SQL on Structurally-Encrypted Databases

Seny Kamara *
Brown University

Tarik Moataz†
Telecom Bretagne &
Colorado State University

Abstract

We show how to encrypt a relational database in such a way that it can efficiently support a large class of SQL queries. Our construction is based solely on structured encryption and does not make use of any property-preserving encryption (PPE) schemes such as deterministic and order-preserving encryption. As such, our approach leaks considerably less than PPE-based solutions which have recently been shown to reveal a lot of information in certain settings (Naveed et al., *CCS '15*). Our construction achieves asymptotically optimal query complexity under very natural conditions on the database and queries.

1 Introduction

The problem of encrypted search has received a lot of attention from Industry, Academia and Government due to its potential applications to cloud computing and database security. Most of the progress in this area, however, has been in the setting of keyword search on encrypted documents. While this has many applications in practice (e.g., email, NoSQL databases, desktop search engines, cloud document storage), much of the data produced and consumed in practice is stored and processed in *relational databases*. A relational database is, roughly speaking, a set of tables with rows representing entities/items and columns representing their attributes. The relational database model was proposed by Codd [14] and most relational DBs are queried using the structured query language (SQL) which is a special-purpose declarative language introduced by Chamberlain and Boyce [10].

The problem of encrypted relational DBs is the "holy-grail" of database security. As far as we know, it was first explicitly considered by Hacigümüs, Iyer, Li and Mehrotra [20] who described a quantization-based approach which leaks the range within which an item falls. In [29], Popa, Redfield, Zeldovich and Balakrishnan describe a system called CryptDB that can support a non-trivial subset of SQL on encrypted relational DBs without quantization. CryptDB achieves this in part by making use of property-preserving encryption (PPE) schemes like deterministic and order-preserving (OPE) encryption, which reveal equality and order, respectively. The high-level technique is to replace the plaintext operations needed to execute a SQL query (e.g., equality tests

*seny@brown.edu. Work done in part at Microsoft Research.
†tarik.moataz@colostate.edu. Work done in part at Microsoft Research.

and comparisons) by the same operations on PPE-encrypted ciphertexts. This approach was later adopted by other systems including Cipherbase [2] and SEEED [19]. While this leads to systems that are efficient and legacy-friendly, it was recently shown by Naveed, Kamara and Wright [26], that such PPE-based EDB systems can leak a lot of information when used in certain settings like electronic medical records (EMRs). In light of this result, the major open problem in encrypted search and, more generally, in database security is whether it is possible to efficiently execute SQL queries on encrypted DBs with less leakage than the PPE-based solutions.

Our contributions. In this work, we address this problem and propose the first solution for SQL on encrypted DBs that does not make use of either PPE or general-purpose primitives like fully-homomorphic encryption (FHE) or oblivious RAM (ORAM). As such, our scheme leaks considerably less than any of the previously-known practical approaches and is much more practical than any leakage-free solution. Our approach is efficient and handles a large sub-class of SQL queries and an even larger class if we allow for a small amount of post-processing at the client.

More precisely, our construction handles the class of *conjunctive queries* [1] [11] which corresponds to SQL queries of the form

$$\mathsf{Select}\ \textit{attributes}\ \mathsf{From}\ \textit{tables}\ \mathsf{Where}\ (\mathsf{a}_1 = X_1 \wedge \cdots \wedge \mathsf{a}_\ell = X_\ell),$$

where a_1 through a_n are attributes in the DB schema and X_1 through X_n are either attributes or constants. While the class of conjunctive queries is smaller than the class supported by the PPE-based solutions, it is one of the most well-studied and useful classes of queries. Furthermore, as mentioned above, if one allows for a small amount of post-processing at the client, we show how to extend the expressiveness of our solution to a much wider sub-class.

With respect to efficiency, we show that the query complexity of our scheme is asymptotically optimal in time and space when $t, s_1, \ldots, s_t, h \ll m$, where t denotes the number of tables in the query, s_i denotes the number of columns in the ith table, h denotes the number of attributes in the Select term of the query and m is the number of rows in the tables (assuming, for ease of exposition, that all tables have m rows). Note that this is a very natural condition which is satisfied in practice by most SQL databases and queries. [2].

1.1 Our Techniques

The PPE-based approach to EDBs essentially replaces the plaintext execution of a SQL query with an encrypted execution of the query by executing the server's low-level operations (i.e., comparisons and equality tests) directly on the encrypted cells. This can be done thanks to the properties of PPE which guarantee that operations on plaintexts can be done on ciphertexts as well. This "plug-and-play" approach makes the design of EDBs relatively straightforward since the only requirement is to replace plaintext cells with PPE-encrypted cells. Given the complexity of relational DBs and of SQL queries it is not a-priori clear how to solve this problem without PPE or without resorting to general-purpose solutions like fully-homomorphic encryption (FHE) or oblivious RAM (ORAM).

[1] We stress that conjunctive queries in the context of relational databases (and as used throughout this work) is completely unrelated to conjunctive *keyword* queries as studied in the searchable encryption literature (e.g., in [8]). In particular, our scheme does not make use of any searchable encryption schemes for conjunctive keyword queries.

[2] Our approach incurs only a small asymptotic overhead even when these parameters are unrealistically large and on the order of m

Conceptual approach. Our first step towards a solution is in isolating some of the conceptual difficulties of the problem. Relational DBs are relatively simple from a data structure perspective since they just consist of a set of two-dimensional arrays. The high-level challenge stems from SQL and, in particular, from its complexity (it can express first-order logic) and the fact that it is *declarative*. To overcome this we restrict ourselves to a simpler but widely applicable and well-studied subset of SQL queries (see above) and we take a more procedural view of SQL. More precisely, we work with the *relational algebra* formulation of SQL which is more amenable to cryptographic techniques. The relational algebra was introduced by Codd [14] as a way to formalize queries on relational databases. Roughly speaking, it consists of all the queries that can be expressed from a set of basic operations. It was later shown by Chandra and Merlin [11] that three of these operations (selection, projection and cross product) capture a large class of useful queries called *conjunctive queries* that have particularly nice theoretical properties. Since their introduction, conjunctive queries have been studied extensively in the database literature.

The subset of the relational algebra expressed by the selection, projection and Cartesian product operators is also called the *SPC algebra*. By working in the SPC algebra, we not only get a procedural representation of SQL queries, but we also reduce the problem to handling just three basic operations. Conceptually, this is reminiscent of the benefits one gets by working with circuits in secure multi-party computation and FHE. Another important advantage of working in the SPC algebra is that it admits a *normal form*; that is, every SPC query can be written in a standard form. By working with this normal form, we get another benefit of general-purpose solutions which are that we can design and analyze a *single* construction that handles *all* SPC queries. Note, however, that like circuit representations the SPC normal form is not always guaranteed to be the most efficient.

The SPC algebra. As mentioned, the SPC algebra consists of all queries that can be expressed by a combination of the select, project and cross product operators which, at a high-level, work as follows. The select operator σ_{Ψ} takes as input a table T and outputs the rows of T that satisfy the predicate Ψ. The project operator $\pi_{\mathsf{a}_1,\ldots,\mathsf{a}_h}$ takes as input a table T and outputs the columns of T indexed by $\mathsf{a}_1,\ldots,\mathsf{a}_h$. Finally, the cross product operator $\mathsf{T}_1 \times \mathsf{T}_2$ takes two tables as input and outputs a third table consisting of rows in the Cartesian product of T_1 and T_2 when viewed as sets of rows. An SPC query in normal form over a database $\mathsf{DB} = (\mathsf{T}_1,\ldots,\mathsf{T}_n)$ has the form,

$$\pi_{\mathsf{a}_1,\cdots,\mathsf{a}_h}\Big([a_1] \times \cdots [a_f] \times \sigma_{\Psi}(\mathsf{T}_{i_1} \times \cdots \times \mathsf{T}_{i_t})\Big),$$

where Ψ is of the form $\mathsf{a}_1 = X_1 \wedge \cdots \wedge \mathsf{a}_\ell = X_\ell$ where $\mathsf{a}_1,\ldots,\mathsf{a}_\ell$ are attributes in the schema of DB and $X_1,\ldots,X_\ell$ are either attributes or constants. So, concretely, our problem reduces to the problem of encrypting a relational database $\mathsf{DB} = (\mathsf{T}_1,\ldots,\mathsf{T}_n)$ in such a way that it can support SPC queries in normal form.

Structured encryption & constructive queries. The concrete problem stated above is a structured encryption problem: namely, how can we encrypt a data structure (i.e., DB) such that it can support a specific type of query (i.e., SPC normal form queries). Structured encryption (STE) is a generalization of searchable symmetric encryption (SSE) introduced by Chase and Kamara [12] which allows a client to encrypt a data structure so that it can be privately queried. As discussed above, relational DBs are relatively simple structures (i.e., two-dimensional arrays)

but the queries they support are complex. This is in opposition to most STE schemes which handle relatively complex structures such as dictionaries [12, 7], multi-maps (i.e., inverted indexes) [15, 23, 8], trees [22], graphs [12, 25] and web graphs [12], but support relatively simple queries. The main difficulty in the case of relational DBs and, in particular, in handling SPC queries is that queries are *constructive* in the sense that they produce new data structures from the original base structure. Intuitively, handling constructive queries (without interaction) is particularly challenging because the intermediate and final structures that have to be created by the server to answer the query are *dependent* on the query and, therefore, cannot be constructed by the client in the setup/pre-processing phase.

An important observation about relational DBs that underlies our approach, however, is that while SPC queries are constructive, they are not arbitrarily so. In other words, the tables needed to answer an SPC query are not completely arbitrary but are structured in a way that can be predicted at setup. What is query-dependent is the *content* of these tables but, crucially, all of that content is already stored in the original database. So the challenge then is to provide the server with the means to construct the appropriate intermediate and final tables and to design encrypted structures that will allow it to efficiently find the (encrypted) content it needs to populate those tables.

Cross products & the location map. By taking a closer look at the SPC normal form, one can see that the first intermediate table needed to answer a query is the cross product $\mathsf{T}' = \mathsf{T}_{i_1} \times \cdots \mathsf{T}_{i_t}$. Ignoring the cross products with $[a_1], \ldots, [a_f]$ for ease of exposition, the remaining intermediate tables as well as the final table are "sub-tables" of T' that result from selecting a subset of rows (according to Ψ) and keeping a subset of columns (according to $\mathsf{a}_1, \ldots, \mathsf{a}_h$). As we show in Section 3, it turns out that from only t and the dimensions of T_{i_1} through T_{i_t}, we can derive not only the dimensions of T' but also a map Λ that maps any cell in DB to its location in T'. We refer to this map as the *location map* and provide it to the server so that it can place the necessary encrypted content in T'. To help the server find the necessary content in the first place, we create a set of encrypted structures which, roughly speaking, can be viewed as different representations of the database. For example, one of the encrypted structures we build provides a row-wise representation of the database whereas another one provides a column-wise representation. By using these various representations and by "combining" them in an appropriate manner, we can generate tokens for the server to recover the (encrypted) database cells that result from applying the select and projection operators to T'. Once the server can recover those encrypted cells it can place them in T' using the location map and prune T' to produce the final table.

Note that, as described, our solution is not practical as the server needs to generate and operate on a m^t-by-$\sum_{j=1}^{t} s_{i_j}$ table, where s_{i_j} denotes the number of columns in T_{i_j}. Fortunately, this not necessary (we describe this naive approach in Section 5 only for ease of exposition) and queries can be processed in optimal time and space using a more complex algorithm which we describe in Section 5.1.

A note on our techniques. We stress that our approach to handling the SPC algebra is completely novel and has no resemblance to how these queries are usually handled on plaintext databases. In other words, our approach does not simply replicate standard data structures and algorithms from databases. In fact, as far as we know, currently the only way to re-use standard data structures and algorithms on encrypted data is to use PPE as done, for example, in [29, 2, 19].

2 Related Work

Searchable & structured encryption. Encrypted search was first considered explicitly by Song, Wagner and Perrig in [30] which introduced the notion of searchable symmetric encryption (SSE). Goh provided the first security definition for SSE and a solution based on Bloom filters with linear search complexity. Chang and Mitzenmacher proposed an alternative security definition and construction, also with linear search complexity. Curtmola, Garay, Kamara and Ostrovsky introduced and formulated the notion of adaptive semantic security for SSE [15] together with optimal-time and optimal-space constructions. Chase and Kamara introduced the notion of structured encryption which generalizes SSE to arbitrary data structures [12]. Kurosawa and Ohtaki proposed a universally composable variant of adaptive semantic security [24]. Kamara, Papamanthou and Roeder [23] gave the first optimal-time dynamic SSE scheme. Cash, Jutla, Jarecki, Krawczyk, Rosu and Steiner [8] proposed the first optimal-time scheme that handles conjunctive keyword search and Faber, Krawczyk, Jarecki, Nguyen, Rosu and Steiner show how to extend it to rich queries (e.g., range, substring and wildcard queries) [16] . Naveed, Prabakharan and Gunther [27] propose an optimal-time dynamic SSE scheme based on blind storage. Cash, Jaeger, Jarecki, Jutla, Krawczyk, Rosu and Steiner [7] show how to construct optimal-time SSE schemes with low I/O complexity and Cash and Tessaro [9] gave lower bounds on the locality of adaptively-secure SSE schemes. SSE has also been considered in the multi-user setting [15, 21]. Pappas, Krell, Vo, Kolesnikov, Malkin, Choi, George, Keromytis, Bellovin [28] present another approach for multi-user encrypted search based on garbled circuits and Bloom filters that can support Boolean formulas, ranges and stemming. Other approaches for encrypted search include oblivious RAMs (ORAM) [18], secure multi-party computation [4], functional encryption [6] and fully-homomorphic encryption [17] as well as solutions based on deterministic encryption [3] and order-preserving encryption (OPE) [5].

Encrypted relational databases. As far as we know the first encrypted relational DB solution was proposed by Hacigümüs, Iyer, Li and Mehrotra [20] and was based on quantization. Roughly speaking, the attribute space of each column is partitioned into bins and each element in the column is replaced with its bin number. Popa, Redfield, Zeldovich and Balakrishnan proposed CryptDB [29]. CryptDB was the first non-quantization-based solution and can handle a large subset of SQL. Instead of quantization, CryptDB relies on PPE like deterministic [3] and OPE [1, 5]. The CryptDB design influenced the Cipherbase system from Arasu *et al.* [2] and the SEEED system from Grofig *et al.* [19]. In [26], Naveed, Kamara and Wright study the security of these PPE-based solutions in the context of medical data.

3 Preliminaries

Notation. The set of all binary strings of length n is denoted as $\{0,1\}^n$, and the set of all finite binary strings as $\{0,1\}^*$. $[n]$ is the set of integers $\{1,\ldots,n\}$. We write $x \leftarrow \chi$ to represent an element x being sampled from a distribution χ, and $x \xleftarrow{\$} X$ to represent an element x being sampled uniformly at random from a set X. The output x of an algorithm $\mathcal{A}$ is denoted by $x \leftarrow \mathcal{A}$. Given a sequence $\mathbf{v}$ of n elements, we refer to its ith element as v_i or $\mathbf{v}[i]$. If S is a set then $\#S$ refers to its cardinality. If s is a string then $|s|$ refers to its bit length.

Basic cryptographic primitives. A private-key encryption scheme is a set of three polynomial-time algorithms $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ such that Gen is a probabilistic algorithm that takes a security parameter k and returns a secret key K; Enc is a probabilistic algorithm that takes a key K and a message m and returns a ciphertext c; Dec is a deterministic algorithm that takes a key K and a ciphertext c and returns m if K was the key under which c was produced. Informally, a private-key encryption scheme is secure against chosen-plaintext attacks (CPA) if the ciphertexts it outputs do not reveal any partial information about the plaintext even to an adversary that can adaptively query an encryption oracle.

Data types. An *abstract data type* is a collection of objects together with a set of operations defined on those objects. Examples include sets, dictionaries (also known as key-value stores or associative arrays) and graphs. The operations associated with an abstract data type fall into one of two categories: query operations, which return information about the objects; and update operations, which modify the objects. If the abstract data type supports only query operations it is *static*, otherwise it is *dynamic*. In this work we only consider static types with a single operation.

Data structures. A *data structure* for a given data type is a representation in some computational model [3] of an object of the given type. Typically, the representation is optimized to support the type's query operation as efficiently as possible. For data types that support multiple queries, the representation is often optimized to efficiently support as many queries as possible. As a concrete example, the dictionary *type* can be represented using various data structures depending on which queries one wants to support efficiently. Hash tables support Get and Put in expected $O(1)$ time whereas balanced binary search trees support both operations in worst-case $\log(n)$ time. For ease of understanding and to match colloquial usage, we sometimes blur the distinction between data types and structures. So, for example, when referring to a *dictionary structure* or a *multi-map structure* what we are referring to is an unspecified instantiation of the dictionary or multi-map data type.

We make use of several basic data types including arrays, dictionaries and multi-maps which we recall here. An array A of capacity n stores n items at locations 1 through n and supports read and write operations. We write $v := \mathsf{A}[i]$ to denote reading the item at location i and $\mathsf{A}[i] := v$ the operation of storing an item at location i. A dictionary DX of capacity n is a collection of n label/value pairs $\{(\ell_i, v_i)\}_{i \leq n}$ and supports get and put operations. We write $v_i := \mathsf{DX}[\ell_i]$ to denote getting the value associated with label ℓ_i and $\mathsf{DX}[\ell_i] := v_i$ to denote the operation of associating the value v_i in DX with label ℓ_i. A multi-map MM with capacity n is a collection of n label/tuple pairs $\{(\ell_i, \mathbf{t}_i)_i\}_{i \leq n}$ that supports get and put operations. Similarly to dictionaries, we write $\mathbf{t}_i := \mathsf{MM}[\ell_i]$ to denote getting the tuple associated with label ℓ_i and $\mathsf{MM}[\ell_i] := \mathbf{t}_i$ to denote operation of associating the tuple $\mathbf{t}_i$ to label ℓ_i. Multi-maps are the abstract data type instantiated by an inverted index. In the encrypted search literature multi-maps are sometimes referred to as indexes, databases or tuple-sets (T-sets). We refer to the set of all possible queries a data structure supports as its *query space* and to the set of its possible responses as its *response space*. For some data structure DS we sometimes write $\mathsf{DS} : \mathbf{Q} \rightarrow \mathbf{R}$ to mean that DS has query and response spaces $\mathbf{Q}$ and $\mathbf{R}$, respectively.

Relational databases. A relational database $\mathsf{DB} = (\mathsf{T}_1, \ldots, \mathsf{T}_n)$ is a set of *tables* where each table T_i is a two-dimensional array with rows corresponding to an entity (e.g., a customer or an employee) and columns corresponding to attributes (e.g., age, height, salary). For any given

[3] In this work, the underlying model will always be the word RAM.

attribute, we refer to the set of all possible values that it can take as its *space* (e.g., integers, booleans, strings). We define the *schema* of a table T to be its set of attributes and denote it $\mathbb{S}(\mathsf{T})$. The schema of a database $\mathsf{DB} = (\mathsf{T}_1, \ldots, \mathsf{T}_n)$ is then the set $\mathbb{S}(\mathsf{DB}) = \bigcup_i \mathbb{S}(\mathsf{T}_i)$. We assume the attributes in $\mathbb{S}(\mathsf{DB})$ are unique and represented as positive integers. We denote a table T's number of rows as $\|\mathsf{T}\|_r$ and its number of columns as $\|\mathsf{T}\|_c$.

We sometimes view tables as a tuple of rows and write $\mathbf{r} \in \mathsf{T}$ and sometimes as a tuple of columns and write $\mathbf{c} \in \mathsf{T}^\intercal$. Similarly, we write $\mathbf{r} \in \mathsf{DB}$ and $\mathbf{c} \in \mathsf{DB}^\intercal$ for $\mathbf{r} \in \bigcup_i \mathsf{T}_i$ and $\mathbf{c} \in \bigcup_i \mathsf{T}_i^\intercal$, respectively. For a row $\mathbf{r} \in \mathsf{T}_i$, its table identifier $\mathsf{tbl}(\mathbf{r})$ is i and its row rank $\mathsf{rrk}(\mathbf{r})$ is its position in T_i when viewed as a tuple of rows. Similarly, for a column $\mathbf{c} \in \mathsf{T}_i^\intercal$, its table identifier $\mathsf{tbl}(\mathbf{c})$ is i and its column rank $\mathsf{crk}(\mathbf{c})$ is its position in T_i when viewed as a tuple of columns. For any row $\mathbf{r} \in \mathsf{DB}$ and column $\mathbf{c} \in \mathsf{DB}^\intercal$, we refer to the pairs $(\mathsf{tbl}(\mathbf{r}), \mathsf{rrk}(\mathbf{r}))$ and $(\mathsf{tbl}(\mathbf{c}), \mathsf{crk}(\mathbf{c}))$, respectively, as their *coordinates* in DB. We write $\mathbf{r}[i]$ and $\mathbf{c}[i]$ to refer to the ith element of a row $\mathbf{r}$ and column $\mathbf{c}$. The coordinate of the jth cell in row $\mathbf{r} \in \mathsf{T}_i$ is the triple $(i, \mathsf{rrk}(\mathbf{r}), j)$. Given a column $\mathbf{c} \in \mathsf{DB}^\intercal$, we denote its corresponding attribute by $\mathsf{att}(\mathbf{c})$. For any pair of attributes $\mathsf{a}_1, \mathsf{a}_2 \in \mathbb{S}(\mathsf{DB})$ such that $\mathsf{dom}(\mathsf{a}_1) = \mathsf{dom}(\mathsf{a}_2)$, $\mathsf{DB}_{\mathsf{a}_1=\mathsf{a}_2}$ denotes the set of row pairs $\{(\mathbf{r}_1, \mathbf{r}_2) \in \mathsf{DB}^2 : \mathbf{r}_1[\mathsf{a}_1] = \mathbf{r}_2[\mathsf{a}_2]\}$. For any attribute $\mathsf{a} \in \mathbb{S}(\mathsf{DB})$ and constant $a \in \bigcup_{\mathsf{a} \in \mathbb{S}(\mathsf{DB})} \mathsf{dom}(\mathsf{a})$, $\mathsf{DB}_{\mathsf{a}=a}$ is the set of rows $\{\mathbf{r} \in \mathsf{DB} : \mathbf{r}[\mathsf{a}] = a\}$.

SQL. In practice, relational databases are queried using the special-purpose language SQL, introduced by Chamberlain and Boyce [10]. SQL is a declarative language and can be used to modify and query a relational DB. In this work, we only focus on its query operations. Informally, SQL queries typically have the form

$$\mathsf{Select}\ \textit{attributes}\ \mathsf{From}\ \textit{tables}\ \mathsf{Where}\ \textit{condition},$$

where *attributes* is a set of attributes/columns, *tables* is a set of tables and *condition* is a predicate over the rows of *tables* and can itself contain a nested SQL query. More complex queries can be obtained using Group-by, Order-by and aggregate operators (i.e., max, min, average etc.) but the simple form above already captures a large subset of SQL. The most common class of queries on relational DBs are *conjunctive queries* [11] which have the above form with the restriction that *condition* is a conjunction of equalities over attributes and constants. In particular, this means there are no nested queries in *condition*. More precisely, conjunctive queries have the form

$$\mathsf{Select}\ \textit{attributes}\ \mathsf{From}\ \textit{tables}\ \mathsf{Where}\ (\mathsf{a}_1 = X_1 \wedge \cdots \wedge \mathsf{a}_\ell = X_\ell),$$

where a_i is an attribute in $\mathbb{S}(\mathsf{DB})$ and X_i can be either an attribute or a constant.

The SPC algebra. It was shown by Chandra and Merlin [11] that conjunctive queries could be expressed as a subset of Codd's relational algebra which is an imperative query language based on a set of basic operators. In particular, they showed that three operators *select*, *project* and *cross product* were enough. The *select* operator σ_Ψ is parameterized with a predicate Ψ and takes as input a table T and outputs a new table T' that includes the rows of T that satisfy the predicate Ψ. The *projection* operator $\pi_{\mathsf{a}_1,\ldots,\mathsf{a}_h}$ is parameterized by a set of attributes $\mathsf{a}_1, \ldots, \mathsf{a}_h$ and takes as input a table T and outputs a table T' that consists of the columns of T indexed by a_1 through a_n. The *cross product* operator $\times$ takes as input two tables T_1 and T_2 and outputs a new table $\mathsf{T}' = \mathsf{T}_1 \times \mathsf{T}_2$ such that each row of T' is an element of the Cartesian product between the set of rows of T_1 and the set of rows of T_2. The query language that results from any combination of select, project and cross product is referred to as the *SPC algebra*. We formalize this in Definition 3.1 below.

Definition 3.1 (SPC algebra)**.** *Let* $\mathsf{DB} = (\mathsf{T}_1, \ldots, \mathsf{T}_n)$ *be a relational database. The SPC algebra consists of any query that results from the combination of the following operators:*

- $\mathsf{T}' \leftarrow \sigma_\Psi(\mathsf{T})$*: the* select *operator is parameterized with a predicate* Ψ *of form* $\mathsf{a}_1 = X_1 \wedge \cdots \wedge \mathsf{a}_\ell = X_\ell$*, where* $\mathsf{a}_i \in \mathbb{S}(\mathsf{DB})$ *and* X_i *is either a constant* a *in the domain of* a_i *(type-1) or an attribute* $\mathsf{x}_j \in \mathbb{S}(\mathsf{DB})$ *(type-2). It takes as input a table* $\mathsf{T} \in \mathsf{DB}$ *and outputs a table* $\mathsf{T}' = \{\mathbf{r} \in \mathsf{T} : \Psi(\mathbf{r}) = 1\}$*, where terms of the form* $\mathsf{a}_i = \mathsf{x}_j$ *are satisfied if* $\mathbf{r}(\mathsf{a}_i) = \mathbf{r}(\mathsf{x}_j)$ *and terms of the form* $\mathsf{a}_i = a$ *are satisfied if* $\mathbf{r}(\mathsf{a}_i) = a$*.*

- $\mathsf{T}' \leftarrow \pi_{\mathsf{a}_1,\ldots,\mathsf{a}_h}(\mathsf{T})$*: the* project *operator is parameterized by a set of attributes* $\mathsf{a}_1, \ldots, \mathsf{a}_h \in \mathbb{S}(\mathsf{DB})$*. It takes as input a table* $\mathsf{T} \in \mathsf{DB}$ *and outputs a table* $\mathsf{T}' = \{\langle \mathbf{r}(\mathsf{a}_1), \ldots, \mathbf{r}(\mathsf{a}_h)\rangle : \mathbf{r} \in \mathsf{T}\}$*.*

- $\mathsf{T}_3 \leftarrow \mathsf{T}_1 \times \mathsf{T}_2$*: the* cross product *operator takes as input two tables* T_1 *and* T_2 *and outputs a table* $\mathsf{T}_3 = \{\langle \mathbf{r}, \mathbf{v}\rangle : \mathbf{r} \in \mathsf{T}_1 \text{ and } \mathbf{v} \in \mathsf{T}_2\}$*, where* $\langle \mathbf{r}, \mathbf{v}\rangle$ *is the concatenation of rows* $\mathbf{r}$ *and* $\mathbf{v}$*.*

Intuitively, the connection between conjunctive SQL queries and the SPC algebra can be seen as follows: Select corresponds to the projection operator, From to the cross product and Where to the (SPC) select operator.

SPC normal form. Any query in the SPC algebra can be reduced to a *normal form* using a certain set of well-known identities. The normal form of an SPC query over a relational database $\mathsf{DB} = (\mathsf{T}_1, \ldots, \mathsf{T}_n)$ has the form:

$$\pi_{\mathsf{a}_1,\cdots,\mathsf{a}_h}\Big([a_1] \times \cdots [a_f] \times \sigma_\Psi(\mathsf{T}_{i_1} \times \cdots \times \mathsf{T}_{i_\ell})\Big),$$

where $a_1, \ldots, a_f \in \bigcup_{\mathsf{a}\in\mathbb{S}(\mathsf{DB})} \mathsf{dom}(\mathsf{a})$ and $[a_j]$ is the 1×1 table that holds a_j. Here, the attributes $\mathsf{a}_1, \ldots, \mathsf{a}_h$ in the projection are either in $\mathbb{S}(\mathsf{DB})$ or refer to the columns generated by $[a_1]$ through $[a_h]$. In the latter case, we say that they are *virtual attributes* and are in $\mathbb{S}(\mathsf{VDB})$, where VDB is the *virtual database* defined as $\mathsf{VDB} = ([a_1], \ldots, [a_f])$.

We note that converting SQL queries of the above form to SPC queries is a well-studied problem with highly-optimized solutions. In particular, the SPC queries that result from such a translation are "compact" in the sense that the number of projects, selects and cross products in the resulting SPC query is the same as the number of attributes, tables and conditions, respectively, in the original SQL query (for an overview of SQL-to-SPC translation we refer the reader to [31]). In addition, the transformation of SPC queries into their normal form can also be done efficiently based on a standard set of identities and also results in "compact" normal form queries in the sense above.

The location map. An important notion for our purposes is what we refer to as the *location map* which, intuitively, maps the cell coordinates in DB to their location in the cross product $\mathsf{T}_{i_1} \times \cdots \times \mathsf{T}_{i_t}$. Here, we assume the tables $\mathsf{T}_{i_1}, \ldots, \mathsf{T}_{i_t}$ all have the same number of rows m. More precisely, let M be a $m^t \times \sum_i s_i$ two-dimensional array. The location map $\Lambda_{t,m,\mathbf{s}}$ is parameterized by the number of tables in the cross product $t \geq 1$, the size of the tables $m \geq 1$ and a vector $\mathbf{s} = (s_1, \ldots, s_t)$ such that $s_i = \|\mathsf{T}_i\|_c$. It takes as input the coordinates of any cell in DB and outputs

a set of locations $(\alpha, \beta) \in [m^t] \times [\sum_i s_i]$ in M. Concretely, the location map is defined as

$$\Lambda_{t,m,\mathbf{s}}(z,i,j) = \Big\{ \Big((i-1) \cdot m^{t-z} + h \cdot m^{t-z+1}, j + \sum_{\ell=1}^{z-1} \|\mathsf{T}_\ell\|_c \Big), \ldots, \Big(i \cdot m^{t-z} + h \cdot m^{t-z+1}, j + \sum_{\ell=1}^{z-1} \|\mathsf{T}_\ell\|_c \Big) \Big\}_{h \in \{1,\cdots,m^{z-1}\}},$$

where $z \in \{i_1, \ldots, i_t\}$, $i \in [m]$ and $j \in \|\mathsf{T}_z\|_c$.

Note that given a position $(\alpha, \beta) \in [m^t] \times [\sum_i s_i]$, and a specific order of tables $\mathsf{T}_{i_1}, \cdots, \mathsf{T}_{i_z}$, the original coordinates of the cell in the tables can be computed *efficiently* such that

$$(z,i,j) = \Lambda^{-1}_{t,m,\mathbf{s}}(\alpha, \beta),$$

where $i = \alpha \bmod m^{t-z}$ and $j = \beta \bmod \sum_{\ell=1}^{z} \|\mathsf{T}_z\|_c$.

4 Definitions

In this Section, we define the syntax and security of STE schemes. A STE scheme encrypts data structures in such a way that they can be privately queried. There are several natural forms of structured encryption. The original definition of [12] considered schemes that encrypt both a structure and a set of associated data items (e.g., documents, emails, user profiles etc.). In [13], the authors also describe *structure-only* schemes which only encrypt structures. Another distinction can be made between *interactive* and *non-interactive* schemes. Interactive schemes produce encrypted structures that are queried through an interactive two-party protocol, whereas non-interactive schemes produce structures that can be queried by sending a single message, i.e, the token. One can also distinguish between *response-hiding* and *response-revealing* schemes: the latter reveal the query response to the server whereas the former do not.

In this work, we focus on non-interactive structure-only schemes. Our main construction, SPX, is response-hiding but makes use of response-revealing schemes as building blocks. As such, we define both forms below. At a high-level, non-interactive STE works as follows. During a setup phase, the client constructs an encrypted structure EDS under a key K from a plaintext structure DS. The client then sends EDS to the server. During the query phase, the client constructs and sends a token tk generated from its query q and secret key K. The server then uses the token tk to query EDS and recover either a response r or an encryption ct of r depending on whether the scheme is response-revealing or response-hiding.

Definition 4.1 (Response-revealing structured encryption [12])**.** *A response-revealing structured encryption scheme* $\Sigma = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Query})$ *consists of three polynomial-time algorithms that work as follows:*

- $(K, \mathsf{EDS}) \leftarrow \mathsf{Setup}(1^k, \mathsf{DS})$*: is a probabilistic algorithm that takes as input a security parameter* 1^k *and a structure* DS *and outputs a secret key* K *and an encrypted structure* EDS*.*
- $\mathsf{tk} \leftarrow \mathsf{Token}(K, q)$*: is a (possibly) probabilistic algorithm that takes as input a secret key* K *and a query* q *and returns a token* tk*.*

- $\{\perp, r\} \leftarrow \mathsf{Query}(\mathsf{EDS}, \mathsf{tk})$*: is a deterministic algorithm that takes as input an encrypted structure* EDS *and a token* tk *and outputs either* $\perp$ *or a response.*

We say that a response-revealing structured encryption scheme Σ *is correct if for all* $k \in \mathbb{N}$*, for all* $\mathsf{poly}(k)$*-size structures* $\mathsf{DS} : \mathbf{Q} \to \mathbf{R}$*, for all* (K, EDS) *output by* $\mathsf{Setup}(1^k, \mathsf{DS})$ *and all sequences of* $m = \mathsf{poly}(k)$ *queries* $q_1, \ldots, q_m$*, for all tokens* tk_i *output by* $\mathsf{Token}(K, q_i)$*,* $\mathsf{Query}(\mathsf{EDS}, \mathsf{tk}_i)$ *returns* $\mathsf{DS}(q_i)$ *with all but negligible probability.*

Definition 4.2 (Response-hiding structured encryption [12])**.** *A response-hiding structured encryption scheme* $\Sigma = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Query}, \mathsf{Dec})$ *consists of four polynomial-time algorithms such that* Setup *and* Token *are as in Definition 4.1 and* Query *and* Dec *are defined as follows:*

- $\{\perp, \mathrm{ct}\} \leftarrow \mathsf{Query}(\mathsf{EDS}, \mathsf{tk})$*: is a deterministic algorithm that takes as input an encrypted structured* EDS *and a token* tk *and outputs either* $\perp$ *or a ciphertext* ct*.*
- $r \leftarrow \mathsf{Dec}(K, \mathrm{ct})$*: is a deterministic algorithm that takes as input a secret key* K *and a ciphertext* ct *and outputs a response* r*.*

We say that a response-hiding structured encryption scheme Σ *is correct if for all* $k \in \mathbb{N}$*, for all* $\mathsf{poly}(k)$*-size structures* $\mathsf{DS} : \mathbf{Q} \to \mathbf{R}$*, for all* (K, EDS) *output by* $\mathsf{Setup}(1^k, \mathsf{DS})$ *and all sequences of* $m = \mathsf{poly}(k)$ *queries* $q_1, \ldots, q_m$*, for all tokens* tk_i *output by* $\mathsf{Token}(K, q_i)$*,* $\mathsf{Dec}_K\Big(\mathsf{Query}\Big(\mathsf{EDS}, \mathsf{tk}_i\Big)\Big)$ *returns* $\mathsf{DS}(q_i)$ *with all but negligible probability.*

Security. The standard notion of security for structured encryption guarantees that an encrypted structure reveals no information about its underlying structure beyond the setup leakage $\mathcal{L}_{\mathsf{S}}$ and that the query algorithm reveals no information about the structure and the queries beyond the query leakage $\mathcal{L}_{\mathsf{Q}}$. If this holds for non-adaptively chosen operations then this is referred to as non-adaptive semantic security. If, on the other hand, the operations are chosen adaptively, this leads to the stronger notion of adaptive semantic security. This notion of security was introduced by Curtmola *et al.* in the context of SSE [15] and later generalized to structured encryption in [12].

Definition 4.3 (Adaptive semantic security [15, 12])**.** *Let* $\Sigma = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Query})$ *be a response-revealing structured encryption scheme and consider the following probabilistic experiments where* $\mathcal{A}$ *is a stateful adversary,* $\mathcal{S}$ *is a stateful simulator,* $\mathcal{L}_{\mathsf{S}}$ *and* $\mathcal{L}_{\mathsf{Q}}$ *are leakage profiles and* $z \in \{0,1\}^*$*:*

- $\mathbf{Real}_{\Sigma,\mathcal{A}}(k)$*: given* z *the adversary* $\mathcal{A}$ *outputs a structure* DS*. It receives* EDS *from the challenger, where* $(K, \mathsf{EDS}) \leftarrow \mathsf{Setup}(1^k, \mathsf{DS})$*. The adversary then* adaptively *chooses a polynomial number of queries* $q_1, \ldots, q_m$*. For all* $i \in [m]$*, the adversary receives* $\mathsf{tk} \leftarrow \mathsf{Token}(K, q_i)$*. Finally,* $\mathcal{A}$ *outputs a bit* b *that is output by the experiment.*
- $\mathbf{Ideal}_{\Sigma,\mathcal{A},\mathcal{S}}(k)$*: given* z *the adversary* $\mathcal{A}$ *generates a structure* DS *which it sends to the challenger. Given* z *and leakage* $\mathcal{L}_{\mathsf{S}}(\mathsf{DS})$ *from the challenger, the simulator* $\mathcal{S}$ *returns an encrypted data structure* EDS *to* $\mathcal{A}$*. The adversary then* adaptively *chooses a polynomial number of operations* $q_1, \ldots, q_m$*. For all* $i \in [m]$*, the simulator receives a tuple* $(\mathsf{DS}(q_i), \mathcal{L}_{\mathsf{Q}}(\mathsf{DS}, q_i))$ *and returns a token* tk_i *to* $\mathcal{A}$*. Finally,* $\mathcal{A}$ *outputs a bit* b *that is output by the experiment.*

We say that Σ *is adaptively* $(\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{Q}})$*-semantically secure if for all* PPT *adversaries* $\mathcal{A}$*, there exists a* PPT *simulator* $\mathcal{S}$ *such that for all* $z \in \{0,1\}^*$*, the following expression is negligible in* k*:*

$$|\Pr[\,\mathbf{Real}_{\Sigma,\mathcal{A}}(k) = 1\,] - \Pr[\,\mathbf{Ideal}_{\Sigma,\mathcal{A},\mathcal{S}}(k) = 1\,]|$$

The security definition for *response-hiding* schemes can be derived from Definition 4.3 by giving the simulator $(\perp, \mathcal{L}_{\mathsf{Q}}(\mathsf{DS}, q_i))$ instead of $(\mathsf{DS}(q_i), \mathcal{L}_{\mathsf{Q}}(\mathsf{DS}, q_i))$.

5 Our Construction

We describe SPX, our structured encryption scheme for SPC queries on relational DBs. More precisely, the scheme encrypts relational database structures DB which we define as tuples of two-dimensional arrays $(\mathsf{T}_1, \ldots, \mathsf{T}_n)$ which support SPC queries in normal form. The scheme makes black-box use of a response-revealing multi-map encryption scheme $\Sigma_{\mathsf{MM}} = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Get})$, of a response-revealing dictionary encryption scheme $\Sigma_{\mathsf{DX}} = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Get})$ and of a symmetric-key encryption scheme $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. Note that encrypted multi-maps and dictionaries can be instantiated using a variety of schemes [15, 12, 23, 8, 7, 27].

Our construction is detailed in Figs. 1, 2 and 3 and we provide high-level overviews of its setup, token generation and query algorithms. We stress that, here, we only describe a *naive* query algorithm that is not efficient but that is relatively easy to understand. In Section 5.1 below, we describe our optimized query algorithm which is optimal under natural conditions.

Setup. The Setup algorithm takes as input a relational database $\mathsf{DB} = (\mathsf{T}_1, \ldots, \mathsf{T}_n)$ and starts by creating three multi-maps MM_R, MM_C and MM_V, each of which stores a different representation of the database. MM_R stores its row-wise representation, by which we mean that it maps row coordinates (i.e., the row table and rank) to encrypted rows. Analogously, MM_C stores a column-wise representation of DB and maps column coordinates (i.e., the column table and rank) to encrypted columns. Setup then encrypts MM_R and MM_C with Σ_{MM}, resulting in EMM_R and EMM_C. It then uses MM_V to store a value-wise representation of the database in the sense that the latter maps the value of a cell to *tokens* for the rows in DB that store that same value (where the tokens are for EMM_R). Again, it uses Σ_{MM} to encrypt MM_V, resulting in EMM_V. Now, recall the SPC normal form:

$$\pi_{\mathsf{a}_1,\cdots,\mathsf{a}_h}\Big([a_1] \times \cdots [a_f] \times \sigma_{\Psi}(\mathsf{T}_{i_1} \times \cdots \times \mathsf{T}_{i_t})\Big).$$

At a high-level, EMM_C will enable projection operations and EMM_V combined with EMM_R will enable type-1 select operations (i.e., of the form $\mathsf{a}_i = a_i$). To support type-2 select operations, however, we need additional structures.

For this, Setup will create, for all columns $\mathbf{c} \in \mathsf{DB}^{\mathsf{T}}$, a multi-map $\mathsf{MM}_{\mathbf{c}}$ that maps pairs of the form

$$\Big\langle \langle \mathsf{tbl}(\mathbf{c}), \mathsf{crk}(\mathbf{c}) \rangle, \langle \mathsf{tbl}(\mathbf{c}'), \mathsf{crk}(\mathbf{c}') \rangle \Big\rangle$$

to tokens for the rows $(\mathbf{r}_1, \mathbf{r}_2) \in \mathsf{DB}_{\mathsf{att}(\mathbf{c})=\mathsf{att}(\mathbf{c}')}$, where $\mathbf{c}' \in \mathsf{DB}^{\mathsf{T}}$ is a column with the same domain as $\mathbf{c}$. It then encrypts each of these multi-maps with Σ_{MM}, resulting in a set $\{\mathsf{EMM}_{\mathbf{c}}\}_{\mathbf{c}\in\mathsf{DB}^{\mathsf{T}}}$, and creates a dictionary DX that maps the attributes of each column $\mathsf{att}(\mathbf{c})$, for all $\mathbf{c} \in \mathsf{DB}^{\mathsf{T}}$, to $\mathsf{EMM}_{\mathbf{c}}$. Finally, it encrypts DX with Σ_{DX}, resulting in an encrypted dictionary EDX.

Setup outputs the key $K = (K_R, K_C, K_V, K_D, \{K_{\mathbf{c}}\}_{\mathbf{c}\in\mathsf{DB}^{\mathsf{T}}}, K_1)$, where K_R, K_C, K_V, K_D and $\{K_{\mathbf{c}}\}_{\mathbf{c}\in\mathsf{DB}^{\mathsf{T}}}$ are the keys for EMM_R, EMM_C, EMM_V, EDX and $\{\mathsf{EMM}_{\mathbf{c}}\}_{\mathbf{c}\in\mathsf{DB}^{\mathsf{T}}}$, respectively and K_1 is a key for SKE. The encrypted database is

$$\mathsf{EDB} = (\mathsf{EMM}_R, \mathsf{EMM}_C, \mathsf{EMM}_V, \mathsf{EDX}).$$

Token. The Token algorithm takes as input a secret key K and an SPC query q in normal form. The algorithm first generates tokens for the projections. Recall that in the SPC normal form, projections can occur either on virtual or real attributes. So, for all $i \in [h]$, if $\mathsf{a}_i \in \mathbb{S}(\mathsf{VDB})$, Token sets $\mathsf{ytk}_i = (1, \mathsf{a}_i)$; otherwise, if $\mathsf{a}_i \in \mathbb{S}(\mathsf{DB})$, it creates a projection token $\mathsf{ptk}_i \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}_{K_C}\left(\left\langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i)\right\rangle\right)$ and sets $\mathsf{ytk}_i = (0, \mathsf{ptk}_i)$.

For every constant a_1 through a_f it computes $\mathrm{e}_{a_1} \leftarrow \mathsf{Enc}_{K_1}(a_1)$ through $\mathrm{e}_{a_f} \leftarrow \mathsf{Enc}_{K_1}(a_f)$. Then, for each term $\mathsf{a}_i = X_i$ of Ψ it does the following. If the term is a type-1 query of the form $\mathsf{a}_i = a_i$, it computes an EMM_V token vtk_i for the pair that consists of the constant a_i and the coordinates of a_i and sets $\mathsf{stk}_i = \mathsf{vtk}_i$. Throughout, we refer to s-tokens of this form as type-1 tokens. On the other hand, if the term is a type-2 query of the form $\mathsf{a}_i = \mathsf{x}_i$, then it computes two tokens dtk_i and jtk_i. The first, dtk_i is an EDX token for the coordinates of a_i and the second, jtk_i, is an $\mathsf{EMM}_{\mathsf{a}_i}$ token for the pair that consists of the coordinates of a_i and x_i. It then sets $\mathsf{stk}_i = (\mathsf{dtk}_i, \mathsf{jtk}_i)$. We refer to s-tokens of this form as type-2 tokens. Token then creates a vector $\mathbf{s}$ that holds the (column) size of $\mathsf{T}_{i_1}, \ldots, \mathsf{T}_{i_t}$ and outputs the token

$$\mathsf{tk} = \left(t, \mathbf{s}, (\mathrm{e}_{a_i})_{i\in[f]}, (\mathsf{ytk}_i)_{i\in[h]}, (\mathsf{stk}_i)_{i\in[\ell]}\right).$$

We first present an *inefficient* but algorithmic clear description of Query in 3. We then provide a realistic *efficient* instantiation of Query in 4 and 5.

Naive Query. We begin by describing naive version of the Query algorithm. While this version is not efficient, it is easier to understand and will serve as an introduction to the more efficient but more complex variant we describe in below. The naive version uses the basic location map defined in Section 3, whereas the efficient variant below uses the extended version.

The Query algorithm takes as input the encrypted database EDB and a token tk. Given the arity of the cross product t, the size of the columns m and the column-sizes $\mathbf{s}$ of the tables in the product, it instantiates an empty $m^t \times \sum_i s_i$ matrix M.

It then processes the s-tokens stk_1 through stk_ℓ. If stk_i is type-1, it uses it to query EMM_V. This results in a tuple $\mathbf{t}$ of row tokens which it then uses to query the EMM_R structure. More precisely, for all $\mathsf{rtk} \in \mathbf{t}$, it computes

$$\left\langle \mathrm{ct}_1, \ldots, \mathrm{ct}_d, \mathrm{tb}, \mathrm{rrk}\right\rangle \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_R, \mathsf{rtk}),$$

where $\langle \mathrm{ct}_1, \ldots, \mathrm{ct}_d\rangle$ is an encrypted row and $(\mathrm{tb}, \mathrm{rrk})$ are its coordinates. Note that these will be encryptions of the rows in DB that hold the constant a_i at column a_i, where a_i and a_i are the attribute/constant pair in the type-1 select query that underlies vtk_i. For each ciphertext ct_i of every encrypted row above, it then computes $(\alpha, \beta) \leftarrow \Lambda_{t,m,\mathbf{s}}(\mathrm{tb}, \mathrm{rrk}, i)$ and sets $\mathsf{M}[\alpha, \beta] := \mathrm{ct}_i$. On the other hand, if stk_i is type-2, the algorithm uses dtk_i to query EDX, resulting in an encrypted multi-map $\mathsf{EMM}_{\mathbf{c}}$ which it then queries with jtk_i. This last operation results in a tuple $\mathbf{t}$ of *pairs* of row tokens for EMM_R. For each pair of tokens, it queries EMM_R, resulting in a pair of encrypted rows with their coordinates. It uses the location map on each pair of coordinates to recover locations (α, β) and (α', β'). If $\alpha = \alpha'$, it further uses the location map to place each ciphertext in the encrypted rows in M.

After processing the s-tokens, it processes the y-tokens $(\mathsf{ytk}_1, \ldots, \mathsf{ytk}_h)$ as follows. It starts by initializing a 1-by-1 matrix VA. If ytk_i has the form $(1, \mathsf{a})$, for some attribute a, it sets $\mathsf{VA} = \mathsf{VA} \times \mathrm{e}_{\mathsf{a}}$. On the other hand, if $\mathsf{ytk}_i = (0, \mathsf{ptk}_i)$, it computes

$$\langle \mathrm{ct}_1, \ldots, \mathrm{ct}_m, \mathrm{tb}, \mathrm{crk} \rangle \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_C, \mathsf{ptk}_i),$$

and for all $j \in [m]$ it sets $\mathsf{M}[\alpha_j, \beta_j] := \mathrm{ct}_j$, where $(\alpha_j, \beta_j) \leftarrow \Lambda_{t,m,\mathbf{s}}(\mathrm{tb}, \mathrm{crk}, j)$. It then sets $\mathsf{ET} := \mathsf{VA} \times \mathsf{M}$ and outputs ET after eliminating all columns that contain at least one empty cell.

Decryption. The Dec algorithm takes as input a secret key K and the encrypted table ET returned by the server and simply decrypts each cell of ET. Note that we omit the formal description of Dec from Figs. 1, 2, 3, 4 and 5.

5.1 An Optimal Query Algorithm

The naive Query algorithm above makes use of a m^t-by-$\sum_{i=1}^{t} s_i$ array which can be very large. Fortunately, we do not need to process queries in this manner and we now describe a more efficient approach that builds the result table directly *without needing to generate the cross product of the tables*. Our optimized Query algorithm is relatively complex compared to the naive algorithm but considerably more efficient. An important component in this variant of the query algorithm is an additional map we refer to as the *row location map*.

The row location map. The row location map takes as input a tuple of b table-identifier/row-rank pairs $((\mathrm{tb}_1, \mathsf{rrk}_1), \ldots, (\mathrm{tb}_b, \mathsf{rrk}_b))$, where $b \leq t$, and outputs a set of locations for these rows in the cross product $\mathsf{M} = \mathsf{T}_{i_1} \times \cdots \times T_{i_t}$. Here, we assume the input to the row location map has the form $((i_1, \mathsf{rrk}_1), \ldots, (i_b, \mathsf{rrk}_b))$; that is, the rows belong to tables T_{i_1} to T_{i_b} (all our inputs to the row location map will have this form). It outputs a row location $\alpha \in [m^t]$ in M. More precisely, the row location map is defined as

$$\Gamma_{t,m,b}((z_1, l_1), \cdots, (z_b, l_b)) = \Big\{(\gamma - 1) \cdot m^{t-b} + 1, \ldots, \gamma \cdot m^{t-b}\Big\},$$

where $\gamma = l_1 \cdot m^{b-1} + l_2 \cdot m^{b-2} + \cdots + l_b$, z_i $(l_1, \cdots, l_b) \in [m]^b$. Note that if $b = t$, $\Gamma_{t,m,b}$ outputs a single location γ in M.

Overview. Our optimized query algorithm is described in detail in Figs. (4) and (5). At a high-level, there are four steps: (1) a recovery step where we process all the s-tokens and recover sets of rows for each token; (2) an intersection step, where we take per-table intersections of these sets (i.e., we take intersections of all sets that hold rows belonging to the same table); (3) an augmented cross-product step where we combine the remaining rows across sets; and (4) a placement step, where we use the location map to place the merged rows in the results table.

We first process the s-tokens. If a token is type-1, we recover a set of encrypted rows together with their coordinates (i.e., their table identifiers and row ranks). If, on the other hand, it is type-2 then we recover *pairs* of coordinates. We denote by R_i the ith *result set*; that is, the set of coordinates (or pair of coordinates) recovered from processing stk_i. If stk_i is type-1 we say that R_i is type-1 and if stk_i is type-2 then we say that R_i is type-2. Note that if stk_i is type-1 then all the coordinates we recover from processing it have the same table identifier. If stk_i is type-2, then the

Let $\Sigma_{\mathsf{DX}} = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Get})$ be a response-revealing dictionary encryption scheme, $\Sigma_{\mathsf{MM}} = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Get})$ be a response-revealing multi-map encryption scheme and $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a symmetric-key encryption scheme. Consider the DB encryption scheme $\mathsf{SPX} = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Query}, \mathsf{Dec})$ defined as follows [a]:

- $\mathsf{Setup}(1^k, \mathsf{DB})$:
  1. initialize a dictionary DX;
  2. initialize multi-maps MM_R, MM_C and MM_V;
  3. initialize multi-maps $(\mathsf{MM}_{\mathsf{a}})_{\mathsf{a} \in \mathbb{S}(\mathsf{DB})}$;
  4. for all $\mathbf{r} \in \mathsf{DB}$ set
  $$\mathsf{MM}_R\Big[\Big\langle \mathsf{tbl}(\mathbf{r}), \mathsf{rrk}(\mathbf{r}) \Big\rangle\Big] := \Big(\mathsf{Enc}_{K_1}(r_1), \ldots \mathsf{Enc}_{K_1}(r_{\#\mathbf{r}}), \mathsf{tbl}(\mathbf{r}), \mathsf{rrk}(\mathbf{r})\Big);$$
  5. compute $(K_R, \mathsf{EMM}_R) \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Setup}\big(1^k, \mathsf{MM}_R\big)$;
  6. for all $\mathbf{c} \in \mathsf{DB}^{\intercal}$, set
  $$\mathsf{MM}_C\Big[\Big\langle \mathsf{tbl}(\mathbf{c}), \mathsf{crk}(\mathbf{c}) \Big\rangle\Big] := \Big(\mathsf{Enc}_{K_1}(c_1), \ldots \mathsf{Enc}_{K_1}(c_{\#\mathbf{c}}), \mathsf{tbl}(\mathbf{c}), \mathsf{crk}(\mathbf{c})\Big);$$
  7. compute $(K_C, \mathsf{EMM}_C) \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Setup}\big(1^k, \mathsf{MM}_C\big)$;
  8. for all $\mathbf{c} \in \mathsf{DB}^{\intercal}$,
     (a) for all $v \in \mathbf{c}$ and $\mathbf{r} \in \mathsf{DB}_{\mathbf{c}=v}$,
        i. compute $\mathsf{rtk}_{\mathbf{r}} \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}_{K_R}\Big(\big\langle \mathsf{tbl}(\mathbf{r}), \mathsf{rrk}(\mathbf{r}) \big\rangle\Big)$,
     (b) set
     $$\mathsf{MM}_V\Big[\Big\langle v, \Big\langle \mathsf{tbl}(\mathsf{att}(\mathbf{c})), \mathsf{crk}(\mathsf{att}(\mathbf{c})) \Big\rangle\Big\rangle\Big] := \Big(\mathsf{rtk}_{\mathbf{r}}\Big)_{\mathbf{r} \in \mathsf{DB}_{\mathbf{c}=v}};$$
  9. compute $(K_V, \mathsf{EMM}_V) \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Setup}\big(1^k, \mathsf{MM}_V\big)$;
  10. for all $\mathbf{c} \in \mathsf{DB}^{\intercal}$,
      (a) for all $\mathbf{c}' \in \mathsf{DB}^{\intercal}$ such that $\mathsf{dom}(\mathsf{att}(\mathbf{c}')) = \mathsf{dom}(\mathsf{att}(\mathbf{c}))$,
         i. initialize an empty tuple $\mathbf{t}$;
         ii. for all $i, j \in [m]$ such that $\mathbf{c}[i] = \mathbf{c}'[j]$,
            A. compute $\mathsf{rtk}_i \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}_{K_R}\Big(\Big\langle \mathsf{tbl}(\mathbf{c}), i \Big\rangle\Big)$;
            B. compute $\mathsf{rtk}_j \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}_{K_R}\Big(\Big\langle \mathsf{tbl}(\mathbf{c}'), j \Big\rangle\Big)$;
            C. add $(\mathsf{rtk}_i, \mathsf{rtk}_j)$ to $\mathbf{t}$;
         iii. set
         $$\mathsf{MM}_{\mathbf{c}}\Big[\Big\langle \big\langle \mathsf{tbl}(\mathbf{c}), \mathsf{crk}(\mathbf{c}) \big\rangle, \big\langle \mathsf{tbl}(\mathbf{c}'), \mathsf{crk}(\mathbf{c}') \big\rangle \Big\rangle\Big] := \mathbf{t};$$
      (b) compute $(K_{\mathbf{c}}, \mathsf{EMM}_{\mathbf{c}}) \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Setup}\big(1^k, \mathsf{MM}_{\mathbf{c}}\big)$;
      (c) set $\mathsf{DX}\big[\big\langle \mathsf{tbl}(\mathbf{c}), \mathsf{crk}(\mathbf{c}) \big\rangle\big] = \mathsf{EMM}_{\mathbf{c}}$;
  11. compute $(K_D, \mathsf{EDX}) \leftarrow \Sigma_{\mathsf{DX}}.\mathsf{Setup}(1^k, \mathsf{DX})$;
  12. output $K = \big(K_R, K_C, K_V, K_D, \{K_{\mathbf{c}}\}_{\mathbf{c} \in \mathsf{DB}^{\intercal}}\big)$ and $\mathsf{EDB} = \big(\mathsf{EMM}_R, \mathsf{EMM}_C, \mathsf{EMM}_V, \mathsf{EDX}\big)$;

[a] Note that we omit the description of Dec since it simply decrypts every cell of ET.

Figure 1: SPX: a relational DB encryption scheme (Part 1).

- $\mathsf{Token}(K, q)$:
  1. parse q as $\pi_{\mathsf{a}_1,\cdots,\mathsf{a}_h}\Big([a_1] \times \cdots \times [a_f] \times \sigma_\Psi(\mathsf{T}_{i_1} \times \cdots \times \mathsf{T}_{i_t})\Big)$;
  2. for all $i \in [h]$,
     (a) if $\mathsf{a}_i \in \mathbb{S}(\mathsf{DB})$, compute
     $$\mathsf{ptk}_i \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}_{K_C}\Big(\Big\langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i) \Big\rangle\Big);$$
     and set $\mathsf{ytk}_i := (0, \mathsf{ptk}_i)$;
     (b) if $\mathsf{a}_i \in \mathbb{S}(\mathsf{VDB})$, set $\mathsf{ytk}_i := (1, \mathsf{a}_i)$;
  3. parse Ψ as $\mathsf{a}_1 = X_1 \wedge \cdots \wedge \mathsf{a}_\ell = X_\ell$;
  4. for all $i \in [\ell]$,
     (a) if $\mathsf{a}_i = X_i$ is type-1, compute
     $$\mathsf{vtk}_i \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}_{K_V}\Big(\Big\langle X_i, \Big\langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i) \Big\rangle\Big\rangle\Big),$$
     and set $\mathsf{stk}_i = \mathsf{vtk}_i$;
     (b) if $\mathsf{a}_i = X_i$ is type-2, compute
     $$\mathsf{jtk}_i \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}_{K_{\mathsf{a}_i}}\Big(\Big\langle \big\langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i) \big\rangle, \big\langle \mathsf{tbl}(X_i), \mathsf{crk}(X_i) \big\rangle\Big\rangle\Big)$$
     and
     $$\mathsf{dtk}_i \leftarrow \Sigma_{\mathsf{DX}}.\mathsf{Token}_{K_D}\Big(\Big\langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i) \Big\rangle\Big)$$
     and set $\mathsf{stk}_i = (\mathsf{dtk}_i, \mathsf{jtk}_i)$
  5. for all $i \in [f]$, compute $\mathrm{e}_i \leftarrow \mathsf{Enc}_{K_1}(a_i)$;
  6. let $\mathbf{s} = (\|\mathsf{T}_{i_1}\|_c, \cdots, \|\mathsf{T}_{i_t}\|_c)$;
  7. output $\mathsf{tk} = (t, \mathbf{s}, (\mathrm{e}_i)_{i\in[f]}, (\mathsf{ytk})_{i\in[h]}, (\mathsf{stk}_i)_{i\in[\ell]})$;

Figure 2: SPX: a relational DB encryption scheme (Part 2).

- $\mathsf{Query}(\mathsf{EDB}, \mathsf{tk})$:
  1. parse EDB as $\big(\mathsf{EMM}_R, \mathsf{EMM}_C, \mathsf{EMM}_V, \mathsf{EDX}\big)$;
  2. parse $\mathsf{tk} = (t, \mathbf{s}, (\mathrm{e}_i)_{i\in[f]}, (\mathsf{ytk})_{i\in[h]}, (\mathsf{stk}_i)_{i\in[\ell]})$;
  3. initialize an m^t-by-$\sum_{i=1}^{t} s_i$ matrix M and a 1-by-1 matrix VA;
  4. for all $i \in [\ell]$,
     (a) if stk_i is type-1, parse it as $\mathsf{stk}_i = \mathsf{vtk}_i$ and,
        i. compute $\mathbf{t} \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{vtk}_i, \mathsf{EMM}_V)$;
        ii. for all $\mathsf{rtk} \in \mathbf{t}$,
           A. $\langle \mathrm{ct}_1, \ldots, \mathrm{ct}_d, \mathrm{tb}, \mathrm{rrk}\rangle \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_R, \mathsf{rtk})$;
           B. set $\mathsf{M}[\alpha, \beta] := \mathrm{ct}_j$ for all $j \in [d]$ and all $(\alpha, \beta) \in \Lambda_{t,m,\mathbf{s}}(\mathrm{tb}, \mathrm{rrk}, j)$;
     (b) if stk_i is type-2, parse it as $\mathsf{stk}_i = (\mathsf{dtk}_i, \mathsf{jtk}_i)$ and,
        i. compute $\mathsf{EMM}_{\mathbf{c}} \leftarrow \Sigma_{\mathsf{DX}}.\mathsf{Get}(\mathsf{EDX}, \mathsf{dtk}_i)$;
        ii. compute $\mathbf{t} \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_{\mathbf{c}}, \mathsf{jtk}_i)$;
        iii. for all $(\mathsf{rtk}, \mathsf{rtk}') \in \mathbf{t}$,
           A. compute $\langle \mathrm{ct}_1, \ldots, \mathrm{ct}_d, \mathrm{tb}, \mathrm{rrk}\rangle \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_R, \mathsf{rtk})$;
           B. compute $\langle \mathrm{ct}'_1, \ldots, \mathrm{ct}'_{d'}, \mathrm{tb}', \mathrm{rrk}'\rangle \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_R, \mathsf{rtk}')$;
           C. for all $(\alpha, \beta) \in \Lambda_{t,m,\mathbf{s}}(\mathrm{tb}, \mathrm{rrk}, 1)$ and $(\alpha', \beta') \in \Lambda_{t,m,\mathbf{s}}(\mathrm{tb}', \mathrm{rrk}', 1)$ s.t. $\alpha = \alpha'$,
              – set $\mathsf{M}[\alpha, \beta] := \mathrm{ct}_j$ for all $j \in [d]$ and all $(\alpha, \beta) \in \Lambda_{t,m,\mathbf{s}}(\mathrm{tb}, \mathrm{rrk}, j)$;
              – set $\mathsf{M}[\alpha', \beta'] := \mathrm{ct}'_z$ for all $z \in [d']$ and all $(\alpha', \beta') \in \Lambda_{t,m,\mathbf{s}}(\mathrm{tb}', \mathrm{rrk}', z)$;
  5. for all $i \in [h]$,
     (a) if $\mathsf{ytk}_i = (1, \mathsf{a})$, compute $\mathsf{VA} := \mathsf{VA} \times [\mathrm{e}_{\mathsf{a}}]$;
     (b) if $\mathsf{ytk}_i = (0, \mathsf{ptk}_i)$,
        i. compute $\langle \mathbf{c}, \langle \mathrm{tb}, \mathrm{crk}\rangle\rangle \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_C, \mathsf{ptk}_i)$;
        ii. for all $\gamma \in [m]$ and all $(\alpha, \beta) \in \Lambda_{t,m,\mathbf{s}}(\mathrm{tb}, \gamma, \mathrm{crk})$, set $\mathsf{M}[\alpha, \beta] := \mathbf{c}[\gamma]$;
  6. compute $\mathsf{ET} := \mathsf{VA} \times \mathsf{M}$;
  7. delete all columns with a least one empty row in ET;
  8. output ET

Figure 3: SPX: a relational DB encryption scheme (Part 3).

first coordinates of the all the coordinate pairs in R_i have the same table identifier and, similarly, the second coordinates of all the coordinate pairs have the same table identifiers. As such, we can assign a unique table identifier to each type-1 set and two table identifiers to each type-2 set. To make this explicit, we write type-1 sets as R_{tb} and type-2 sets as $R_{\mathrm{tb}_1,\mathrm{tb}_2}$. In the following, when referring to the table identifier of a type-2 set, we mean the table identifier of its first elements.

Our next step is to take per-table intersections. More precisely, let $T \subseteq [n]$ be the set of table identifiers of the recovered sets. Now, for all identifiers $\mathrm{tb} \in T$, we generate the set I_{tb} as the intersection between all the result sets with table identifier tb (this could include type-1 and type-2 sets). Note that we could be taking intersections of type-1 sets, which hold coordinates, and type-2 sets, which hold pairs of coordinates. This is, of course, undefined so in such a case the intersection is defined to be the set of coordinate pairs in the type-2 set whose first coordinate is in the type-1 set. This results in one set I_{tb} per table, which holds either coordinates (if it results from the intersection of type-1 sets) or pairs of coordinates (if it results from the intersection of two type-2 sets or the intersection of one type-1 set and one type-2 set).

The next step is to take a cross product between the sets $\{I_{\mathrm{tb}}\}_{\mathrm{tb}\in T}$. Note that when doing this, there could be instances where we need to create a new tuple from two *pairs* of coordinates. In such a case, we only create the new tuple if the first elements of the pairs are equal. If they are not, we discard the two pairs. We refer to this process as an *augmented cross-product* and denote its result as $\mathcal{C}$, which is a set of tuples of coordinates. Now, for every coordinate tuple in $\mathcal{C}$, we do the following. We recover the encrypted rows associated with the coordinates in the tuple. We then query the location map on the coordinates in the tuple to get a location i in M. We then store this information in a multi-map $\mathsf{TR_3}$; that is, we set $\mathsf{TR_3}$ to map the location i to the encrypted rows associated with the coordinates in the tuple. For correctness, we add dummy elements to the encrypted rows. This is necessary in case the project operation is on an attribute that is not included in the select operation.

The final step consists of retrieving the desired columns based on the y-tokens. More precisely, we use the y-token to query EMM_C which returns an encrypted column together with its table identifier tb and column rank crk. Then, for every row i in the column, we lookup $\mathsf{TR_3}[i]$ to recover an encrypted row. If the element at location crk in the encrypted row is not a dummy, we add it to our result table. On the other hand, if it is a dummy, we use i and the column rank crk to generate a location in M on which we apply the inverse of the location map. This returns a cell coordinate which we use to fetch an encrypted cell from the results of the EMM_C query. Finally, we replace the dummy element with this cell.

5.2 Efficiency

We now turn to analyzing the search and storage efficiency of our construction with the optimized Query algorithm.

Search complexity. Given an SPC query

$$\pi_{\mathsf{a}_1,\cdots,\mathsf{a}_h}\Big([a_1]\times\cdots[a_f]\times\sigma_\Psi(\mathsf{T}_{i_1}\times\cdots\times\mathsf{T}_{i_\ell})\Big),$$

the size of the result is linear in

$$\mathcal{Q} = h\cdot(m^{h'}\cdot\Pi_{i=1}^{\ell}\#R_i),$$

- $\mathsf{Query}(\mathsf{EDB}, \mathsf{tk})$:
  1. parse EDB as $\big(\mathsf{EMM}_R, \mathsf{EMM}_C, \mathsf{EMM}_V, \mathsf{EDX}\big)$;
  2. parse $\mathsf{tk} = (t, \mathbf{s}, (\mathrm{e}_i)_{i\in[f]}, (\mathsf{ytk})_{i\in[h]}, (\mathsf{stk}_i)_{i\in[\ell]})$;
  3. initialize six multi-maps TR_1, TR_2, TR_3, CT_1, CT_2, and Result;
  4. initialize two non-duplicate sets TB_1, and TB_2;
  5. for all $i \in [\ell]$,
     (a) initialize an empty set R_i;
     (b) if stk_i is type-1, parse it as $\mathsf{stk}_i = \mathsf{vtk}_i$ and,
        i. compute $\mathbf{t} \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_V, \mathsf{vtk}_i)$;
        ii. for all $\mathsf{rtk} \in \mathbf{t}$,
           A. $\langle \mathrm{ct}_1, \ldots, \mathrm{ct}_d, \mathrm{tb}, \mathrm{rrk}\rangle \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_R, \mathsf{rtk})$;
           B. set $R_i = R_i \bigcup \{\mathrm{rrk}\}$;
           C. set $\mathsf{CT}_1[(\mathrm{tb}, \mathrm{rrk})] := \langle \mathrm{ct}_1, \ldots, \mathrm{ct}_d\rangle$;
           D. add tb to TB_1, and set $\mathsf{TR}_1[\mathrm{tb}] := i$;
     (c) if stk_i is type-2, parse it as $\mathsf{stk}_i = (\mathsf{dtk}_i, \mathsf{jtk}_i)$ and,
        i. compute $\mathsf{EMM}_{\mathbf{c}} \leftarrow \Sigma_{\mathsf{DX}}.\mathsf{Get}(\mathsf{EDX}, \mathsf{dtk}_i)$;
        ii. compute $\mathbf{t} \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_{\mathbf{c}}, \mathsf{jtk}_i)$;
        iii. for all $(\mathsf{rtk}, \mathsf{rtk}') \in \mathbf{t}$,
           A. compute $\langle \mathrm{ct}_1, \ldots, \mathrm{ct}_d, \mathrm{tb}, \mathrm{rrk}\rangle \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_R, \mathsf{rtk})$;
           B. compute $\langle \mathrm{ct}'_1, \ldots, \mathrm{ct}'_{d'}, \mathrm{tb}', \mathrm{rrk}'\rangle \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_R, \mathsf{rtk}')$;
           C. set $R_i = R_i \bigcup \{(\mathrm{rrk}, \mathrm{rrk}')\}$;
           D. set $\mathsf{CT}_2[(\mathrm{tb}, \mathrm{rrk})] := \langle \mathrm{ct}_1, \ldots, \mathrm{ct}_d\rangle$;
           E. set $\mathsf{CT}_2[(\mathrm{tb}', \mathrm{rrk}')] := \langle \mathrm{ct}'_1, \ldots, \mathrm{ct}'_{d'}\rangle$;
           F. add $(\mathrm{tb}, \mathrm{tb}')$ to TB_2, and set $\mathsf{TR}_2[(\mathrm{tb}, \mathrm{tb}')] := i$;
  6. for all $\mathrm{tb} \in \mathsf{TB}_1$, set $R_{\mathrm{tb}} = \bigcap_{i\in\mathbf{g}} R_g$, where $\mathbf{g} := \mathsf{TR}_1[\mathrm{tb}]$;
  7. for all $(\mathrm{tb}_1, \mathrm{tb}_2) \in \mathsf{TB}_2$, set $R_{(\mathrm{tb}_1,\mathrm{tb}_2)} = \bigcap_{i\in\mathbf{g}} R_g$, where $\mathbf{g} := \mathsf{TR}_2[(\mathrm{tb}_1, \mathrm{tb}_2)]$;
  8. for all $(\mathrm{tb}_1, \mathrm{tb}_2)$ in TB_2,
     (a) for all tb in TB_1, if $\mathrm{tb} = \mathrm{tb}_i$, for $i = 1$ or 2,
        – set $S_{(\mathrm{tb}_1,\mathrm{tb}_2)} = \{(\mathrm{rrk}_1, \mathrm{rrk}_2) \mid (\mathrm{rrk}_1, \mathrm{rrk}_2) \in R_{(\mathrm{tb}_1,\mathrm{tb}_2)} \wedge \mathrm{rrk}_1 \in R_{\mathrm{tb}}\}$, and delete R_{tb};
     (b) for all $(\mathrm{tb}'_1, \mathrm{tb}'_2)$ in TB_2 such that $\mathrm{tb}_1 = \mathrm{tb}'_1$ and $\mathrm{tb}_2 \neq \mathrm{tb}'_2$,
        – set $P = \{\mathrm{rrk}_1 \mid (\mathrm{rrk}_1, \mathrm{rrk}_2) \in R_{(\mathrm{tb}_1,\mathrm{tb}_2)}\}$;
        – set $S_{(\mathrm{tb}'_1,\mathrm{tb}'_2)} = \{(\mathrm{rrk}_1, \mathrm{rrk}_2) \mid (\mathrm{rrk}_1, \mathrm{rrk}_2) \in R_{(\mathrm{tb}'_1,\mathrm{tb}'_2)} \wedge \mathrm{rrk}_1 \in P\}$;

Figure 4: An optimized Query algorithm (Part 1).

- Continuation of $\mathsf{Query}(\mathsf{EDB}, \mathsf{tk})$:

  9. initialize counter $\mu = 0$;
  10. for all $(\mathrm{tb}_1, \mathrm{tb}_2) \in \mathsf{TB}_2$,
      (a) initialize two sets $\mathcal{T}_\mu$ and $\mathcal{C}_\mu$;
      (b) set $\mathcal{C}_\mu = S_{(\mathrm{tb}_1, \mathrm{tb}_2)}$;
      (c) add tb_1, and tb_2 to $\mathcal{T}_\mu$;
      (d) for all $(\mathrm{tb}'_1, \mathrm{tb}'_2) \in \mathsf{TB}_2$,
          i. if $(\mathrm{tb}'_1, \mathrm{tb}'_2) \neq (\mathrm{tb}_1, \mathrm{tb}_2)$ and $\mathrm{tb}'_i = \mathrm{tb}_i$ for $i = 1$ or 2,
             A. set $P = \{\mathrm{rrk}_i \mid (\mathrm{rrk}_1, \mathrm{rrk}_2) \in R_{(\mathrm{tb}'_1, \mathrm{tb}'_2)}\}$;
             B. set $\mathcal{C}_i = \big\{(\mathrm{rrk}_1, \cdots, \mathrm{rrk}_e, \mathrm{rrk}) \mid (\mathrm{rrk}_1, \cdots, \mathrm{rrk}_e) \in \mathcal{C}_i \wedge \big(\exists j \in [e] \text{ s.t. } \mathrm{rrk} = \mathrm{rrk}_j \bigwedge \mathrm{rrk} \in P\ \big)\big\}$;
             C. remove $(\mathrm{tb}'_1, \mathrm{tb}'_2)$ from TB_2 and set $\mu = \mu + 1$;
             D. add tb'_i to $\mathcal{T}_\mu$;
  11. set $w = \kappa + h_1 + \cdots + h_\mu$, where $h_i = \#\mathcal{C}_i$;
  12. for all $\mathrm{rrk}_1, \cdots, \mathrm{rrk}_\kappa \in R_{\mathrm{tb}_1}, \cdots, R_{\mathrm{tb}_\kappa}$ and for all $(\mathrm{rrk}^1_1, \cdots, \mathrm{rrk}^1_{h_1}), \cdots, (\mathrm{rrk}^\mu_1, \cdots, \mathrm{rrk}^\mu_{h_\mu}) \in \mathcal{C}_1, \cdots, \mathcal{C}_\mu$
      (a) for all i in

$$\Gamma_{t,m,w}\Big(\big(\mathrm{tb}_1, \mathrm{rrk}_1\big), \cdots, \big(\mathrm{tb}_\kappa, \mathrm{rrk}_\kappa\big), \big(\mathcal{T}_1[1], \mathrm{rrk}^1_1\big), \cdots, \big(\mathcal{T}_1[h_1], \mathrm{rrk}^1_{h_1}\big), \cdots \big(\mathcal{T}_\mu[1], \mathrm{rrk}^\mu_1\big) \cdots \big(\mathcal{T}_\mu[h_\mu], \mathrm{rrk}^\mu_{h_\mu}\big), 1\Big),$$

set,

$$\begin{aligned}\mathsf{TR}_3[i] := \Big\langle & \mathsf{CT}_1\big[(\mathrm{tb}_1, \mathrm{rrk}_1)\big], \cdots, \mathsf{CT}_1\big[(\mathrm{tb}_h, \mathrm{rrk}_h)\big], \\ & \mathsf{CT}_2\big[(\mathcal{T}_1[1], \mathrm{rrk}^1_1)\big], \cdots, \mathsf{CT}_2[(\mathcal{T}_1[h_1], \mathrm{rrk}^1_{h_1})\big], \\ & \cdots, \\ & \mathsf{CT}_2\big[(\mathcal{T}_\mu[1], \mathrm{rrk}^\mu_1)\big], \cdots, \mathsf{CT}_2\big[(\mathcal{T}_\mu[h_\mu], \mathrm{rrk}^\mu_{h_\mu})\big], \underbrace{\mathsf{dummy}, \cdots, \mathsf{dummy}}_{\sum_{i=w+1}^{t} s_i \text{ time}}\Big\rangle\end{aligned}$$

  13. for all $i \in [h]$,
      (a) if $\mathsf{ytk}_i = (0, \mathsf{ptk}_i)$,
          i. compute $\langle \mathbf{c}, \langle \mathrm{tb}, \mathrm{crk}\rangle\rangle \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_C, \mathsf{ptk}_i)$;
          ii. set Θ be the range of TR_3;
          iii. set $\mathrm{crk} = \mathrm{crk} + \sum_{i=1}^{\mathrm{tb}} s_i$;
          iv. if $\mathrm{crk} \leq \sum_{i=1}^{w} s_i$, set

$$\mathsf{Result}[\mathrm{crk}] := \langle \mathsf{TR}_3[\Theta_1]_{\mathrm{crk}}, \cdots, \mathsf{TR}_3[\Theta_{|\Theta|}]_{\mathrm{crk}}\rangle;$$

          v. if $\mathrm{crk} > \sum_{i=1}^{w} s_i$, set

$$\mathsf{Result}[\mathrm{crk}] := \langle \mathbf{c}[\Lambda^{-1}_{t,m,\mathbf{s}}(\Theta_1, \mathrm{crk})_2], \cdots, \mathbf{c}[\Lambda^{-1}_{t,m,\mathbf{s}}(\Theta_{|\Theta|}, \mathrm{crk})_2]\rangle;$$

      (b) if $\mathsf{ytk}_i = (1, \mathsf{a})$, set $\mathsf{Result}[\mathsf{a}] := \langle \underbrace{\mathrm{e}_\mathsf{a}, \cdots, \mathrm{e}_\mathsf{a}}_{|\Theta|}\rangle$;
  14. output ET such that every row equals an entry of a label in Result;

Figure 5: An optimized Query algorithm (Part 2).

where R_i is the set of rows that match the ith term in the select (i.e., the ith result set), and $0 \leq h' \leq h$ is the number of attributes in the project operator $\pi_{\mathsf{a}_1,\ldots,\mathsf{a}_h}$ that are *not* included in the select operator σ_Ψ. Note that here we assume the worst-case where these projection attributes (i.e., the ones not involved in σ_Ψ) all belong to distinct tables. In $\mathcal{Q}$, the term $\Pi_{i=1}^{\ell} \# R_i$ is the number of rows in the tables $\mathsf{T}_{i_1}, \ldots, \mathsf{T}_{i_\ell}$ (not in their cross product) that satisfy σ_Ψ. The term $m^{h'}$ is the number of rows in $\mathsf{T}_{i_1}, \ldots, \mathsf{T}_{i_\ell}$ that include one of the attributes in $\pi_{\mathsf{a}_1,\ldots,\mathsf{a}_h}$ that are not involved in σ_Ψ (assuming all tables have m rows). Finally, h is the number of columns/attributes in the projection operation.

$\mathcal{Q}$ represents a worst-case lower bound on both the running time [4] and the space complexity of a plaintext SPC query. Note that in practice the number h' of attributes in $\pi_{\mathsf{a}_1,\ldots,\mathsf{a}_h}$ that are not involved in the select operator σ_Ψ is usually very small. The best and most common case is when $h' = 0$ which leads to $\mathcal{Q} = h \cdot \Pi_{i=1}^{\ell} \# R_i$.

We now show that the optimized Query algorithm presented in Section 5.1, requires $O(\mathcal{Q})$ time and space when $t, s_1, \ldots, s_n, h, \ell \ll m$ which is common in practice. Consider a token

$$\mathsf{tk} = \Big(t, \mathbf{s}, (\mathrm{e}_{a_i})_{i \in [f]}, (\mathsf{ytk}_i)_{i \in [h]}, (\mathsf{stk}_i)_{i \in [\ell]}\Big),$$

and an encrypted database $\mathsf{EDB} = (\mathsf{EMM}_R, \mathsf{EMM}_C, \mathsf{EMM}_V, \mathsf{EDX})$. For ease of exposition, all tables have the same number of rows m.

When processing the s-tokens we use four multi-maps TR_1, TR_2, CT_1, CT_2, and two sets TB_1 and TB_2 that store the table identifiers in the result sets. Recall that TR_1, TR_2 both have size $O(\ell)$ since there are at most ℓ s-tokens in tk. TB_1 and TB_2 have respectively size $O(t)$, $O(\max(t, \ell))$ since there are at most t tables involved in the query, and ℓ possible type-2 pairs. The multi-maps CT_1 and CT_2 store the encrypted rows that are recovered by processing the s-tokens. That is, they are of size $O(\sum_{i=1}^{\ell} \# R_i \cdot s_i)$, where s_i is the number of cells in the ith table. Note that the size of CT_1 and CT_2 dominates the other data structures. Also, note that the running time is $O(\sum_{i=1}^{\ell} \# R_i \cdot s_i)$.

Computing intersections is performed in $O(\tau^2 \cdot \max_{i \in [\ell]} \# R_i)$ steps, where $\tau = \max(t, \ell)$. The intersection goes over all possible tables identifiers in TB_1 and TB_2. Note that the worst case *running time* occurs when the terms of the select operation are all type-2 and have the same attributes, e.g., $\mathsf{a}_1 = \mathsf{a}_1 \bigwedge \cdots \bigwedge \mathsf{a}_1 = \mathsf{a}_1$. On the other hand, the worst case *space complexity* occurs when there are no common attributes between terms in the select operation and this requires $O(\sum_{i=1}^{\ell} \# R_i \cdot s_i)$ space.

When computing the augmented cross product, the worst case with respect to space occurs when the result sets have no elements in common so we need to keep all pairs. The cost in space of this cross product will be

$$O\Big(m^{h'} \cdot (\Pi_{i=1}^{\ell} \# R_i) \cdot \sum_{i=1}^{t} s_i\Big) = O\Big(\frac{\mathcal{Q}}{h} \cdot \sum_{i=1}^{t} s_i\Big).$$

On the other hand, the worst-case with respect to time occurs when all terms in the select operation share an attribute in common. In such a case the running time is,

$$O\Big(m^{h'} \cdot (\Pi_{i=1}^{\ell} \# R_i) \cdot \sum_{i=1}^{t} s_i + \tau^2 \cdot \max_{i \in [\ell]} \# R_i\Big) = O\Big(\frac{\mathcal{Q}}{h} \cdot \sum_{i=1}^{t} s_i\Big).$$

[4] This is assuming, of course, one is not using pre-computation.

The generation of the final encrypted table ET requires us to move items from TR_3 to Result and then into ET. The space and time complexity of this operation is dominated by the size of TR_3 which is $O(h^{-1} \cdot \mathcal{Q} \cdot \sum_{i=1}^{t} s_i)$.

In summary, Query requires $O(h^{-1} \cdot \mathcal{Q} \cdot \sum_{i=1}^{t} s_i)$ space and time. If we view $h^{-1} \cdot \sum_{i=1}^{t} s_i$ as a constant since, in practice $t, s_1, \ldots, s_t, h \ll m$, both the running time and the space complexity are $O(\mathcal{Q})$.

Storage complexity. For a database $\mathsf{DB} = (\mathsf{T}_1, \ldots, \mathsf{T}_n)$, SPX produces four encrypted multi-maps EMM_R, EMM_C, EMM_V and EDX. Let $s = \sum_i \|\mathsf{T}_i\|_c$. For ease of exposition, we again assume each table has m rows. Finally, note that standard multi-map encryption schemes [15, 23, 8, 7] produce encrypted structures with storage overhead that is linear in sum of the tuple sizes. Using such a scheme as the underlying multi-map encryption scheme, we have that EMM_R and EMM_C will be $O(\sum_{\mathbf{r} \in \mathsf{DB}} \#\mathbf{r})$ and $O(\sum_{\mathbf{c} \in \mathsf{DB}^\intercal} \#\mathbf{c})$, respectively, since the former maps the coordinates of each row in DB to their (encrypted) row and the latter maps the coordinates of very column to their (encrypted) columns. Since EMM_V maps each cell in DB to tokens for the rows that contain the same value, it requires $O(\sum_{\mathbf{c} \in \mathsf{DB}^\intercal} \sum_{v \in \mathbf{c}} \#\mathsf{DB}_{\mathbf{c}=v})$ storage. EDX maps the coordinates of each column $\mathbf{c} \in \mathsf{DB}^\intercal$ to an encrypted multi-map $\mathsf{EMM}_{\mathbf{c}}$ which in turn maps each pair of form $(\mathbf{c}, \mathbf{c}')$ such that $\mathsf{dom}(\mathbf{c}) = \mathsf{dom}(\mathbf{c}')$ to a tuple of tokens for rows in $\mathsf{DB}_{\mathbf{c}=\mathbf{c}'}$. As such, EDX will have size

$$O\Big(\sum_{\mathbf{c} \in \mathsf{DB}^\intercal} \sum_{\mathbf{c}' : \mathsf{dom}(\mathbf{c}') = \mathsf{dom}(\mathbf{c})} \#\mathsf{DB}_{\mathbf{c}=\mathbf{c}'} \Big).$$

Note that the expression will vary greatly depending on the number of columns in DB with the same domain. In the worst case, all columns will have a common domain and the expression will be a sum of $s(s-1)$ terms of the form $\#\mathsf{DB}_{\mathbf{c}=\mathbf{c}'}$. In the best case, none of the columns will share a domain and EDX will be empty. In practice, however, we expect there to be some relatively small number of columns with common domains.

6 Security and Leakage

We now show that our construction is adaptively-semantically secure with respect to a well-specified leakage profile. Part of the subtlety in our security analysis is that some of the leakage is "black-box" in the sense that it comes from the underlying schemes and part of it "non-black-box" in the sense that it comes directly from the SPX construction itself. Throughout our discussion of SPX's leakage, we consider both its black-box leakage (i.e., when the underlying schemes are left abstract) and its concrete leakage (i.e., when the underlying schemes are instantiated). To instantiate the underlying schemes, we consider any of a standard set of SSE constructions from the literature [15, 23, 12, 8, 7, 27] which all have the same leakage profile, i.e., the search pattern which reveals if and when a query is repeated. [5] In particular, these SSE schemes can be used to instantiate both Σ_{MM} and Σ_{DX} since the former is a generalization of the latter.

[5] Since the schemes are response-revealing, we do not consider the "access pattern leakage" of SSE schemes (which is defined as the search response) to be leakage in our context.

Setup leakage. The setup leakage of SPX captures what an adversary can learn before performing any query operation. The setup leakage of SPX is

$$\mathcal{L}_{\mathsf{S}}^{\mathsf{spx}}(\mathsf{DB}) = \Big(\mathcal{L}_{\mathsf{S}}^{\mathsf{dx}}(\mathsf{DX}), \mathcal{L}_{\mathsf{S}}^{\mathsf{mm}}\left(\mathsf{MM}_R\right), \mathcal{L}_{\mathsf{S}}^{\mathsf{mm}}\left(\mathsf{MM}_C\right), \mathcal{L}_{\mathsf{S}}^{\mathsf{mm}}\left(\mathsf{MM}_V\right)\Big),$$

where $\mathcal{L}_{\mathsf{S}}^{\mathsf{dx}}$ and $\mathcal{L}_{\mathsf{S}}^{\mathsf{mm}}$ are the setup leakages of Σ_{DX} and Σ_{MM}, respectively. If the latter are instantiated with standard encrypted multi-map constructions, the setup leakage of SPX will consist of the number of rows and columns in DB and the size of the dictionary. Note that standard encrypted dictionary constructions leak only the maximum size of the values they store so the size of the $\mathsf{EMM}_{\mathbf{c}}$'s will be hidden (up to the maximum size).

Query leakage. The query leakage is more complex and is defined as follows,

$$\mathcal{L}_{\mathsf{Q}}^{\mathsf{spx}}(\mathsf{DB}, q) = \Big(\mathrm{XPP}(\mathsf{DB}, q), \mathrm{PrP}(\mathsf{DB}, q), \mathrm{SelP}(\mathsf{DB}, q)\Big),$$

where each individual pattern is described next.

Cross product. The first leakage pattern is the *cross product* pattern which is defined as

$$\mathrm{XPP}(\mathsf{DB}, q) = \Big\{t, \|\mathsf{T}_{i_1}\|_c, \ldots, \|\mathsf{T}_{i_\ell}\|_c, \left(|a_i|\right)_{i\in[f]}\Big\},$$

and includes the number and size of the tables needed in the query as well as the size of the virtual attributes.

Projection. The second leakage pattern is the *projection pattern* which is defined as

$$\mathrm{PrP}(\mathsf{DB}, q) = \Big(\mathcal{P}(\mathsf{a}_1), \ldots, \mathcal{P}(\mathsf{a}_h)\Big),$$

where

$$\mathcal{P}(\mathsf{a}_i) = \begin{cases} \Big(\mathtt{real}, \mathcal{L}_{\mathsf{Q}}^{\mathsf{mm}}\Big(\mathsf{MM}_C, \Big\langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i)\Big\rangle\Big), \left(|c_j|\right)_{j\in[\#\mathbf{c}_i]}, \mathrm{AccP}(\mathbf{c}_i)\Big) & \text{if } \mathsf{a}_i \in \mathbb{S}(\mathsf{DB}); \\ \Big(\mathtt{virtual}, \mathsf{a}_i\Big) & \text{if } \mathsf{a}_i \in \mathbb{S}(\mathsf{VDB}), \end{cases}$$

where $\mathbf{c}_i \in \mathsf{DB}^{\mathsf{T}}$ is the column with attribute a_i and $\mathrm{AccP}(\mathbf{c}_i)$ indicates if and when the column $\mathbf{c}_i$ has been accessed before. PrP captures the leakage produced when the server queries MM_C and for every attribute a_i reveals whether the attribute is real or virtual. If it is real, it also reveals the size of the items in the projected column and if and when this column has been accessed before. Notice that it also reveals the Σ_{MM} query leakage on the coordinates of the projected attribute. If the latter is instantiated with any of the standard multi-map encryption schemes then this leakage will reveal whether the attribute a_i has appeared in a previous query. If the attribute is virtual, it just reveals the attribute.

Selection. The third leakage pattern is the *selection pattern* which is defined as

$$\mathrm{SelP}(\mathsf{DB}, q) = \Big(\mathcal{Z}(\mathsf{a}_1, X_1), \ldots, \mathcal{Z}(\mathsf{a}_\ell, X_\ell)\Big).$$

If $\mathsf{a}_i = X_i$ is type-1, then $\mathcal{Z}(\mathsf{a}_i, X_i)$ is defined as

$$\mathcal{Z}(\mathsf{a}_i, X_i) = \Big(\text{type-1}, \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\Big(\mathsf{MM}_V, \Big\langle X_i, \Big\langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i)\Big\rangle\Big\rangle\Big), \Big\{\mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\Big(\mathsf{MM}_R, \Big\langle \mathsf{tbl}(\mathbf{r}), \mathsf{rrk}(\mathbf{r})\Big\rangle\Big), \mathrm{AccP}(\mathbf{r})\Big\}_{\mathbf{r}\in\mathsf{DB}_{\mathsf{a}_i=X_i}}\Big).$$

where $\mathrm{AccP}(\mathbf{r})$ indicates whether the row $\mathbf{r}$ has been accessed before. $\mathcal{Z}(\mathsf{a}_i, X_i)$ captures the leakage produced when the server queries MM_V and uses the resulting row tokens to then query MM_R. It reveals whether the selection term is of type-1 and the Σ_{MM} query leakage on the constant X_i and the coordinates of the attribute a_i. In addition, it also leaks the Σ_{MM} query leakage on the coordinates of the rows in $\mathsf{DB}_{\mathsf{a}_i=X_i}$ as well as if and when they have been accessed before. If the encrypted multi-maps are instantiated with standard constructions, $\mathcal{Z}(\mathsf{a}_i, X_i)$ amounts to: if and when the pair (X_i, a_i) has been queried before and if and when any of the rows in $\mathsf{DB}_{\mathsf{a}_i=X_i}$ have been accessed in the past.

If, on the other hand, $\mathsf{a}_i = X_i$ is type-2, then $\mathcal{Z}(\mathsf{a}_i, X_i)$ is defined as

$$\begin{aligned}\mathcal{Z}(\mathsf{a}_i, X_i) = \Big(&\text{type-2}, \mathcal{L}^{\mathsf{dx}}_{\mathsf{Q}}\Big(\mathsf{DX}, \Big\langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i)\Big\rangle\Big), \mathcal{L}^{\mathsf{mm}}_{\mathsf{S}}(\mathsf{MM}_{\mathsf{a}_i}), \mathrm{AccP}(\mathsf{EMM}_{\mathsf{a}_i}),\\ &\mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\Big(\mathsf{MM}_{\mathsf{a}_i}, \Big\langle \langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i)\rangle, \langle \mathsf{tbl}(X_i), \mathsf{crk}(X_i)\rangle\Big\rangle\Big),\\ &\Big\{\mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\Big(\mathsf{MM}_R, \langle \mathsf{tbl}(\mathbf{r}_1), \mathsf{rrk}(\mathbf{r}_1)\rangle\Big), \mathrm{AccP}(\mathbf{r}_1),\\ &\mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\Big(\mathsf{MM}_R, \langle \mathsf{tbl}(\mathbf{r}_2), \mathsf{rrk}(\mathbf{r}_2)\rangle\Big), \mathrm{AccP}(\mathbf{r}_2)\Big\}_{(\mathbf{r}_1,\mathbf{r}_2)\in\mathsf{DB}_{\mathsf{a}_i=X_i}}\Big),\end{aligned}$$

where $\mathrm{AccP}(\mathbf{r})$, $\mathrm{AccP}(\mathbf{r}_1)$, $\mathrm{AccP}(\mathbf{r}_2)$ and $\mathrm{AccP}(\mathsf{EMM}_{\mathsf{a}_i})$ indicate if and when $\mathbf{r}$, $\mathbf{r}_1$, $\mathbf{r}_2$ and $\mathsf{EMM}_{\mathsf{a}_i}$ have been accessed before. In this case, $\mathcal{Z}(\mathsf{a}_i, X_i)$ captures the leakage produced when the server queries EDX to retrieve some $\mathsf{EMM}_{\mathsf{a}}$ which it in turn queries to retrieve row tokens with which to query EMM_R. It reveals whether the selection term is type-2, the Σ_{DX} query leakage on the coordinates of a_i, the Σ_{MM} *setup* leakage on $\mathsf{MM}_{\mathsf{a}_i}$ and if and when $\mathsf{EMM}_{\mathsf{a}_i}$ has been accessed in the past. In addition, it reveals the query leakage of Σ_{MM} on the coordinates of a_i and X_i and, for every pair of rows $(\mathbf{r}_1, \mathbf{r}_2)$ in $\mathsf{DB}_{\mathsf{a}_i=X_i}$, their Σ_{MM} query leakage and if and when they were accessed in the past. Again, if instantiated with standard encrypted multi-maps, this would amount to the type of the selection, if and when a_i had been queried in the past, the number of columns in DB that share a domain with a_i, if and when the pair (a_i, X_i) has appeared in previous queries and, for every pair of rows in $\mathsf{DB}_{\mathsf{a}_i=X_i}$, if and when these rows have been accessed in the past.

Comparison to PPE-based solutions. As mentioned in Section 1, PPE-based solutions can handle a large class of SQL queries which includes conjunctive queries. To support conjunctive queries, however, these solutions have to rely on deterministic encryption. For example, to handle a

type-1 select on a column $\mathbf{c}$, they will reveal a deterministic encryptions of $\mathbf{c}$ (i.e., every element of the column is encrypted under the same key). To handle a type-2 select between two columns $\mathbf{c}_1$ and $\mathbf{c}_2$, they will reveal deterministic encryptions of both columns (under the same key). In turn, this will provide the *frequency* information on entire columns to the server. Depending on the setting, frequency patterns can be particularly dangerous, as shown in [26].

We note that, in comparison, SPX leaks considerably less. First, it does not leak any frequency information on entire columns or rows. For type-1 selects, it only leaks information about the attribute in the select and the rows that match the term. For type-2 selects, it only leaks information about the pair of attributes (a_i, X_i) in the select and the rows that match the term. Note that this leakage is only a function of the attributes in the query and of the rows that match it, whereas the leakage in PPE-based solutions is a function of entire columns. Moreover, in the case of SPX, if the underlying multi-map and dictionary schemes are instantiated with standard constructions, the information leaked about the attributes and matching rows is "repetition" type of information, i.e., if and when they have appeared in the past. Analogously, the project operations in SPX only leak information about the attributes in the project and the columns that match it and the information being leaked "repetition" type of information.

6.1 Security of Our Construction

We now prove that SPX is adaptively semantically-secure with respect to the leakage profile described in the previous sub-section.

Theorem 6.1. *If Σ_{DX} is adaptively $(\mathcal{L}_{\mathsf{S}}^{\mathsf{dx}}, \mathcal{L}_{\mathsf{Q}}^{\mathsf{dx}})$-semantically secure and Σ_{MM} is adaptively $(\mathcal{L}_{\mathsf{S}}^{\mathsf{mm}}, \mathcal{L}_{\mathsf{Q}}^{\mathsf{mm}})$-secure, then SPX is $(\mathcal{L}_{\mathsf{S}}^{\mathsf{spx}}, \mathcal{L}_{\mathsf{Q}}^{\mathsf{spx}})$-semantically secure.*

The proof of Theorem 6.1 in Appendix A.

7 Handling Extended Queries

In addition to the basic Select-From-Where structure, SQL queries can also include additional clauses to post-process the results of a query. The most common examples are the Group-by, Order-by and various aggregate functions which include Sum, Average, Median, Count, Mode, Max and Min. Here, we refer to SQL queries with such additional clauses as *extended queries*. Though SPX cannot handle extended queries explicitly we note that the additional clauses can be executed at the client. In cases where the result table R is such that $\|\mathsf{R}\|_r \ll \sum_{i=1}^{n} \|\mathsf{T}_i\|_r$, outsourcing a database with SPX will still be less computationally-expensive than executing the queries locally.

Order-by. The Order-by clause orders the rows in the result table R according to a specified attribute in either ascending or descending order. These queries have the form:

Select *attributes* From *tables* Where *conditions* Order-by *attribute*.

Ordering R at the client can be done in $O(m \log m)$ time where $m = \|\mathsf{R}\|_r$. Note that the client has to perform $O(m)$ work just to decrypt the result table so, asymptotically-speaking, ordering R does not add a lot. In practice, however, Order-by operations are often performed on integer or string columns and in the latter case this can be expensive if the strings are long. A simple optimization is for the client to compute the order of the rows during the setup phase and to add a column that

includes a CPA-secure encryption of each row's order. When the client retrieves the result-table R it can then decrypt the column and order the rows using a *numerical* sorting algorithm (as opposed to lexicographic).

Group by. Another common SQL clause is Group-by which is used to partition the rows in the result table by a specified attribute. Group-by is typically used with aggregate functions (i.e., Sum, Max etc.). Such queries have the form:

Select *attributes attributes* From *tables* Where *conditions* Group-by*attributes*.

Group-by clauses can be handled at the client in $O(m)$ time, where $m = \|\mathsf{R}\|_r$.

Aggregates. Often, one needs to compute a function on the rows of the result table (e.g., computing a sum of salaries from employee table). In SQL, this is handled with an aggregate function. For example, a query with the Sum function has the form:

Select Sum(*attribute*) From *tables* Where *conditions*.

Some of the aggregates can be handled by the server by extending SPX in the natural way. For example, to handle Count the server can simply return the number of rows in the encrypted result table ET. To handle Sum on a given attribute a, it suffices to encrypt the cells of that column with an additively-homomorphic encryption scheme and have the server return the sum of the resulting column. The remaining functions like Max, Min, Median, Mode, have to be handled at the client.

References

[1] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order preserving encryption for numeric data. In *ACM SIGMOD International Conference on Management of Data*, pages 563–574, 2004.

[2] A. Arasu, S. Blanas, K. Eguro, R. Kaushik, D. Kossmann, R. Ramamurthy, and R. Venkatesan. Orthogonal security with cipherbase. In *CIDR*, 2013.

[3] M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In A. Menezes, editor, *Advances in Cryptology – CRYPTO '07*, Lecture Notes in Computer Science, pages 535–552. Springer, 2007.

[4] A. Ben-David, N. Nisan, and B. Pinkas. Fairplaymp: a system for secure multi-party computation. In *ACM Conference on Computer and Communications Security (CCS 2008)*, pages 257–266. ACM, 2008.

[5] A. Boldyreva, N. Chenette, Y. Lee, and A. O'neill. Order-preserving symmetric encryption. In *Advances in Cryptology - EUROCRYPT 2009*, pages 224–241, 2009.

[6] D. Boneh, G. di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Advances in Cryptology – EUROCRYPT '04*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522. Springer, 2004.

[7] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *Network and Distributed System Security Symposium (NDSS '14)*, 2014.

[8] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology - CRYPTO '13*. Springer, 2013.

[9] D. Cash and S. Tessaro. The locality of searchable symmetric encryption. In *Advances in Cryptology - EUROCRYPT 2014*, 2014.

[10] Donald D Chamberlin and Raymond F Boyce. SEQUEL: A structured english query language. In *ACM SIGFIDET workshop on Data description, access and control (SIGMOD '74)*, pages 249–264. ACM, 1974.

[11] A. Chandra and P. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *ACM Symposium on Theory of Computing (STOC '77)*, pages 77–90. ACM, 1977.

[12] M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *Advances in Cryptology - ASIACRYPT '10*, volume 6477 of *Lecture Notes in Computer Science*, pages 577–594. Springer, 2010.

[13] M. Chase and S. Kamara. Structured encryption and controlled disclosure. Technical Report 2011/010.pdf, IACR Cryptology ePrint Archive, 2010.

[14] E. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[15] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In *ACM Conference on Computer and Communications Security (CCS '06)*, pages 79–88. ACM, 2006.

[16] S. Faber, S. Jarecki, H. Krawczyk, Q. Nguyen, M. Rosu, and M. Steiner. Rich queries on encrypted data: Beyond exact matches. In *European Symposium on Research in Computer Security (ESORICS '15). Lecture Notes in Computer Science*, volume 9327, pages 123–145, 2015.

[17] C. Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.

[18] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM*, 43(3):431–473, 1996.

[19] P. Grofig, M. Haerterich, I. Hang, F. Kerschbaum, M. Kohler, A. Schaad, A. Schroepfer, and W. Tighzert. Experiences and observations on the industrial implementation of a system to search over outsourced encrypted data. In *Sicherheit*, pages 115–125, 2014.

[20] H. Hacigümü**c**s, B. Iyer, C. Li, and S. Mehrotra. Executing sql over encrypted data in the database-service-provider model. In *ACM SIGMOD International Conference on Management of Data*, pages 216–227. ACM, 2002.

[21] S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Outsourced symmetric private information retrieval. In *ACM Conference on Computer and Communications Security (CCS '13)*, pages 875–888, 2013.

[22] S. Kamara and C. Papamanthou. Parallel and dynamic searchable symmetric encryption. In *Financial Cryptography and Data Security (FC '13)*, 2013.

[23] S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *ACM Conference on Computer and Communications Security (CCS '12)*. ACM Press, 2012.

[24] K. Kurosawa and Y. Ohtaki. Uc-secure searchable symmetric encryption. In *Financial Cryptography and Data Security (FC '12)*, Lecture Notes in Computer Science, pages 285–298. Springer, 2012.

[25] X. Meng, S. Kamara, K. Nissim, and G. Kollios. Grecs: Graph encryption for approximate shortest distance queries. In *ACM Conference on Computer and Communication Security (CCS '15)*, 2015.

[26] M. Naveed, S. Kamara, and C. Wright. Inference attacks on property-preserving encrypted databases. In *ACM Conference on Computer and Communication Security (CCS '15)*, 2015.

[27] M. Naveed, M. Prabhakaran, and C. Gunter. Dynamic searchable encryption via blind storage. In *IEEE Symposium on Security and Privacy (S&P '14)*, 2014.

[28] V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S.-G. Choi, W. George, A. Keromytis, and S. Bellovin. Blind seer: A scalable private dbms. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 359–374. IEEE, 2014.

[29] R. Ada Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: Protecting confidentiality with encrypted query processing. In *SOSP*, pages 85–100, 2011.

[30] D. Song, D. Wagner, and A. Perrig. Practical techniques for searching on encrypted data. In *IEEE Symposium on Research in Security and Privacy*, pages 44–55. IEEE Computer Society, 2000.

[31] J. Van den Bussche and S. Vansummeren. Translating sql into the relational algebra. 2009. http://cs.ulb.ac.be/public/_media/teaching/infoh417/sql2alg_eng.pdf.

A Proof of Theorem 6.1

Theorem 6.1. *If* Σ_{DX} *is adaptively* $(\mathcal{L}_{\mathsf{S}}^{\mathsf{dx}}, \mathcal{L}_{\mathsf{Q}}^{\mathsf{dx}})$*-semantically secure and* Σ_{MM} *is adaptively* $(\mathcal{L}_{\mathsf{S}}^{\mathsf{mm}}, \mathcal{L}_{\mathsf{Q}}^{\mathsf{mm}})$*-secure, then* SPX *is* $(\mathcal{L}_{\mathsf{S}}^{\mathsf{spx}}, \mathcal{L}_{\mathsf{Q}}^{\mathsf{spx}})$*-semantically secure.*

Proof. Let $\mathcal{S}_{\mathsf{DX}}$ and $\mathcal{S}_{\mathsf{MM}}$ be the simulators guaranteed to exist by the adaptive security of Σ_{DX} and Σ_{MM} and consider the SPX simulator $\mathcal{S}$ that works as follows. Given $\mathcal{L}_{\mathsf{S}}^{\mathsf{spx}}(\mathsf{DB})$, $\mathcal{S}$ simulates EDB by computing $\mathsf{EDX} \leftarrow \mathcal{S}_{\mathsf{DX}}(\mathcal{L}_{\mathsf{S}}^{\mathsf{dx}}(\mathsf{DX}))$, $\mathsf{EMM}_R \leftarrow \mathcal{S}_{\mathsf{MM}}(\mathcal{L}_{\mathsf{S}}^{\mathsf{mm}}(\mathsf{MM}_R))$, $\mathsf{EMM}_C \leftarrow \mathcal{S}_{\mathsf{MM}}(\mathcal{L}_{\mathsf{S}}^{\mathsf{mm}}(\mathsf{MM}_C))$ and $\mathsf{EMM}_V \leftarrow \mathcal{S}_{\mathsf{MM}}(\mathcal{L}_{\mathsf{S}}^{\mathsf{mm}}(\mathsf{MM}_V))$ and outputting

$$\mathsf{EDB} = (\mathsf{EMM}_R, \mathsf{EMM}_C, \mathsf{EMM}_V, \mathsf{EDX}).$$

Recall that SPX is response-hiding so $\mathcal{S}$ receives $(\perp, \mathcal{L}^{\mathsf{spx}}_{\mathsf{Q}}(\mathsf{DB}, q))$ as input in the $\mathbf{Real}_{\mathsf{SPX},\mathcal{S}}(k)$ experiment. Given this input, $\mathcal{S}$ simulates a token

$$\mathsf{tk} = \Big(t, \mathbf{s}, (\mathrm{e}_i)_{i\in[f]}, (\mathsf{ytk}_i)_{i\in[h]}, (\mathsf{stk}_i)_{i\in[\ell]}\Big)$$

as follows. It retrieves t and $\mathbf{s}$ directly from $\mathrm{XPP}(\mathsf{DB}, q)$. It then samples $K_1 \xleftarrow{\$} \{0,1\}^k$ and, for all $i \in [f]$, computes $\mathrm{e}_i \leftarrow \mathsf{Enc}_{K_1}(\mathbf{0}^{|a_i|})$, where $|a_i|$ is also from $\mathrm{XPP}(\mathsf{DB}, q)$.

For all $i \in [h]$, if

$$\mathcal{P}(\mathsf{a}_i) = \Big(\mathtt{real}, \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\Big(\mathsf{MM}_C, \Big\langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i)\Big\rangle\Big), (|c_j|)_{j\in[\#\mathbf{c}]}, \mathrm{AccP}(\mathbf{c})\Big)$$

it sets

$$\mathsf{ytk}_i := \mathsf{ptk}_i \leftarrow \mathcal{S}_{\mathsf{MM}}\left((\mathrm{ct}_j)_{j\in[\#\mathbf{c}]}, \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\left(\mathsf{MM}_C, \Big\langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i)\Big\rangle\right)\right),$$

where $\mathrm{ct}_j \leftarrow \mathsf{Enc}_{K_1}(\mathbf{0}^{|c_j|})$ if the column has never been accessed before and where ct_j is the previously used ciphertext otherwise. If, on the other hand, $\mathcal{P}(\mathsf{a}_i) = (\mathtt{virtual}, \mathsf{a}_i)$ it sets $\mathsf{ytk}_i := \mathsf{a}_i$.

Then, for all $i \in [\ell]$, it simulates stk_i as follows. If $\mathcal{Z}(\mathsf{a}_i)$ is type-1, it first computes for all $\mathbf{r} \in \mathsf{DB}_{\mathsf{a}_i = X_i}$,

$$\mathsf{rtk}_{\mathbf{r}} \leftarrow \mathcal{S}_{\mathsf{MM}}\Big((\mathrm{ct}_j)_{j\in[\#\mathbf{r}]}, \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\Big(\mathsf{MM}_R, \Big\langle \mathsf{tbl}(\mathbf{r}), \mathsf{rrk}(\mathbf{r})\Big\rangle\Big)\Big)$$

where $\mathrm{ct}_j \leftarrow \mathsf{Enc}_{K_1}(\mathbf{0}^{|r_j|})$ if $\mathbf{r}$ has never been accessed and ct_j is the previously used ciphertext otherwise. It then computes the token

$$\mathsf{stk}_i := \mathsf{vtk}_i \leftarrow \mathcal{S}_{\mathsf{MM}}\Big((\mathsf{rtk}_{\mathbf{r}})_{\mathbf{r}\in\mathsf{DB}_{\mathsf{a}_i=X_i}}, \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\Big(\mathsf{MM}_V, \Big\langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i)\Big\rangle\Big)\Big).$$

If, on the other hand, $\mathcal{Z}(\mathsf{a}_i)$ is type-2, $\mathcal{S}$ starts by computing

$$\mathsf{dtk}_i \leftarrow \mathcal{S}_{\mathsf{DX}}\Big(\mathsf{EMM}_{\mathsf{a}_i}, \mathcal{L}^{\mathsf{dx}}_{\mathsf{Q}}\Big(\ \langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i)\rangle\ \Big)\Big)$$

where $\mathsf{EMM}_{\mathsf{a}_i} \leftarrow \mathcal{S}_{\mathsf{MM}}(\mathcal{L}^{\mathsf{mm}}_{\mathsf{S}}(\mathsf{MM}_{\mathsf{a}_i}))$ if it has never been accessed before and where $\mathsf{EMM}_{\mathsf{a}_i}$ is the previously-used structure otherwise. It then computes, for all $(\mathbf{r}_1, \mathbf{r}_2) \in \mathsf{DB}_{\mathsf{a}_i=X_i}$,

$$\mathsf{rtk}_1 \leftarrow \mathcal{S}_{\mathsf{MM}}\Big((\mathrm{ct}_j)_{j\in[\#\mathbf{r}_1]}, \mathcal{L}^{\mathsf{MM}}_{\mathsf{Q}}\Big(\mathsf{MM}_R, \langle \mathsf{tbl}(\mathbf{r}_1), \mathsf{rrk}(\mathbf{r}_1)\rangle\ \Big)\Big)$$

and

$$\mathsf{rtk}_2 \leftarrow \mathcal{S}_{\mathsf{MM}}\Big((\mathrm{ct}'_j)_{j\in[\#\mathbf{r}_2]}, \mathcal{L}^{\mathsf{MM}}_{\mathsf{Q}}\Big(\mathsf{MM}_R, \langle \mathsf{tbl}(\mathbf{r}_2), \mathsf{rrk}(\mathbf{r}_2)\rangle\ \Big)\Big),$$

where $\mathrm{ct}_j \leftarrow \mathsf{Enc}_{K_1}(\mathbf{0}^{|r_1[j]|})$ and $\mathrm{ct}'_j \leftarrow \mathsf{Enc}_{K_1}(\mathbf{0}^{|r_2[j]|})$ if they have never been accessed and ct_j and ct'_j are the previously-used ciphertexts otherwise. Finally, it computes

$$\mathsf{jtk}_i \leftarrow \mathcal{S}_{\mathsf{MM}}\Big(\Big\{\mathsf{rtk}_{\mathbf{r}_1}, \mathsf{rtk}_{\mathbf{r}_2}\Big\}_{(\mathbf{r}_1,\mathbf{r}_2)\in\mathsf{DB}_{\mathsf{a}_i=X_i}}, \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\Big(\mathsf{MM}_{\mathsf{a}_i}, \Big\langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{tbl}(X_i)\Big\rangle\Big)\Big),$$

and sets $\mathsf{stk}_i = (\mathsf{dtk}_i, \mathsf{jtk}_i)$.

It remains to show that for all probabilistic polynomial-time adversaries $\mathcal{A}$, the probability that $\mathbf{Real}_{\mathsf{SPX},\mathcal{A}}(k)$ outputs 1 is negligibly-close to the probability that $\mathbf{Ideal}_{\mathsf{SPX},\mathcal{A},\mathcal{S}}(k)$ outputs 1. We do this using the following sequence of games:

Game_0 : is the same as a $\mathbf{Real}_{\mathsf{SPX},\mathcal{A}}(k)$ experiment. For ease of exposition, we define Loc_Ψ as the tuple of left-hand-side attributes in the type-2 terms of Ψ; i.e., the a_i's in the terms of form $\mathsf{a}_i = \mathsf{x}_i$.

Game_1 : is the same as Game_0, except that EMM_C is replaced with the output of $\mathcal{S}_{\mathsf{MM}}(\mathcal{L}^{\mathsf{mm}}_{\mathsf{S}}(\mathsf{MM}_C))$ and, for every normal form query q, each real y-token ytk_i is replaced with the output of

$$\mathcal{S}_{\mathsf{MM}}\left((\mathrm{ct}_j)_{j\in[\#\mathbf{c}]}, \mathrm{AccP}(\mathbf{c}), \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\left(\mathsf{MM}_C, \left\langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i)\right\rangle\right)\right).$$

Game_2 : is the same as Game_1, except that EMM_V is replaced with the output of $\mathcal{S}_{\mathsf{MM}}(\mathcal{L}^{\mathsf{mm}}_{\mathsf{S}}(\mathsf{MM}_V))$ and, for every normal form query q, every type-1 s-token stk_i is replaced with the output of

$$\mathcal{S}_{\mathsf{MM}}\Big((\mathsf{rtk}_{\mathbf{r}})_{\mathbf{r}\in\mathsf{DB}_{\mathsf{a}_i=X_i}}, \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\Big(\mathsf{MM}_V, \Big\langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i)\Big\rangle\Big)\Big).$$

Game_3 : is the same as Game_2, except that EDX is replaced with the output of $\mathcal{S}_{\mathsf{DX}}(\mathcal{L}^{\mathsf{dx}}_{\mathsf{S}}(\mathsf{DX}))$ and, for every normal form query q, every dictionary token dtk_i in type-2 s-tokens $\mathsf{stk}_i = (\mathsf{dtk}_i, \mathsf{jtk}_i)$ are replaced with the output of

$$\mathcal{S}_{\mathsf{DX}}\Big(\mathsf{EMM}_{\mathsf{a}_i}, \mathrm{AccP}(\mathsf{EMM}_{\mathsf{a}_i}), \mathcal{L}^{\mathsf{dx}}_{\mathsf{Q}}\Big(\ \langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i)\rangle\ \Big)\Big).$$

Game_{3+l} for $l \in [\#\mathrm{Loc}_\Psi]$: is the same as Game_{2+l}, except that $\mathsf{EMM}_{\mathrm{Loc}_\Psi[l]}$ is replaced with the output of $\mathcal{S}_{\mathsf{MM}}(\mathcal{L}^{\mathsf{mm}}_{\mathsf{S}}(\mathsf{MM}_{\mathrm{Loc}_\Psi[l]}))$ and for all type-2 terms $\mathsf{a}_i = \mathsf{x}_i$ in Ψ such that $\mathsf{a}_i = \mathrm{Loc}_\Psi[l]$, the join tokens jtk_i in the type-2 s-token $\mathsf{stk}_i = (\mathsf{dtk}_i, \mathsf{jtk}_i)$ is replaced with the output of

$$\begin{aligned}\mathcal{S}_{\mathsf{MM}}\Big(&\Big\{\mathsf{rtk}_{\mathbf{r}_1}, \mathsf{rtk}_{\mathbf{r}_2}\Big\}_{(\mathbf{r}_1,\mathbf{r}_2)\in\mathsf{DB}_{\mathrm{Loc}_\Psi[l]=\mathsf{x}_i}},\\ &\mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\Big(\mathsf{MM}_{\mathrm{Loc}_\Psi[l]}, \Big\langle \mathsf{tbl}(\mathrm{Loc}_\Psi[l]), \mathsf{crk}(\mathrm{Loc}_\Psi[l])\Big\rangle,\\ &\Big\langle \mathsf{tbl}(\mathsf{x}_i), \mathsf{crk}(\mathsf{x}_i)\Big\rangle\Big)\Big).\end{aligned}$$

$\mathsf{Game}_{4+\#\mathrm{Loc}_\Psi}$: is the same as $\mathsf{Game}_{3+\#\mathrm{Loc}_\Psi}$, except that EMM_R is replaced with the output of $\mathcal{S}_{\mathsf{MM}}(\mathcal{L}^{\mathsf{mm}}_{\mathsf{S}}(\mathsf{MM}_R))$ and every row token $\mathsf{rtk}_{\mathbf{r}}$ for a row $\mathbf{r}$ is replaced with the output of of

$$\mathcal{S}_{\mathsf{MM}}\Big(\{\mathrm{ct}_j\}_{j\in[\#\mathbf{r}]}, \mathrm{AccP}(\mathbf{r}), \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\Big(\mathsf{MM}_R, \Big\langle \mathsf{tbl}(\mathbf{r}), \mathsf{rrk}(\mathbf{r})\Big\rangle\Big)\Big)$$

where $\mathrm{ct}_j \leftarrow \mathsf{Enc}_{K_1}(r_j)$.

$\mathsf{Game}_{5+\#\mathrm{Loc}_\Psi}$: is the same as $\mathsf{Game}_{4+\#\mathrm{Loc}_\Psi}$, except that every SKE encryption ct of a message m is replaced with $\mathrm{ct} \leftarrow \mathsf{Enc}_{K_1}(\mathbf{0}^{|m|})$. Note that message size information is always available to the simulator $\mathcal{S}$ through its leakage.

$\mathsf{Game}_{6+\#\mathrm{Loc}_\Psi}$: is the same as $\mathsf{Game}_{5+\#\mathrm{Loc}}$, except that for every token tk for a normal form query q, t and $\mathbf{s}$ are replaced with t and $\mathbf{s}$ from $\mathrm{XPP}(\mathsf{DB}, q)$. Note that $\mathsf{Game}_{6+\#\mathrm{Loc}_\Psi}$ is identical to $\mathbf{Ideal}_{\mathsf{SPX},\mathcal{A},\mathcal{S}}(k)$.

Claim 1. *For all* PPT *adversaries* $\mathcal{A}$*,*

$$\left|\Pr\left[\,\mathsf{Game}_0 = 1\,\right] - \Pr\left[\,\mathsf{Game}_1 = 1\,\right]\right| \leq \mathsf{negl}(k).$$

We show that if there exists a PPT adversary $\mathcal{A}$ that contradicts the claim, then there exists a PPT adversary $\mathcal{B}$ that breaks the adaptive semantic security of Σ_{MM} with respect to an arbitrary PPT simulator $\mathcal{S}'$. $\mathcal{B}$ starts by running $\mathcal{A}$. When $\mathcal{A}$ outputs $\mathsf{DB} = (\mathsf{T}_1, \cdots, \mathsf{T}_n)$, $\mathcal{B}$ creates a dictionary DX, a column multi-map MM_C, a row multi-map MM_R, a value multi-map MM_V and a set of multi-maps $\{\mathsf{MM}_{\mathbf{c}}\}_{\mathbf{c}\in\mathsf{DB}^{\intercal}}$, from DB. Note that this can be done by executing Steps 1 through 10 of $\mathsf{SPX.Setup}$. $\mathcal{B}$ then outputs MM_C. Upon receiving $\mathsf{EMM}_C^{\star}$—from either a $\mathbf{Real}_{\Sigma_{\mathsf{MM}},\mathcal{B}}(k)$ experiment or an $\mathbf{Ideal}_{\Sigma_{\mathsf{MM}},\mathcal{B},\mathcal{S}'}(k)$ experiment—$\mathcal{B}$ sends ERD to $\mathcal{A}$, where

$$\mathsf{EDB} = (\mathsf{EMM}_R, \mathsf{EMM}_C^{\star}, \mathsf{EMM}_V, \mathsf{EDX}),$$

with $(K_D, \mathsf{EDX}) \leftarrow \Sigma_{\mathsf{DX}}.\mathsf{Setup}(1^k, \mathsf{DX})$ such that for all $\mathbf{c} \in \mathsf{DB}^{\intercal}$, $\mathsf{DX}[\mathbf{c}] = \mathsf{EMM}_{\mathbf{c}}$ where $(K_{\mathbf{c}}, \mathsf{EMM}_{\mathbf{c}}) \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Setup}(1^k, \mathsf{MM}_{\mathbf{c}})$, $(K_R, \mathsf{EMM}_R) \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Setup}(1^k, \mathsf{MM}_R)$ and $(K_V, \mathsf{EMM}_V) \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Setup}(1^k, \mathsf{MM}_V)$.

Whenever $\mathcal{A}$ outputs a normal form query

$$q = \pi_{\mathsf{a}_1,\cdots,\mathsf{a}_h}\Big([a_1] \times \cdots \times [a_f] \times \sigma_{\Psi}(\mathsf{T}_{i_1} \times \cdots \times \mathsf{T}_{i_t})\Big),$$

where $\Psi = \mathsf{a}_1 = X_1 \wedge \cdots \wedge \mathsf{a}_\ell = X_\ell$, $\mathcal{B}$ does the following. For all $i \in [h]$, $\mathcal{B}$ outputs $\Big\langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i) \Big\rangle$ as its own query to $\mathsf{EMM}_C^{\star}$ and receives $\mathsf{ytk}_i^{\star}$. It then sends to $\mathcal{A}$ the token

$$\mathsf{tk} = (t, \mathbf{s}, (\mathrm{e}_i)_{i\in[f]}, (\mathsf{ytk})_{i\in[h]}^{\star}, (\mathsf{stk}_i)_{i\in[\ell]});$$

where t is the number of queried tables, $s = \sum_{j=1}^{t} \|T_{i_j}\|_c$ and for all $i \in [f]$, $\mathrm{e}_i \leftarrow \mathsf{Enc}_{K_1}(a_i)$. For all $i \in [\ell]$, if $\mathsf{a}_i = X_i$ is type-1, then $\mathsf{stk}_i = \mathsf{vtk}_i$ such that $\mathsf{vtk}_i \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}_{K_V}\Big(\Big\langle X_i, \Big\langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i) \Big\rangle \Big\rangle\Big)$. Otherwise if $\mathsf{a}_i = X_i$ is type-2, then $\mathsf{stk}_i = (\mathsf{dtk}_i, \mathsf{jtk}_i)$ such that $\mathsf{dtk}_i \leftarrow \Sigma_{\mathsf{DX}}.\mathsf{Token}_{K_D}\Big(\Big\langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i) \Big\rangle\Big)$ and $\mathsf{jtk}_i \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}_{K_{\mathsf{a}_i}}\Big(\Big\langle \langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i) \rangle, \langle \mathsf{tbl}(X_i), \mathsf{crk}(X_i) \rangle \Big\rangle\Big)$.

After answering all of $\mathcal{A}$'s queries, $\mathcal{A}$ outputs a bit which $\mathcal{B}$ returns as its own output. Note that if $\mathcal{B}$ is executed in a $\mathbf{Real}_{\Sigma_{\mathsf{MM}},\mathcal{B}}(k)$ experiment then, by construction, $\mathcal{A}$'s view is exactly its view in Game_0. On the other hand, if $\mathcal{B}$ is executed in an $\mathbf{Ideal}_{\Sigma_{\mathsf{MM}},\mathcal{B},\mathcal{S}'}(k)$ experiment, then $\mathcal{A}$'s view is, by construction, exactly its view in Game_1. It follows by our initial assumption that

$$\left|\Pr\left[\,\mathbf{Real}_{\Sigma_{\mathsf{MM}},\mathcal{B}}(k) = 1\,\right] - \Pr\left[\,\mathbf{Ideal}_{\Sigma_{\mathsf{MM}},\mathcal{B},\mathcal{S}'}(k) = 1\,\right]\right|$$

is non-negligible, which is a contradiction.

□

Claim 2. *For all* PPT *adversaries* $\mathcal{A}$*,*

$$\left|\Pr\left[\,\mathsf{Game}_1 = 1\,\right] - \Pr\left[\,\mathsf{Game}_2 = 1\,\right]\right| \leq \mathsf{negl}(k).$$

We show that if there exists a PPT adversary $\mathcal{A}$ that contradicts the claim, then there exists a PPT adversary $\mathcal{B}$ that breaks the adaptive semantic security of Σ_{MM} with respect to an arbitrary PPT simulator $\mathcal{S}'$. $\mathcal{B}$ starts by running $\mathcal{A}$. When $\mathcal{A}$ outputs $\mathsf{DB} = (\mathsf{T}_1, \cdots, \mathsf{T}_n)$, $\mathcal{B}$ creates a dictionary DX, a column multi-map MM_C, a row multi-map MM_R, a value multi-map MM_V and a set of multi-maps $\{\mathsf{MM}_{\mathbf{c}}\}_{\mathbf{c}\in\mathsf{DB}^{\mathsf{T}}}$, from DB. $\mathcal{B}$ then outputs MM_V. Upon receiving $\mathsf{EMM}_V^\star$—from either a $\mathbf{Real}_{\Sigma_{\mathsf{MM}},\mathcal{B}}(k)$ experiment or an $\mathbf{Ideal}_{\Sigma_{\mathsf{MM}},\mathcal{B},\mathcal{S}'}(k)$ experiment—$\mathcal{B}$ sends ERD to $\mathcal{A}$, where

$$\mathsf{EDB} = (\mathsf{EMM}_R, \mathsf{EMM}_C, \mathsf{EMM}_V^\star, \mathsf{EDX}),$$

with $\mathsf{EMM}_C \leftarrow \mathcal{S}_{\mathsf{MM}}(\mathcal{L}_{\mathsf{S}}^{\mathsf{mm}}(\mathsf{MM}_C))$, $(K_D, \mathsf{EDX}) \leftarrow \Sigma_{\mathsf{DX}}.\mathsf{Setup}(1^k, \mathsf{DX})$ such that for all $\mathbf{c} \in \mathsf{DB}^{\mathsf{T}}$, $\mathsf{DX}[\mathbf{c}] = \mathsf{EMM}_{\mathbf{c}}$ where $(K_{\mathbf{c}}, \mathsf{EMM}_{\mathbf{c}}) \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Setup}(1^k, \mathsf{MM}_{\mathbf{c}})$ and $(K_R, \mathsf{EMM}_R) \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Setup}(1^k, \mathsf{MM}_R)$.

Whenever $\mathcal{A}$ outputs a normal form query q. $\mathcal{B}$ does the following. For all $i \in [\ell]$ such that $\mathsf{a}_i = X_i$ is type-2, $\mathcal{B}$ outputs

$$\left\langle X_i, \left\langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i) \right\rangle \right\rangle$$

as its own query to $\mathsf{EMM}_V^\star$ and receives $\mathsf{vtk}_i^\star$. It then sends to $\mathcal{A}$ the token

$$\mathsf{tk} = (t, \mathbf{s}, (\mathrm{e}_i)_{i\in[f]}, (\mathsf{ytk})_{i\in[h]}, (\mathsf{stk}_i)_{i\in[\ell]}^\star);$$

where t is the number of queried tables, $s = \sum_{j=1}^{t} \|T_{i_j}\|_c$ and for all $i \in [f]$, $\mathrm{e}_i \leftarrow \mathsf{Enc}_{K_1}(a_i)$. If $\mathsf{a}_i \in \mathbb{S}(\mathsf{DB})$,

$$\mathsf{ytk}_i := \mathsf{ptk}_i \leftarrow \mathcal{S}_{\mathsf{MM}}\left((\mathrm{ct}_j)_{j\in[\#\mathbf{c}]}, \mathrm{AccP}(\mathsf{a}_i), \mathcal{L}_{\mathsf{Q}}^{\mathsf{mm}}\left(\mathsf{MM}_C, \left\langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i) \right\rangle\right)\right).$$

Given $\mathrm{AccP}(\mathsf{a}_i)$, ct_j is either equal to $\mathsf{Enc}_{K_1}(c_j)$ if the column has never been accessed before, or the previously used ciphertext otherwise. If, on the other hand, $\mathsf{a}_i \in \mathbb{S}(\mathsf{VDB})$, $\mathsf{ytk}_i := \mathsf{a}_i$. For all $i \in [\ell]$, if $\mathsf{a}_i = X_i$ is type-2, then $\mathsf{stk}_i = (\mathsf{dtk}_i, \mathsf{jtk}_i)$ such that $\mathsf{dtk}_i \leftarrow \Sigma_{\mathsf{DX}}.\mathsf{Token}_{K_D}\left(\left\langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i) \right\rangle\right)$ and $\mathsf{jtk}_i \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}_{K_{\mathsf{a}_i}}\left(\left\langle \langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i)\rangle, \langle \mathsf{tbl}(X_i), \mathsf{crk}(X_i)\rangle \right\rangle\right)$.

After answering all of $\mathcal{A}$'s queries, $\mathcal{A}$ outputs a bit which $\mathcal{B}$ returns as its own output. Note that if $\mathcal{B}$ is executed in a $\mathbf{Real}_{\Sigma_{\mathsf{MM}},\mathcal{B}}(k)$ experiment then, by construction, $\mathcal{A}$'s view is exactly its view in Game_1. On the other hand, if $\mathcal{B}$ is executed in an $\mathbf{Ideal}_{\Sigma_{\mathsf{MM}},\mathcal{B},\mathcal{S}'}(k)$ experiment, then $\mathcal{A}$'s view is, by construction, exactly its view in Game_2. It follows by our initial assumption that

$$\left|\Pr\left[\,\mathbf{Real}_{\Sigma_{\mathsf{MM}},\mathcal{B}}(k) = 1\,\right] - \Pr\left[\,\mathbf{Ideal}_{\Sigma_{\mathsf{MM}},\mathcal{B},\mathcal{S}'}(k) = 1\,\right]\right|$$

is non-negligible, which is a contradiction.

□

Claim 3. *For all* PPT *adversaries* $\mathcal{A}$*,*

$$\left|\Pr\left[\,\mathsf{Game}_2 = 1\,\right] - \Pr\left[\,\mathsf{Game}_3 = 1\,\right]\right| \leq \mathsf{negl}(k).$$

We show that if there exists a PPT adversary $\mathcal{A}$ that contradicts the claim, then there exists a PPT adversary $\mathcal{B}$ that breaks the adaptive semantic security of Σ_{DX} with respect to an arbitrary PPT simulator $\mathcal{S}'$. $\mathcal{B}$ starts by running $\mathcal{A}$. When $\mathcal{A}$ outputs $\mathsf{DB} = (\mathsf{T}_1, \cdots, \mathsf{T}_n)$, $\mathcal{B}$ creates a dictionary DX, a column multi-map MM_C, a row multi-map MM_R, a value multi-map MM_V and a set of multi-maps $\{\mathsf{MM}_{\mathbf{c}}\}_{\mathbf{c}\in\mathsf{DB}^{\intercal}}$, from DB. $\mathcal{B}$ then outputs DX. Upon receiving $\mathsf{EDX}^{\star}$—from either a $\mathbf{Real}_{\Sigma_{\mathsf{DX}},\mathcal{B}}(k)$ experiment or an $\mathbf{Ideal}_{\Sigma_{\mathsf{DX}},\mathcal{B},\mathcal{S}'}(k)$ experiment—$\mathcal{B}$ sends ERD to $\mathcal{A}$, where

$$\mathsf{EDB} = (\mathsf{EMM}_R, \mathsf{EMM}_C, \mathsf{EMM}_V, \mathsf{EDX}^{\star}),$$

with $\mathsf{EMM}_C \leftarrow \mathcal{S}_{\mathsf{MM}}(\mathcal{L}^{\mathsf{mm}}_{\mathsf{S}}(\mathsf{MM}_C))$, $\mathsf{EMM}_V \leftarrow \mathcal{S}_{\mathsf{MM}}(\mathcal{L}^{\mathsf{mm}}_{\mathsf{S}}(\mathsf{MM}_V))$ and $(K_R, \mathsf{EMM}_R) \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Setup}(1^k, \mathsf{MM}_R)$.

Whenever $\mathcal{A}$ outputs a normal form query q, $\mathcal{B}$ does the following. For all $i \in [\ell]$ such that $\mathsf{a}_i = X_i$ is type-2, $\mathcal{B}$ outputs $\left\langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i) \right\rangle$ as its own query to $\mathsf{EDX}^{\star}$ and receives $\mathsf{dtk}^{\star}_i$. It then sends to $\mathcal{A}$ the token

$$\mathsf{tk} = (t, \mathbf{s}, (\mathrm{e}_i)_{i\in[f]}, (\mathsf{ytk})_{i\in[h]}, (\mathsf{stk}_i)^{\star}_{i\in[\ell]});$$

where t is the number of queried tables, $s = \sum_{j=1}^{t} \|T_{i_j}\|_c$ and for all $i \in [f]$, $\mathrm{e}_i \leftarrow \mathsf{Enc}_{K_1}(a_i)$. If $\mathsf{a}_i \in \mathbb{S}(\mathsf{DB})$,

$$\mathsf{ytk}_i := \mathsf{ptk}_i \leftarrow \mathcal{S}_{\mathsf{MM}}\left((\mathrm{ct}_j)_{j\in[\#\mathbf{c}]}, \mathrm{AccP}(\mathsf{a}_i), \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\left(\mathsf{MM}_C, \left\langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i) \right\rangle\right)\right).$$

Given $\mathrm{AccP}(\mathsf{a}_i)$, ct_j is either equal to $\mathsf{Enc}_{K_1}(c_j)$ if the column has never been accessed before, or the previously used ciphertext otherwise. If, on the other hand, $\mathsf{a}_i \in \mathbb{S}(\mathsf{VDB})$, $\mathsf{ytk}_i := \mathsf{a}_i$. For all $i \in [\ell]$, if $\mathsf{a}_i = X_i$ is type-1, $\mathsf{stk}_i := \mathsf{vtk}_i \leftarrow \mathcal{S}_{\mathsf{MM}}\left((\mathsf{rtk}_{\mathbf{r}})_{\mathbf{r}\in\mathsf{DB}_{\mathsf{a}_i=X_i}}, \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\left(\mathsf{MM}_V, \left\langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i) \right\rangle\right)\right)$, with $\mathsf{rtk}_{\mathbf{r}} \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}_{K_R}\left(\langle \mathsf{tbl}(\mathbf{r}), \mathsf{rrk}(\mathbf{r}) \rangle\right)$ for $\mathbf{r} \in \mathsf{DB}_{\mathsf{a}_i=X_i}$. If $\mathsf{a}_i = X_i$ is type-2, then $\mathsf{stk}_i = (\mathsf{dtk}^{\star}_i, \mathsf{jtk}_i)$ such that $\mathsf{jtk}_i \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}_{K_{\mathsf{a}_i}}\left(\left\langle \langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i) \rangle, \langle \mathsf{tbl}(X_i), \mathsf{crk}(X_i) \rangle \right\rangle\right)$.

After answering all of $\mathcal{A}$'s queries, $\mathcal{A}$ outputs a bit which $\mathcal{B}$ returns as its own output. Note that if $\mathcal{B}$ is executed in a $\mathbf{Real}_{\Sigma_{\mathsf{DX}},\mathcal{B}}(k)$ experiment then, by construction, $\mathcal{A}$'s view is exactly its view in Game_2. On the other hand, if $\mathcal{B}$ is executed in an $\mathbf{Ideal}_{\Sigma_{\mathsf{DX}},\mathcal{B},\mathcal{S}'}(k)$ experiment, then $\mathcal{A}$'s view is, by construction, exactly its view in Game_3. It follows by our initial assumption that

$$\left| \Pr\left[\mathbf{Real}_{\Sigma_{\mathsf{DX}},\mathcal{B}}(k) = 1 \right] - \Pr\left[\mathbf{Ideal}_{\Sigma_{\mathsf{DX}},\mathcal{B},\mathcal{S}'}(k) = 1 \right] \right|$$

is non-negligible, which is a contradiction.

□

Claim 4. *For all $l \in [\#\mathrm{Loc}_{\Psi}]$, all* PPT *adversaries $\mathcal{A}$,*

$$\left| \Pr\left[\mathsf{Game}_{2+l} = 1 \right] - \Pr\left[\mathsf{Game}_{3+l} = 1 \right] \right| \leq \mathsf{negl}(k).$$

We show that if there exists a PPT adversary $\mathcal{A}$ that contradicts the claim, then there exists a PPT adversary $\mathcal{B}$ that breaks the adaptive semantic security of Σ_{MM} with respect to an arbitrary PPT simulator $\mathcal{S}'$. $\mathcal{B}$ starts by running $\mathcal{A}$. When $\mathcal{A}$ outputs $\mathsf{DB} = (\mathsf{T}_1, \cdots, \mathsf{T}_n)$, $\mathcal{B}$ creates a dictionary

DX, a column multi-map MM_C, a row multi-map MM_R, a value multi-map MM_V and a set of multi-maps $\{\mathsf{MM}_{\mathbf{c}}\}_{\mathbf{c}\in\mathsf{DB}^\intercal}$, from DB. $\mathcal{B}$ then outputs $\mathsf{MM}_{\mathrm{Loc}_\Psi[l]}$. Upon receiving $\mathsf{EMM}^\star_{\mathrm{Loc}_\Psi[l]}$—from either a $\mathbf{Real}_{\Sigma_{\mathsf{MM}},\mathcal{B}}(k)$ experiment or an $\mathbf{Ideal}_{\Sigma_{\mathsf{MM}},\mathcal{B},\mathcal{S}'}(k)$ experiment—$\mathcal{B}$ sends ERD to $\mathcal{A}$, where

$$\mathsf{EDB} = (\mathsf{EMM}_R, \mathsf{EMM}_C, \mathsf{EMM}_V, \mathsf{EDX}),$$

with $\mathsf{EMM}_C \leftarrow \mathcal{S}_{\mathsf{MM}}(\mathcal{L}^{\mathsf{mm}}_{\mathsf{S}}(\mathsf{MM}_C))$, $\mathsf{EMM}_V \leftarrow \mathcal{S}_{\mathsf{MM}}(\mathcal{L}^{\mathsf{mm}}_{\mathsf{S}}(\mathsf{MM}_V))$, $\mathsf{EDX} \leftarrow \mathcal{S}_{\mathsf{DX}}(\mathcal{L}^{\mathsf{dx}}_{\mathsf{S}}(\mathsf{DX}))$ and $(K_R, \mathsf{EMM}_R) \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Setup}(1^k, \mathsf{MM}_R)$.

Whenever $\mathcal{A}$ outputs a normal form query q, $\mathcal{B}$ does the following. For all $i \in [\ell]$ such that $\mathsf{a}_i = X_i$ with $\mathsf{a}_i = \mathrm{Loc}_\Psi[l]$, $\mathcal{B}$ outputs

$$\left\langle \langle \mathsf{tbl}(\mathrm{Loc}_\Psi[l]), \mathsf{crk}(\mathrm{Loc}_\Psi[l])\rangle, \langle \mathsf{tbl}(X_i), \mathsf{crk}(X_i)\rangle \right\rangle$$

as its own query to $\mathsf{EMM}^\star_{\mathrm{Loc}_\Psi[l]}$ and receives $\mathsf{jtk}^\star_i$. It then sends to $\mathcal{A}$ the token

$$\mathsf{tk} = (t, \mathbf{s}, (\mathrm{e}_i)_{i\in[f]}, (\mathsf{ytk})_{i\in[h]}, (\mathsf{stk}_i)^\star_{i\in[\ell]});$$

where t is the number of queried tables, $s = \sum_{j=1}^t \|T_{i_j}\|_c$ and for all $i \in [f]$, $\mathrm{e}_i \leftarrow \mathsf{Enc}_{K_1}(a_i)$. If $\mathsf{a}_i \in \mathbb{S}(\mathsf{DB})$,

$$\mathsf{ytk}_i := \mathsf{ptk}_i \leftarrow \mathcal{S}_{\mathsf{MM}}\left((\mathrm{ct}_j)_{j\in[\#\mathbf{c}]}, \mathrm{AccP}(\mathsf{a}_i), \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\left(\mathsf{MM}_C, \left\langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i)\right\rangle\right)\right).$$

Given $\mathrm{AccP}(\mathsf{a}_i)$, ct_j is either equal to $\mathsf{Enc}_{K_1}(c_j)$ if the column has never been accessed before, or the previously used ciphertext otherwise. If, on the other hand, $\mathsf{a}_i \in \mathbb{S}(\mathsf{VDB})$, $\mathsf{ytk}_i := \mathsf{a}_i$. For all $i \in [\ell]$, if $\mathsf{a}_i = X_i$ is type-1, $\mathsf{stk}_i := \mathsf{vtk}_i \leftarrow \mathcal{S}_{\mathsf{MM}}\left((\mathsf{rtk}_{\mathbf{r}})_{\mathbf{r}\in\mathsf{DB}_{\mathsf{a}_i=X_i}}, \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\left(\mathsf{MM}_V, \left\langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i)\right\rangle\right)\right)$, with $\mathsf{rtk}_{\mathbf{r}} \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}_{K_R}\left(\langle \mathsf{tbl}(\mathbf{r}), \mathsf{rrk}(\mathbf{r})\rangle\right)$ for $\mathbf{r} \in \mathsf{DB}_{\mathsf{a}_i=X_i}$. If $\mathsf{a}_i = X_i$ is type-2, then $\mathsf{stk}_i = (\mathsf{dtk}_i, \mathsf{jtk}^\star_i)$ equals:

- if $z = l$, $\mathsf{dtk}_i \leftarrow \mathcal{S}_{\mathsf{DX}}\left(\mathsf{EMM}^\star_{\mathrm{Loc}_\Psi[l]}, \mathcal{L}^{\mathsf{dx}}_{\mathsf{Q}}\left(\langle \mathsf{tbl}(\mathrm{Loc}_\Psi[l]), \mathsf{crk}(\mathrm{Loc}_\Psi[l])\rangle \right)\right)$ and $\mathsf{jtk}_i = \mathsf{jtk}^\star_i$
- for all $z < l$, $\mathsf{dtk}_i \leftarrow \mathcal{S}_{\mathsf{DX}}\left(\mathsf{EMM}_{\mathrm{Loc}_\Psi[z]}, \mathrm{AccP}(\mathsf{EMM}_{\mathrm{Loc}_\Psi[z]}), \mathcal{L}^{\mathsf{dx}}_{\mathsf{Q}}\left(\langle \mathsf{tbl}(\mathrm{Loc}_\Psi[z]), \mathsf{crk}(\mathrm{Loc}_\Psi[z])\rangle \right)\right)$ with $\mathrm{AccP}(\mathsf{EMM}_{\mathrm{Loc}_\Psi[z]})$ is a leakage that captures when and where $\mathsf{EMM}_{\mathrm{Loc}_\Psi[z]}$ was generated. Thus, it can be either equal to $\mathsf{EMM}_{\mathrm{Loc}_\Psi[z]} \leftarrow \mathcal{S}_{\mathsf{MM}}(\mathcal{L}^{\mathsf{mm}}_{\mathsf{S}}(\mathsf{MM}_{\mathrm{Loc}_\Psi[z]}))$, or, a previously simulated one, and

$$\mathsf{jtk}_i \leftarrow \mathcal{S}_{\mathsf{MM}}\left(\left\{\mathsf{rtk}_{\mathbf{r}_1}, \mathsf{rtk}_{\mathbf{r}_2}\right\}_{(\mathbf{r}_1,\mathbf{r}_2)\in\mathsf{DB}_{\mathrm{Loc}_\Psi[z]=X_i}}, \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\left(\mathsf{MM}_{\mathrm{Loc}_\Psi[z]}, \left\langle \mathsf{tbl}(\mathrm{Loc}_\Psi[z]), \mathsf{tbl}(X_i)\right\rangle\right)\right),$$

with $\mathsf{rtk}_{\mathbf{r}_1} \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}_{K_R}\left(\langle \mathsf{tbl}(\mathbf{r}_1), \mathsf{rrk}(\mathbf{r}_1)\rangle\right)$ and $\mathsf{rtk}_{\mathbf{r}_2} \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}_{K_R}\left(\langle \mathsf{tbl}(\mathbf{r}_2), \mathsf{rrk}(\mathbf{r}_2)\rangle\right)$.

- for all $z > l$, $\mathsf{dtk}_i \leftarrow \mathcal{S}_{\mathsf{DX}}\Big(\mathsf{EMM}_{\mathrm{Loc}_\Psi[z]}, \mathrm{AccP}(\mathsf{EMM}_{\mathrm{Loc}_\Psi[z]}), \mathcal{L}^{\mathsf{dx}}_{\mathsf{Q}}\Big(\langle \mathsf{tbl}(\mathrm{Loc}_\Psi[z]), \mathsf{crk}(\mathrm{Loc}_\Psi[z])\rangle \Big)\Big)$ with either $\mathsf{EMM}_{\mathrm{Loc}_\Psi[z]} \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Setup}(1^k, \mathsf{MM}_{\mathrm{Loc}_\Psi[z]}$, or a previously generated one, and

$$\mathsf{jtk}_i \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}_{K_{\mathsf{a}_i}}\Big(\Big\langle \langle \mathsf{tbl}(\mathrm{Loc}_\Psi[z]), \mathsf{crk}(\mathrm{Loc}_\Psi[z])\rangle, \langle \mathsf{tbl}(X_i), \mathsf{crk}(X_i)\rangle \Big\rangle\Big).$$

After answering all of $\mathcal{A}$'s queries, $\mathcal{A}$ outputs a bit which $\mathcal{B}$ returns as its own output. Note that if $\mathcal{B}$ is executed in a $\mathbf{Real}_{\Sigma_{\mathsf{MM}},\mathcal{B}}(k)$ experiment then, by construction, $\mathcal{A}$'s view is exactly its view in Game_{2+l}. On the other hand, if $\mathcal{B}$ is executed in an $\mathbf{Ideal}_{\Sigma_{\mathsf{MM}},\mathcal{B},\mathcal{S}'}(k)$ experiment, then $\mathcal{A}$'s view is, by construction, exactly its view in Game_{3+l}. It follows by our initial assumption that

$$\Big|\Pr\left[\,\mathbf{Real}_{\Sigma_{\mathsf{MM}},\mathcal{B}}(k) = 1\,\right] - \Pr\left[\,\mathbf{Ideal}_{\Sigma_{\mathsf{MM}},\mathcal{B},\mathcal{S}'}(k) = 1\,\right]\Big|$$

is non-negligible, which is a contradiction.

□

Claim 5. *For all* PPT *adversaries* $\mathcal{A}$*,*

$$\Big|\Pr\left[\,\mathsf{Game}_{3+\#\mathrm{Loc}_\Psi} = 1\,\right] - \Pr\left[\,\mathsf{Game}_{4+\#\mathrm{Loc}_\Psi} = 1\,\right]\Big| \leq \mathsf{negl}(k).$$

We show that if there exists a PPT adversary $\mathcal{A}$ that contradicts the claim, then there exists a PPT adversary $\mathcal{B}$ that breaks the adaptive semantic security of Σ_{MM} with respect to an arbitrary PPT simulator $\mathcal{S}'$. $\mathcal{B}$ starts by running $\mathcal{A}$. When $\mathcal{A}$ outputs $\mathsf{DB} = (\mathsf{T}_1, \cdots, \mathsf{T}_n)$, $\mathcal{B}$ creates a dictionary DX, a column multi-map MM_C, a row multi-map MM_R, a value multi-map MM_V and a set of multi-maps $\{\mathsf{MM}_{\mathbf{c}}\}_{\mathbf{c}\in\mathsf{DB}^\intercal}$, from DB. $\mathcal{B}$ then outputs MM_R. Upon receiving $\mathsf{EMM}^\star_R$—from either a $\mathbf{Real}_{\Sigma_{\mathsf{MM}},\mathcal{B}}(k)$ experiment or an $\mathbf{Ideal}_{\Sigma_{\mathsf{MM}},\mathcal{B},\mathcal{S}'}(k)$ experiment—$\mathcal{B}$ sends ERD to $\mathcal{A}$, where

$$\mathsf{EDB} = (\mathsf{EMM}^\star_R, \mathsf{EMM}_C, \mathsf{EMM}_V, \mathsf{EDX}),$$

with $\mathsf{EMM}_C \leftarrow \mathcal{S}_{\mathsf{MM}}(\mathcal{L}^{\mathsf{mm}}_{\mathsf{S}}(\mathsf{MM}_C))$, $\mathsf{EMM}_V \leftarrow \mathcal{S}_{\mathsf{MM}}(\mathcal{L}^{\mathsf{mm}}_{\mathsf{S}}(\mathsf{MM}_V))$ and $\mathsf{EDX} \leftarrow \mathcal{S}_{\mathsf{DX}}(\mathcal{L}^{\mathsf{dx}}_{\mathsf{S}}(\mathsf{DX}))$.

Whenever $\mathcal{A}$ outputs a normal form query q, $\mathcal{B}$ does the following. $\mathcal{B}$ outputs $\Big\langle \mathsf{tbl}(\mathbf{r}), \mathsf{rrk}(\mathbf{r})\Big\rangle$ as its own query to $\mathsf{EMM}^\star_R$ and receives $\mathsf{rtk}^\star_{\mathbf{r}}$. It then sends to $\mathcal{A}$ the token

$$\mathsf{tk} = (t, \mathbf{s}, (\mathrm{e}_i)_{i\in[f]}, (\mathsf{ytk})_{i\in[h]}, (\mathsf{stk}_i)_{i\in[\ell]});$$

where t is the number of queried tables, $s = \sum_{j=1}^{t} \|T_{i_j}\|_c$ and for all $i \in [f]$, $\mathrm{e}_i \leftarrow \mathsf{Enc}_{K_1}(a_i)$. If $\mathsf{a}_i \in \mathbb{S}(\mathsf{DB})$,

$$\mathsf{ytk}_i := \mathsf{ptk}_i \leftarrow \mathcal{S}_{\mathsf{MM}}\left(\left(\mathrm{ct}_j\right)_{j\in[\#\mathbf{c}]}, \mathrm{AccP}(\mathsf{a}_i), \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\left(\mathsf{MM}_C, \Big\langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i)\Big\rangle\right)\right).$$

Given $\mathrm{AccP}(\mathsf{a}_i)$, ct_j is either equal to $\mathsf{Enc}_{K_1}(c_j)$ if the column has never been accessed before, or the previously used ciphertext otherwise. If, on the other hand, $\mathsf{a}_i \in \mathbb{S}(\mathsf{VDB})$, $\mathsf{ytk}_i := \mathsf{a}_i$. For all $i \in [\ell]$, if $\mathsf{a}_i = X_i$ is type-1, $\mathsf{stk}_i := \mathsf{vtk}_i \leftarrow \mathcal{S}_{\mathsf{MM}}\Big(\left(\mathsf{rtk}^\star_{\mathbf{r}}\right)_{\mathbf{r}\in\mathsf{DB}_{\mathsf{a}_i=X_i}}, \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\Big(\mathsf{MM}_V, \Big\langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i)\Big\rangle\Big)\Big)$. If

$\mathsf{a}_i = X_i$ is type-2, then $\mathsf{stk}_i = (\mathsf{dtk}_i, \mathsf{jtk}_i)$ such that $\mathsf{dtk}_i \leftarrow \mathcal{S}_{\mathsf{DX}}\Big(\mathsf{EMM}_{\mathsf{a}_i}, \mathrm{AccP}(\mathsf{EMM}_{\mathsf{a}_i}), \mathcal{L}^{\mathsf{dx}}_{\mathsf{Q}}\Big(\langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i)\rangle \Big)\Big)$ with $\mathrm{AccP}(\mathsf{EMM}_{\mathsf{a}_i})$ is a leakage that captures when and where $\mathsf{EMM}_{\mathsf{a}_i}$ was generated. Thus, it can be either equal to $\mathsf{EMM}_{\mathsf{a}_i} \leftarrow \mathcal{S}_{\mathsf{MM}}(\mathcal{L}^{\mathsf{mm}}_{\mathsf{S}}(\mathsf{MM}_{\mathsf{a}_i}))$, or, a previously simulated one, and

$$\mathsf{jtk}_i \leftarrow \mathcal{S}_{\mathsf{MM}}\Bigg(\Big\{\mathsf{rtk}^{\star}_{\mathbf{r}_1}, \mathsf{rtk}^{\star}_{\mathbf{r}_2}\Big\}_{(\mathbf{r}_1,\mathbf{r}_2)\in \mathsf{DB}_{\mathsf{a}_i = X_i}}, \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\Big(\mathsf{MM}_{\mathsf{a}_i}, \Big\langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{tbl}(X_i)\Big\rangle\Big)\Bigg).$$

After answering all of $\mathcal{A}$'s queries, $\mathcal{A}$ outputs a bit which $\mathcal{B}$ returns as its own output. Note that if $\mathcal{B}$ is executed in a $\mathbf{Real}_{\Sigma_{\mathsf{MM}},\mathcal{B}}(k)$ experiment then, by construction, $\mathcal{A}$'s view is exactly its view in $\mathsf{Game}_{3+\#\mathrm{Loc}_\Psi}$. On the other hand, if $\mathcal{B}$ is executed in an $\mathbf{Ideal}_{\Sigma_{\mathsf{MM}},\mathcal{B},\mathcal{S}'}(k)$ experiment, then $\mathcal{A}$'s view is, by construction, exactly its view in $\mathsf{Game}_{4+\#\mathrm{Loc}_\Psi}$. It follows by our initial assumption that

$$\Big|\Pr\left[\,\mathbf{Real}_{\Sigma_{\mathsf{MM}},\mathcal{B}}(k) = 1\,\right] - \Pr\left[\,\mathbf{Ideal}_{\Sigma_{\mathsf{MM}},\mathcal{B},\mathcal{S}'}(k) = 1\,\right]\Big|$$

is non-negligible, which is a contradiction.

□

Claim 6. *For all* PPT *adversaries* $\mathcal{A}$*,*

$$\Big|\Pr\left[\,\mathsf{Game}_{4+\#\mathrm{Loc}_\Psi} = 1\,\right] - \Pr\left[\,\mathsf{Game}_{5+\#\mathrm{Loc}_\Psi} = 1\,\right]\Big| \leq \mathsf{negl}(k).$$

We show that if there exists a PPT adversary $\mathcal{A}$ that contradicts the claim, then there exists a PPT adversary $\mathcal{B}$ that breaks the adaptive semantic security of $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with respect to an arbitrary PPT simulator $\mathcal{S}'$. $\mathcal{B}$ starts by running $\mathcal{A}$. When $\mathcal{A}$ outputs $\mathsf{DB} = (\mathsf{T}_1, \cdots, \mathsf{T}_n)$, $\mathcal{B}$ creates a dictionary DX, a column multi-map MM_C, a row multi-map MM_R, a value multi-map MM_V and a set of multi-maps $\{\mathsf{MM}_{\mathbf{c}}\}_{\mathbf{c}\in\mathsf{DB}^{\intercal}}$, from DB. $\mathcal{B}$ then outputs all message m for all cells. Upon receiving $\mathrm{ct}^{\star}$ for all cells in DB—from either a $\mathbf{Real}_{\mathsf{SKE},\mathcal{B}}(k)$ experiment or an $\mathbf{Ideal}_{\mathsf{SKE},\mathcal{B},\mathcal{S}'}(k)$ experiment—. $\mathcal{B}$ sends ERD to $\mathcal{A}$, where

$$\mathsf{EDB} = (\mathsf{EMM}_R, \mathsf{EMM}_C, \mathsf{EMM}_V, \mathsf{EDX}),$$

with $\mathsf{EMM}_C \leftarrow \mathcal{S}_{\mathsf{MM}}(\mathcal{L}^{\mathsf{mm}}_{\mathsf{S}}(\mathsf{MM}_C))$, $\mathsf{EMM}_V \leftarrow \mathcal{S}_{\mathsf{MM}}(\mathcal{L}^{\mathsf{mm}}_{\mathsf{S}}(\mathsf{MM}_V))$, $\mathsf{EDX} \leftarrow \mathcal{S}_{\mathsf{DX}}(\mathcal{L}^{\mathsf{dx}}_{\mathsf{S}}(\mathsf{DX}))$ and $\mathsf{EMM}_R \leftarrow \mathcal{S}_{\mathsf{MM}}(\mathcal{L}^{\mathsf{mm}}_{\mathsf{S}}(\mathsf{MM}_R))$.

Whenever $\mathcal{A}$ outputs a query χ, $\mathcal{B}$ does the following. For all $i \in [f]$, $\mathcal{B}$ outputs a_i and receives $\mathrm{e}^{\star}_i$. It then sends to $\mathcal{A}$ the token

$$\mathsf{tk} = (t, \mathbf{s}, (\mathrm{e}_i)^{\star}_{i\in[f]}, (\mathsf{ytk})_{i\in[h]}, (\mathsf{stk}_i)_{i\in[\ell]});$$

where t is the number of queried tables and $s = \sum_{j=1}^{t} \|T_{i_j}\|_c$. If $\mathsf{a}_i \in \mathbb{S}(\mathsf{DB})$,

$$\mathsf{ytk}_i := \mathsf{ptk}_i \leftarrow \mathcal{S}_{\mathsf{MM}}\left(\left(\mathrm{ct}^{\star}_j\right)_{j\in[\#\mathbf{c}]}, \mathrm{AccP}(\mathsf{a}_i), \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\left(\mathsf{MM}_C, \left\langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i)\right\rangle\right)\right).$$

Given $\mathrm{AccP}(\mathsf{a}_i)$, $\mathrm{ct}_j\star$ is either equal to a simulated ciphertext if the column has never been accessed before, or the previously used ciphertext otherwise. If, on the other hand, $\mathsf{a}_i \in \mathbb{S}(\mathsf{VDB})$, $\mathsf{ytk}_i := \mathsf{a}_i$. For all $i \in [\ell]$, if $\mathsf{a}_i = X_i$ is type-1, $\mathsf{stk}_i := \mathsf{vtk}_i \leftarrow \mathcal{S}_{\mathsf{MM}}\Big(\left(\mathsf{rtk}_{\mathbf{r}}\right)_{\mathbf{r}\in\mathsf{DB}_{\mathsf{a}_i = X_i}}, \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\Big(\mathsf{MM}_V, \Big\langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i)\Big\rangle\Big)\Big)$.

If $\mathsf{a}_i = X_i$ is type-2, then $\mathsf{stk}_i = (\mathsf{dtk}_i, \mathsf{jtk}_i)$ $\mathsf{dtk}_i \leftarrow \mathcal{S}_{\mathsf{DX}}\Big(\mathsf{EMM}_{\mathsf{a}_i}, \mathrm{AccP}(\mathsf{EMM}_{\mathsf{a}_i}), \mathcal{L}^{\mathsf{dx}}_{\mathsf{Q}}\Big(\langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{crk}(\mathsf{a}_i)\rangle\Big)\Big)$ with $\mathrm{AccP}(\mathsf{EMM}_{\mathsf{a}_i})$ is a leakage that captures when and where $\mathsf{EMM}_{\mathsf{a}_i}$ was generated. Thus, it can be either equal to $\mathsf{EMM}_{\mathsf{a}_i} \leftarrow \mathcal{S}_{\mathsf{MM}}(\mathcal{L}^{\mathsf{mm}}_{\mathsf{S}}(\mathsf{MM}_{\mathsf{a}_i}))$, or, a previously simulated one, and

$$\mathsf{jtk}_i \leftarrow \mathcal{S}_{\mathsf{MM}}\Big(\Big\{\mathsf{rtk}_{\mathbf{r}_1}, \mathsf{rtk}_{\mathbf{r}_2}\Big\}_{(\mathbf{r}_1,\mathbf{r}_2)\in \mathsf{DB}_{\mathsf{a}_i = X_i}}, \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\Big(\mathsf{MM}_{\mathsf{a}_i}, \Big\langle \mathsf{tbl}(\mathsf{a}_i), \mathsf{tbl}(X_i)\Big\rangle\Big)\Big)\Big),$$

with $\mathsf{rtk}_{\mathbf{r}}$ in both cases above equals

$$\mathsf{rtk}_{\mathbf{r}} \leftarrow \mathcal{S}_{\mathsf{MM}}\Big(\{\mathrm{ct}^{\star}_j\}_{j\in[\#\mathbf{r}]}, \mathrm{AccP}(\mathbf{r}), \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\Big(\mathsf{MM}_R, \Big\langle \mathsf{tbl}(\mathbf{r}), \mathsf{rrk}(\mathbf{r})\Big\rangle\Big)\Big),$$

where $\mathrm{AccP}(\mathbf{r})$ captures when and where $\mathbf{r}$ has been queried. Thus, $\mathrm{ct}^{\star}_j$ can be either freshly generated, or, re-used from a previously generated one.

After answering all of $\mathcal{A}$'s queries, $\mathcal{A}$ outputs a bit which $\mathcal{B}$ returns as its own output. Note that if $\mathcal{B}$ is executed in a $\mathbf{Real}_{\mathsf{SKE},\mathcal{B}}(k)$ experiment then, by construction, $\mathcal{A}$'s view is exactly its view in $\mathsf{Game}_{4+\#\mathrm{Loc}_\Psi}$. On the other hand, if $\mathcal{B}$ is executed in an $\mathbf{Ideal}_{\mathsf{SKE},\mathcal{B},\mathcal{S}'}(k)$ experiment, then $\mathcal{A}$'s view is, by construction, exactly its view in $\mathsf{Game}_{5+\#\mathrm{Loc}_\Psi}$. It follows by our initial assumption that

$$\Big|\Pr\left[\,\mathbf{Real}_{\mathsf{SKE},\mathcal{B}}(k) = 1\,\right] - \Pr\left[\,\mathbf{Ideal}_{\mathsf{SKE},\mathcal{B},\mathcal{S}'}(k) = 1\,\right]\Big|$$

is non-negligible, which is a contradiction.

□

Claim 7. *For all* PPT *adversaries* $\mathcal{A}$*,*

$$\Big|\Pr\left[\,\mathsf{Game}_{5+\#\mathrm{Loc}_\Psi} = 1\,\right] = \Pr\left[\,\mathsf{Game}_{6+\#\mathrm{Loc}_\Psi} = 1\,\right]\Big|.$$

It is clear that the view of both games is the same.

■