great interest, become even more important. To partially address privacy needs, database-management systems can use private database retrieval protocols, where clients submit queries and receive matching records in a way so that clients do not learn anything new about the database records (other than the content of the matching records), and database servers do not learn which queries are submitted. The research literature has attempted to address these issues, by studying private database retrieval protocols in limited database models and with limited efficiency properties. In this paper we address some of these limitations, by using a practical database model, and proposing a participant and privacy model in which data is outsourced to a third party (in encrypted form) and practical private database retrieval protocols are possible.

Our contribution. Continuing the well-studied areas of private information retrieval (PIR) [4, 17] and keyword search (KS) [3, 2, 9], we propose a more practical database model, capturing record payloads, multiple attributes, possibly equal attribute values across different database records, and multiple answers to a given query and insertion/deletion of database records. In this model, we define suitable correctness, privacy and efficiency requirements, by showing previously not discussed technical reasons as to why we cannot use the exact same requirements from the PIR and KS areas.

We then design a first database retrieval protocol that satisfies the desired privacy properties (i.e., the server learns no information about the query value other than the number of matching records, and the client learns no information about the database other than matching database records) in our novel and practical database model (i.e., a time-varying, multi-attribute and multiple-occurrence table). This protocol is based on oblivious pseudo-random function evaluation protocols and PIR protocols and still has server time complexity linear in the database size (a source of great inefficiency in previous PIR and KS protocols), a drawback dealt with in our next result.

After expanding the participant model with a third party (i.e., a cloud server, as in the database-as-a-service model [13]), we design a second protocol, *only based on pseudo-random functions (implemented as block ciphers)*, where both server and third party run queries in logarithmic time and the following privacy properties hold: the server learns nothing about the query value, the client learns nothing about the database in addition to the payloads associated with the matching records, and the third party learns nothing about the query value or the database content, other than the repeating of queries from the client and repeated access to the encrypted data structures received by the server at initialization. Thus, we solve the long-standing problem of achieving efficient server runtime at the (arguably small) cost of a 'third-party'-server and some 'access-pattern' type leakage to this third party. We stress that this protocol has efficient running time not only in an asymptotic sense, but in a sense that makes it ready for real-life applications (where such form of leakage to the third party is tolerable), answering another long-standing question in the PIR area. In our implementation, we reached our main design goal of achieving response time to be *only a small constant* slower than commercial non-private protocols like MySQL. Solving a number of technical challenges posed by the new data model (including a reduction step via a novel multiplicity database) using simple and practical techniques was critical to achieve this goal. The privacy loss traded for such a practicality property is rather minimal, as neither the client nor the server learn anything new, and the third party does not learn anything about the plain database content or the plain queries made, but just an 'unlabeled histogram' describing the relative occurrences of (encrypted) matching records and (encrypted) query values within the protocol. (Techniques from [14] can be used to mitigate privacy loss from such leakage as well.) In all our protocols, we only consider privacy against a semi-honest adversary, as there are known techniques, based on the general paradigm of [11], to compile such protocols and achieve privacy against malicious adversaries. Almost all formal definitions and proofs are omitted due to space restriction.

Related work. Our work revisits and extends work in the PIR and KS areas, which have received a large amount of attention in the cryptography literature. (See, e.g., [18].) Both areas consider rather theoretical data models, as we now discuss. In PIR, a database is

modeled as a string of n bits, and the query value is an index $i \in \{1, \ldots, n\}$. In KS, the data models differ depending on the specific paper we consider; the closest data model to ours is the one from [9], where a database is a set of records with per-record payloads, but has a single attribute, admits a single matching record per query and no record insertions/deletions. The inefficiency of the server runtime in PIR and KS protocols has been well documented (see, e.g., [20]). Some results attempted to use a third party and make the PIR query subprotocol more efficient but require a practically inefficient preprocessing phase [7].

In both PIR and KS, the database owner is the server that hosts the plain data. In some related research areas, such as oblivious RAM (starting with [12]), and searchable symmetric encryption (starting with [19]), the database owner is the client that uses the server to host a carefully prepared and encrypted version of the data. In public-key encryption with keyword search (starting with [2]) the data is provided in encrypted form by multiple independent servers to a third party that helps satisfying a client's query. Because of these critical differences, results in the cited areas do not solve the problem addressed by PIR and KS and use substantially different techniques, which are not directly comparable to ours.

In the project supporting this paper, other performing teams like ours came up with different and interesting solutions to similar problems. The closest published work from any of these teams that we are aware of consists of [15]. Among the many differences, the cited paper uses random oracles and public-key cryptography operations, which we since random oracles have been proved to likely not exist and public-key cryptography operations are known to be less efficient than symmetric-key ones. Finally, our database retrieval protocols well combine with database policy compliance protocols from [6] that allow a server to authorize (or not) queries by a client according to a specific policy, while maintaining privacy of queries and policy.

2 Models and Requirements

Data and Query Models. We model a *database* as an n-row, m-column matrix $D = (A_1, \ldots, A_m)$, where each column is associated with an *attribute*, denoted as A_j, for $j = 1, \ldots, m$. The first $m - 1$ columns are *keyword attributes*, and the last column $P = A_m$, is a *payload* attribute. A *database entry* is denoted as keyword $A_j(i)$ or, in the case $j = m$, as payload $p(i) = A_m(i)$. The database *schema* is the collection of parameters n, m and of the description of each *domain* associated with each of the m attributes, and to which database entries belong. The database schema is assumed to be publicly known to all parties. A database row is also called *record*, and is assumed to have the same length ℓ_r (if data is not already in this form, techniques from [8] are used to efficiently achieve this property), where ℓ_r is constant with respect to n. We consider a *database update* as an addition, deletion, or change of a single record, and refer to the *current database* (resp., *current database schema*), as the database (resp., database schema) obtained after any previously occurred updates. A *query* q is modeled to refer to one or more database attributes and to contain one or more *query values* from the relative attribute domains. We will mainly consider KS queries, such as: "SELECT $*$ FROM $main$ WHERE attribute_name $= v$," where v is the query value.

A *valid response* to such a query consists of all payloads $p(i)$, for $i \in \{1, \ldots, n\}$, such that $A_j(i) = v$, if attribute_name $= A_j$, for some $j \in \{1, \ldots, m-1\}$. We say that the records in a valid response *match* the query. We will also discuss extensions of our techniques to other query types, such as conjunctions (via logical AND gates) of KS queries. The number of records containing the same query value v at attribute A_j is also called the j*-multiplicity* of v in the database, and is briefly denoted as $m_j(v)$. KS queries are always made to a specific attribute A_j, for some $j \in \{1, \ldots, m-1\}$, and therefore in their discussion the index j is omitted to simplify notation and discussion.

Participant Models. We consider the following *efficient* (i.e., running in probabilistic polynomial-time in a common security parameter 1^σ) participants. The *client* is the party, denoted as C, that is interested in retrieving data from the database. The *server* is the party, denoted as S, holding the database (in the clear), and is interested in allowing clients to retrieve data. The *third party*, denoted as TP, helps the client to carry out the database retrieval functionality and the server to satisfy efficiency requirements during the associated protocol. By *2-party model* we denote the participant model that includes C, S and no third party. By *3-party model* we denote the participant model that includes C, S, and TP. (See Figure 1,2 for a comparison of the two participant models.)

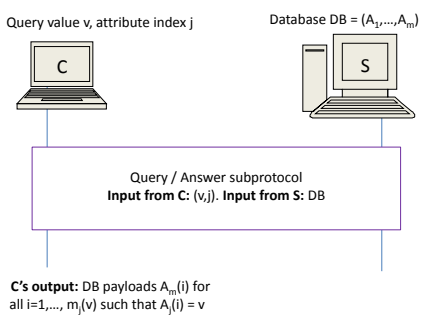

Fig. 1. Structure of our 2-party DR protocol

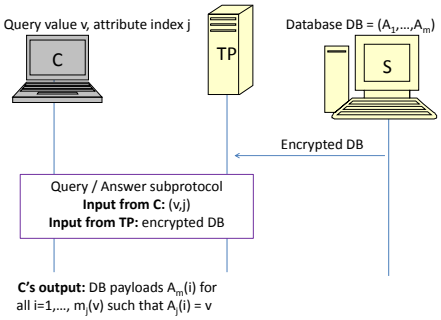

Fig. 2. Structure of our 3-party DR protocol

Database retrieval protocols. In the above data, query, and participant models, we consider a *database retrieval* (briefly, DR) protocol as an evolution of the KS protocol, as defined in [9] (in turn, an evolution of the PIR protocol, as defined in [17]), with the following three main model and functionality extensions, which are consistent with the functioning of typically deployed databases: (1) databases contain multiple attributes (or columns with keywords); (2) each database attribute can have multiple occurrences of the same keyword; and (3) database entries may change as a result of record addition and deletion. Specifically, we define a DR protocol as a triple $(\mathsf{Init}, \mathsf{Query}, \mathsf{Update})$ of subprotocols, as follows. The initialization subprotocol Init is used to set up data structures and cryptographic keys before C's queries are executed. The query subprotocol Query allows C to make a single query to retrieve (possibly multiple) matching database records. The record update subprotocol Update allows S to periodically update the data

structures and cryptographic keys set during the Init subprotocol, as a result of a record addition or deletion. As a first attempt, we target DR protocols that satisfy the following (informal) list of *requirements*:

1. *Correctness*: the DR protocol allows a client to obtain all payloads from the current database associated with records that match its issued query;
2. *Privacy*: the DR protocol preserves privacy of database content and query values, ideally only revealing what is leaked by system parameters known to all parties and by the intended functionality output (i.e., all payloads in matching records to C);
3. *Efficiency*: the protocol should have low time, communication and round complexity; ideally, a constant number of messages per query, and time and communication sublinear in the number n of database records.

It turns out that, as written, these requirement cannot be exactly satisfied, and the missing properties are different depending on whether we consider the 2-party or the 3-party model. Thus, we continue with formal definitions common to both models, and then defer different formal definitions of privacy and efficiency to later sections.

Preliminary Requirement Notations: Let σ be a security parameter. A function is *negligible* if for all sufficiently large natural numbers $\sigma \in \mathcal{N}$, it is $< 1/p(\sigma)$, for any polynomial p. A *DR protocol execution* is a sequence of executions of subprotocols $(\mathsf{Init}, \mathsf{qu}_1, \ldots, \mathsf{qu}_\mathsf{q})$, where $qu_i \in \{\mathsf{Query}, \mathsf{Update}\}$, for $i = 1, \ldots, q$, for some q polynomial in σ, and all subprotocols are run on inputs provided by the involved parties (i.e., a database from S, query values from C, and database updates from S). We only consider *stateless* Query subprotocols, in that they can depend on the outputs of Init and Update subprotocols but not on the output of previous Query subprotocols.

Correctness: for any DR protocol execution, and any inputs provided by the participants, in any execution of a Query subprotocol, the probability that C obtains all records in the current database that match C's query value input to this subprotocol, is 1.

Background primitives. A *random function* R is a function that is chosen with distribution uniform across all possible functions with some pre-defined input and output domains. A keyed function $F(k, \cdot)$ is a *pseudo-random function* (PRF, first defined in [10]) if, after key k is randomly chosen, no efficient algorithm allowed to query an oracle function O can distinguish whether O is $F(k, \cdot)$ or O is a random function R (over the same input and output domain), with probability greater than $1/2$ plus a negligible quantity. An *oblivious pseudo-random function evaluation protocol* (oPRFeval, first defined in [9]) is a protocol between two parties A, having as input a string k, and B, having as input a key x for a PRF F. The protocol's outcome is a private function evaluation of the value $F(k, x)$, returned to B (thus, without revealing any information about x to B, or any information about k to A). oPRFeval protocols were constructed in [9, 16] using number-theoretic hardness assumptions.

A *single-database private information retrieval protocol* (PIR, first defined in [17]) is a protocol between two parties A, having as input a value $i \in \{1, \ldots, n\}$, and B, having as input a database represented as a sequence of equal-length values $x = (x[1], \ldots, x[n])$ The protocol consists in a private retrieval of the value $x[i]$, returned to A (thus, without revealing any information about i to B or about $x[1], \ldots, x[i-1], x[i+1], \ldots, x[n]$ to A). A *semi-private* PIR protocol is a protocol where privacy only consists of preventing to reveal any information about i to B. Several PIR and semi-private

PIR protocols have been presented in the cryptographic literature, starting with [17], using number-theoretic hardness assumptions.

3 Two-Party Database Retrieval

We define privacy and efficiency requirements in the 2-party model in Section 3.1 and describe our first DR protocol (for KS queries in the 2-party model) in Section 3.2.

3.1 Privacy and Efficiency in the 2-Party Model

Informally, our privacy and efficiency requirements are modified with respect to our first attempt in Section 2, so that a 2-party DR protocol can leak the number of matching records to S, has communication complexity sublinear in n if the number of matching records is also sublinear in n, and allows S to run in time linear in the number n of database records. A formal treatment follows.

Privacy: Informally speaking, we require the subprotocols in a DR protocol execution do not leak information beyond the following:

1. Init: the database schema, as part of overall system parameters, will be known to all participants;
2. Query, based on query value v, attribute index j, and the current database: the pair $(j, m_j(v))$ will be known by S and all payloads $\{p(i) : i = i(1), \ldots, i(m_j(v))\}$ such that $A_j(i(1)) = \cdots = A_j(i((m_j(v)))) = v$ will be obtained by C, as a consequence of the correctness requirement;
3. Update: on input a record addition or deletion to the current database, the current database (after the update) will be known by S, as a consequence of the correctness requirement, and the current database schema (after the update), as part of overall system parameters, will be known by all participants.

Given this characterization of the intended leakage, a formal privacy definition can be derived using known definition techniques from simulation-based security and composable security frameworks often used in the cryptography literature.

Consistently with the literature on secure function evaluation, PIR and KS protocols, it might have seemed reasonable to just require that no new information about the query value is revealed to S, as we hoped to achieve in our first attempt in Section 2. It turns out that this level of privacy cannot be obtained, as we now explain. Let us consider a specific execution of subprotocol Query within a DR protocol execution. Since more than one record may match C's query value v, by the correctness property, at the end of this execution of subprotocol Query, C must be able to compute all payloads $p(i(1)), \ldots, p(i(m_j(v)))$ corresponding to records matching v, where $m_j(v)$ denotes the multiplicity of v in the j-th database attribute A_j. Moreover, since the executions of the Init, Update protocols only leak the database schema to C, and since subprotocol Query is stateless, all previous subprotocol executions in the DR protocol execution, do not help in computing all matching records to this specific execution of subprotocol Query. In other words, C must be able to compute all matching records from the communication received during this specific execution of subprotocol Query. By a standard application of Shannon's source coding theorem (see, e.g., [5]), this implies that

the communication exchanged in this execution is an upper bound to the entropy H of payloads $p(i(1)), \ldots, p(i(m_j(v)))$ (over the probability distribution of the source that generates the database payloads). Thus, we obtain the following

Proposition 1. For any v, let $cc(v)$ denote the number of bits exchanged in an execution of subprotocol Query on input query value v and attribute index j. It holds that $cc(v) \geq H(p(i(1)), \ldots, p(i(m_j(v))))$.

In Proposition 1, the entropy term is maximized when database payloads are randomly and independently distributed, in which case the communication exchanged in each execution of subprotocol Query leaks an upper bound on the value $m_j(v)$, i.e., the number of matching records, to *both* C and S. Accordingly, we target the design of protocols that may leak $m_j(v)$ also to S during each execution of subprotocol Query on input query value v and attribute index j (as formulated in the above privacy requirement).

Efficiency: A DR protocol's *round complexity* (resp., *communication complexity*) is the max number of messages (resp., the max length of all messages), as a function of the system parameters n, m, σ, required by any of the $\mathsf{Init}, \mathsf{Query}, \mathsf{Update}$ subprotocols, for any inputs to them. A DR protocol's *S-time complexity* for subprotocol π is the max running time (as a function of the system parameters n, m, σ) required by S in subprotocol $\pi \in \{\mathsf{Init}, \mathsf{Query}, \mathsf{Update}\}$, over all possible inputs to it. Asymptotic requirements consistent with the literature on PIR and KS protocols include the following: (1) the communication complexity of each execution of protocol Query is sublinear in n; (2) the S-time complexity in each execution of protocol Query is sublinear in n. Requirement (1) is achieved by PIR and KS protocols in the literature when up to a single record is sent as a reply to each query. However, in our DR protocols, a query could be matched by a possibly linear number of records; accordingly, we only require the communication complexity to be sublinear *whenever so is the number of matching records*. Requirement (2), when coupled with the privacy requirement that an execution of the Init protocol only leaks minimal information to C, is known to be unachievable in the 2-party model (or otherwise privacy of the query value v would not hold against the server), as discussed in many papers including [17, 9].

3.2 Our Protocol

Our 2-party DR protocol for KS queries follows the general structure outlined in Figure 1 and satisfies the following

Theorem 1. Under the existence of oPRFeval protocols [9] and (single-database) PIR protocols [17], there exists (constructively) a DR protocol $\pi_1 = (\mathsf{Init}_1, \mathsf{Query}_1, \mathsf{Update}_1)$ for KS queries in the 2-party model, satisfying:

1. correctness
2. privacy against C (i.e., it only leaks the matching records to C);
3. privacy against S (i.e., it only leaks the queried attribute and the number of matching records to S);
4. communication complexity of Query_1 is $o(n)$ if so is the number of matching records;

5. round complexity of Query_1 is $O(\log n)$;
6. the S-time complexity in Query_1 (resp. Init_1) (resp., Update_1) is $O(n)$ (resp., is $O(n)$) (resp., is $O(\log n)$).

The protocol π_1 claimed in Theorem 1 is presented in two steps: first, we describe a DR protocol $\pi_0 = (\mathsf{Init}_0, \mathsf{Query}_0)$ in the restricted data model where all keywords in each database attribute are distinct, and no record additions or deletions happen, and then we describe the DR protocol π_1 that builds on π_0 to remove these restrictions.

The DR protocol π_0. Informally speaking, this protocol is a combination of a KS protocol, denoted as Protocol 2 in [9] (in turn building on a semi-private KS protocol from [3]), and an oPRFeval protocol for computing a pseudo-random function f, as follows.

Init_0. On input database $D = (A_1, \ldots, A_m)$, S returns a pseudo-random version of the database, denoted as $prD = (prA_1, \ldots, prA_{m-1}, A_m)$, and computed by replacing keyword entries $A_j(i)$ with pseudo-random values $prA_j(i) = f_k(A_j(i))$ for all $j = 1, \ldots, m-1$ and $i = 1, \ldots, n$, where k is a random key. (That is, keyword entries are replaced by pseudo-random versions of them, but payloads in A_m remain unchanged).

Query_0. On input query value v and attribute index j from C, and key k and database prD from S, the following steps are run:

1. C and S run an oPRFeval protocol to return $f_k(v)$ to C;
2. C sends j to S
3. C and S run the semi-private KS protocol from [3], where C uses $f_k(v)$ as query value and S provides (prA_j, A_m) as a 2-column database. At the end of the protocol, C can compute the record $A_m(i)$ such that $prA_j(i) = f_k(v)$, if any.

Here, the semi-private KS protocol from [3] consists of using a PIR protocol, such as the one from [17], to probe a conventional search data structure built by S on top of the pseudo-database keywords in a way that is oblivious to S.

Protocol π_0 can be shown to be a DR protocol in the following restricted data model: (1) no database records are added or deleted; (2) for each $j = 1, \ldots, m-1$ the database entries $A_j(1), \ldots, A_j(n)$ relative to the j-th attribute are all distinct. We remove these two restrictions by combining the following ideas: S can transform the original database into one where each column A_j has distinct keywords (or payload), by computing a *padded database*, where keywords are padded with a multiplicity counter; S can compute a preliminary *multiplicity database* to let C obtain the multiplicity of its query value from S using protocol π_0; given the multiplicity of the query value, C can request the matching records by making one query for each matching record to the padded database, using again protocol π_0; updating the padded and multiplicity databases and associated data structures can be done efficiently by careful choices of padding and data structures. We stress that revealing the multiplicity value to C does not provide any more information than sending the matching records (which C is entitled to receive). A more formal description follows.

The DR protocol π_1. Based on the above DR protocol $\pi_0 = (\mathsf{Init}_0, \mathsf{Query}_0)$, we define protocol $\pi_1 = (\mathsf{Init}_1, \mathsf{Query}_1, \mathsf{Update}_1)$ as follows.

Init_1. On input database $D = (A_1, \ldots, A_m)$, S builds an associated padded database $pD = (pA_1, \ldots, pA_m)$, as follows. The payload pA_m is equal to A_m. Then, for each $j = 1, \ldots, m-1$, and each $i = 1, \ldots, n$, the keyword $pA_j(i)$ is defined as

$(A_j(i), mc(i,j))$ where $mc(i,j)$ is the multiplicity counter from $\{1,\ldots,n\}$ such that $A_j(i)$ is the $mc(i,j)$-th occurrence of value $A_j(i)$ within array $(A_j(1),\ldots,A_j(n))$; Then S builds an associated multiplicity database $mD = (mA_1,\ldots,mA_m)$, as follows. For each $j = 1,\ldots,m-1$, and each $i = 1,\ldots,n$, the keyword $mA_j(i)$ is defined exactly as $pA_j(i)$. The payload $mA_m(i)$ is denoted as $m'_j(v)$, where $m'_j(v)$ is defined as $m_j(v)$ when $mc(i,j) = 1$ and a null string $\perp$ of the same length otherwise. Finally, S runs Init_0 to compute pseudo-random versions $prpD$ and $prmD$ of databases pD and mD, respectively, and to send the schema for databases mD and pD to C.

Query_1. On input query value v and attribute index j for C, and key k and databases $prpD, prmD$ for S, the following steps are run:

1. C and S run subprotocol Query_0 on input database $prmD$ from S, and query value $(v,1)$ and attribute index j from C. Let $p(i)$ be the payload obtained by C at the end of this protocol, for some $i \in \{1,\ldots,n\}$. If $p(i) = \perp$ then the protocol stops. Otherwise C sets the value $m_j(v) = p(i)$.
2. For $t = 1,\ldots,m_j(v)$, C and S run subprotocol Query_0 on input database $prpD$ from S, and query value (v,t) and attribute index j from C.

Update_1. In subprotocol Update_1, we consider record addition and record deletion operations, and on each of them, we need to efficiently update any values changing as a result of these operations; specifically:

1. the data structures required in π_0;
2. the multiplicity value $m'_j(v)$ used in mD;
3. the multiplicity counters $mc(i,j)$ in the padding structures used in pD and mD;
4. the database schema for both pD and mD; and
5. the pseudo-random databases $prpD$ and $prmD$.

First, with respect to (1), we observe that the data structures used in π_0 are those from [3] and are conventional data structures with the only requirement of performing search in logarithmic time. Our added requirement of having efficient insertion and deletion does not require a modification of the data structures, since many of the data structures used in [3] (e.g., binary search trees) can be used to perform logarithmic-time search, insertion and deletion. Then, the values in (2), (4), and (5) can be updated in time constant with respect to n, as follows. As for (2), updating the multiplicity value $m'_j(v)$ only requires to change one payload entry $mA_m(i)$ in the multiplicity database mD, regardless of the multiplicity of v, and can be done in $O(\log n)$. As for (4), S can just update the database schema for both pD and mD and send those to C. As for (5), S can again use key k to recompute new $f_k(pD_j(i))$ and $f_k(mD_j(i))$ values.

With respect to values in (3), we now describe how to update the multiplicity counters $mc(i,j)$ after a record update. In the addition case, a record would also be added in the databases pD and mD, and the multiplicity counter $mc(i,j)$ in the added record is set to $m_j(v)+1$. In the deletion case, a record would also be deleted in the databases pD and mD, and we cannot stop there as this would likely create a discontinuity in the sequence of multiplicity counters $mc(i,j)$ (e.g., when $m_j(v) \geq 3$, deleting the record with multiplicity counter 2 would leave only records with counters 1 and 3). Instead, S resets the multiplicity counter of i-th record such that $mc(i,j) = m_j(v)$ as equal to the multiplicity counter of the just deleted record, thus keeping no discontinuity in the sequence of multiplicity counters.

4 Three-Party Database Retrieval

We define privacy and efficiency requirements in the 3-party model in Section 4.1 and describe our second DR protocol (for KS queries in the 3-party model) in Section 4.2.

4.1 Privacy and Efficiency in the 3-Party Model

Informally, our privacy and efficiency requirements are modified with respect to our first attempt in Section 2, so that a 3-party DR protocol can leak the number of matching records as well as 'access-pattern' leakage, to TP, and has communication complexity sublinear in n if the number of matching records is also sublinear in n. (In particular, we keep the requirement that S and TP have to run in time sublinear in the number n of database records.) A formal treatment follows.

Privacy: The privacy leakage we allow in the 3-party model has two differences with respect to the 2-party model: the number of matching records is now leaked to TP instead of S; moreover, the following additional leakage to TP is allowed: repeated (or not) occurrences of the same query made by C, and repeated (or not) accesses to the same initialization information sent by S to TP at the end of the initialization protocol. Informally speaking, we require the subprotocols in a DR protocol execution in the 3-party model to not leak any information beyond the following:

1. Init: the database schema, as part of overall system parameters, will be known to all participants and an additional string eds (for encrypted data structures) will be known to TP; here, eds is encrypted under one or more keys unknown to TP and its length is known from quantities in the database schema;
2. Query, based on query value v, attribute index j, and the current database: all payloads $\{p(i) : i = i(1), .., i(m_j(v))\}$ such that $A_j(i(1)) = \cdot\cdot = A_j(i((m_j(v)))) = v$ will be obtained by C, as a consequence of the correctness requirement; the pair $(j, m_j(v)))$, all bits in eds read by TP according to the instructions in the Query protocol, and which previous executions of Query used the same query value v, will be known to TP;
3. Update: on input a record addition or deletion to the current database, the current database (after the update) will be known to S, as a consequence of the correctness requirement, and the current database schema (after the update), as part of overall system parameters, will be known by all participants; all bits in eds read and/or modified by TP according to the instructions in the Update protocol will be known to TP, who will also determine up to one record previously or currently present in the database containing the same query value as the one in the added/deleted record.

Given this characterization of the intended leakage, a formal privacy definition can be derived using known definition techniques from simulation-based security and composable security frameworks often used in the cryptography literature.

Using a direct extension of Proposition 1 to the 3-party model, we can prove an analogue result with respect to leaking $m_j(v)$ to the coalition of S and TP. Thus, different 3-party DR protocols could leak $m_j(v)$ only to S, or only to TP, or somehow split this leakage between S and TP. Having to choose between one of these options, we made the practical consideration that privacy against S (i.e., the data owner) is typically of

greater interest than privacy against TP (i.e., the cloud server helping C retrieve data from S) in many applications, and therefore we focused in this paper on seeking protocols that leak $m_j(v)$ to TP and nothing at all to S. The other definitional choice of leaking repetition patterns (even though not actual data) to TP is not due to a theoretical limitation, but seems a rather small privacy price to pay towards achieving the very efficient S and TP time-complexity requirements discussed below.

Efficiency: The definition of a 3-party DR protocol's *round complexity*, *communication complexity* and *S-time-complexity* are naturally extended from those of 2-party DR protocols. We also define the *TP-time-complexity* by naturally adapting the server time-complexity definition. As for 2-party protocols, we require that 3-party DR protocols have communication complexity to be sublinear whenever so is the number of matching records. Contrarily to 2-party protocols, we do require that the S-time complexity and the TP-time complexity in each execution of protocol Query is sublinear in n, whenever is the number of matching records is sublinear in n (and achieving this property is one of the major goals in the constructions in this paper). We also target a practical *response time* efficiency requirement: the response time within a Query execution is only a small constant c worse than the response time within the same subprotocol for a non-private protocol such as MySQL.

4.2 Our Protocol

Our 3-party DR protocol for KS queries follows the general structure outlined in Figure reffi-3party and satisfies the following

Theorem 2. Under the existence of a PRF, there exists (constructively) a DR protocol $\pi_2 = (\mathsf{Init}_2, \mathsf{Query}_2, \mathsf{Update}_2)$ for KS queries in the 3-party model, satisfying:

1. correctness
2. privacy against C (i.e., it only leaks the matching records to C);
3. privacy against S (i.e., it does not leak anything to S);
4. privacy against TP (i.e., it only leaks number of matching records, the repetition of query values and the repeated access to initialization encrypted data structures);
5. communication complexity of Query_2 (resp. Init_2) (resp., Update_2) is $o(n)$ if so is the number of matching records (resp., is $O(n)$) (resp., is $O(\log n)$);
6. S-time complexity in Query_2 (resp. Init_2) (resp., Update_2) is $O(1)$ (resp., is $O(n)$) (resp., is $O(\log n)$);
7. TP-time complexity in Query_2 (resp. Init_2) (resp., Update_2) is $o(n)$ if the number of matching records is $o(n/logn)$ (resp., is $O(n)$) (resp., is $O(\log n)$);
8. round complexity of Query_2 is $O(1)$.

Informally, our protocol π_2 is obtained by performing several improvements in the 3-party model to protocol π_1 (which was designed in the 2-party model). As done for π_1, we first construct a DR protocol $\pi'_0 = (\mathsf{Init}'_0, \mathsf{Query}'_0)$ in a restricted data model, and then a DR protocol π_2 that uses π'_0 and the ideas of multiplicity database and padded database to obtain a DR protocol in our more general data model.

The DR protocol π'_0. Informally speaking, this protocol is a simplified construction of π_0, taking advantage of the 3-party model.

Init'_0. On input database $D = (A_1, \ldots, A_m)$, where $A_1, \ldots, A_{m-1}$ contain keywords and A_m contains a payload, S first computes a shuffled version sD of database D; that is, $sD = (sA_1, \ldots, sA_m)$, where $sA_j(\rho(i)) = A_j(i)$ for $j = 1, \ldots, m$ and $i = 1, \ldots, n$, where ρ is a random permutation over $\{1, \ldots, n\}$. Then, similarly as in Init_0, S computes a pseudo-random version of the database sD, denoted as $prsD = (prsA_1, \ldots, prsA_m)$, and computed by replacing keyword entries $sA_j(i)$ with pseudo-random entries $prsA_j(i) = f_k(sA_j(i))$ for all $j = 1, \ldots, m$ and $i = 1, \ldots, n$, where k is a random key and f is a pseudo-random permutation (here, using a permutation instead of a function will later facilitate C's decryption of any received payloads). Moreover, S generates a search data structure (i.e., a binary search tree $iTree_j$) over the keywords $prsA_j(1), \ldots, prsA_j(n)$ in $prsD$, for $j = 1, \ldots, m-1$. Then, S sends $(iTree_1, \ldots, iTree_{m-1}), prsD$ to TP and the key k to C, in addition to sending the database schema to both parties.

Query'_0. In this protocol, C takes as input key k, a query value v and attribute index j, S takes as input key k and database D, and TP takes as input $(iTree_1, \ldots, iTree_{m-1})$, $prsD$. On these inputs, the protocol goes as follows:

1. C computes $f_k(v)$ and sends $(f_k(v), j)$ to TP
2. TP searches for $f_k(v)$ in the search data structure $iTree_j$
3. If TP finds i such that $f_k(v) = prsA_j(i)$, then
   TP sends the associated (encrypted) payload $prsA_m(i)$ to C
   C computes the (plain) payload as $f_k^{-1}(prsA_m(i))$

We note that Query'_0 significantly simplifies Query_0 in that C can directly compute $f_k(v)$, without need to run an oPRFeval protocol, and TP can directly search for $f_k(v)$ in the data structure $iTree_j$, without need to run a semi-private PIR protocol.

Protocol π'_0 can be shown to be a DR protocol in the 3-party model and in the following restricted data model: (1) the database entries do not change; (2) for each $j \in \{1, \ldots, m\}$ the database entries $A_j(1), \ldots, A_j(n)$ relative to the j-th attribute are all distinct. We remove these two restrictions exactly as done in Section 3: we generate π_2 from π'_0 using a padded database, a multiplicity database, and composing $m_j(v)+1$ times protocol π'_0. A more formal description is included for completeness.

The DR protocol π_2. Based on $\pi'_0 = (\mathsf{Init}'_0, \mathsf{Query}'_0, \mathsf{Update}'_0)$, we define protocol $\pi_2 = (\mathsf{Init}_2, \mathsf{Query}_2, \mathsf{Update}_2)$ as follows.

Init_2. On input database $D = (A_1, \ldots, A_m)$, S first runs Init'_0, thus sending $prsD$ and $(iTree_1, \ldots, iTree_{m-1})$ to TP and k to C, and D's schema to both C and TP. Then S builds an associated padded database $pD = (pA_1, \ldots, pA_m)$, and an associated multiplicity database $mD = (mA_1, \ldots, mA_m)$, exactly as done in Init_1. Finally, S runs Init'_0 to compute pseudo-random and shuffled versions $prspD$ and $prsmD$ of databases pD and mD, respectively, and to send the schema for databases mD and pD to C.

Query_2. On input query value v and attribute index j for C, and key k and databases $prspD, prsmD$ for S, the following steps are run:

1. C and TP run subprotocol Query'_0 on input database $prsmD$ from TP, and query value $(v, 1)$ and attribute index j from C. Let $p(i)$ be the payload obtained by C at the end of this protocol, for some $i \in \{1, \ldots, n\}$. If $p(i) = \perp$ then the protocol stops. Otherwise C sets the value $m_j(v) = p(i)$.

2. For $t = 1, \ldots, m_j(v)$, C and TP run subprotocol Query'_0 on input database $prspD$ from TP, and query value (v, t) and attribute index j from C, who thus obtains payloads $p(i(1)), \ldots, p(i(m_j(v)))$, $i(1), \ldots, i(m_j(v)) \in \{1, \ldots, n\}$, from $prspD$;
3. For $t = 1, \ldots, m_j(v)$, C computes the t-th original payload from D as $f_k^{-1}(p(i(t)))$, where k is the key obtained during the execution of Init'_0.

Update_2. We consider record addition and record deletion operations, and on each of them, we need to efficiently update any changed values; specifically:

1. the data structures required in π'_0;
2. the multiplicity value $m'_j(v)$ used in mD;
3. the multiplicity counters $mc(i, j)$ in the padding structures used in pD and mD;
4. the database schema for pD and mD;
5. the data structures $(iTree_1, \ldots, iTree_{m-1})$; and
6. the pseudo-random and shuffled databases $prsmD$ and $prspD$.

The updates for values in (1)-(4) are done as in Update_1. The updates to the data structure $iTree$ are conventional data structure updates in the presence of one insertion or one deletion. As for (6), S can again use key k to recompute new $f_k(pD_j(i))$ and $f_k(mD_j(i))$ values. Several approaches would work to update the permutation ρ; here is an example: upon a record insertion, the newly inserted record is considered the $(n + 1)$-th record in D and $\rho(n + 1)$ is defined as $= n + 1$; upon a deletion of the i-th record, the value $\rho(n)$ is defined as $= i$.

5 Extension: Conjunction Formulae

We extend the KS protocols from Sections 3, 4 to the conjunction formulae over KS queries; that is, the AND of c KS queries, for some $c \geq 1$.

A first approach would be to process, in parallel, an individual KS query for each term in the conjunction, and then having TPcompute the intersection across the matching sets. This approach is clearly undesirable both for privacy and efficiency reasons. With respect to privacy, such a protocol would reveal to C the multiplicity of the query value in each conjunct, which is not necessarily computable from the number of records matching all conjuncts. With respect to efficiency, it is not hard to find examples of conjunctive queries for which at least one conjunct is matched by a linear number of records, but the number of records matching all conjuncts is sublinear in n. On such queries, the communication complexity would be linear even though the information to which C is entitled by the DR protocol functionality is sublinear.

To avoid both problems, we designed the following *'combined-index' approach*. In the initialization subprotocol, a specific combined index on the tuple of attribute values can be created by concatenating the attribute names and query values, thus allowing the conjunction to be treated as a single keyword query. For example, a conjunctive query of $A_1 = a_1$ and $A_2 = a_2$ becomes a single KS query $A_1A_2 = a_1a_2$. The combined index for A_1A_2 is treated exactly as indices for single attributes. The initialization, insertion and deletion subprotocols are the same as those for KS queries, working on the cartesian product of the domain. C runs the KS query algorithm using the combined attribute values. Good properties of the method include: support of conjunctive queries with an arbitrary number of conjuncts; it is only necessary to create a single combined index

for each set of attribute values as opposed to each permutation of the attribute values (i.e. only an index for A_1A_2 is necessary, not one for A_1A_2 and one for A_2A_1); and querying on the single index is faster than querying on multiple indices and determining the intersection of the results. The unattractive property is that it requires one index for each supported conjunctions, resulting in an exponential (in c) number of indices if all possible conjunctions have to be supported. Still, many practical formulae can be addressed, as the scenario where the number of attributes is constant with respect to n is the most typical, and storage is an inexpensive resource nowadays.

6 Performance Evaluation

We present performance results for KS queries and boolean formulae over KS queries for a small number of queries executed using our 3-party protocol π_2 (implemented with an additional SSL/TLS layer). MySQL was used for obtaining baseline results in a non-private setting using the same schema and records as for π_2.

Setup. We used 5 different database sizes: 10K, 100K, 1M, 10M, and 100M records. All records were stored in a single database table containing the following columns:

Column	**Type**	**Format**	**Population Approach**
FirstName	VARCHAR	16 Bytes	Randomly chosen, average 1K multiplicity
LastName	VARCHAR	16 Bytes	Randomly chosen, average 1K multiplicity
Gender	ENUM	Male or Female	Randomly chosen
Number	INTEGER	8 Bytes	Randomly chosen and distinct
DoB	DATE	YYYY-MM-DD	Randomly chosen in [1940-01-01, 1990-12-31]
Notes1	TEXT	64 Bytes	Random sentences from ebooks in [1]
Notes2	TEXT	256 Bytes	Random sentences from ebooks in [1]

The S and TP processes and an instance of MySQL server version 5.5.28 were running on a Dell PowerEdge R710 server with two Intel Xeon X5650 2.66Ghz processors, 48GB of memory, 64-bit Ubuntu 12.04.1 operating system, and connected to a Dell PowerVault MD1200 disk array with 12 2TB 7.2K RPM SAS drives in RAID6 configuration. Database clients were running on a Dell PowerEdge R810 server with two Intel Xeon E7-4870 2.40GHz processors, 64 GB of memory, 64-bit Red Hat Enterprise Linux Server release 6.3 operating system, and connected to the Dell PowerEdge R710 server via switched Gigabit Ethernet. Regular TCP/IP connections were used for MySQL. We built MySQL indices on *FirstName*, *LastName*, *Gender*, and *Number*. We built keyword indices for our protocol π_2 on *FirstName*, *LastName*, *Number*, and a combined index on *FirstName, Gender*. The following query templates were used for executing database queries. Each query was executed five times using different values, and the average query response was used.

Q1: SELECT * FROM main WHERE Number = value
Q2: SELECT * FROM main WHERE FirstName = value AND Gender = value
Q3: SELECT * FROM main WHERE FirstName = value AND LastName = value
Q4: SELECT * FROM main WHERE FirstName = value OR LastName = value

We implemented a B^+-tree as the search data structure (used as attribute index) due to its sub-linear search performance for disk-resident data. We note that for query Q3, we

did not build a combined index, but, for sake of performance comparisons, we implemented an alternative conjunction protocol (a work in progress, omitted here) with yet unclear privacy properties. Also, for Q4, we implemented a disjunction protocol where the parties simply run the query protocol for both queries at the disjuncts (also a work in progress, since disjunction protocols with improved privacy may be desirable).

Results. Figures 3, 4 show the performance results for our protocol π_2 and MySQL, respectively. For all database sizes, Q1-Q4 matched 1, 500, 1000 and 2000 records, respectively. We observe that the database size has minimal impact on query response time. This was expected since the same number of records were matched by each query.

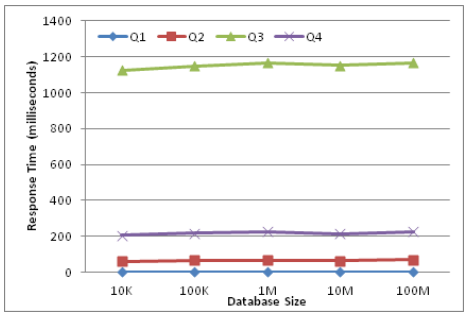

Fig. 3. Our 3-party query performance

Fig. 4. MySQL query performance

Our protocol π_2's performance ($\sim$2ms) is better than MySQL ($\sim$5ms) for Q1 despite the additional overhead of SSL/TLS communications, because of the minimalistic approach used by our implementation, in terms of simpler query execution and data structures, compared to traditional database management systems. When the number of matched records increases, MySQL outperforms our protocol by a factor of around 8.6, 119.1, and 14.4 for Q2, Q3, and Q4, respectively. Unlike MySQL, where once the first match is found in the B^+-tree data structure used for the queried column then a simple forward scan of the tree leaf pages is needed for locating all matches, our protocol performs a separate scan of the search data structure (a B^+-tree) for each matched value. Because these scans are for ciphertext values, it is highly unlikely that any two scans will traverse the same path from the root of the B^+-tree down to the same leaf page. Thus, the number of I/O operations required for fetching B^+-tree leaf pages into the in-memory cache for locating matched records is proportional to the number of matched records. Our protocol's response times in Q3 are more than 17 times slower than response times in Q2 despite the fact that they have the same structure and, in addition, Q3 matched twice as many records as Q2. Furthermore, Q3 is 5 times slower than Q4 although Q4 resulted in twice as many matched records. The main reasons for this are the inefficiency of the protocol used for Q3 as well as the fact that Q2 was answered using a combined index, while Q3 and Q4 required scanning of two B^+-tree indices.

Acknowledgements. This work was supported by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior National Business Center (DoI/

NBC). The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation hereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoI/NBC, or the U.S. Government.

References

1. Project gutenberg. *http://www.gutenberg.org*.
2. D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *EUROCRYPT*, pages 506–522, 2004.
3. B. Chor, N. Gilboa, and M. Naor. Private information retrieval by keywords. *IACR Cryptology ePrint Archive*, 1998.
4. B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.
5. T. M. Cover and J. A. Thomas. *Elements of information theory (2. ed.)*. Wiley, 2006.
6. G. Di Crescenzo, J. Feigenbaum, D. Gupta, E. Panagos, J. Perry, and R. N. Wright. Practical and privacy-preserving policy compliance for outsourced data. In *WAHC*, 2014.
7. G. Di Crescenzo, Y. Ishai, and R. Ostrovsky. Universal service-providers for private information retrieval. *J. Cryptology*, 14(1):37–74, 2001.
8. G. Di Crescenzo and D. Shallcross. On minimizing the size of encrypted databases, In *DBSec*, 2014.
9. M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In *TCC*, pages 303–324, 2005.
10. O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.
11. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.
12. O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious RAMs. *J. ACM*, 43(3):431–473, 1996.
13. H. Hacigümüs, B. R. Iyer, C. Li, and S. Mehrotra. Executing SQL over encrypted data in the database-service-provider model. In *SIGMOD Conference*, pages 216–227, 2002.
14. M. S. Islam, M. Kuzu, and M. Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *NDSS*, 2012.
15. S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner. Outsourced symmetric private information retrieval. In *ACM Conference on Computer and Communications Security*, pages 875–888, 2013.
16. S. Jarecki and X. Liu. Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In *TCC*, pages 577–594, 2009.
17. E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *FOCS*, pages 364–373, 1997.
18. R. Ostrovsky and W. Skeith. A survey of single-database private information retrieval: Techniques and applications. In *Public Key Cryptography*, pages 393–411, 2007.
19. D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, pages 44–55, 2000.
20. S. Wang, X. Ding, R. H. Deng, and F. Bao. Private information retrieval using trusted hardware. In *ESORICS*, pages 49–64, 2006.