SQL on Structurally-Encrypted Databases

Seny Kamara [*]
Brown University

Tarik Moataz[†]
Brown 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 (STE) 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 is efficient and—under some conditions on the database and queries—can have asymptotically-optimal query complexity. We also show how to extend our solution to be dynamic while maintaining the scheme's optimal query complexity.

[*] seny@brown.edu. Work done in part at Microsoft Research.

[†] tarik_moataz@brown.edu. Work done in part at Microsoft Research, IMT Atlantique and Colorado State.

Contents

1 Introduction

The problem of encrypted search has received 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 [17] and most relational DBs are queried using the structured query language (SQL) which is a special-purpose declarative language introduced by Chamberlain and Boyce [13].

The problem of encrypted relational DBs is one of the "holy-grails" of database security. As far as we know, it was first explicitly considered by Hacigümüs, Iyer, Li and Mehrotra [24] who described a quantization-based approach which leaks the range within which an item falls. In [35], Popa, Redfield, Zeldovich and Balakrishnan describe a system called CryptDB that can support a non-trivial subset of SQL 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 approach is to replace the plaintext operations needed to execute a SQL query (e.g., equality tests and comparisons) by the same operations on PPE-encrypted ciphertexts. This approach was later adopted by other systems including Cipherbase [2] and SEEED [22]. While this leads to systems that are efficient and legacy-friendly, it was shown by Naveed, Kamara and Wright [32] that 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).[1] As such, our scheme leaks less than any of the previously-known practical approaches and is more practical than any zero-leakage (ZL) solution. Our approach is efficient and handles a 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* [2] [14] which corresponds to SQL queries of the form

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

where att_1 through att_ℓ are attributes in the DB schema and X_1 through X_n are either attributes or

[1] While the dynamic construction we present in Appendix E makes use of ORAM to achieve forward-security, it is only used to store and manage one of several data structures needed by the scheme. In other words, ORAM is not used to store and manage the entire database.

[2] 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 [11, 27]). In particular, our scheme does not make use of any searchable encryption schemes for conjunctive keyword queries and our problem cannot be solved by applying these schemes directly on tables.

constants. For ease of exposition, we mainly focus on conjunctive queries with Where predicates that are *uncorrelated* which, very roughly speaking, means that the attributes are not the same across terms (we refer the reader to Section 5 for a precise definition). The case of correlated predicates is quite involved so it is presented in Appendix C. 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 wider sub-class.

With respect to efficiency, we show that the query complexity of our scheme is asymptotically optimal in time and space when $(s_1 + \cdots + s_t)/h = O(1)$, where t denotes the number of tables in the query, s_i denotes the number of columns in the ith table and h denotes the number of attributes in the Select term of the query. Towards analyzing the asymptotic complexity of our solution, we precisely characterize the result size of an SPC query as a function of the query and of the underlying relational database. This analysis, provided in Appendix A, could be of independent interest.

We also show how to extend our construction to be dynamic and to support two traditional SQL update operations: row addition and row deletions. Surprisingly, our dynamic construction has the same asymptotic efficiency as our static construction. Finally, we show how to extend our dynamic construction to be forward-secure at the cost of a poly-logarithmic overhead for updates but maintaining the same query complexity.

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).

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. 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 [17] 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 [14] 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 cross 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 $\pi_{\mathsf{att}_1,\ldots,\mathsf{att}_h}$ takes as input a table T and outputs the columns of T indexed by $\mathsf{att}_1,\ldots,\mathsf{att}_h$. 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 cross 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,

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

where $[a_j]$ is a 1×1 table that holds a constant a_j for all $j \in [f]$, Ψ is of the form $\mathsf{att}_1 = X_1 \wedge \cdots \wedge \mathsf{att}_\ell = X_\ell$ where $\mathsf{att}_1,\ldots,\mathsf{att}_\ell$ 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 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 create those tables.

Handling SPC normal form queries. 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 $\mathsf{T}' = \mathsf{T}_{i_1} \times \cdots \times \mathsf{T}_{i_t}$. 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 $\mathsf{att}_1,\ldots,\mathsf{att}_h$). Handling such a query naively requires one to first compute the cross product of the tables which can be prohibitively large. As we show in Section 5, however, SPC normal form queries can be rewritten in a different and

optimized form we introduce called the heuristic normal form (HNF). We then show how to encrypt the database in such a way that we can handle queries in their HNF form. At a high level, we achieve this by creating a set of encrypted structures that store different representations of the database. For example, one of the encrypted structures stores a row-wise representation of the database whereas another stores 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 rows needed for it to process the query in its HNF form.

Dynamism. We show how to extend our static construction to be dynamic. This is challenging as we want to maintain the scheme's query complexity while not introducing additional leakage. From a functionality perspective, we restrict our attention to row additions and deletions and leave as important open problem the handling of more complex update operations. While real-world databases also handle edits, we note that these two update operations are already interesting in practice and non-trivial to achieve. As discussed above, we store different encrypted representations of the database. One of these representations, however, stores parts of the database that are highly inter-correlated. The difficulty this poses is that we cannot simply add or remove items from this structure as any change affects all the other items stored in the structure. We introduce a two-party protocol to solve this challenge without the client having to trivially download the entire structure and without leaking too much information to the server. We then show how to extend this solution to be forward-secure at the cost of a poly-logarithmic blowup (for updates). This is achieved by storing and managing one of the structures in an oblivious RAM.

Black-box leakage. We describe and analyze our scheme using algorithms that make black-box use of several lower-level STE schemes (e.g., multi-map and dictionary encryption schemes). We believe this approach has several advantages. The first is that it results in modular constructions that are easier to describe and analyze. The second is that our schemes can benefit from any improvement in the underlying building blocks. This holds with respect to efficiency but also with respect to leakage because we prove our construction secure with respect to a black-box leakage profile (which we later instantiate to get a concrete leakage profile).

A note on our techniques. We stress that our approach to handling the SPC algebra is very different from how these queries are handled on plaintext databases. In other words, our approach does not simply replicate standard data structures and algorithms from the database literature. In fact, our approach to handling SPC queries could be of independent interest for plaintext relational databases.

2 Related Work

Searchable & structured encryption. Encrypted search was first considered explicitly by Song et al. in [36] 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. Curtmola et al. introduced and formulated the notion of adaptive semantic security for SSE [18] together with optimal-time and optimal-space constructions. Chase and Kamara introduced the notion of structured encryption which generalizes SSE to arbitrary data structures [15]. Cash et al. [10] show how to construct optimal-time SSE schemes with low I/O complexity and Cash and

Tessaro [12] gave lower bounds on the locality of adaptively-secure SSE schemes. Asharov et al. build SSE schemes with optimal locality, optimal space overhead and nearly-optimal read efficiency [3]. Garg et al. [19] presented a new SSE construction with reduced leakage leveraging oblivious RAM and garbled RAM techniques. Bost [8] proposed an efficient forward-secure SSE construction based on trapdoor permutations. SSE has also been considered in the multi-user setting [18, 26]. Pappas et al. [34] proposed a multi-user SSE construction based on garbled circuits and Bloom filters that can support Boolean formulas, ranges and stemming. Other approaches for encrypted search include oblivious RAMs (ORAM) [21], secure multi-party computation [5], functional encryption [7] and fully-homomorphic encryption [20] as well as solutions based on deterministic encryption [4] and order-preserving encryption (OPE) [6].

Encrypted relational databases. As far as we know the first encrypted relational DB solution was proposed by Hacigümüs et al. [24] 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 et al. proposed CryptDB [35]. 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 encryption [4] and OPE [1, 6]. The CryptDB design influenced the Cipherbase system from Arasu *et al.* [2] and the SEEED system from Grofig *et al.* [22]. In [32], Naveed et al. study the security of these PPE-based solutions in the context of medical data. Recently, Grubbs, Ristenpart and Shmatikov [23] point out pitfalls in integrating encrypted database solutions in real-world database management systems (DBMS).

Attacks on SSE. While we do not consider the problem of designing an SSE scheme in this work, we do use SSE schemes as building blocks. Several works have proposed attacks that try to exploit the leakage of SSE. This includes the query-recovery attacks of Islam et al. [25], of Cash et al. [9] and of Zhang et al. [40]. Recently, Abdelraheem et al. [31], presented attacks on encrypted relational databases. We briefly mention here that although the attacks in [31] are ostensibly on relational EDBs, they are not related to or applicable to our construction. For more details on these attacks and their relation to our work we refer the reader to Section 6.

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 structures. We make use of several basic data types including dictionaries and multi-maps which we recall here. 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 $\{(\ell_i, \mathbf{t}_i)\}_{i\leq n}$ 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. Note that tuples may have different lengths. 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 $\mathsf{DS} : \mathbf{Q} \to \mathbf{R}$ 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 attribute, we refer to the set of all possible values that it can take as its *domain* (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 $\chi(\mathbf{r}) \stackrel{def}{=} (\mathsf{tbl}(\mathbf{r}), \mathsf{rrk}(\mathbf{r}))$ and $\chi(\mathbf{c}) \stackrel{def}{=} (\mathsf{tbl}(\mathbf{c}), \mathsf{crk}(\mathbf{c}))$, respectively, as their *coordinates* in DB. Similarly, we denote by $\chi(\mathsf{att})$ the coordinate of column $\mathbf{c}$ with attribute $\mathsf{att} \in \mathbb{S}(\mathsf{DB})$ such that $\chi(\mathsf{att}) = \chi(\mathbf{c})$. 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 $\mathsf{att}_1, \mathsf{att}_2 \in \mathbb{S}(\mathsf{DB})$ with the same domain such that $\mathsf{dom}(\mathsf{att}_1) = \mathsf{dom}(\mathsf{att}_2)$, $\mathsf{DB}_{\mathsf{att}_1 = \mathsf{att}_2}$ denotes the set of row pairs $\{(\mathbf{r}_1, \mathbf{r}_2) \in \mathsf{DB}^2 : \mathbf{r}_1[\mathsf{att}_1] = \mathbf{r}_2[\mathsf{att}_2]\}$. For any attribute $\mathsf{att} \in \mathbb{S}(\mathsf{DB})$ and constant $a \in \mathsf{dom}(\mathsf{att})$, $\mathsf{DB}_{\mathsf{att}=a}$ is the set of rows $\{\mathbf{r} \in \mathsf{DB} : \mathbf{r}[\mathsf{att}] = a\}$.

SQL. In practice, relational databases are queried using the special-purpose language SQL, introduced by Chamberlain and Boyce [13]. 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* [14] 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

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

where att_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 [14] 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 $\pi_{\mathsf{att}_1,\ldots,\mathsf{att}_h}$ is parameterized by a set of attributes $\mathsf{att}_1, \ldots, \mathsf{att}_h$ and takes as input a table T and outputs a table T' that consists of the columns of T indexed by att_1 through att_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 cross 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* $\mathsf{att}_1 = X_1 \wedge \cdots \wedge \mathsf{att}_\ell = X_\ell$*, where* $\mathsf{att}_i \in \mathbb{S}(\mathsf{DB})$ *and* X_i *is either a constant* a *in the domain of* att_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* $\mathsf{att}_i = \mathsf{x}_j$ *are satisfied if* $\mathbf{r}[\mathsf{att}_i] = \mathbf{r}[\mathsf{x}_j]$ *and terms of the form* $\mathsf{att}_i = a$ *are satisfied if* $\mathbf{r}[\mathsf{att}_i] = a$*.*
- $\mathsf{T}' \leftarrow \pi_{\mathsf{att}_1,\ldots,\mathsf{att}_h}(\mathsf{T})$*: the* project *operator is parameterized by a set of attributes* $\mathsf{att}_1, \ldots, \mathsf{att}_h \in \mathbb{S}(\mathsf{DB})$*. It takes as input a table* $\mathsf{T} \in \mathsf{DB}$ *and outputs a table* $\mathsf{T}' = \{\langle \mathbf{r}[\mathsf{att}_1], \ldots, \mathbf{r}[\mathsf{att}_h]\rangle : \mathbf{r} \in \mathsf{T}\}$*.*
- $\mathsf{R} \leftarrow \mathsf{T}_1 \times \mathsf{T}_2$*: the* cross product *operator takes as input two tables* T_1 *and* T_2 *and outputs a result table* $\mathsf{R} = \{\langle \mathbf{r}, \mathbf{v}\rangle : \mathbf{r} \in \mathsf{T}_1 \text{ and } \mathbf{v} \in \mathsf{T}_2\}$*, 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:

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

where $a_1, \ldots, a_f \in \bigcup_{\mathsf{att}\in\mathbb{S}(\mathsf{DB})} \mathsf{dom}(\mathsf{att})$ and $[a_j]$ is the 1×1 table that holds a_j. Here, the attributes $\mathsf{att}_1, \ldots, \mathsf{att}_h$ in the projection are either in $\mathbb{S}(\mathsf{DB})$ or refer to the columns generated by $[a_1]$ through $[a_f]$. 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 $\mathsf{VDB} = ([a_1], \ldots, [a_f])$.

One of the advantages of working in the relational algebra is that it allows for powerful optimization techniques. Given a query, we can use several identities to rewrite the query so that it

can be executed more efficiently. The topic of query optimization is a large and important area of research in both database theory and engineering and real-world database management systems crucially rely on sophisticated query optimization algorithms. The main disadvantage of working with SPC queries in normal form is that their execution is extremely expensive, i.e., exponential in t. Furthermore, it is a-priori unclear how one could use standard query optimization techniques over encrypted data. We will see in Section 5, however, that these challenges can be overcome.

We note that while executing normal form SPC queries is prohibitively expensive, converting conjunctive SQL queries to normal form SPC queries is a well-studied problem with highly-optimized solutions. In particular, the 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 [39]).

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 [15] considered schemes that encrypt both a structure and a set of associated data items (e.g., documents, emails, user profiles etc.). In [16], 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 [15])**.** *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 [15])**.** *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 [18] and later generalized to structured encryption in [15].

Definition 4.3 (Adaptive semantic security [18, 15])**.** *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 $z \in \{0,1\}^*$:*

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.*

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 there exists a PPT *simulator $\mathcal{S}$ such that for all* PPT *adversaries $\mathcal{A}$, 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 SPX: A Relational Database Encryption Scheme

In this Section we describe our main construction SPX. We start by giving a high-level overview of two of the main techniques we rely on. The first is how we index the DB to in order to handle HNF queries efficiently. The second is how we use the "chaining" technique from [15] to build complex encrypted structures from simpler ones.

Database indexing. The first step of our construction is to build different representations of the database, each designed to handle a particular operation of the SPC algebra. These representations are designed—when combined in an appropriate manner— to support the *efficient* processing of SPC queries. We use four representations. The first is a *row-wise* representation of the database instantiated as a multi-map MM_R that maps the coordinate of every row in the DB (recall that a coordinate is a row rank / table identifier pair) to the contents of the row. The second representation is a *column-wise* representation of the DB. Similarly, we create a multi-map MM_C that maps the coordinate of every column to the contents of that. The third representation, contrary to MM_R and MM_C, does not store any content of the table but the equality relation among values in the database. For this, we create a multi-map MM_V that maps each value in every column to all the rows that contain the same value. Finally, the fourth representation is a set of multi-maps, one for every column $\mathbf{c}$ in the DB. Each multi-map, $\mathsf{MM}_{\mathbf{c}}$, maps a pair of column coordinates to all the rows that have the same value in both those columns. Now, using multi-map and dictionary encryption schemes, we encrypt all these representations. This results in the encrypted multi-maps $\mathsf{EMM}_R, \mathsf{EMM}_C, \mathsf{EMM}_V$ and an encrypted dictionary EDX (which stores all the all EMM_c's).

Chaining and constructive queries. The different representations we just described are designed so that, given an SPC query, the server can generate the intermediate (encrypted) tables needed to produce the final (encrypted) result/table. To do this, the server will need to make further intermediate queries on these (intermediate) encrypted tables. This type of query evaluation is *constructive* in the sense that the intermediate and final encrypted tables are not the result of pre-processing at setup time but are constructed at query time by the server as a function of the query and the underlying DB. To handle this, we use the chaining technique of [15]. At a high level, the idea is to store query tokens for one encrypted structure as the responses of another encrypted structure. By carefully chaining the various the encrypted multi-maps (EMMs) described above, we can handle constructive queries by first querying some subset of the EMMs to recover either tokens for EMMs further down the chain or encrypted content which we will use to populate intermediate tables. This process proceeds further down the chain until the final result/table is constructed.

Security and efficiency. The database representations we choose along with the careful chaining of their encryptions provide us a way to control both the efficiency and the security of scheme. While intermediate results/tables will vary depending on the query, the chaining sequence remains the same for any SPC query written in our heuristic normal form. The chaining sequence is important because it determines the leakage profile of the construction. We analyze the security of our scheme in black-box manner; that is, we provide a black-box leakage profile that is a function of the leakage profile of the underlying encrypted multi-map and encrypted dictionaries used. This allows us to isolate the leakage that is coming from the underlying building blocks and the leakage that is coming directly from our construction. This further enables us to reason about and decide which concrete

instantiations to use as building blocks so that we can choose the kind of leakage/performance tradeoff that is most appropriate.

From an efficiency standpoint, we show that when SPX is instantiated with optimal-time encrypted multi-map and dictionary schemes, it can achieve optimal query complexity and linear storage complexity (in the size of the DB) under natural assumptions about the database.

5.1 (Plaintext) Database Indexing

As detailed above, SPX relies on several ideas and techniques. Some of these are cryptographic and some are not. To better explain these techniques we will progressively build our solution; starting with a naive plaintext algorithm for evaluating SPC queries and ending with a detailed description of SPX.

The naive SPC algorithm. The naive way to evaluate an SPC normal form query

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

on a database $\mathsf{DB}=(\mathsf{T}_1,\ldots,\mathsf{T}_n)$ is to first compute $\mathsf{R}_1:=\mathsf{T}_{i_1}\times\cdots\times\mathsf{T}_{i_t}$, then $\mathsf{R}_2:=\sigma_\Psi(\mathsf{R}_1)$, then $\mathsf{R}_3:=[a_1]\times\cdots\times[a_f]\times\mathsf{R}_2$ and finally $\mathsf{R}:=\pi_{\mathsf{att}_1,\ldots,\mathsf{att}_h}(\mathsf{R}_3)$. This algorithm is dominated by the cross product computation which is $O(m^t\cdot\sum_{i=1}^t s_i)$, where $m=\max_{i=1}^t\|\mathsf{T}_i\|_r$ and $s_i=\|\mathsf{T}_i\|_c$. The exponential blowup in t is the main reason normal form SPC queries are never used in practice. In addition, since m is usually very large the naive algorithm is prohibitive even for small t.

The benefit of working with the SPC normal form is generality; that is, we can handle an entire class of queries by finding a solution for a single well-specified query form. The disadvantage, however, is that normal form queries take exponential time to evaluate even on a plaintext database.

Heuristic normal form (HNF). We show that certain optimizations can be applied to the SPC normal form so that its evaluation time only induces a multiplicative factor of $\sum_{i=1}^t s_i/h$ over the optimal evaluation time on a plaintext database. We refer to this new normal form as the *heuristic normal form*. In some cases, this multiplicative factor is a constant as it does not depend on the size of the result and, in such cases, the HNF evaluation is optimal. The idea is inspired by a query optimization heuristic from database theory which takes advantage of a distributive property between the select and cross product operators. For example, if the predicate $\Psi=(\mathsf{att}_1=a_1\wedge\cdots\wedge\mathsf{att}_\ell=a_\ell)$ is only composed of type-1 terms and if, for all $i\in[\ell]$, $\mathsf{att}_i\in\mathsf{T}_i$, and the number of terms in Ψ equals the number of tables in the cross product, $\ell=t$, then we have the identity

$$\sigma_\Psi(\mathsf{T}_1\times\cdots\times\mathsf{T}_t)=\sigma_{\mathsf{att}_1=a_1}(\mathsf{T}_1)\times\cdots\times\sigma_{\mathsf{att}_t=a_t}(\mathsf{T}_t).$$

In the database literature this is known as "pushing selects through products" and, depending on the selectivity of the terms, it can greatly reduce the cost of the evaluation. We extend this approach to arbitrary conjunctive predicates which can have both type-1 and type-2 terms. Optimizing these queries is quite involved because the terms can have complex dependencies. In the following, we say that a query is *correlated* if its predicate Ψ satisfies any of the following properties: (1) two or more type-2 terms share a common attribute; (2) a type-1 and type-2 term share a common attribute; (3) the attributes of two or more type-2 terms are from the same table; and (4) the attributes from a type-1 and type-2 term are from the same table. We say that a query is *uncorrelated* if it is not

correlated. For ease of exposition, we only describe here how to handle uncorrelated queries and treat the case of correlated queries in Appendix C.

HNF for uncorrelated queries. If Ψ is uncorrelated, we process each term of Ψ and apply the following rules. Let φ be an empty query. If there are $p \geq 1$ type-1 terms $\mathsf{att}_1 = a_1, \ldots, \mathsf{att}_p = a_p$ from some table T, then we set

$$\varphi := \varphi \times \Big(\sigma_{\mathsf{att}_1=a_1}(\mathsf{T}) \cap \cdots \cap \sigma_{\mathsf{att}_p=a_p}(\mathsf{T})\Big),$$

and remove these terms from Ψ. If the term has form $\mathsf{att}_1 = \mathsf{att}_2$ (i.e., is type-2), where att_1 and att_2 are from tables T_1 and T_2, respectively, then we set

$$\varphi := \varphi \times \sigma_{\mathsf{att}_1=\mathsf{att}_2}(\mathsf{T}_1 \times \mathsf{T}_2).$$

Note that if att_1 and att_2 are from the same table T, then $\mathsf{T}_1 = \mathsf{T}_2 = \mathsf{T}$ above.

At the end of this rewriting process, we say that the query

$$\pi_{\mathsf{att}_1,\cdots,\mathsf{att}_h}\Big([a_1] \times \cdots [a_f] \times \varphi\Big)$$

is in the heuristic SPC normal form or simply the heuristic normal form.

Indexing. In database systems, select and project operations can be executed in one of two ways: with or without an index. In an unindexed execution, the database management system evaluates the operation using sequential scan. For example, to evaluate the operation $\sigma_{\mathsf{att}=a}(\mathsf{T})$, it scans the rows of T and returns the ones that satisfy $\mathsf{att} = a$. In an indexed execution, on the other hand, the database management system uses a pre-computed data structure (e.g., an index) to find the relevant rows in sub-linear time. Here, we give an overview of how one can index the database to support efficient heuristic normal form queries. Note that our indexing strategy is really designed so that we can support heuristic normal form queries on encrypted data (which we discuss below) so it is not necessarily the most natural way to index a plaintext database.

Given a database $\mathsf{DB} = (\mathsf{T}_1, \ldots, \mathsf{T}_n)$, we first create a multi-map MM_R that stores, for all $\mathbf{r} \in \mathsf{DB}$, the pair

$$\Big(\chi(\mathbf{r}), \mathbf{r}\Big).$$

In other words, the multi-map MM_R maps row coordinates to rows. We then create a second multi-map MM_C that maps column coordinates to columns. Following this, we build a third multi-map, MM_V, that maps every value/column pair $(v, \chi(\mathbf{c}))$ in the database to the coordinates of the rows that hold v in column $\mathbf{c}$. That is, for all columns $\mathbf{c} \in \mathsf{DB}^{\mathsf{T}}$ and all values $v \in \mathbf{c}$, MM_V stores the pair

$$\Big(\Big\langle v, \chi(\mathbf{c})\Big\rangle, \Big(\chi(\mathbf{r})\Big)_{\mathbf{r} \in \mathsf{DB}_{\mathsf{att}(\mathbf{c})=v}}\Big).$$

Finally, we build a set of multi-maps for every column $\mathbf{c} \in \mathsf{DB}^{\mathsf{T}}$. More precisely, for all columns $\mathbf{c} \in \mathsf{DB}^{\mathsf{T}}$ we create the multi-map $\mathsf{MM}_{\mathbf{c}}$ which maps the coordinates of $\mathbf{c}$ and any other column $\mathbf{c}'$

that has the same domain as $\mathbf{c}$, to the coordinates of rows $\mathbf{r}$ and $\mathbf{r}'$ such that $\mathbf{r}[\mathbf{c}] = \mathbf{r}'[\mathbf{c}']$. More precisely, for all $\mathbf{c}' \in \mathsf{DB}^{\mathsf{T}}$ such that $\mathsf{dom}(\mathbf{c}') = \mathsf{dom}(\mathbf{c})$, $\mathsf{MM}_{\mathbf{c}}$ stores pairs

$$\left(\left\langle \chi(\mathbf{c}), \chi(\mathbf{c}') \right\rangle, \left(\chi(\mathbf{r}), \chi(\mathbf{r}')\right)_{(\mathbf{r},\mathbf{r}') \in \mathsf{DB}_{\mathsf{att}(\mathbf{c})=\mathsf{att}(\mathbf{c}')}}\right).$$

To speed up access to the multi-map $\mathsf{MM}_{\mathbf{c}}$, we store it in a dictionary DX. That is, for all $\mathbf{c} \in \mathsf{DB}^{\mathsf{T}}$, we set

$$\mathsf{DX}[\chi(\mathbf{c})] := \mathsf{MM}_{\mathbf{c}}.$$

Note that, in practice, we could store a pointer to $\mathsf{MM}_{\mathbf{c}}$ in the dictionary instead.

Indexed execution of HNF queries. We now show how to perform an *indexed* execution of heuristic normal form queries using these structures. In Appendix B, we provide a concrete example that walks through our indexed HNF approach. For clarity, we use a small database composed of two tables and a simple SQL query. We hope that this example clarifies some of the ideas behind our construction.

Recall that HNF queries have form

$$\pi_{\mathsf{att}_1,\cdots,\mathsf{att}_h}\Big([a_1] \times \cdots \times [a_f] \times \varphi\Big),$$

where $\varphi = \varphi_1 \times \cdots \times \varphi_d$ with each φ_i having form either $\sigma_{\mathsf{att}_1=a_1}(\mathsf{T}) \cap \cdots \cap \sigma_{\mathsf{att}_p=a_p}(\mathsf{T})$ or $\sigma_{\mathsf{att}_1=\mathsf{att}_2}(\mathsf{T}_1 \times \mathsf{T}_2)$. We process each φ_i and create a set R_i of rows as follows:

- **(Case 1)** If φ_i has form $\sigma_{\mathsf{att}_1=a_1}(\mathsf{T}) \cap \cdots \cap \sigma_{\mathsf{att}_p=a_p}(\mathsf{T})$ we recover for each term $\sigma_{\mathsf{att}_j=a_j}(\mathsf{T})$ a set R'_j by computing

  $$\Big(\chi(\mathbf{r})\Big)_{\mathbf{r} \in \mathsf{DB}_{\mathsf{att}_j=a_j}} := \mathsf{MM}_V\Big[\Big\langle a_j, \chi(\mathsf{att}_j)\Big\rangle\Big]$$

  and querying MM_R on each of the returned row coordinates. We then set

  $$R_i = R'_1 \cap \cdots \cap R'_p.$$

- **(Case 2)** If φ_i has form $\sigma_{\mathsf{att}_1=\mathsf{att}_2}(\mathsf{T}_1 \times \mathsf{T}_2)$, we first compute $\mathsf{MM}_{\mathsf{att}_1} := \mathsf{DX}[\chi(\mathsf{att}_1)]$ and

  $$\Big(\chi(\mathbf{r}_1), \chi(\mathbf{r}_2)\Big)_{(\mathbf{r}_1,\mathbf{r}_2) \in \mathsf{DB}_{\mathsf{att}_1=\mathsf{att}_2}} := \mathsf{MM}_{\mathsf{att}_1}\Big[\Big\langle \chi(\mathsf{att}_1), \chi(\mathsf{att}_2)\Big\rangle\Big].$$

  Then we query MM_R on all of the returned row coordinates to produce a set

  $$R_i := \Big\{\mathbf{r}_1 \times \mathbf{r}_2\Big\}_{(\mathbf{r}_1,\mathbf{r}_2) \in \mathsf{DB}_{\mathsf{att}_1=\mathsf{att}_2}}.$$

After processing $\varphi_1, \ldots, \varphi_d$, we compute a temporary table

$$\mathsf{S} := [a_1] \times \cdots \times [a_f] \times R_1 \times \cdots \times R_d.$$

We then consider the set of attributes in the project operation that are in tables that appear in the select operation. Specifically, this is the set:

$$I = \left\{ \mathsf{att} \in S : \mathsf{att} \in \bigcup_{j=1}^{t} \mathbb{S}(\mathsf{T}_{i_j}) \right\},$$

where $S \stackrel{def}{=} \{\mathsf{att}_1, \ldots, \mathsf{att}_h\}$. Suppose I has $z \geq 1$ elements which we denote $(\mathsf{att}_1^i, \ldots, \mathsf{att}_z^i)$.We compute

$$\mathsf{W} := \pi_{\mathsf{att}_1^i, \ldots, \mathsf{att}_z^i}(\mathsf{S}).$$

We then consider the attributes in the project operation that are not in the tables that appear in the select operation; that is, the set $O = S \setminus I$. Suppose O has $h - z$ elements which we denote $(\mathsf{att}_1^o, \ldots, \mathsf{att}_{h-z}^o)$. For all $1 \leq j \leq h - z$, we compute

$$\mathbf{c}_j := \mathsf{MM}_c[\chi(\mathsf{att}_j^o)].$$

Finally, we generate the result table

$$\mathsf{R} := \mathbf{c}_1 \times \cdots \times \mathbf{c}_{h-z} \times \mathsf{W},$$

where the $\mathbf{c}_j$'s are viewed as single-column tables.

5.2 Detailed Construction

We now describe our SPX construction at a high-level, refer to Figs. 1 and 2 for more details. 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 $\Sigma_{\mathsf{DX}} = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Get})$, 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 [18, 15, 28, 11, 10, 33].

Overview. At a high-level, the Setup algorithm takes as input a database $\mathsf{DB} = (\mathsf{T}_1, \ldots, \mathsf{T}_n)$, creates the multi-maps MM_R, MM_C, MM_V, $\{\mathsf{MM}_{\mathbf{c}}\}_{\mathbf{c} \in \mathsf{DB}^\intercal}$ and the dictionary DX, as described above, and then encrypts each structure with the appropriate structured encryption scheme. The Token algorithm works by parsing the heuristic normal form query and generating appropriate tokens for each structure so as to enable the server to perform an indexed execution of the query (over encrypted data) as described in the previous paragraph.

Setup. The Setup algorithm takes as input a relational database $\mathsf{DB} = (\mathsf{T}_1, \ldots, \mathsf{T}_n)$ and indexes it as above. This results in three multi-maps MM_R, MM_V and MM_C and a dictionary DX that stores pointers to an additional set of multi-maps $\{\mathsf{MM}_{\mathbf{c}}\}_{\mathbf{c} \in \mathsf{DB}^\intercal}$. The algorithm then encrypts every row $\mathbf{r}$ in MM_R using SKE. In other words, MM_R now holds value/tuple pairs of the form

$$\Big(\chi(\mathbf{r}), \Big(\mathsf{Enc}_{K_1}(r_1), \ldots, \mathsf{Enc}_{K_1}(r_{\#\mathbf{r}})\Big)\Big).$$

It then encrypts MM_R with Σ_{MM} which results in a key K_R and an encrypted multi-map EMM_R. It then encrypts every column $\mathbf{c}$ in MM_C using SKE in the same manner as above and encrypts MM_C with Σ_{MM}. This results in K_C and an encrypted multi-map EMM_C.

Now for all $\mathbf{r} \in \mathsf{DB}$, it replaces all occurrences of $\chi(\mathbf{r})$ in MM_V and $\{\mathsf{MM}_{\mathbf{c}}\}_{\mathbf{c}\in\mathsf{DB}^{\mathsf{T}}}$ with

$$\mathsf{rtk}_{\mathbf{r}} := \Sigma_{\mathsf{MM}}.\mathsf{Token}(K_R, \chi(\mathbf{r})).$$

It then encrypts MM_V and $\{\mathsf{MM}_{\mathbf{c}}\}_{\mathbf{c}\in\mathsf{DB}}$ with Σ_{MM} which results in keys K_V and $\{K_{\mathbf{c}}\}_{\mathbf{c}\in\mathsf{DB}^{\mathsf{T}}}$ and encrypted multi-maps EMM_V and $\{\mathsf{EMM}_{\mathbf{c}}\}_{\mathbf{c}\in\mathsf{DB}^{\mathsf{T}}}$. It then stores pairs $(\chi(\mathbf{c}), \mathsf{EMM}_{\mathbf{c}})_{\mathbf{c}\in\mathsf{DB}^{\mathsf{T}}}$ in a dictionary DX and encrypts DX with Σ_{DX} which results in a key K_D and an encrypted dictionary EDX.

Finally, the Setup algorithms then outputs the key

$$K = (K_1, K_R, K_V, K_C, K_D, \{K_{\mathbf{c}}\}_{\mathbf{c}\in\mathsf{DB}^{\mathsf{T}}}),$$

and the encrypted database

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

Token. The Token algorithm takes as input a secret key K and a query q in SPC normal form. It first transforms it in heuristic normal form:

$$\pi_{\mathsf{att}_1,\cdots,\mathsf{att}_h}\Big([a_1] \times \cdots [a_f] \times \varphi_1 \times \cdots \times \varphi_d\Big).$$

For all $i \in [h]$, if the project attribute att_i does not appear in $\varphi_1 \times \cdots \times \varphi_d$, the algorithm computes

$$\mathsf{ptk}_i := \Sigma_{\mathsf{MM}}.\mathsf{Token}(K_C, \chi(\mathsf{att}_i)),$$

and sets $\mathsf{ytk}_i = (\mathsf{ptk}_i, \mathtt{out})$; otherwise it sets

$$\mathsf{ptk}_i := \mathtt{pos}_i,$$

where $\mathtt{pos}_i \in \left[\sum_{j=1}^{t} \|\mathsf{T}_{i_j}\|_c\right]$ denotes the position of the attribute in the tables referenced in $\varphi_1 \times \cdots \times \varphi_d$. It then sets $\mathsf{ytk}_i = (\mathsf{ptk}_i, \mathtt{in})$.

For every constant a_1 through a_f it computes $\mathrm{e}_1 \leftarrow \mathsf{Enc}_{K_1}(a_1)$ through $\mathrm{e}_f \leftarrow \mathsf{Enc}_{K_1}(a_f)$. It then processes φ_1 through φ_d and for each φ_i it does the following:

- **(Case 1)** if φ_i has form $\sigma_{\mathsf{att}_1=a_1}(\mathsf{T}) \cap \cdots \cap \sigma_{\mathsf{att}_p=a_p}(\mathsf{T})$, it sets

  $$\mathsf{stk}_i := (\mathsf{itk}_1, \ldots, \mathsf{itk}_p),$$

  where, for all $j \in [p]$,

  $$\mathsf{itk}_j := \Sigma_{\mathsf{MM}}.\mathsf{Token}(K_V, \langle a_j, \chi(\mathsf{att}_j)\rangle).$$

- **(Case 2)** if φ_i has form $\sigma_{\mathsf{att}_1=\mathsf{att}_2}(\mathsf{T}_1 \times \mathsf{T}_2)$ it sets $\mathsf{stk}_i := (\mathsf{dtk}_i, \mathsf{jtk}_i)$, where

  $$\mathsf{dtk}_i := \Sigma_{\mathsf{DX}}.\mathsf{Token}(K_D, \chi(\mathsf{att}_1))$$

  and

  $$\mathsf{jtk}_i := \Sigma_{\mathsf{MM}}.\mathsf{Token}\Big(K_{\mathbf{c}}, \Big\langle \chi(\mathsf{att}_1), \chi(\mathsf{att}_2)\Big\rangle\Big).$$

Finally, it outputs the token

$$\mathsf{tk} = \Big((\mathsf{ytk}_i)_{i\in[h]}, (e_i)_{i\in[f]}, (\mathsf{stk}_i)_{i\in[d]}\Big).$$

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 $(\mathsf{MM}_{\mathsf{att}})_{\mathsf{att} \in \mathbb{S}(\mathsf{DB})}$;
  4. for all $\mathbf{r} \in \mathsf{DB}$ set
  $$\mathsf{MM}_R\big[\chi(\mathbf{r})\big] := \Big(\mathsf{Enc}_{K_1}(r_1), \ldots \mathsf{Enc}_{K_1}(r_{\#\mathbf{r}}), \chi(\mathbf{r})\Big);$$
  5. compute $(K_R, \mathsf{EMM}_R) \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Setup}\big(1^k, \mathsf{MM}_R\big)$;
  6. for all $\mathbf{c} \in \mathsf{DB}^{\mathsf{T}}$, set
  $$\mathsf{MM}_C\big[\chi(\mathbf{c})\big] := \Big(\mathsf{Enc}_{K_1}(c_1), \ldots \mathsf{Enc}_{K_1}(c_{\#\mathbf{c}}), \chi(\mathbf{c})\Big);$$
  7. compute $(K_C, \mathsf{EMM}_C) \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Setup}\big(1^k, \mathsf{MM}_C\big)$;
  8. for all $\mathbf{c} \in \mathsf{DB}^{\mathsf{T}}$,
     (a) for all $v \in \mathbf{c}$ and $\mathbf{r} \in \mathsf{DB}_{\mathbf{c}=v}$,
        i. compute $\mathsf{rtk}_{\mathbf{r}} \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}\Big(K_R, \chi(\mathbf{r})\Big)$,
     (b) set
     $$\mathsf{MM}_V\Big[\big\langle v, \chi(\mathbf{c})\big\rangle\Big] := \Big(\mathsf{rtk}_{\mathbf{r}}\Big)_{\mathbf{r} \in \mathsf{DB}_{\mathbf{c}=v}};$$
  9. compute $(K_V, \mathsf{EMM}_V) \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Setup}\big(1^k, \mathsf{MM}_V\big)$;
  10. for all $\mathbf{c} \in \mathsf{DB}^{\mathsf{T}}$,
      (a) for all $\mathbf{c}' \in \mathsf{DB}^{\mathsf{T}}$ 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 $\mathsf{rtk}_i \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}\Big(K_R, \Big\langle \mathsf{tbl}(\mathbf{c}), i \Big\rangle\Big)$;
            B. compute $\mathsf{rtk}_j \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}\Big(K_R, \Big\langle \mathsf{tbl}(\mathbf{c}'), j \Big\rangle\Big)$;
            C. add $(\mathsf{rtk}_i, \mathsf{rtk}_j)$ to $\mathbf{t}$;
         iii. set
         $$\mathsf{MM}_{\mathbf{c}}\Big[\Big\langle \chi(\mathbf{c}), \chi(\mathbf{c}') \Big\rangle\Big] := \mathbf{t};$$
      (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 $\mathsf{DX}\big[\chi(\mathbf{c})\big] = \mathsf{EMM}_{\mathbf{c}}$;
  11. compute $(K_D, \mathsf{EDX}) \leftarrow \Sigma_{\mathsf{DX}}.\mathsf{Setup}(1^k, \mathsf{DX})$;
  12. output $K = \big(K_R, K_C, K_V, K_D, \{K_{\mathbf{c}}\}_{\mathbf{c} \in \mathsf{DB}^{\mathsf{T}}}\big)$ 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 R.

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

- $\mathsf{Token}(K, q)$:
  1. rewrite q as $\pi_{\mathsf{att}_1,\cdots,\mathsf{att}_h}\Big([a_1] \times \cdots \times [a_f] \times \varphi_1 \times \cdots \times \varphi_d\Big)$;
  2. for all $i \in [h]$,
     (a) if att_i is not in $\varphi_1 \times \cdots \times \varphi_d$, compute $\mathsf{ptk}_i := \Sigma_{\mathsf{MM}}.\mathsf{Token}\Big(K_C, \chi(\mathsf{att}_i)\Big)$, and set $\mathsf{ytk}_i := (\mathsf{ptk}_i, \mathsf{out})$;
     (b) else set $\mathsf{ptk}_i := (\mathsf{ptk}_i, \mathsf{in})$, where $\mathsf{ptk}_i = \mathtt{pos}_i$ and $\mathtt{pos}_i$ is att_i's position in the tables in $\varphi_1 \times \cdots \times \varphi_d$;
  3. for all $i \in [d]$,
     (a) if φ_i has form $\sigma_{\mathsf{att}_1=a_1}(\mathsf{T}) \cap \cdots \cap \sigma_{\mathsf{att}_p=a_p}(\mathsf{T})$, set $\mathsf{stk}_i := (\mathsf{itk}_1, \ldots, \mathsf{itk}_p)$, where, for all $j \in [p]$, $\mathsf{itk}_j := \Sigma_{\mathsf{MM}}.\mathsf{Token}\big(K_V, \langle a_i, \chi(\mathsf{att}_i)\rangle\big)$;
     (b) if φ_i has form $\sigma_{\mathsf{att}_1=\mathsf{att}_2}(\mathsf{T}_1 \times \mathsf{T}_2)$, set $\mathsf{stk}_i := (\mathsf{dtk}_i, \mathsf{jtk}_i)$ where
     $$\mathsf{dtk}_i := \Sigma_{\mathsf{DX}}.\mathsf{Token}\big(K_D, \chi(\mathsf{att}_1)\big) \quad \text{and} \quad \mathsf{jtk}_i := \Sigma_{\mathsf{MM}}.\mathsf{Token}\big(K_{\mathbf{c}}, \big\langle \chi(\mathsf{att}_1), \chi(\mathsf{att}_2)\big\rangle\big);$$
  4. for all $i \in [f]$, compute $\mathrm{e}_i \leftarrow \mathsf{Enc}_{K_1}(a_i)$;
  5. output $\mathsf{tk} = \big((\mathsf{ytk})_{i\in[h]}, (\mathrm{e}_i)_{i\in[f]}, (\mathsf{stk}_i)_{i\in[d]}\big)$;
- $\mathsf{Query}(\mathsf{EDB}, \mathsf{tk})$:
  1. parse EDB as $\big(\mathsf{EMM}_R, \mathsf{EMM}_C, \mathsf{EMM}_V, \mathsf{EDX}\big)$ and tk as $\big((\mathsf{ytk})_{i\in[h]}, (\mathrm{e}_i)_{i\in[f]}, (\mathsf{stk}_i)_{i\in[d]}\big)$;
  2. for $i \in [d]$,
     (a) if stk_i has form $(\mathsf{itk}_1, \ldots, \mathsf{itk}_p)$,
        i. for $j \in [p]$,
           A. instantiate an empty set R'_j,
           B. compute $(\mathsf{rtk}_1, \ldots, \mathsf{rtk}_s) := \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_V, \mathsf{itk}_j)$,
           C. for all $m \in [s]$, set $R'_j := R'_j \bigcup \{\Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_R, \mathsf{rtk}_m)\}$,
        ii. set $R_i = R'_1 \cap \cdots \cap R'_p$;
     (b) if stk_i has form $(\mathsf{dtk}_i, \mathsf{jtk}_i)$,
        i. compute $\mathsf{EMM} := \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EDX}, \mathsf{dtk}_i)$,
        ii. compute $\big((\mathsf{rtk}_1, \mathsf{rtk}'_1), \ldots, (\mathsf{rtk}_s, \mathsf{rtk}'_s)\big) := \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}, \mathsf{jtk}_i)$,
        iii. compute
        $$\mathbf{ct}_1 := \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_R, \mathsf{rtk}_1), \ldots, \mathbf{ct}_s := \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_R, \mathsf{rtk}_s),$$
        and
        $$\mathbf{ct}'_1 := \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_R, \mathsf{rtk}'_1), \ldots, \mathbf{ct}'_s := \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_R, \mathsf{rtk}'_s),$$
        iv. set $R_i = \Big\{\mathbf{ct}_j \times \mathbf{ct}'_j\Big\}_{j\in[s]}$;
  3. set $\mathsf{S} = \mathrm{e}_{a_1} \times \cdots \times \mathrm{e}_{a_f} \times R_1 \times \cdots \times R_d$;
  4. for all the z sub-tokens ytk with form $(\mathsf{ptk}_i, \mathsf{in})$, compute,
  $$\mathsf{W} := \pi_{\mathsf{ptk}_1,\cdots,\mathsf{ptk}_z}(\mathsf{S});$$
  5. for all the $(h - z)$ sub-tokens ytk with form $(\mathsf{ptk}_i, \mathsf{out})$, compute,
  $$\mathbf{ct}_i := \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_C, \mathsf{ptk}_i);$$
  6. output $\mathsf{R} := \mathbf{ct}_1 \times \cdots \times \mathbf{ct}_{h-z} \times \mathsf{W}$;

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

Query. The Query algorithm works like the plaintext indexed HNF query evaluation algorithm we described above. Given a token

$$\mathsf{tk} = \left((\mathsf{ytk}_i)_{i\in[z]}, (e_i)_{i\in[f]}, (\mathsf{stk}_i)_{i\in[d]} \right)$$

as input, it process the sub-tokens $(\mathsf{stk}_1, \ldots, \mathsf{stk}_d)$. For each stk_i it recovers a set of encrypted rows R_i as follows:

- **(Case 1)** if stk_i has form $(\mathsf{itk}_1, \ldots, \mathsf{itk}_p)$, then it recovers, for all $j \in [p]$, a set R'_j by first computing

  $$(\mathsf{rtk}_1, \ldots, \mathsf{rtk}_s) := \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_V, \mathsf{itk}_j).$$

  It then computes

  $$\mathbf{ct}_1 := \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_R, \mathsf{rtk}_1), \ldots, \mathbf{ct}_s := \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_R, \mathsf{rtk}_s),$$

  and sets $R'_j := \{\mathbf{ct}_1, \ldots, \mathbf{ct}_s\}$. Finally, it sets $R_i = R'_1 \cap \cdots \cap R'_p$.

- **(Case 2)** if stk_i has form $(\mathsf{dtk}_i, \mathsf{jtk}_i)$ it first computes

  $$\mathsf{EMM}_{\mathbf{c}} := \Sigma_{\mathsf{DX}}.\mathsf{Get}(\mathsf{EDX}, \mathsf{dtk}_i)$$

  and

  $$\left((\mathsf{rtk}_1, \mathsf{rtk}'_1), \ldots, (\mathsf{rtk}_s, \mathsf{rtk}'_s) \right) := \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_{\mathbf{c}}, \mathsf{jtk}_i).$$

  It then computes

  $$\mathbf{ct}_1 := \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_R, \mathsf{rtk}_1), \ldots, \mathbf{ct}_s := \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_R, \mathsf{rtk}_s),$$

  and

  $$\mathbf{ct}'_1 := \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_R, \mathsf{rtk}'_1), \ldots, \mathbf{ct}'_s := \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_R, \mathsf{rtk}'_s).$$

  Finally, it sets $R_i = \left\{ \mathbf{ct}_j \times \mathbf{ct}'_j \right\}_{j\in[s]}$.

After processing stk_1 through stk_d, it creates the temporary encrypted table

$$\mathsf{S} = \mathrm{e}_{a_1} \times \cdots \times \mathrm{e}_{a_f} \times R_1 \times \cdots \times R_d.$$

Let $(\mathsf{ytk}^i_1, \ldots, \mathsf{ytk}^i_z)$ be the ytk sub-tokens with form $(\mathsf{ptk}_i, \mathsf{in})$. It then computes

$$\mathsf{W} := \pi_{\mathsf{ptk}_1, \cdots, \mathsf{ptk}_z}(\mathsf{S}).$$

Let $(\mathsf{ytk}^o_1, \ldots, \mathsf{ytk}^o_{h-z})$ be the ytk sub-tokens with form $(\mathsf{ptk}_i, \mathsf{out})$. For all $i \in [h-z]$, it computes

$$\mathbf{ct}_i := \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_C, \mathsf{ptk}_i).$$

Finally, it generates the response table

$$\mathsf{R} := \mathbf{ct}_1 \times \cdots \times \mathbf{ct}_{h-z} \times \mathsf{W},$$

where the encrypted column $\mathbf{ct}_i$ is viewed as a single-column table.

Decryption. The Dec algorithm takes as input a secret key K and the response table R returned by the server and simply decrypts each cell of R.

5.3 Efficiency

We now turn to analyzing the search and storage efficiency of our construction.

Search complexity. Consider an SPC query written in its heuristic normal form

$$\pi_{\mathsf{att}_1,\cdots,\mathsf{att}_h}\Big([a_1]\times\cdots[a_f]\times\varphi_1\times\cdots\times\varphi_d\Big).$$

We show in Appendix A that the size of the result table over a plaintext database (in cells) is linear in

$$\#\mathsf{R} = h\cdot(m^{h-z}\cdot\prod_{i=1}^{d}\#R_i), \tag{1}$$

where $z = \#\Big\{\mathsf{att}\in S : \mathsf{att}\in\bigcup_{j=1}^{t}\mathbb{S}(\mathsf{T}_{i_j})\Big\}$ and $= S \stackrel{def}{=} \{\mathsf{att}_1,\ldots,\mathsf{att}_h\}$, and R_i is the set of rows returned by the evaluation of the term φ_i.

Theorem 5.1. *If Σ_{DX} and Σ_{MM} are optimal dictionary and multi-map encryption schemes, then the time and space complexity of the* Query *algorithm presented in Section 5.2 is*

$$O\left(\frac{\#\mathsf{R}}{h}\cdot\sum_{i=1}^{t}s_i\right),$$

where h is the number of selected attributes, s_i the number of attributes of the ith table for all $i\in[t]$, and $\#\mathsf{R}$ is the size of the result table over plaintext database as shown in Eq. 1.

Proof. Let

$$\mathsf{tk} = \Big((\mathsf{ytk}_i)_{i\in[h]}, (e_i)_{i\in[f]}, (\mathsf{stk}_i)_{i\in[d]}\Big),$$

be the token for the above query and let $\mathsf{EDB} = (\mathsf{EMM}_R, \mathsf{EMM}_C, \mathsf{EMM}_V, \mathsf{EDX})$. For ease of exposition, we assume here that all tables have the same number of rows m. When processing the select tokens, the Query algorithm will encounter two cases:

(Case 1). if φ_i has form $\sigma_{\mathsf{att}_{i,1}=a_{i,1}}(\mathsf{T}_i)\times\cdots\times\sigma_{\mathsf{att}_{i,p_i}=a_{i,p_i}}(\mathsf{T}_i)$, then stk_i will have form $(\mathsf{itk}_{i,1},\ldots,\mathsf{itk}_{i,p_i})$ and the Query algorithm will query EMM_V on $\mathsf{itk}_{i,1},\ldots,\mathsf{itk}_{i,p_i}$ to recover p_i row tokens which it will in turn use to query EMM_R to recover sets of encrypted rows $R_{i,1},\ldots,R_{i,p_i}$. It then takes the intersection between these sets such that $R_i = R_{i,1}\cap\cdots\cap R_{i,p_i}$. This requires

$$O\Big(p_i\cdot\max_{j\in[p_i]}\#R_{i,j}+\sum_{j=1}^{p_i}\#R_{i,j}\Big) = O\Big(p_i\cdot\max_{j\in[p_i]}\#R_{i,j}\Big) = O\Big(p_i\cdot\max_{j\in[p_i]}\#\mathsf{DB}_{\mathsf{att}_{i,j}=a_{i,j}}\Big)$$

time and $O(\max_{i\in[p_i]}\#\mathsf{DB}_{\mathsf{att}_{i,j}=a_{i,j}})$ space.

(Case 2). if φ_i has form $\sigma_{\mathsf{att}_{i,1}=\mathsf{att}_{i,2}}(\mathsf{T}_{i,1} \times \mathsf{T}_{i,2})$, then stk_i has form $(\mathsf{dtk}_i, \mathsf{jtk}_i)$ and the Query algorithm queries the encrypted dictionary EDX, from which it retrieves the encrypted multi-map $\mathsf{EMM_c}$. The algorithm then queries $\mathsf{EMM_c}$ on jtk_i to recover a tuple of $\#\mathsf{DB}_{\mathsf{att}_{i,1}=\mathsf{att}_{i,2}}$ row tokens which it then uses to query EMM_R. This requires $O(\#\mathsf{DB}_{\mathsf{att}_{i,1}=\mathsf{att}_{i,2}})$ time and space. To sum up, Query processes the select tokens in

$$\begin{aligned} O\Big(\sum_{i=1}^{u} p_i \cdot \max_{j\in[p_i]} \#\mathsf{DB}_{\mathsf{att}_{i,j}=a_{i,j}} + \sum_{i=u+1}^{d} \#\mathsf{DB}_{\mathsf{att}_{i,1}=\mathsf{att}_{i,2}}\Big) &\leq \sum_{i=1}^{u} p_i \cdot m + \sum_{i=1}^{d} \#R_i \\ &= O\Big(\ell \cdot m + \sum_{i=1}^{d} \#R_i\Big) \end{aligned}$$

time and $O\big(\max(d \cdot m, \sum_{i=1}^{d} \#R_i)\big)$ space, where ℓ is the number of terms in the predicate Ψ of the query's SPC normal form. The equality above holds since $\max_{j\in[p_i]} \#\mathsf{DB}_{\mathsf{att}_{i,j}=a_{i,j}} \leq m$, and $\sum_{i=1}^{u} p_i \leq \ell$.

After evaluating the select tokens, the algorithm computes the cross product $\mathsf{S} = R_1 \times \cdots \times R_d$ between all the R_i. The complexity of this operation is linear in $O(\prod_{i=1}^{d} \#R_i)$.

When processing the project tokens ytk that have form $(\mathsf{ptk}, \mathsf{out})$, the algorithm first retrieves $h - z$ columns from EMM_C. Both the time and space complexities of this step are $O((h-z) \cdot m)$. It then computes the cross product between the $h - z$ retrieved columns and S, which has both time and space complexity

$$O\left(m^{h-z} \cdot \left(\prod_{i=1}^{d} \#R_i\right) \cdot \left(h - z + \sum_{i=1}^{t} s_i\right)\right).$$

Finally, the server outputs the h desired columns. In summary, Query requires

$$O\left(\ell \cdot m + \sum_{i=1}^{d} \#R_i + \prod_{i=1}^{d} \#R_i + m^{h-z} \cdot \left(\prod_{i=1}^{d} \#R_i\right) \cdot \left(h - z + \sum_{i=1}^{t} s_i\right)\right)$$

space and time which is

$$O\left(m^{h-z} \cdot \left(\prod_{i=1}^{d} \#R_i\right) \cdot \left(h - z + \sum_{i=1}^{t} s_i\right)\right)$$

when $\ell \cdot m + \sum_{i=1}^{d} \#R_i \ll \prod_{i=1}^{d} \#R_i$. It follows by Eq (1) that this is

$$O\left(\frac{\#\mathsf{R}}{h} \cdot \left(h - z + \sum_{i=1}^{t} s_i\right)\right) = O\left(\frac{\#\mathsf{R}}{h} \cdot \sum_{i=1}^{t} s_i\right)$$

since $z \leq h$.

■

Corollary 5.2. *If $h^{-1} \cdot \sum_{i=1}^{t} s_i$ is a constant in $\#\mathsf{R}$, then both time and space complexity are in $O(\#\mathsf{R})$, which is optimal.*

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. For ease of exposition, we again assume each table has m rows. Finally, note that standard multi-map encryption schemes [18, 28, 11, 10] 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 $O(\sum_{\mathbf{c}\in\mathsf{DB}^\mathsf{T}} \#\mathbf{c})$, 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 $O(\sum_{\mathbf{c}\in\mathsf{DB}^\mathsf{T}} \sum_{v\in\mathbf{c}} \#\mathsf{DB}_{\mathsf{att}(\mathbf{c})=v})$ storage. EDX maps the coordinates of each column $\mathbf{c} \in \mathsf{DB}^\mathsf{T}$ to an encrypted multi-map $\mathsf{EMM}_\mathbf{c}$ which in turn maps each pair of form $(\mathbf{c}, \mathbf{c}')$ such that $\mathsf{dom}(\mathsf{att}(\mathbf{c})) = \mathsf{dom}(\mathsf{att}(\mathbf{c}'))$ to a tuple of tokens for rows in $\mathsf{DB}_{\mathsf{att}(\mathbf{c})=\mathsf{att}(\mathbf{c}')}$. As such, EDX will have size

$$O\Big(\sum_{\mathbf{c}\in\mathsf{DB}^\mathsf{T}} \sum_{\mathbf{c}':\mathsf{dom}(\mathsf{att}(\mathbf{c}'))=\mathsf{dom}(\mathsf{att}(\mathbf{c}))} \#\mathsf{DB}_{\mathsf{att}(\mathbf{c})=\mathsf{att}(\mathbf{c}')}\Big).$$

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 $O((\sum_i \|\mathsf{T}_i\|_c)^2)$ terms of the form $\#\mathsf{DB}_{\mathsf{att}(\mathbf{c})=\mathsf{att}(\mathbf{c}')}$. 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.

In Appendix B, we provide a concrete example of the storage overhead of an encrypted database.

6 Security and Leakage of SPX

We show that our construction is adaptively-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 building blocks and part of it "non-black-box" in the sense that it comes directly from the SPX construction. 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 [18, 28, 15, 11, 10, 33] which all have the same leakage profile, i.e., the search pattern which reveals if and when a query is repeated. [3] 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.

6.1 Setup leakage

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

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

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

[3] 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.

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).

6.2 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

$$\mathrm{XPP}(\mathsf{DB}, q) = \Big\{\big(|a_i|\big)_{i\in[f]}\Big\},$$

and includes the size of the virtual attributes.

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

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

where

$$\mathcal{P}(\mathsf{att}_i) = \begin{cases} \Big(\mathtt{out}, \mathcal{L}_{\mathsf{Q}}^{\mathsf{mm}}\Big(\mathsf{MM}_C, \chi(\mathsf{att}_i)\Big), \big(|c_j|\big)_{j\in[\#\mathbf{c}_i]}, \mathrm{AccP}(\mathbf{c}_i)\Big) & \text{if } \mathsf{att}_i \in S \setminus I; \\ \Big(\mathtt{in}, \mathsf{att}_i\Big) & \text{otherwise,} \end{cases}$$

where $I = \Big\{\mathsf{att} \in S : \mathsf{att} \in \bigcup_{j=1}^{t} \mathbb{S}(\mathsf{T}_{i_j})\Big\}$ and $S \stackrel{def}{=} \{\mathsf{att}_1, \ldots, \mathsf{att}_h\}$, $\mathbf{c}_i \in \mathsf{DB}^{\mathsf{T}}$ denotes the column with attribute att_i and $\mathrm{AccP}(\mathbf{c}_i)$ indicates the access pattern, i.e., 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 att_i reveals whether the attribute was `in` or `out` of the set composed of the attributes in the predicate Ψ. If it is `out`, 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 att_i has appeared in a previous query. If the attribute is `in`, it just reveals the attribute.[4]

The projection pattern discloses the frequency of accesses made to a particular attribute. An adversary can learn the size of the accessed columns, and therefore the number of entries that a specific table has. The impact of such leaked information depends on the auxiliary information the attacker possesses. In some settings, just knowing the size of the table can be sufficient for an adversary to know the targeted information, but this is a general problem that can be addressed by padding, for instance.

[4]To be more precise, it reveals *only* the position of the attribute in the heuristic normal form. The position, however, is independent of the attribute itself.

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

$$\mathrm{SelP}(\mathsf{DB}, q) = \Big(\mathcal{Z}(\varphi_1), \ldots, \mathcal{Z}(\varphi_d)\Big).$$

If φ_i has form $\sigma_{\mathsf{att}_{i,1}=a_{i,1}}(\mathsf{T}) \cap \cdots \cap \sigma_{\mathsf{att}_{i,p_i}=a_{i,p_i}}(\mathsf{T})$, then $\mathcal{Z}(\varphi_i)$ is defined as

$$\mathcal{Z}(\varphi_i) = \Big(\text{case-1}, p_i, \Big(\mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\Big(\mathsf{MM}_V, \Big\langle X_{i,j}, \chi(\mathsf{att}_{i,j})\Big\rangle\Big), \Big\{\mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\Big(\mathsf{MM}_R, \chi(\mathbf{r})\Big), \mathrm{AccP}(\mathbf{r})\Big\}_{\mathbf{r}\in\mathsf{DB}_{\mathsf{att}_{i,j}=X_{i,j}}}\Big)_{j\in[p_i]}\Big),$$

where $\mathrm{AccP}(\mathbf{r})$ indicates whether the row $\mathbf{r}$ has been accessed before. $\mathcal{Z}(\varphi_i)$ 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 case-1, the Σ_{MM} query leakage on the constant a_j, and the coordinates of the attribute $\mathsf{att}_{i,j}$, for all $j \in [p_i]$ where p_i represents the number of attributes $\mathsf{att}_{i,j}$ that are in the same table T. In addition, it also leaks the Σ_{MM} query leakage on the coordinates of the rows in $\mathsf{DB}_{\mathsf{att}_{i,j}=a_{i,j}}$ as well as if and when they have been accessed before, for all $j \in [p_i]$. If the encrypted multi-maps are instantiated with standard constructions, $\mathcal{Z}(\varphi_i)$ amounts to: if and when the pair $(a_{i,j}, \mathsf{att}_{i,j})$ has been queried before and if and when any of the rows in $\mathsf{DB}_{\mathsf{att}_{i,j}=a_{i,j}}$ have been accessed in the past, for all $j \in [p_i]$.

If, on the other hand, φ_i has form $\sigma_{\mathsf{att}_{i,1}=\mathsf{att}_{i,2}}(\mathsf{T}_{i,1} \times \mathsf{T}_{i,2})$, then $\mathcal{Z}(\varphi_i)$ is defined as

$$\begin{aligned}\mathcal{Z}(\varphi_i) = \Big(&\text{case-2}, \mathcal{L}^{\mathsf{dx}}_{\mathsf{Q}}\Big(\mathsf{DX}, \chi(\mathsf{att}_{i,1})\Big), \mathcal{L}^{\mathsf{mm}}_{\mathsf{S}}(\mathsf{MM}_{\mathsf{att}_{i,1}}), \mathrm{AccP}(\mathsf{EMM}_{\mathsf{att}_{i,1}}),\\ &\mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\Big(\mathsf{MM}_{\mathsf{att}_{i,1}}, \Big\langle \chi(\mathsf{att}_{i,1}), \chi(\mathsf{att}_{i,2})\Big\rangle\Big), \Big\{\mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\Big(\mathsf{MM}_R, \chi(\mathbf{r}_1)\Big),\\ &\mathrm{AccP}(\mathbf{r}_1), \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\Big(\mathsf{MM}_R, \chi(\mathbf{r}_2)\Big), \mathrm{AccP}(\mathbf{r}_2)\Big\}_{(\mathbf{r}_1,\mathbf{r}_2)\in\mathsf{DB}_{\mathsf{att}_{i,1}=\mathsf{att}_{i,2}}}\Big),\end{aligned}$$

where $\mathrm{AccP}(\mathbf{r}_1)$, $\mathrm{AccP}(\mathbf{r}_2)$ and $\mathrm{AccP}(\mathsf{EMM}_{\mathsf{att}_i})$ indicate if and when $\mathbf{r}_1$, $\mathbf{r}_2$ and $\mathsf{EMM}_{\mathsf{att}_{i,1}}$ have been accessed before. In this case, $\mathcal{Z}(\varphi_i)$ captures the leakage produced when the server queries EDX to retrieve some $\mathsf{EMM}_{\mathsf{att}_{i,1}}$ which it in turn queries to retrieve row tokens with which to query EMM_R. It reveals whether the selection term is of case-2, the Σ_{DX} query leakage on the coordinates of $\mathsf{att}_{i,1}$, the Σ_{MM} *setup* leakage on $\mathsf{MM}_{\mathsf{att}_{i,1}}$ and if and when $\mathsf{EMM}_{\mathsf{att}_{i,1}}$ has been accessed in the past. In addition, it reveals the query leakage of Σ_{MM} on the coordinates of $\mathsf{att}_{i,1}$ and $\mathsf{att}_{i,2}$ and, for every pair of rows $(\mathbf{r}_1, \mathbf{r}_2)$ in $\mathsf{DB}_{\mathsf{att}_{i,1}=\mathsf{att}_{i,2}}$, 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 $\mathsf{att}_{i,1}$ had been queried in the past, the number of columns in DB that share a domain with $\mathsf{att}_{i,1}$, if and when the pair $(\mathsf{att}_{i,1}, \mathsf{att}_{i,2})$ has appeared in previous queries and, for every pair of rows in $\mathsf{DB}_{\mathsf{att}_{i,1}=\mathsf{att}_{i,2}}$, if and when these rows have been accessed in the past.

Of all the leakage patterns, the selection pattern is the one that leaks the most. If φ_i is of case-1, then an adversary can know the number of rows that contain the same value at a particular column(s), and this applies to all the p_i attributes in φ_i. The adversary can also learn the frequency with which a particular row has been accessed, and also the size of that row. If many queries have been performed on the same table and the same column, then the adversary can build a frequency histogram of that specific column's contents. Otherwise if φ_i is of case-2, then the server learns how many rows are equal to each other in both columns.

6.3 Security of SPX

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

Theorem 6.1. *If* SKE *is RCPA secure,* 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 is in Appendix F.

6.4 SPX and SSE Attacks

Our construction make use of SSE to instantiate the underlying encrypted multi-maps. There are several known attacks that try to exploit the leakage of various SSE schemes such as the inference attacks of Islam et al. [25] and of Cash et al. [9] and the file injection attacks of Cash et al. [9] and Zhang et al. [40]. It is not exactly clear what the impact of these attacks would be to our setting since our construction handles more complex objects and has a different leakage profile than standard SSE schemes. What is clear, however, is that our scheme leaks *more* than standard SSE schemes so presumably the techniques from these works could be extended to apply to our construction.

We note, however, that, as shown in [9], the SSE attacks from [25] and [9] do not work in practice. That is, to achieve even a moderate success rate the adversary needs to know a large percentage of the client's data (in addition to other knowledge such as some fraction of the client's queries). Specifically, in the experiments reported in [9], the adversary needs to know about 90% of the client's data in order to recover about 10% of its queries. Similarly, the counting attack from [9] requires the adversary to know 80% of the client's data in order to recover 40% of the client's queries (note that the success rate of the counting attack is not linear so knowing even 75% of the client's data is not enough for the adversary to learn even 1% of the client's queries). In a model where the adversary does not know any of the client's data a-priori—which is the standard model for SSE and structured encryption—neither the IKK attack nor counting attack of [9] can recover *any* queries at all. [5]

Unlike the previously mentioned attacks, the file injection attacks of [40] are effective in practice but are only applicable against dynamic SSE schemes and in scenarios where the adversary can inject data into the encrypted structure. This is the case, for example, if one were to use a dynamic SSE scheme to encrypt an email archive since the server/adversary could send the client malicious emails. In our setting, we assume the data is generated by the client and is not publicly modifiable *after the setup*. However, if our dynamic scheme SPX^+ were used in a setting where row injections are possible then, presumably, attacks like those of [40] could be designed and some queries could be disclosed. As suggested in [40], one countermeasure in this case is to use forward-secure constructions. In Appendix E, we discuss how to make SPX^+ forward-secure.

Recently, Abdelraheem et al. [31] presented an attack on relational databases encrypted with SSE. We stress, however, that the attack of [31] only applies to a very specific and naive SSE-based relational EDB construction described in that work and first used for experiments in [11] (e.g., the construction does not handle any non-trivial SQL query). While it is not clear at all how this attack

[5] Note that in adversarial settings where the adversary can know such a large fraction of the client's data, it would be more appropriate to use a primitive designed for public data like private information retrieval rather than SSE which is explicitly designed for private data.

would apply to our construction, we point out that the attack relies on strong assumptions. In particular, it works only for databases with attributes whose domain sizes are unique. In addition, it relies on the adversary knowing the attributes in the database and their domain sizes. Furthermore, the adversary also needs to know, for each attacked column, which domain element appears the most frequently, the second most frequently etc. Finally, the attack needs to solve an NP-complete problem that can be solved in pseudo-polynomial time only for databases with a small number of rows and small attribute domains (experimental results were conducted for databases with $32,561$ rows and domain sizes that range from 2 to 41 and execution times were not reported).

6.5 SPX with Zero-Leakage Building Blocks

In the previous Sections, we described the leakage of SPX when instantiated with encrypted multi-map schemes with the "standard" leakage profile. We also discussed known attacks against these schemes. Here, we are interested in the leakage profile of SPX when the underlying building blocks are ZL. By a ZL encrypted structure, we mean that its query operations only reveals information that can be derived from the security parameter or other public parameters. We write this as $\mathcal{L}_{\mathsf{Q}}(\mathsf{DS}, q) = \perp$, for any query q in its corresponding query space. When instantiated with ZL building blocks, the query leakage of SPX decreases considerably but its setup leakage remains the same. Specifically, the projection pattern becomes $\mathrm{PrP}(\mathsf{DB}, q) = \Big(\mathcal{P}(\mathsf{att}_1), \ldots, \mathcal{P}(\mathsf{att}_h)\Big)$, where

$$\mathcal{P}(\mathsf{att}_i) = \begin{cases} \Big(\mathtt{out}, \big(|c_j|\big)_{j \in [\#\mathbf{c}_i]}, \mathrm{AccP}(\mathbf{c}_i)\Big) & \text{if } \mathsf{att}_i \in S \setminus I; \\ \Big(\mathtt{in}, \mathsf{att}_i\Big) & \text{otherwise.} \end{cases}$$

The selection pattern SelP becomes $\mathrm{SelP}(\mathsf{DB}, q) = \Big(\mathcal{Z}(\varphi_1), \ldots, \mathcal{Z}(\varphi_d)\Big)$, where if φ_i has form $\sigma_{\mathsf{att}_{i,1}=a_{i,1}}(\mathsf{T}) \cap \cdots \cap \sigma_{\mathsf{att}_{i,p_i}=a_{i,p_i}}(\mathsf{T})$, then $\mathcal{Z}(\varphi_i)$ is defined as

$$\mathcal{Z}(\varphi_i) = \Big(\text{case-1}, p_i, \Big\{\mathrm{AccP}(\mathbf{r})\Big\}_{\mathbf{r} \in \mathsf{DB}_{\mathsf{att}_{i,j}=X_{i,j}}, j \in [p_i]}\Big),$$

Otherwise if, φ_i has form $\sigma_{\mathsf{att}_{i,1}=\mathsf{att}_{i,2}}(\mathsf{T}_{i,1} \times \mathsf{T}_{i,2})$, then $\mathcal{Z}(\varphi_i)$ is defined as

$$\mathcal{Z}(\varphi_i) = \Big(\text{case-2}, \mathcal{L}^{\mathsf{mm}}_{\mathsf{S}}(\mathsf{MM}_{\mathsf{att}_{i,1}}), \mathrm{AccP}(\mathsf{EMM}_{\mathsf{att}_{i,1}}), \Big\{\mathrm{AccP}(\mathbf{r}_1), \mathrm{AccP}(\mathbf{r}_2)\Big\}_{(\mathbf{r}_1, \mathbf{r}_2) \in \mathsf{DB}_{\mathsf{att}_{i,1}=\mathsf{att}_{i,2}}}\Big).$$

We are aware of two ZL encrypted multi-map constructions. The first can be derived from an SSE construction of Garg, Mohassel and Papamanthou [19] that itself is based on the TWORAM construction. We note that the SSE scheme proposed in that work is not ZL but it can be made so with a careful parametrization of its block size. [6] The second construction is an unpublished scheme

[6] The issue is that while the TWORAM-based SSE construction hides the search pattern, it still reveals the size of the query responses and this is non-trivial leakage because it can often be correlated with the search pattern.

called FZL. [7] Of course, ZL schemes come with an additional efficiency overhead. For example, if the TWORAM-based constructions is used in SPX its time and space complexity would incur an additive overhead of

$$\widetilde{O}\Big((2\ell + h) \cdot m \cdot \log(n \cdot m) + d \cdot m^2 \cdot \log\Big(\sum_{i \in [n]} \|\mathsf{T}_i\|_c \cdot m\Big)\Big),$$

where n is the number of tables in DB.

6.6 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 case-1 query on a table T, they will reveal a deterministic encryptions of all the accessed attributes $\mathbf{c}$ in T (i.e., every element of every column is encrypted under the same key). To handle a case-2 query 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 the entire columns to the server. Depending on the setting, frequency patterns can be particularly dangerous, as shown in [32].

SPX leaks considerably less. First, it does not leak any frequency information on entire columns or rows. For case-1 queries, it only leaks information about the attributes in the query and the rows that match the term. For case-2 queries, it only leaks information about the pair of attributes $(\mathsf{att}_{i,1}, \mathsf{att}_{i,2})$ 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.

Formally, the setup leakage of PPE-based solutions like CryptDB is

$$\mathcal{L}_{\mathsf{S}}^{\mathsf{ppe}}(\mathsf{DB}) = \Big(\|\mathsf{T}_i\|_c, \|\mathsf{T}_i\|_r\Big)_{i \in [n]},$$

where n is the number of tables in DB. Given a SQL query q, the query leakage is

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

where XPP, PrP and SelP are the cross product, projection and selection patterns (defined as in the leakage profile of SPX), and $\mathrm{FrP}(\mathsf{DB}, q)$ is the frequency pattern which leaks frequency information on all queried columns. It is easy to see that even when SPX is instantiated with non-ZL building blocks, its query leakage is a subset of the query leakage of the PPE-based solutions. Note that, not only is FrP relatively easy to exploit [32], it is also *persistent* in the sense that it is available not only to an adversary that has the query tokens and witnesses or executes the query operation but also to a "snapshot" adversary which only has access to the encrypted DB. This is not the case for SPX.

[7] This construction is not publicly available but we received a pre-print through private communication with the authors.

A remark on leakage. Ideally, one would hope to better understand how significant the leakage of practical encrypted search solutions are but we currently lack any theoretical framework to conduct such an analysis. In other words, the best we can currently do is to give a precise leakage profile and prove that our constructions do not leak anything beyond that profile. For the same reason, the best we can currently do to compare two leakage profiles is to show that one is a subset of the other (and in some cases, this is not even possible).

7 Extensions

In Appendix D, we show how to extend SPX to handle additional post-processing operations including Group-by, Order-by and various aggregate functions such as Sum, Average, Median, Count, Mode, Max and Min.

In addition, due to its modularity, SPX can be extended to be dynamic without re-designing it entirely. We refer to the dynamic version of SPX as SPX^+ and describe it in Appendix E. Note that SPX^+ maintains the same query complexity and query leakage as SPX. We also discuss how to use ORAM to make SPX^+ forward-secure at the cost of a poly-logarithmic overhead for updates and without affecting the query complexity.

8 Future Directions and Open Problems

In this work, we proposed the first encrypted relational database scheme purely based on STE techniques. As such, our construction offers more security than the PPE-based solutions and are more efficient than solutions based on general-purpose techniques like ORAM simulation or FHE. Our work leaves open several interesting questions. The first is whether our techniques can be extended to handle the *full* relational algebra which, effectively, is the entire SQL. To achieve this, our solution would have to be extended to handle negations and disjunctions (set unions) in the Where clause of the SQL query. We believe this to be challenging. A second problem is to handle SQL queries with ranges in the Where clause. A first step towards achieving this would be to improve the state of the art in encrypted range queries. In particular, finding schemes with improved leakage profiles is important since recent work [29, 30] has described powerful attacks against the state of the art encrypted search solutions (under some assumptions on the data and queries).

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] G. Asharov, M. Naor, G. Segev, and I. Shahaf. Searchable symmetric encryption: Optimal locality in linear space via two-dimensional balanced allocations. In *ACM on Symposium on Theory of Computing (STOC '16)*, 2016.

[4] M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In A. Menezes, editor, *Advances in Cryptology – CRYPTO '07*, 2007.

[5] 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.

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

[7] D. Boneh, G. di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Advances in Cryptology – EUROCRYPT '04*, 2004.

[8] R. Bost. Sophos - Forward Secure Searchable Encryption. In *ACM Conference on Computer and Communications Security (CCS '16)*, 2016.

[9] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart. Leakage-abuse attacks against searchable encryption. In *ACM Conference on Communications and Computer Security (CCS '15)*, pages 668–679. ACM, 2015.

[10] 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.

[11] 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.

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

[13] D. D. Chamberlin and R. F. Boyce. SEQUEL: A structured english query language. In *(SIGMOD '74)*, 1974.

[14] A. Chandra and P. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *(STOC '77)*, 1977.

[15] M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *Advances in Cryptology - ASIACRYPT '10*, volume 6477, 2010.

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

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

[18] 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.

[19] S. Garg, P. Mohassel, and C. Papamanthou. TWORAM: efficient oblivious RAM in two rounds with applications to searchable encryption. In *Advances in Cryptology - CRYPTO 2016*, 2016.

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

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

[22] 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.

[23] P. Grubbs, T. Ristenpart, and V. Shmatikov. Why your encrypted database is not secure. In *HotOS*, 2017.

[24] H. Hacigümü**c**s, B. Iyer, C. Li, and S. Mehrotra. Executing sql over encrypted data in the database-service-provider model. In *(SIGMOD'02)*, 2002.

[25] M. S. Islam, M. Kuzu, and M. Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *Network and Distributed System Security Symposium (NDSS '12)*, 2012.

[26] 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.

[27] S. Kamara and T. Moataz. Boolean searchable symmetric encryption with worst-case sub-linear complexity. In *Advances in Cryptology - EUROCRYPT '17*, 2017.

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

[29] G. Kellaris, G. Kollios, K. Nissim, and A. O. Neill. Generic attacks on secure outsourced databases. In *ACM Conference on Computer and Communications Security (CCS '16)*, 2016.

[30] M. Lacharité, B. Minaud, and K. G. Paterson. Improved reconstruction attacks on encrypted data using range query leakage. *IACR Cryptology ePrint Archive*, 2017:701, 2017.

[31] T. A. Mohamed Ahmed Abdelraheem and C. Gehrmann. Inference and record-injection attacks on searchable encrypted relational databases. Technical Report 2017/024, 2017. http://eprint.iacr.org/2017/024.

[32] M. Naveed, S. Kamara, and C. V. Wright. Inference attacks on property-preserving encrypted databases. In *ACM Conference on Computer and Communications Security (CCS)*, CCS '15, pages 644–655. ACM, 2015.

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

[34] 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.

[35] R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: Protecting confidentiality with encrypted query processing. In *ACM Symposium on Operating Systems Principles (SOSP)*, pages 85–100, 2011.

[36] D. Song, D. Wagner, and A. Perrig. Practical techniques for searching on encrypted data. In *IEEE Symposium on Research in Security and Privacy*, 2000.

[37] E. Stefanov, C. Papamanthou, and E. Shi. Practical dynamic searchable encryption with small leakage. In *Network and Distributed System Security Symposium (NDSS '14)*, 2014.

[38] E. Stefanov, M. van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas. Path oram: An extremely simple oblivious ram protocol. In *ACM Conference on Computer and Communications Security (CCS '13)*, 2013.

[39] 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.

[40] Y. Zhang, J. Katz, and C. Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In *USENIX Security Symposium*, 2016.

A SQL Combinatorics

To evaluate the efficiency of our solution, we will need to analyze its asymptotic time and space complexity as a function of various DB and query parameters. To enable this analysis, we show here how to compute several quantities of interest. Throughout this section, we assume $\mathsf{DB} = (\mathsf{T}_1, \ldots, \mathsf{T}_n)$ and that each table T_i has $m_i \geq 1$ rows.

Project on cross product. Let R be the table that results from applying the project operator $\pi_{\mathsf{att}_1,\ldots,\mathsf{att}_h}$ on $\mathsf{T}_{i_1} \times \cdots \times \mathsf{T}_{i_t}$. The size of R in cells is

$$\#\mathsf{R} = h \cdot \Pi_{i=1}^{t} m_i.$$

Select from cross product. Here, we are interested in the cell-size of tables that result from applying the projection operator to cross products:

$$\mathsf{R} := \sigma_{\Psi}(\mathsf{T}_{i_1} \times \cdots \times \mathsf{T}_{i_t}).$$

To calculate the size of the result table, we will re-arrange the predicate Ψ in various ways that make counting easier. In each case the re-written predicate, denoted $\widehat{\Psi}$, will have the form $\widehat{\Psi} = \psi_1 \wedge \cdots \wedge \psi_d$, where each ψ_i will have form $\psi_i = \mathsf{att}_{i,1} = X_{i,1} \wedge \cdots \wedge \mathsf{att}_{i,p_i} = X_{i,p_i}$. We will refer to the ψ_i as sub-predicates of Ψ and write $\mathsf{tbl}(\Psi)$ and $\mathsf{tbl}(\psi_i)$ to refer to the set of tables associated to the attributes in Ψ and ψ_i, respectively. More precisely, we define

$$\mathsf{tbl}(\Psi) = \bigcup_{i=1}^{\ell} \mathsf{tbl}(\mathsf{att}_i) \cup \mathsf{tbl}(X_i),$$

where $\mathsf{tbl}(a) = \emptyset$ for any constant a. Similarly, we define

$$\mathsf{tbl}(\psi_i) = \bigcup_{j=1}^{p_i} \mathsf{tbl}(\mathsf{att}_{i,j}) \cup \mathsf{tbl}(X_{i,j}).$$

We will consider three cases: (1) when Ψ is composed only of type-1 terms; (2) when Ψ is composed only of type-2 terms; and (3) when Ψ is composed of both type-1 and type-2 terms.

For type-1 queries, Ψ has the form $\mathsf{att}_1 = a_1 \wedge \cdots \wedge \mathsf{att}_\ell = a_\ell$, where $a_i \in \mathsf{dom}(\mathsf{att}_i)$. Note that ℓ can be different than t; that is, the number of attributes in the predicate can be different from the number of tables in the cross product since some attributes can belong to the same table. In such a case, we have to be careful not to overcount. In particular, for a given table we should only count rows that satisfy *all* the terms associated to that table. To do this, we first rewrite Ψ and gather all the attributes that belong to the same table together. In other words, we have $\widehat{\Psi} = \psi_1 \wedge \cdots \wedge \psi_d$, where all the attributes in ψ_i belong to the same table. In this case, note that σ_{ψ_i} is of **(case 1)** when the SPC query is written in its heuristics normal form. The size of the result table R is then

$$\#\mathsf{R} = \Big(\sum_{i \in \mathsf{tbl}(\widehat{\Psi})} \|\mathsf{T}_i\|_c \Big) \cdot \prod_{i=1}^{d} \# \bigcap_{j=1}^{p_i} R_{i,j}$$

where $R_{i,j} = \mathsf{DB}_{\mathsf{att}_{i,j}=a_{i,j}}$ is the set of rows that satisfy the jth term of ψ_i. We have to take the intersection $\bigcap_{j=1}^{p_i} R_{i,j}$ because we can only count the rows that that satisfy *all* the terms in ψ_i. Note that in the case where all attributes belong to different tables, the size of R is

$$\#\mathsf{R} = \Big(\sum_{i \in \mathsf{tbl}(\Psi)} \|\mathsf{T}_i\|_c \Big) \cdot \prod_{i=1}^{\ell} \#R_i,$$

where $R_i = \mathsf{DB}_{\mathsf{att}_i=a_i}$. This is the worst case since $\mathsf{tbl}(\widehat{\Psi}) \subseteq \mathsf{tbl}(\Psi)$, $d \leq \ell$ and we are taking intersections of the $R_{i,j}$'s in the first case.

The case of type-2 queries is more complex because there are two constraints that need to be taken into account. Similar to the type-1 queries, we need to consider cases where attributes in the predicate belong to the same table. In addition, however, we also need to consider cases where different terms include the same attribute: e.g., predicates of the form $\mathsf{att}_1 = \mathsf{att}_3 \wedge \mathsf{att}_2 = \mathsf{att}_3$. To handle this case, we rewrite Ψ as $\widehat{\Psi} = \psi_1 \wedge \cdots \wedge \psi_d$, where ψ_i includes all terms of Ψ with attributes that either belong to the same table or that share an attribute in common. Note that in this case, the ψ_i's can include attributes from *different* tables; that is, $\#\mathsf{tbl}(\psi_i) \geq 1$. The size of result table R is then equal to,

$$\#\mathsf{R} = \Big(\sum_{i \in \mathsf{tbl}(\widehat{\Psi})} \|\mathsf{T}_i\|_c \Big) \cdot \prod_{i=1}^{d} \#R_i, \tag{2}$$

where the set R_i is the concatenation of rows that satisfy ψ_i or, more precisely,

$$R_i = \Big\{ \langle \mathbf{r}_1, \ldots, \mathbf{r}_{p_i} \rangle : \psi_i\Big(\langle \mathbf{r}_1, \ldots, \mathbf{r}_{p_i} \rangle \Big) = 1 \Big\}.$$

Finally, we consider the case where Ψ contains terms that are either type-1 (i.e., equal to a constant) or type-2 (i.e., equal to another attribute). The challenges in this case are similar to ones for type-2 only since the constraint for the type-1 case is already taken into account in the way we counted the size of the result table for type-2. Thus, $\#\mathsf{R}$ in this case has a similar form as Eq. (2). For the simple case where all attributes in Ψ belong to different tables, we have

$$\#\mathsf{R} = \Big(\sum_{i \in \mathsf{tbl}(\Psi)} \|\mathsf{T}_i\|_c \Big) \cdot \prod_{i=1}^{d} \#R_i$$

where $R_i = \mathsf{DB}_{\mathsf{att}_i=X_i}$. Note that captures the **(case 2)** of the HNF query.

SPC normal form. We are now ready to compute the size of a result table R that results from applying an SPC normal form query

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

to a database $\mathsf{DB} = \Big(\mathsf{T}_1, \ldots, \mathsf{T}_n \Big)$. If $\widehat{\Psi} = \psi_1 \wedge \cdots \wedge \psi_d$ is the re-arranged form of $\Psi = (\mathsf{att}_1 = X_1 \wedge \cdots \wedge \mathsf{att}_\ell = X_\ell)$ such that each sub-predicate ψ_i contains terms of Ψ with attributes that either

ID	Name	Course
A05	Alice	16
A12	Bob	18

Name	Grade
Alice	A
Bob	B
Eve	A

Figure 3: Plaintext database DB.

share a common table or are dependent, then we have

$$\#\mathsf{R} = h \cdot \left(m^{h-z} \cdot \prod_{i \in \mathsf{tbl}(\widehat{\Psi})} \#R_i \right),$$

where $R_i = \{\langle \mathbf{r}_1, \ldots, \mathbf{r}_{\#\mathsf{tbl}(\psi_i)} \rangle, \psi_i(\langle \mathbf{r}_1, \ldots, \mathbf{r}_{\#\mathsf{tbl}(\psi_i)} \rangle) = 1\}$ and $z = \#(\mathsf{att}(\Psi) \setminus \mathsf{att}(\pi))$, with $\mathsf{att}(\pi)$ and $\mathsf{att}(\Psi)$ denoting the set of attributes in the project operator π and the predicate Ψ, respectively. In other words, z is the number of attributes in the project operator $\pi_{\mathsf{att}_1,\ldots,\mathsf{att}_h}$ that are included in $\Psi = (\mathsf{att}_1 = X_1 \wedge \cdots \wedge \mathsf{att}_\ell = X_\ell)$. Note that in the expression above, we assume the *worst-case* where the projection attributes not in Ψ (i.e., $\mathsf{att}(\pi) \setminus \mathsf{att}(\Psi)$) all belong to distinct tables. The term $\Pi_{i \in \mathsf{tbl}(\widehat{\Psi})} \#R_i$ is the number of rows in tables $\mathsf{tbl}(\Psi)$ that satisfy Ψ. The term m^{h-z} is the number of rows that include all the attributes in $\mathsf{att}(\pi) \setminus \mathsf{att}(\Psi)$.

Note that $\#\mathsf{R}$ is a worst-case lower bound on both the query complexity [8], communication complexity and the space complexity of a *plaintext* SPC query. In practice the number $h - z$ of attributes in $\mathsf{att}(\pi) \setminus \mathsf{att}(\Psi)$ is typically very small. The best and most common case is when $h = z$ which leads to

$$\#\mathsf{R} = h \cdot \prod_{i \in \mathsf{tbl}(\widehat{\Psi})} \#R_i.$$

B A Concrete Example of Indexed HNF

Our examples will rely on a small database DB composed of two tables T_1 and T_2 that have two and three rows, respectively. The schema of T_1 is $\mathbb{S}(\mathsf{T}_1) = (\mathsf{ID}, \mathsf{Name}, \mathsf{Course})$ and that of T_2 is $\mathbb{S}(\mathsf{T}_2) = (\mathsf{Name}, \mathsf{Grade})$. These tables are described in Fig. 3.

Fig. 4 shows the result of applying our method to index the database $\mathsf{DB} = (\mathsf{T}_1, \mathsf{T}_2)$, as detailed in Section 5. There are four multi-maps MM_R, MM_C, MM_V and $\mathsf{MM}_{\mathrm{Name}}$. Here, as we have only one pair of columns with the same domain, there is no need to store $\mathsf{MM}_{\mathrm{Name}}$ in a dictionary DX to hide its size since it is going to be disclosed as setup leakage during the EDB setup phase even if its stored in in a dictionary.

The first multi-map, MM_R, maps every row in each table to its encrypted content. As an instance, the first row of T_1 is composed of three values $(A05, \mathrm{Alice}, 16)$ that will get encrypted and stored in MM_R. Since DB has five rows, MM_R has five pairs. The second multi-map, MM_C, maps each column of every table to its encrypted content. Similarly, as DB is composed of five columns

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

in total, MM_C has five pairs. The third multi-map, MM_V, maps every unique value in every table to its coordinates in the plaintext table. For example, the value A in T_2 exists in two positions, in particular, in the first and third row. Finally, the last multi-map, $\mathsf{MM}_{\text{Name}}$, maps the columns' coordinates to the pair of rows that have the same value. In our example, as the first and second rows of the two tables contain Alice and Bob, respectively, the label/tuple pair

$$\Big(\mathsf{T}_1\|\mathbf{c}_1\|\mathsf{T}_2\|\mathbf{c}_2, ((\mathsf{T}_1\|r_1, \mathsf{T}_2\|r_1), (\mathsf{T}_1\|r_2, \mathsf{T}_2\|r_2))\Big)$$

is added to $\mathsf{MM}_{\text{Name}}$.

Concrete storage overhead. The plaintext database DB is composed of twelve cells excluding the tables attributes. The indexed structure consists of thirty eight pairs. Assuming that a pair and a cell have the same bit length, our indexed HNF representation of the database has a multiplicative storage overhead of 3.16. In particular, each of the multi-maps MM_R, MM_C and MM_V have the same size as the plaintext database (i.e., 12 pairs). This explains the $3\times$ factor. It is worth emphasizing that even if one considers a larger database, the $3\times$ factor remains unchanged. The additive component of the multiplicative factor, i.e., the 0.16, will vary, however, from one database to another depending on the number of columns that with the same domain and the number of equal rows in these columns.

A concrete query. Let us consider the following simple SQL query

$$\mathsf{Select}\ \mathit{ID}\ \mathsf{From}\ \mathsf{T}_1, \mathsf{T}_2\ \mathsf{Where}\ \mathsf{T}_1.\text{Name} = \mathsf{T}_2.\text{Name}.$$

This can be re-written in the normal form as

$$\pi_{\text{ID}}\Big(\sigma_{\mathsf{T}_1.\text{Name}=\mathsf{T}_1.\text{Name}}(\mathsf{T}_1 \times \mathsf{T}_2)\Big).$$

The token[9] is

$$\mathsf{tk} = \Big((1, \mathtt{in}), \perp, (\text{Name}, \mathsf{T}_1\|\text{Name}\|\mathsf{T}_2\|\text{Name})\Big).$$

The server starts by fetching from $\mathsf{MM}_{\text{Name}}$ the tuple corresponding to $\mathsf{T}_1\|\text{Name}\|\mathsf{T}_2\|\text{Name}$, which is equal to $((\mathsf{T}_1\|r_1, \mathsf{T}_2\|r_1), (\mathsf{T}_1\|r_2, \mathsf{T}_2\|r_2))$. For each element in this tuple, the server then fetches the corresponding pair of rows from MM_R. That is for $(\mathsf{T}_1\|r_1, \mathsf{T}_2\|r_1)$, the server will fetch the row r_1 from T_1 and r_2 from T_2 and add their concatenation to a temporary set R_1. The server does the same operations with the second pair and obtains

$$\mathsf{R}_1 = \Big(\Big\langle \mathsf{Enc}_K(A05), \mathsf{Enc}_K(\text{Alice}), \mathsf{Enc}_K(16), \mathsf{Enc}_K(\text{Alice}), \mathsf{Enc}_K(A05), \mathsf{Enc}_K(A)\Big\rangle$$
$$\Big\langle \mathsf{Enc}_K(A12), \mathsf{Enc}_K(\text{Bob}), \mathsf{Enc}_K(18), \mathsf{Enc}_K(\text{Bob}), \mathsf{Enc}_K(A05), \mathsf{Enc}_K(B)\Big\rangle\Big).$$

Finally, the server uses the project token $(1, \mathtt{in})$ to select the first value of each concatenated row in R_1. That is the final result is equal to

$$\mathsf{R} = \Big(\mathsf{Enc}_K(A05), \mathsf{Enc}_K(A12)\Big).$$

[9]For sake of clarity, this example of token generation does not accurately reflect the token algorithm of SPX, but only gives a high level idea of its algorithmic generation.

MM_R	
$\mathsf{T}_1 \| r_1$	$\mathsf{Enc}_K(A05), \mathsf{Enc}_K(\text{Alice}), \mathsf{Enc}_K(16)$
$\mathsf{T}_1 \| r_2$	$\mathsf{Enc}_K(A12), \mathsf{Enc}_K(\text{Bob}), \mathsf{Enc}_K(18)$
$\mathsf{T}_2 \| r_1$	$\mathsf{Enc}_K(\text{Alice}), \mathsf{Enc}_K(A)$
$\mathsf{T}_2 \| r_2$	$\mathsf{Enc}_K(\text{Bob}), \mathsf{Enc}_K(B)$
$\mathsf{T}_2 \| r_3$	$\mathsf{Enc}_K(\text{Eve}), \mathsf{Enc}_K(A)$

MM_C	
$\mathsf{T}_1 \| c_1$	$\mathsf{Enc}_K(1), \mathsf{Enc}_K(2),$
$\mathsf{T}_1 \| c_2$	$\mathsf{Enc}_K(\text{Alice}), \mathsf{Enc}_K(\text{Bob})$
$\mathsf{T}_1 \| c_3$	$\mathsf{Enc}_K(16), \mathsf{Enc}_K(18)$
$\mathsf{T}_2 \| c_1$	$\mathsf{Enc}_K(\text{Alice}), \mathsf{Enc}_K(\text{Bob}), \mathsf{Enc}_K(\text{Eve})$
$\mathsf{T}_2 \| c_2$	$\mathsf{Enc}_K(A), \mathsf{Enc}_K(B), \mathsf{Enc}_K(A)$

MM_V	
$\mathsf{T}_1 \| c_1 \| A05$	T_1, r_1
$\mathsf{T}_1 \| c_1 \| A12$	T_1, r_2
$\mathsf{T}_1 \| c_2 \| \text{Alice}$	T_1, r_1
$\mathsf{T}_1 \| c_2 \| \text{Bob}$	T_1, r_2
$\mathsf{T}_1 \| c_3 \| 16$	T_1, r_1
$\mathsf{T}_1 \| c_3 \| 18$	T_1, r_2
$\mathsf{T}_2 \| c_1 \| \text{Alice}$	T_2, r_1
$\mathsf{T}_2 \| c_1 \| \text{Bob}$	T_2, r_2
$\mathsf{T}_2 \| c_1 \| \text{Eve}$	T_2, r_3
$\mathsf{T}_2 \| c_2 \| A$	$(\mathsf{T}_1, r_1), (\mathsf{T}_2, r_3)$
$\mathsf{T}_2 \| c_2 \| B$	T_2, r_2

$\mathsf{MM}_{\text{Name}}$	
$\mathsf{T}_1 \| c_2 \| \mathsf{T}_2 \| c_1$	$(\mathsf{T}_1 \| r_1, \mathsf{T}_2 \| r_1), (\mathsf{T}_1 \| r_2, \mathsf{T}_2 \| r_2)$

Figure 4: Indexed database.

C General Heuristic Normal Form

We showed in Section 5 how to transform uncorrelated SPC queries into a more efficient form we called the HNF. Here, first we describe how to transform correlated SPC queries into an optimized form we call the *general heuristic normal form* (GHNF). We then show how to extend SPX to handle GHNF queries.

Recall that an SPC query is correlated if its predicate Ψ satisfies either of the following properties: (1) two or more type-2 terms share a common attribute; (2) a type-1 and type-2 term share a common attribute; (3) the attributes of two or more type-2 terms are from the same table; and (4) the attributes from a type-1 and type-2 term are from the same table. To rewrite a correlated SPC query in GHNF we first transform the predicate Ψ into a new predicate $\widehat{\Psi}$ and then show how to rewrite the query into GHNF based on $\widehat{\Psi}$.

General heuristic normal form. Given Ψ let $\widehat{\Psi}$ have form $\widehat{\psi}_1 \wedge \cdots \wedge \widehat{\psi}_d$, where $\widehat{\psi}_i$ includes all terms of Ψ with attributes that either belong to the same table or that share an attribute in common. Furthermore, for all $i \in [d]$, rewrite $\widehat{\psi}_i$ such that all the type-2 terms come before the type-1 terms. More specifically, if there are u type-2 terms and $p_i - u$ type-1 terms then write it in

the following form:

$$\widehat{\psi_i} = \Big(\mathsf{att}_1 = \mathsf{att}'_1 \bigwedge \cdots \bigwedge \mathsf{att}_u = \mathsf{att}'_u \bigwedge \mathsf{att}_{u+1} = a_{u+1} \bigwedge \cdots \bigwedge \mathsf{att}_{p_i} = a_{p_i}\Big).$$

Now, for all $i \in [d]$, process ψ_i and

$$\varphi_i := \sigma_{\mathsf{att}_1=\mathsf{att}'_1}\Big(\mathsf{T}'_1 \times \sigma_{\mathsf{att}_2=\mathsf{att}'_2}\Big(\mathsf{T}'_2 \times \sigma_{\mathsf{att}_3=\mathsf{att}'_3}\Big(\cdots \times \sigma_{\mathsf{att}_u=\mathsf{att}'_u}\Big(\mathsf{T}'_{u-1} \times \mathsf{T}'_u\Big)\Big)\cdots\Big),$$

where

$$\mathsf{T}'_i = \begin{cases} \sigma_{\mathsf{att}_{i_1}=a_{i_1}}(\mathsf{T}_i) \cap \cdots \cap \sigma_{\mathsf{att}_{i_v}=a_{i_v}}(\mathsf{T}_i) & \forall j \text{ s.t. } \mathsf{att}_{i_j} \in \mathsf{T}_i \\ \mathsf{T}_i & \text{otherwise.} \end{cases}$$

Note that both **(Case 1)** and **(Case 2)** in Section 5 can be captured by the GHNF presented above. We say that the query

$$\pi_{\mathsf{att}_1,\cdots,\mathsf{att}_h}\Big([a_1] \times \cdots [a_f] \times \varphi_1 \times \cdots \times \varphi_d\Big)$$

is in the general heuristic normal form.

C.1 Indexed Execution of GHNF Queries

Indexing for GHNF queries is the same as for HNF queries so we refer the reader to Section 5 and focus on how to perform an indexed execution on these queries. For this, we perform the following steps:

- for all $j \in [u]$, compute

$$X_j = \begin{cases} \bigcap_{m\in[v]} \Big(\Big(\chi(\mathbf{r})\Big)_{\mathbf{r}\in\mathsf{DB}_{\mathsf{att}_{i_m}=a_{i_m}}}\Big) & \forall m \text{ s.t. } \mathsf{att}_{i_m} \in \mathsf{T}_j \\ \bot & \text{otherwise.} \end{cases}$$

  where

$$\Big(\chi(\mathbf{r})\Big)_{\mathbf{r}\in\mathsf{DB}_{\mathsf{att}_m=a_m}} := \mathsf{MM}_V\Big[\Big\langle a_{i_m}, \chi(\mathsf{att}_{i_m})\Big\rangle\Big]$$

- then for all $j \in [u]$, rewrite

$$\mathsf{T}'_j = \begin{cases} \Big(\mathsf{MM}_R\Big[\chi(\mathbf{r})\Big]\Big)_{\chi(\mathbf{r})\in X_j} & \text{if } X_j \neq \bot \\ \mathsf{T}_j & \text{otherwise.} \end{cases}$$

- for $j = u-1$ to 1, do the following
  - compute $\mathsf{MM}_{\mathsf{att}_j} := \mathsf{DX}[\chi(\mathsf{att}_j)]$ and

$$\Big(\chi(\mathbf{r}_1), \chi(\mathbf{r}_2)\Big)_{(\mathbf{r}_1,\mathbf{r}_2)\in\mathsf{DB}_{\mathsf{att}_j=\mathsf{att}'_j}} := \mathsf{MM}_{\mathsf{att}_j}\Big[\Big\langle \chi(\mathsf{att}_j), \chi(\mathsf{att}'_j)\Big\rangle\Big].$$

– then compute the valid pairs of rows such that

$$\mathsf{T}'_j := \left(\mathsf{MM}_R\Big[\chi(\mathbf{r}_1)\Big] \times \mathsf{MM}_R\Big[\chi(\mathbf{r}_2)\Big]\right)_{\substack{(\mathbf{r}_1,\mathbf{r}_2)\in \mathsf{DB}_{\mathsf{att}_j=\mathsf{att}'_j} \\ \wedge(\mathbf{r}_1,\mathbf{r}_2)\in(\mathsf{T}'_j,\mathsf{T}'_{j+1})}}.$$

- set the result $R := \mathsf{T}'_1$.

C.2 SPX with GHNF

The Setup algorithm is the same. We only describe the differences in the Token and Query algorithms when compared to the HNF in Section 5.

Token. For the select token generation, it sets

$$\mathsf{stk} := \Big(\mathsf{stk}^1, \mathsf{ltk}^1, \cdots, \mathsf{stk}^{u-1}, \mathsf{ltk}^{u-1}, \mathsf{ltk}^u\Big),$$

where for all $j \in [u-1]$, $\mathsf{stk}^j := (\mathsf{dtk}^j, \mathsf{jtk}^j)$, with

$$\mathsf{dtk}^j := \Sigma_{\mathsf{DX}}.\mathsf{Token}\big(K_D, \chi(\mathsf{att}_j)\big)$$

and,

$$\mathsf{jtk}^j := \Sigma_{\mathsf{MM}}.\mathsf{Token}\Big(K_{\mathbf{c}}, \Big\langle \chi(\mathsf{att}_j), \chi(\mathsf{att}'_j)\Big\rangle\Big)$$

and for all $j \in [u]$,

$$\mathsf{ltk}^j = \begin{cases} (\mathsf{itk}_1, \cdots, \mathsf{itk}_v) & \forall m \text{ s.t. } \mathsf{att}_{i_m} \in \mathsf{T}_i \\ \perp & \text{otherwise.} \end{cases}$$

where,

$$\mathsf{itk}_j := \Sigma_{\mathsf{MM}}.\mathsf{Token}\big(K_V, \langle a_{i_m}, \chi(\mathsf{att}_{i_m})\rangle\big).$$

Query. For the query algorithm, to evaluate the select token, it performs the following operations, for all $j = u-1$ to 1:

- parse $\mathsf{ltk}^j = (\mathsf{ltk}^j_1, \cdots, \mathsf{ltk}^j_v)$ and $\mathsf{stk}^j = (\mathsf{dtk}^j, \mathsf{jtk}^j)$,
- recover, for all $w \in [v]$, a set R'_w by first computing

  $$(\mathsf{rtk}_1, \ldots, \mathsf{rtk}_s) := \Sigma_{\mathsf{MM}}.\mathsf{Query}(\mathsf{EMM}_V, \mathsf{itk}^j_w).$$

  It then computes

  $$\mathbf{ct}_1 := \Sigma_{\mathsf{MM}}.\mathsf{Query}(\mathsf{EMM}_R, \mathsf{rtk}_1), \ldots, \mathbf{ct}_s := \Sigma_{\mathsf{MM}}.\mathsf{Query}(\mathsf{EMM}_R, \mathsf{rtk}_s),$$

  and sets $R'_w := \{\mathbf{ct}_1, \ldots, \mathbf{ct}_s\}$. Finally, it sets $\mathsf{TR}_i = R'_1 \cap \cdots \cap R'_v$.

- first computes $\mathsf{EMM} := \Sigma_{\mathsf{MM}}.\mathsf{Query}(\mathsf{EDX}, \mathsf{dtk}^j)$ and

$$\Big((\mathsf{rtk}_1, \mathsf{rtk}'_1), \ldots, (\mathsf{rtk}_s, \mathsf{rtk}'_s)\Big) := \Sigma_{\mathsf{MM}}.\mathsf{Query}(\mathsf{EMM}, \mathsf{jtk}^j).$$

  It then computes

$$\Big(\mathbf{ct}_1 := \Sigma_{\mathsf{MM}}.\mathsf{Query}(\mathsf{EMM}_R, \mathsf{rtk}_1), \ldots, \mathbf{ct}_s := \Sigma_{\mathsf{MM}}.\mathsf{Query}(\mathsf{EMM}_R, \mathsf{rtk}_s)\Big) \bigcap \mathsf{TR}_j,$$

  and

$$\Big(\mathbf{ct}'_1 := \Sigma_{\mathsf{MM}}.\mathsf{Query}(\mathsf{EMM}_R, \mathsf{rtk}'_1), \ldots, \mathbf{ct}'_s := \Sigma_{\mathsf{MM}}.\mathsf{Query}(\mathsf{EMM}_R, \mathsf{rtk}'_s)\Big) \bigcap \mathsf{TR}_{j+1}.$$

  Finally, it sets $R = \Big(\mathbf{ct}_j \times \mathbf{ct}'_j\Big)_{j \in [s]}$.

D 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:

$$\mathsf{Select}\ \mathsf{Sum}(\textit{attribute})\ \mathsf{From}\ \textit{tables}\ \mathsf{Where}\ \textit{conditions}.$$

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 response table R. To handle Sum on a given attribute att, 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.

E SPX$^+$: SPX dynamic extension

Here, we describe a dynamic version of SPX. We first recall the syntax and security definitions for dynamic STE and detail our construction in Section E.2.

E.1 Dynamic STE

In this Section, we formalize dynamic STE schemes. While our SPX$^+$ construction from Section E.2 handles updates interactively, it makes use of several non-interactive dynamic STE schemes as building blocks. For this reason, we present two definitions of dynamic STE: one for schemes that have interactive updates and one for schemes that have non-interactive updates.

Definition E.1 (Non-interactive dynamic STE)**.** *A non-interactive (or single-round) response-hiding dynamic structured encryption scheme* $\Sigma_{\mathscr{T}} = (\mathsf{Setup}, \mathsf{Token}^{\mathsf{sr}}, \mathsf{Query}, \mathsf{Token}^{\mathsf{up}}, \mathsf{Update}, \mathsf{Dec})$ *for data type* $\mathscr{T}$ *consists of six polynomial-time algorithms that work as follows:*

- $(K, st, \mathsf{EDS}) \leftarrow \mathsf{Setup}(1^k, \mathsf{DS})$*: is a probabilistic algorithm that takes as input a security parameter* 1^k *and a structure* DS *of type* $\mathscr{T}$ *and outputs a secret key* K*, a state* st *and an encrypted structure* EDS*.*
- $\mathsf{tk} \leftarrow \mathsf{Token}^{\mathsf{sr}}(K, st, q)$*: is a (possibly) probabilistic algorithm that takes as input a secret key* K*, a query* q *and a state* st *and returns a token* tk*.*
- $\mathrm{ct} \leftarrow \mathsf{Query}(\mathsf{EDS}, \mathsf{tk})$*: is a (possibly) probabilistic algorithm that takes as input an encrypted structure* EDS *and a token* tk *and outputs either a message* ct*.*
- $(st', \mathsf{utk}) \leftarrow \mathsf{Token}^{\mathsf{up}}(k, st, u)$*: is a (possibly) probabilistic algorithm that takes as input a secret key* K*, a document's identifier* id*, the state* st*, an update* u *and returns an updated state* st' *and an update token* utk*.*
- $\mathsf{EDS}' \leftarrow \mathsf{Update}(\mathsf{EDS}, \mathsf{utk})$*: is a (possibly) probabilistic algorithm that takes as input an encrypted structure* EDS *and an update token* utk *and outputs an updated encrypted structure* EDS$'$*.*
- $r \leftarrow \mathsf{Dec}(K, \mathrm{ct})$*: is a deterministic algorithm that takes as input a secret key* K *and a message* ct *and outputs a response* r*.*

We say that a dynamic 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, st, 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 $\mathsf{Token}(K, q_i)$, and all sequences of $m = \mathsf{poly}(k)$ updates $u_1, \ldots, u_m$, for all update tokens utk_i output by $\mathsf{Token}^{\mathsf{up}}(K, st, u_i)$, and for all structures $\mathsf{EDS} \leftarrow \mathsf{Update}(\mathsf{EDS}, \mathsf{utk}_i)$, for all messages ct output by $\mathsf{Query}(\mathsf{EDS}, \mathsf{tk}_i)$, $\mathsf{Dec}(K, \mathrm{ct})$ returns the correct response with all but negligible probability.

The syntax of a response-revealing dynamic STE scheme can be recovered by omitting the Dec algorithm and having Query output the response r directly.

Definition E.2. *An dynamic structured encryption scheme $\Sigma_{\mathscr{T}} = (\mathsf{Setup}, \mathsf{Token}^{\mathsf{sr}}, \mathsf{Query}, \Pi^+, \mathsf{Dec})$ for data type $\mathscr{T}$ with interactive updates consists of four polynomial-time algorithms and one two-party protocol between the client and server. The* Setup, $\mathsf{Token}^{\mathsf{sr}}$, Query *and* Dec *algorithms are as in Definition E.1 while Π^+ works as as follows:*

- $(st', \mathsf{EDS}') \leftarrow \Pi^+\Big((K, st, u), \mathsf{EDS}\Big)$*: is a two-party protocol between the client and server. It takes as input from the client a key K, state st and an update u and as input from the server an encrypted structure* EDS*. It outputs to the client an updated state st' and to the server an updated encrypted structure* EDS'*.*

Security. The security of dynamic STE schemes is similar to the security of static schemes except that we also consider the update leakage $\mathcal{L}_{\mathsf{U}}$. Intuitively, we require that the dynamic encrypted structure reveals no information about its underlying structure beyond the setup leakage $\mathcal{L}_{\mathsf{S}}$, and that the query and update algorithms reveal no information about the structure and the queries and updates beyond the query leakage $\mathcal{L}_{\mathsf{Q}}$ and update leakage $\mathcal{L}_{\mathsf{U}}$, respectively.

Definition E.3 (Adaptive security for dynamic STE with non-interactive updates [18, 15])**.** *Let $\Sigma_{\mathscr{T}}$ = $(\mathsf{Setup}, \mathsf{Token}^{\mathsf{sr}}, \mathsf{Query}, \mathsf{Token}^{\mathsf{up}}, \mathsf{Update})$ be a non-interactive dynamic STE scheme and consider the following probabilistic experiments where $\mathcal{A}$ is a stateful adversary, $\mathcal{S}$ is a stateful simulator, $\mathcal{L}_{\mathsf{S}}$, $\mathcal{L}_{\mathsf{Q}}$ and $\mathcal{L}_{\mathsf{U}}$ are leakage profiles and $z \in \{0,1\}^*$:*

- $\mathbf{Real}_{\Sigma,\mathcal{A}}(k)$*: given z the adversary $\mathcal{A}$ outputs a structure $\mathsf{DS} : \mathbf{Q} \to \mathbf{R}$ and receives* EDS *from the challenger, where $(K, st, \mathsf{EDS}) \leftarrow \mathsf{Setup}(1^k, \mathsf{DS})$. The adversary* adaptively *chooses a polynomial number of operations $o_1, \ldots, o_m$ where, for all $i \in [m]$, o_i is either a query q_i or an update u_i. If the operation is a query, the adversary receives $\mathsf{tk}_i \leftarrow \mathsf{Token}^{\mathsf{sr}}(K, st, q_i)$. If it is an update, it receives receives $\mathsf{utk}_i \leftarrow \mathsf{Token}^{\mathsf{up}}(K, st, u_i)$. 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 $\mathsf{DS} : \mathbf{Q} \to \mathbf{R}$ 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* adaptively *chooses a polynomial number of operations $o_1, \ldots, o_m$ where, for all $i \in [m]$, o_i is either a query q_i or an udpate u_i. If the operation is a query, the simulator receives query leakage $\mathcal{L}_{\mathsf{Q}}(\mathsf{DS}, q_i)$ and returns a token tk_i to $\mathcal{A}$. If the operation is an update, the simulator receives update leakage $\mathcal{L}_{\mathsf{U}}(\mathsf{DS}, u_i)$ and returns a token utk_i to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs a bit b that is output by the experiment.*

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

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

Definition E.3 can be modified for the case of dynamic STE schemes with interactive updates as follows. In the $\text{REAL}_{\Sigma,\mathcal{A}}$ experiment, if the operation is an update, the adversary executes $\Pi^+((K, st, u_i), \mathsf{EDS})$ with an honest challenger playing the role of the client. In the $\mathbf{Ideal}_{\Sigma,\mathcal{A}}$ experiment, the adversary executes the server-side of Π^+ against the simulator who, given the update leakage $\mathcal{L}_{\mathsf{U}}(\mathsf{DS}, u_i)$, plays the role of the client.

Forward security. An important property of dynamic STE schemes is *forward security* which, informally, guarantees that updates to the encrypted structure cannot be correlated to previous searches. Forward security, along with a construction that achieves it, was introduced in [37] but not formally defined. Recently, a definition was proposed in [8] in the context of SSE. The definition, however, does not seem to be strong enough to capture the intuition described above. The approach of [8] is to define a forward-secure SSE scheme as one whose update leakage $\mathcal{L}_{\mathsf{U}}(\mathsf{DB}, u)$, where $u = \{(\mathsf{id}_i, \mathsf{W}_i)\}_i$, is such that

$$\mathcal{L}_{\mathsf{U}}(\mathsf{DB}, u) = f\left(\left\{(\mathsf{id}_i, \#W_i)\right\}_i\right)$$

for some function f. Essentially, what is guaranteed is that the update leakage only includes information about the file ids that are updated and the number of keywords added to each file. In particular, the intuition is that such a guarantee is strong enough for forward security since no information about previous queries appears as input to f. The difficulty is that the inputs to f could still be correlated with previous queries. As a concrete example, consider a client that makes $t \geq 1$ keyword queries w_1 through w_t. At a later point in time, it adds a file whose first keyword is w_λ, for $\lambda \in [t]$, and with a number of unique keywords equal to λ. Such a scheme would satisfy the above definition (where f is the identity function) but the leakage would reveal to the server that the first keyword of the new file was queried at time $t = \#\mathsf{W}_i$, clearly breaking forward security.

To capture the intuition of forward security, we require that the update leakage be leakage-free in the sense that it reveals nothing about the database or the update beyond what can be derived from the security parameter or a public parameter λ. This is formalized simply by requiring that the scheme be $(\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{Q}}, \mathcal{L}_{\mathsf{U}})$-secure where $\mathcal{L}_{\mathsf{U}}(\mathsf{DB}, u) = (1^k, \lambda)$ which we sometimes write this as $(\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{Q}}, \perp)$-secure. Whether leakage-free updates are necessary to achieve forward-security is not clear. What is clear, however, is that it is sufficient. We also note that the Sophos construction of Bost from [8] seems to satisfy this stronger property.

E.2 Making SPX Dynamic

Due to its modularity, SPX can be extended to be dynamic without re-designing it entirely. In the following, we refer to the dynamic version of SPX as SPX^+.

In addition to the Setup, $\mathsf{Token}^{\mathsf{sr}}$ and Query operations, SPX^+ handles two update operations: (1) row insertion; and (2) targeted row deletion, i.e, deletion of rows that contain a particular value. To handle these updates, SPX^+ has an additional two-party protocol Π^+ between the client and server.

Overview. To handle dynamic databases, the only change we need to make to the Setup, Query and Token algorithms of SPX is to replace the underlying static multi-map and dictionary encryption schemes with dynamic ones. The main challenge is in how to handle updates. To add a row $\mathbf{r}$ to a table T in the database, we need to do four things: (1) add it to EMM_R; (2) add each of its cells to the appropriate columns in EMM_C; (3) add its search token $\mathsf{tk}_{\mathbf{r}}$ to EMM_V; and (4) update the multi-maps stored in EDX. The first step is straightforward and is done by encrypting $\mathbf{r}$ (using the underlying symmetric encryption scheme) and sending an EMM_r update token for the pair that consists of $(\chi(\mathbf{r}), \mathbf{ct})$, where $\mathbf{ct}$ is the encryption of $\mathbf{r}$. The second step can also be handled using the dynamic properties of the underlying dynamic EMM_C structure. The third step can also be done using the dynamic operations of EMM_V.

The main difficulty is in handling the fourth step. The reason steps (1) through (3) are straightforward is because the structures we need to update hold items that are independent of each other. For example, the addition of an encrypted row to EMM_R does not affect the rows that are already stored in EMM_R. The challenge with updating EDX is that the items it holds are highly correlated with each other and an update to EDX affects many of the items it already contains. Consider the following example. Suppose we have two attributes att_1 and att_2 in the database with the same domain. This means that in EDX, we already store an encrypted multi-map that contains all the pairs of rows (in DB) with att_1 equal to att_2. More concretely, each pair consists of two row tokens that are later used to fetch the corresponding encrypted rows from EMM_R. Now, if we want to add a new row $\mathbf{r}$, we have to first determine whether a specific cell in $\mathbf{r}$ is equal to any other cell in any other row in any column with the same domain. Once this information is known, we can appropriately update EDX, but learning this is non-trivial. Our solution is based on a two-round interactive protocol where the client first queries EMM_V to retrieve the tokens of the rows that have the same value as $\mathbf{r}$ in some column (if any). We then combine these row tokens with the token of $\mathbf{r}$ (considering every possible combination) and update EDX by creating a new encrypted multi-map that holds all these pairs.

From a security perspective, the leakage disclosed from accessing EMM_V is non-trivial. The adversary can infer—in the worst case where all columns have the same domain—all the rows that have the same values [10]. To handle this problem, we manage the multi-map MM_V using ORAM so that we do not disclose the access pattern. With this approach, the only information disclosed to the server during an update is the *number* of rows with the same value. We can also use naive padding to leak only an upper bound, but this would add a lot of overhead for large databases.

Details. SPX^+ is composed of three algorithms and a two-party protocol $\mathsf{SPX}^+ = (\mathsf{Setup}, \mathsf{Token}^{\mathsf{sr}}, \mathsf{Query}, \Pi^+)$. As mentioned above, the Setup algorithm is the same $\mathsf{SPX.Setup}$ but using dynamic encrypted structures instead of static ones. Similarly, the $\mathsf{Token}^{\mathsf{sr}}$ and Query algorithms are the same as $\mathsf{SPX.Token}$ and $\mathsf{SPX.Query}$, respectively, except the underlying tokens are for the dynamic structures created during setup. We now provide details about the Π^+ protocol.

The Π^+ protocol takes as input from the client a key K, a state st, a table identifier T, a row $\mathbf{r}$ and the type of operation $\mathsf{op} \in \{\mathsf{edit}^+, \mathsf{edit}^-\}$. It takes as input from the server an encrypted database EDB. Let $\mathbb{S}(\mathsf{T}) = (\mathsf{att}_1, \ldots, \mathsf{att}_p)$, where $p = \|\mathsf{T}\|_c$. It proceeds as follows:

- If $\mathsf{op} = \mathsf{edit}^+$ (i.e., the client wants to add the row $\mathbf{r}$ to table T) the client generates an EMM_R

[10] This is still not as much leakage as that of a PPE-based EDB since the frequency of cells within a column remains hidden, and only cross-column frequency is disclosed for columns with the same domain.

update token utk to add an encryption of $\mathbf{r}$ to EMM_R where, as in SPX, every element of $\mathbf{r}$ is encrypted with a symmetric encryption scheme.

Then, for each value $\mathbf{r}[i]$ (where $i \in [p]$) of $\mathbf{r}$, it retrieves from EMM_V the row tokens of all rows $\mathbf{r}'$ for which there exists some j such that $\mathbf{r}'[j] = \mathbf{r}[i]$. Intuitively, it retrieves the row tokens of all the rows in the database that have some cell value equal to $\mathbf{r}[i]$ (on columns with the same domain). This is done by sending EMM_V search tokens for pairs of the form $\langle \mathbf{r}[i], \chi(\mathsf{att}_j) \rangle$ for all $\mathsf{att}_j \in \mathsf{DB}$ such that $\mathsf{dom}(\mathsf{att}_i) = \mathsf{dom}(\mathsf{att}_j)$. Upon receiving these tokens, the server uses them to query EMM_V and sends back the returned row tokens to the client.

The client then computes the cross product between the set $\{\mathsf{rtk}_{\mathbf{r}}\}$ and the set of returned tokens, where $\mathsf{rtk}_{\mathbf{r}}$ is the EMM_R token for $\mathbf{r}$. This results in a set of token pairs for which the first element is always $\mathsf{rtk}_{\mathbf{r}}$.

These pairs are then be inserted in a set of new multi-maps that will be added to EDX. More precisely, the client creates an encrypted multi-map for every attribute in the update table, i.e., in $\mathbb{S}(\mathsf{T})$, where $\mathbb{S}(\mathsf{T}) = (\mathsf{att}_1, \cdots, \mathsf{att}_p)$. For every attribute att in $\mathbb{S}(\mathsf{T})$, the client instantiates an empty multi-map $\mathsf{MM}_{\mathsf{att}}$. The client then, for all attributes $\mathsf{att}' \in \mathbb{S}(\mathsf{DB} \setminus \mathsf{T})$ that verify $\mathsf{dom}(\mathsf{att}) = \mathsf{dom}(\mathsf{att}')$, sets

$$\mathsf{MM}_{\mathsf{att}_1}[\langle \chi(\mathsf{att}), \chi(\mathsf{att}') \rangle] := \mathbf{t},$$

where $\mathbf{t}$ is a tuple that contains the pairs of row tokens generated in the previous step.

The client then updates EMM_C by generating tokens to add the cells of $\mathbf{r}$ to the appropriate columns. Similarly the client updates EMM_V by generating the update token that will add the new row token to EMM_V.

- if $\mathsf{op} = \mathsf{edit}^-$ (i.e., the client wants to delete all rows equal to $\mathbf{r}$) the client first needs to find the rows equal to $\mathbf{r}$. In order to do this, it generates, for all $i \in [p]$, EMM_V search tokens for $\langle \mathbf{r}[i], \chi(i) \rangle$. The server queries EMM_V on all these tokens and keeps the row tokens that are in the intersection of these query results. This intersection will correspond to the rows whose cells are *all* equal to the cells of $\mathbf{r}$. It then uses these tokens to query EMM_R and returns the coordinates of the encrypted rows to the client. The client then uses these coordinates to create EMM_R deletion tokens.

We give more details of the SPX^+ algorithm in Figs. 5 and 6.

Security and efficiency improvements. Note that even if we store and manage MM_V in an ORAM, and instantiate EMM_R, and EDX with forward-secure encrypted structures, SPX^+ is still not be forward-secure in the strong sense discussed in Appendix E.1. There are two main reasons for this. First, when $\mathsf{op} = \mathsf{edit}^+$, the server will learn the number of rows retrieved from EMM_V. Second, when $\mathsf{op} = \mathsf{edit}^-$, the server learns the number of rows that are equal to $\mathbf{r}$. As discussed in Appendix E.1, these numbers could potentially be correlated with previous searches so to achieve forward-security the scheme should hide even these cardinalities. To handle this, we propose the following modifications:

- in the Setup algorithm, set the multi-map MM_V to

$$\mathsf{MM}_V\left[\left\langle v, \chi(\mathbf{c}) \right\rangle\right] := \left(\left\langle \mathsf{rtk}_{\mathbf{r}}, \chi(\mathbf{r}) \right\rangle\right)_{\mathbf{r} \in \mathsf{DB}_{\mathbf{c}=v}};$$

where $\mathsf{rtk}_{\mathbf{r}} \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}_{K_R}\Big(\chi(\mathbf{r})\Big)$, for all $\mathbf{c} \in \mathsf{DB}^{\mathsf{T}}$, all $v \in \mathbf{c}$ and $\mathbf{r} \in \mathsf{DB}_{\mathbf{c}=v}$. Then, store and manage MM_V using an ORAM. Notice that, here, we added the coordinates of the rows in MM_V. The reason is so that when we delete a row, the server no longer has retrieve tokens from EMM_V, compute intersections and query EMM_R. Instead, the client will query MM_V through the ORAM and recover the coordinates of the rows, take the intersection and generate the delete tokens for EMM_R.

- fix an upper bound that determines the number of queries that are made to the ORAM-managed MM_V for every update. This can be implemented in a more efficient manner by pipelining queries in such a way that even if a query matches more than the fixed threshold, it will be performed during the next query under the assumption that on average the matching tuples are below the fixed threshold. Setting the threshold correctly will not induce a big loss of efficiency.

- for efficiency, the client can create an encrypted multi-map EMM_V in addition to the ORAM-managed MM_V. The ORAM-managed copy is queried during updates while the encrypted one is queried for searches. This preserve the same search efficiency as SPX.

Efficiency. SPX^+ has the same search complexity as SPX (see Section 5.3). For updates, managing MM_V in an ORAM can lead to a logarithmic blow up. Contrary to standard optimal-time dynamic multi-map encryption schemes, ORAM induces at least a logarithmic multiplicative blow up [38]. Let T denote the table to be updated and $\mathcal{H}_i = \{\mathbf{r}' \in \mathsf{DB} : \mathbf{r}'[i] = \mathbf{r}[i]\}$ the set of rows matching $\mathbf{r}$, for all $i \in [\|\mathsf{T}\|_c]$. We write $\mathsf{access}(\mathsf{MM}_V)$ to denote an ORAM access cost to MM_V. The update complexity is equal to:

$$O\Big(\Big(\sum_{i=1}^{[\|\mathsf{T}\|_c} \#\mathcal{H}_i\Big) \cdot \mathsf{access}\left(\mathsf{MM}_V\right)\Big).$$

If we fix a threshold th, both the computation and communication complexity of updates are

$$O\Big(\|\mathsf{T}\|_c \cdot \mathsf{th} \cdot \mathsf{access}(\mathsf{MM}_V)\Big).$$

The storage of SPX^+ is the same as SPX. To see why, the only major modification in SPX^+ consists of initiating EMM_V using an ORAM. Recent constructions of ORAM do not blow up storage.

E.2.1 Security and Leakage

The setup and query leakage of SPX^+ is similar to the SPX. In particular, we have

$$\mathcal{L}_{\mathsf{S}}^{\mathsf{spx}^+}(\mathsf{DB}) = \Big(\mathcal{L}_{\mathsf{S}}^{\mathsf{spx}}(\mathsf{DB})\Big) \qquad \text{and} \qquad \mathcal{L}_{\mathsf{Q}}^{\mathsf{spx}^+}(\mathsf{DB}) = \Big(\mathcal{L}_{\mathsf{Q}}^{\mathsf{spx}}(\mathsf{DB})\Big).$$

Let $\Sigma^+_{\mathsf{DX}} = (\mathsf{Setup}, \mathsf{Token}^{\mathsf{sr}}, \mathsf{Query}, \mathsf{Token}^{\mathsf{up}}, \mathsf{Update})$ be a response-revealing dictionary encryption scheme, $\Sigma^+_{\mathsf{MM}} = (\mathsf{Setup}, \mathsf{Token}^{\mathsf{sr}}, \mathsf{Query}, \mathsf{Token}^{\mathsf{up}}, \mathsf{Update})$ 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 $\mathsf{SPX}^+ = (\mathsf{Setup}, \mathsf{Token}^{\mathsf{sr}}, \mathsf{Query}, \Pi^+)$ defined as follows :

- $\mathsf{Setup}(1^k, \mathsf{DB})$: output $(K, st, \mathsf{EDB}) \leftarrow \mathsf{SPX.Setup}(1^k, \mathsf{DB})$.
- $\mathsf{Token}^{\mathsf{sr}}(K, st, q)$: output $(\mathsf{tk}, st) \leftarrow \mathsf{SPX.Token}(K, st, q)$.
- $\mathsf{Query}(\mathsf{tk}, \mathsf{EDB})$: compute $\mathsf{R} \leftarrow \mathsf{SPX.Query}(\mathsf{EDB}, \mathsf{tk})$.
- $\Pi^+\big((K, st, u), \mathsf{EDS}\big)$:
  1. parse $u = (\mathsf{T}, \mathbf{r}, \mathsf{op})$, $\mathbb{S}(\mathsf{T}) = (\mathsf{att}_1, \cdots, \mathsf{att}_{\|\mathsf{T}\|_c})$, and $st = \big(st_R, st_C, st_V, st_D\big)$;
  2. if $\mathsf{op} = \mathsf{edit}^-$,
     (a) for all $i \in [\|\mathsf{T}\|_c]$, compute $\mathsf{tk}_r \leftarrow \Sigma^+_{\mathsf{MM}}.\mathsf{Token}^{\mathsf{sr}}_{K_V}\Big(st_V, \Big\langle \mathbf{r}[i], \chi(\mathsf{att}_i) \Big\rangle\Big)$;
     (b) server computes $\mathbf{t} \leftarrow \bigcap_{i=1}^{\|\mathsf{T}\|_c} \Big(\Sigma^+_{\mathsf{MM}}.\mathsf{Query}(\mathsf{tk}_r, \mathsf{EMM}_V)\Big)$;
     (c) server sends $\mathbf{t}' := \bigcup_{t \in \mathbf{t}} \Big(\Sigma^+_{\mathsf{MM}}.\mathsf{Query}(t, \mathsf{EMM}_R)\Big)$;
     (d) for all $t \in \mathbf{t}'$, client and server respectively compute

$$(st'_R, \mathsf{utk}_R) \leftarrow \Sigma^+_{\mathsf{MM}}.\mathsf{Token}^{\mathsf{up}}\Big(K_R, st_R, \big(\perp, \chi(t), \mathsf{op}\big)\Big)$$

and,

$$\mathsf{EMM}'_R \leftarrow \Sigma^+_{\mathsf{MM}}.\mathsf{Update}\Big(\mathsf{utk}_R, \mathsf{EMM}_R\Big);$$

Figure 5: SPX^+: a dynamic relational DB encryption scheme (Part 1).

- Continuation of $\Pi^{+}\big((K, st, u), \mathsf{EDS}\big)$:

  3. if $\mathsf{op} = \mathsf{edit}^{+}$,
     (a) compute $\mathrm{ct}_{\mathbf{r}} = \big(\mathsf{Enc}_{K_1}(r_1), \cdots, \mathsf{Enc}_{K_1}(r_{\|\mathsf{T}\|_c})\big)$;
     (b) client and server respectively compute

$$(st'_R, \mathsf{utk}_R) \leftarrow \Sigma^{+}_{\mathsf{MM}}.\mathsf{Token}^{\mathsf{up}}\Big(K_R, st_R, \big(\mathrm{ct}_{\mathbf{r}}, \chi(\mathbf{r}), \mathsf{op}\big)\Big),$$

     and,

$$\mathsf{EMM}'_R \leftarrow \Sigma^{+}_{\mathsf{MM}}.\mathsf{Update}\Big(\mathsf{utk}_R, \mathsf{EMM}_R\Big);$$

     (c) compute $\mathsf{rtk}_{\mathbf{r}} \leftarrow \Sigma^{sf+}_{\mathsf{MM}}.\mathsf{Token}^{\mathsf{sr}}\Big(K_R, st_R, \chi(\mathbf{r})\Big)$;
     (d) for all $i \in [\|\mathsf{T}\|_c]$,
        i. initialize an empty multi-map $\mathsf{MM}_{\mathsf{att}_i}$;
        ii. client and server respectively compute

$$(st'_C, \mathsf{utk}_C) \leftarrow \Sigma^{+}_{\mathsf{MM}}.\mathsf{Token}^{\mathsf{up}}\Big(K_C, st_C, \big(\mathbf{r}[i], \chi(\mathsf{att}_i), \mathsf{op}\big)\Big)$$

        and,

$$\mathsf{EMM}'_C \leftarrow \Sigma^{+}_{\mathsf{MM}}.\mathsf{Update}\Big(\mathsf{utk}_C, \mathsf{EMM}_C\Big);$$

        iii. client and server respectively compute

$$(st'_V, \mathsf{utk}_V) \leftarrow \Sigma^{+}_{\mathsf{MM}}.\mathsf{Token}^{\mathsf{up}}\Big(K_V, st_V, \big(\mathsf{rtk}_{\mathbf{r}}, \langle \mathbf{r}[i], \chi(\mathsf{att}_i)\rangle, \mathsf{op}\big)\Big)$$

        and,

$$\mathsf{EMM}'_V \leftarrow \Sigma^{+}_{\mathsf{MM}}.\mathsf{Update}\Big(\mathsf{utk}_V, \mathsf{EMM}_V\Big);$$

        iv. for all $\mathsf{att} \in \mathbb{S}(\mathsf{DB} \setminus \{\mathsf{T}\})$ where $\mathsf{dom}(\mathsf{att}) = \mathsf{dom}(\mathsf{att}_i)$,
           A. computes $\mathsf{tk}_r \leftarrow \Sigma^{+}_{\mathsf{MM}}.\mathsf{Token}^{\mathsf{sr}}\Big(K_V, st_V, \big\langle \mathbf{r}[i], \chi(\mathsf{att})\big\rangle\Big)$;
           B. server computes $\mathbf{t} \leftarrow \Sigma^{+}_{\mathsf{MM}}.\mathsf{Query}(\mathsf{tk}_r, \mathsf{EMM}_V)$;
           C. set $\mathsf{MM}_{\mathbf{c}}\Big[\Big\langle \chi(\mathsf{att}_i), \chi(\mathsf{att})\Big\rangle\Big] := \{\mathsf{rtk}_r, t\}_{t \in \mathbf{t}}$;
           D. compute $(K_{\mathbf{c}}, \mathsf{EMM}_{\mathbf{c}}) \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Setup}\big(1^k, \mathsf{MM}_{\mathbf{c}}\big)$;
        v. client and server respectively compute

$$(st'_D, \mathsf{utk}_D) \leftarrow \Sigma^{+}_{\mathsf{DX}}.\mathsf{Token}^{\mathsf{up}}\Big(K_D, st_D, \big(\chi(\mathsf{att}_i), \mathsf{EMM}_{\mathbf{c}}, \mathsf{op}\big)\Big)$$

$$\mathsf{EDX}' \leftarrow \Sigma^{+}_{\mathsf{DX}}.\mathsf{Update}\Big(\mathsf{utk}_D, \mathsf{EDX}\Big);$$

Figure 6: SPX^{+}: a dynamic relational DB encryption scheme (Part 2).

Update leakage. We now describe the update leakage of SPX^+. Like the description of the SPX leakage, we proceed in a black box fashion assuming the underlying structures are all forward-secure. If $\mathsf{op} = \mathsf{edit}^+$ we have:

$$\begin{aligned}\mathcal{L}_{\mathsf{U}}^{\mathsf{spx}^+}(\mathsf{DB}, (\mathsf{T}, \mathbf{r}, \mathsf{op})) = &\Big(\|\mathsf{T}\|_c, \mathcal{L}_{\mathsf{U}}\Big(\mathsf{MM}_R, \mathbf{r}, \chi(\mathbf{r}), \mathsf{op}\Big), \\ &\Big(\mathcal{L}_{\mathsf{Q}}\Big(\mathsf{MM}_V, \Big\langle \mathbf{r}[i], \chi(\mathsf{att})\Big\rangle\Big)\Big)_{\substack{i\in[\|\mathsf{T}\|_c], \mathsf{att}\in\mathbb{S}(\mathsf{DB}\setminus\mathsf{T}) \\ \mathsf{dom}(\mathsf{att}_i)=\mathsf{dom}(\mathsf{att})}}, \\ &\Big(\mathcal{L}_{\mathsf{S}}\Big(\mathsf{MM}_{\mathsf{att}_i}\Big), \\ &\mathcal{L}_{\mathsf{U}}\Big(\mathsf{MM}_V, \mathbf{r}[i], \Big\langle \mathbf{r}[i], \chi(\mathsf{att}_i)\Big\rangle, \mathsf{op}\Big), \\ &\mathcal{L}_{\mathsf{U}}\Big(\mathsf{MM}_C, \mathbf{r}[i], \chi(\mathsf{att}_i), \mathsf{op}\Big)\Big)_{i\in[\|\mathsf{T}\|_c]}, \\ &\Big(\mathcal{L}_{\mathsf{U}}\Big(\mathsf{DX}, \perp, \Big\langle \chi(\mathsf{att}_1), \chi(\mathsf{att}_2)\Big\rangle, \mathsf{op}\Big)\Big)_{\substack{\mathsf{att}_1\in\mathbb{S}(\mathsf{T}), \mathsf{att}_2\in\mathbb{S}(\mathsf{DB})\setminus\{\mathsf{T}\}, \\ \mathsf{dom}(\mathsf{att}_1)=\mathsf{dom}(\mathsf{att}_2)}}\Big).\end{aligned}$$

If $\mathsf{op} = \mathsf{edit}^-$:

$$\begin{aligned}\mathcal{L}_{\mathsf{U}}^{\mathsf{spx}^+}(\mathsf{DB}, (\mathsf{T}, \mathbf{r}, \mathsf{op})) = &\Big(\Big(\mathcal{L}_{\mathsf{Q}}\Big(\mathsf{MM}_V, \Big\langle \mathbf{r}[i], \chi(\mathsf{att}_i)\Big\rangle\Big)\Big)_{i\in[\|\mathsf{T}\|_c]}, \\ &\Big(\mathcal{L}_{\mathsf{U}}\Big(\mathsf{MM}_R, \perp, \chi(\mathbf{r}'), \mathsf{op}\Big)\Big)_{\mathbf{r}'\in\mathbf{R}_{\mathbf{r}}}\Big),\end{aligned}$$

where $\mathbf{R}_{\mathbf{r}} = \{\mathbf{r}' \in \mathsf{T} : \forall i \in \|\mathsf{T}\|_c, \mathbf{r}'[i] = \mathbf{r}[i]\}$.

If we consider the case where MM_V is stored and managed with an ORAM with a public parameter that upper bound the ORAM requests to th, then

$$\Big(\mathcal{L}_{\mathsf{Q}}\Big(\mathsf{MM}_V, \Big\langle \mathbf{r}[i], \chi(\mathsf{att}_i)\Big\rangle\Big)\Big)_{i\in[\|\mathsf{T}\|_c]} = \Big(\mathsf{th}, \#\mathbb{S}(\mathsf{T})\Big),$$

which means

$$\mathcal{L}_{\mathsf{U}}^{\mathsf{spx}^+}(\mathsf{DB}, (\mathsf{T}, \mathbf{r}, \mathsf{op})) = \Big(\mathsf{th}, \#\mathbb{S}(\mathsf{T}), \Big(\mathcal{L}_{\mathsf{U}}\Big(\mathsf{MM}_R, \perp, \chi(\mathbf{r}'), \mathsf{op}\Big)\Big)_{\mathbf{r}'\in\mathbf{R}_{\mathbf{r}}}\Big),$$

We can now state the security of SPX^+ with respect to the leakage profile just described.

Theorem E.4. *If* SKE *is RCPA secure,* Σ_{DX}^D *is adaptively* $(\mathcal{L}_{\mathsf{S}}^{\mathsf{dx}}, \mathcal{L}_{\mathsf{Q}}^{\mathsf{dx}}, \mathcal{L}_{\mathsf{U}}^{\mathsf{dx}})$*-semantically secure and* Σ_{MM}^D *is adaptively* $(\mathcal{L}_{\mathsf{S}}^{\mathsf{mm}}, \mathcal{L}_{\mathsf{Q}}^{\mathsf{mm}}, \mathcal{L}_{\mathsf{U}}^{\mathsf{dx}})$*-secure, then* SPX^+ *is* $(\mathcal{L}_{\mathsf{S}}^{\mathsf{spx}^+}, \mathcal{L}_{\mathsf{Q}}^{\mathsf{spx}^+}, \mathcal{L}_{\mathsf{U}}^{\mathsf{spx}^+})$*-semantically secure.*

Since the proof of Theorem E.4 is similar to the proof of Theorem E.4, we defer it to the full version of this work.

F Proof of Theorem 6.1

Theorem 6.1. *If* SKE *is RCPA secure,* Σ_{DX} *is adaptively* $(\mathcal{L}^{\mathsf{dx}}_{\mathsf{S}}, \mathcal{L}^{\mathsf{dx}}_{\mathsf{Q}})$*-semantically secure and* Σ_{MM} *is adaptively* $(\mathcal{L}^{\mathsf{mm}}_{\mathsf{S}}, \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}})$*-secure, then* SPX *is* $(\mathcal{L}^{\mathsf{spx}}_{\mathsf{S}}, \mathcal{L}^{\mathsf{spx}}_{\mathsf{Q}})$*-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 $\mathcal{L}^{\mathsf{spx}}_{\mathsf{S}}(\mathsf{DB})$, $\mathcal{S}$ simulates EDB by computing $\mathsf{EDX} \leftarrow \mathcal{S}_{\mathsf{DX}}(\mathcal{L}^{\mathsf{dx}}_{\mathsf{S}}(\mathsf{DX}))$, $\mathsf{EMM}_R \leftarrow \mathcal{S}_{\mathsf{MM}}(\mathcal{L}^{\mathsf{mm}}_{\mathsf{S}}(\mathsf{MM}_R))$, $\mathsf{EMM}_C \leftarrow \mathcal{S}_{\mathsf{MM}}(\mathcal{L}^{\mathsf{mm}}_{\mathsf{S}}(\mathsf{MM}_C))$ and $\mathsf{EMM}_V \leftarrow \mathcal{S}_{\mathsf{MM}}(\mathcal{L}^{\mathsf{mm}}_{\mathsf{S}}(\mathsf{MM}_V))$ 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 $\mathbf{Ideal}_{\mathsf{SPX},\mathcal{A},\mathcal{S}}(k)$ experiment. Given this input, $\mathcal{S}$ simulates a token

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

as follows. 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 from $\mathrm{XPP}(\mathsf{DB}, q)$.

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

$$\mathcal{P}(\mathsf{att}_i) = \Big(\mathtt{out}, \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\Big(\mathsf{MM}_C, \chi(\mathsf{att}_i)\Big), (|c_j|)_{j\in[\#\mathbf{c}]}, \mathrm{AccP}(\mathbf{c})\Big)$$

it sets

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

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, $\mathcal{P}(\mathsf{att}_i) = (\mathtt{in}, \mathsf{att}_i)$ it sets $\mathsf{ytk}_i := \mathsf{att}_i$.

Then, for all $i \in [d]$, it simulates stk_i as follows. If $\mathcal{Z}(\varphi_i)$ is of case-1, it first computes for all $\mathbf{r} \in \mathsf{DB}_{\mathsf{att}_{i,j}=X_{i,j}}$, for all $j \in [p_i]$,

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

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

$$\mathsf{itk}_{i,j} := \mathbf{tk}_{i,j} \leftarrow \mathcal{S}_{\mathsf{MM}}\Big((\mathsf{rtk}_{\mathbf{r}})_{\mathbf{r}\in\mathsf{DB}_{\mathsf{att}_{i,j}=X_{i,j}}}, \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\Big(\mathsf{MM}_V, \chi(\mathsf{att}_{i,j})\Big)\Big).$$

It then sets

$$\mathsf{stk}_i = (\mathsf{itk}_{i,1}, \cdots, \mathsf{itk}_{i,p_i}).$$

If, on the other hand, $\mathcal{Z}(\varphi_i)$ is of case-2, $\mathcal{S}$ starts by computing

$$\mathsf{dtk}_i \leftarrow \mathcal{S}_{\mathsf{DX}}\Big(\mathsf{EMM}_{\mathsf{att}_i}, \mathcal{L}^{\mathsf{dx}}_{\mathsf{Q}}\Big(\chi(\mathsf{att}_{i,1})\Big)\Big)$$

where $\mathsf{EMM}_{\mathsf{att}_{i,1}} \leftarrow \mathcal{S}_{\mathsf{MM}}(\mathcal{L}^{\mathsf{mm}}_{\mathsf{S}}(\mathsf{MM}_{\mathsf{att}_{i,1}}))$ if it has never been accessed before and where $\mathsf{EMM}_{\mathsf{att}_{i,1}}$ is the previously-used structure otherwise. It then computes, for all $(\mathbf{r}_1, \mathbf{r}_2) \in \mathsf{DB}_{\mathsf{att}_{i,1}=\mathsf{att}_{i,2}}$,

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

and

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

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

$$\mathsf{jtk}_i \leftarrow \mathcal{S}_{\mathsf{MM}}\Big(\Big\{\mathsf{rtk}_{\mathbf{r}_1}, \mathsf{rtk}_{\mathbf{r}_2}\Big\}_{(\mathbf{r}_1,\mathbf{r}_2)\in\mathsf{DB}_{\mathsf{att}_{i,1}=\mathsf{att}_{i,2}}}, \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\Big(\mathsf{MM}_{\mathsf{att}_{i,1}}, \Big\langle \chi(\mathsf{att}_{i,1}), \chi(\mathsf{att}_{i,2})\Big\rangle\Big)\Big),$$

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 attributes $\mathsf{att}_{i,1}$ in the terms of form $\mathsf{att}_{i,1} = \mathsf{att}_{i,2}$.

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 written in its heuristic normal form, each `out` y-token ytk_i is replaced with the output of

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

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 heuristic normal form query q, every case-1 select token stk_i is replaced with the output of

$$\begin{aligned}
&\Big(\mathcal{S}_{\mathsf{MM}}\Big(\big(\mathsf{rtk}_{\mathbf{r}}\big)_{\mathbf{r}\in\mathsf{DB}_{\mathsf{att}_{i,1}=X_{i,1}}}, \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\Big(\mathsf{MM}_V, \chi(\mathsf{att}_{i,1})\Big)\Big),\\
&\cdots,\\
&\mathcal{S}_{\mathsf{MM}}\Big(\big(\mathsf{rtk}_{\mathbf{r}}\big)_{\mathbf{r}\in\mathsf{DB}_{\mathsf{att}_{i,p_i}=X_{i,p_i}}}, \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\Big(\mathsf{MM}_V, \chi(\mathsf{att}_{i,p_i})\Big)\Big)\Big),
\end{aligned}$$

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 heuristic normal form query q, every dictionary token dtk_i in case-2 select tokens $\mathsf{stk}_i = (\mathsf{dtk}_i, \mathsf{jtk}_i)$ are replaced with the output of

$$\mathcal{S}_{\mathsf{DX}}\Big(\mathsf{EMM}_{\mathsf{att}_{i,1}}, \mathrm{AccP}(\mathsf{EMM}_{\mathsf{att}_{i,1}}), \mathcal{L}^{\mathsf{dx}}_{\mathsf{Q}}\Big(\chi(\mathsf{att}_{i,1})\Big)\Big).$$

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 $\mathsf{att}_{i,1} = \mathsf{att}_{i,2}$ in Ψ such that $\mathsf{att}_{i,1} = \mathrm{Loc}_\Psi[l]$, the join tokens jtk_i in the case-2 select token $\mathsf{stk}_i = (\mathsf{dtk}_i, \mathsf{jtk}_i)$ is replaced with the output of

$$\mathcal{S}_{\mathsf{MM}}\Bigg(\Big\{\mathsf{rtk}_{\mathbf{r}_1}, \mathsf{rtk}_{\mathbf{r}_2}\Big\}_{(\mathbf{r}_1,\mathbf{r}_2)\in \mathsf{DB}_{\mathrm{Loc}_\Psi[l]=\mathsf{att}_{i,2}}}, \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\Big(\mathsf{MM}_{\mathrm{Loc}_\Psi[l]}, \Big\langle \chi(\mathrm{Loc}_\Psi[l]), \chi(\mathsf{att}_{i,2})\Big\rangle\Big)\Bigg).$$

$\mathsf{Game}_{4+\#\mathrm{Loc}_\Psi}$: 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.

Note that $\mathsf{Game}_{5+\#\mathrm{Loc}_\Psi}$ is identical to $\mathbf{Ideal}_{\mathsf{SPX},\mathcal{A},\mathcal{S}}(k)$.

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

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

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 $\{\mathsf{MM}_{\mathbf{c}}\}_{\mathbf{c}\in\mathsf{DB}^{\mathsf{T}}}$, 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 $\mathsf{EMM}^\star_C$—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

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

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

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

$$\pi_{\mathsf{att}_1,\cdots,\mathsf{att}_h}\Big([a_1] \times \cdots [a_f] \times \varphi_1 \times \cdots \times \varphi_d\Big).$$

where for all $i \in [d]$, φ_i, **(case 1)**, has form

$$\sigma_{\mathsf{att}_{i,1}=a_{i,1}}(\mathsf{T}_i) \times \cdots \times \sigma_{\mathsf{att}_{i,p_i}=a_{i,p_i}}(\mathsf{T}_i),$$

or, **(case 2)**, has form

$$\sigma_{\mathsf{att}_{i,1}=\mathsf{att}_{i,2}}(\mathsf{T}_{i,1} \times \mathsf{T}_{i,2}).$$

For all $i \in [h]$, $\mathcal{B}$ outputs $\chi(\mathsf{att}_i)$ as its own query to $\mathsf{EMM}^{\star}_C$ and receives $\mathsf{ytk}^{\star}_i$. It then sends to $\mathcal{A}$ the token

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

where for all $i \in [f]$, $\mathrm{e}_i \leftarrow \mathsf{Enc}_{K_1}(a_i)$. For all $i \in [\ell]$, if φ_i is of case-1, then $\mathsf{stk}_i = (\mathsf{itk}_{i,1}, \cdots, \mathsf{itk}_{i,p_i})$ such that for all $j \in [p_i]$, $\mathsf{itk}_{i,j} \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}_{K_V}\left(\left\langle X_{i,j}, \chi(\mathsf{att}_{i,j})\right\rangle\right)$. Otherwise if φ_i is of case-2, then $\mathsf{stk}_i = (\mathsf{dtk}_i, \mathsf{jtk}_i)$ such that $\mathsf{dtk}_i \leftarrow \Sigma_{\mathsf{DX}}.\mathsf{Token}_{K_D}\left(\chi(\mathsf{att}_{i,1})\right)$ and $\mathsf{jtk}_i \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}_{K_{\mathsf{att}_{i,1}}}\left(\left\langle \chi(\mathsf{att}_{i,1}), \chi(\mathsf{att}_{i,2})\right\rangle\right)$.

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_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. 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 $\mathsf{EMM}^{\star}_V$—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

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

with $\mathsf{EMM}_C \leftarrow \mathcal{S}_{\mathsf{MM}}(\mathcal{L}^{\mathsf{mm}}_{\mathsf{S}}(\mathsf{MM}_C))$, $(K_D, \mathsf{EDX}) \leftarrow \Sigma_{\mathsf{DX}}.\mathsf{Setup}(1^k, \mathsf{DX})$ 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 heuristic normal form query q. $\mathcal{B}$ does the following. For all $i \in [d]$ and $j \in [p_i]$ such that φ_i is of case-1, $\mathcal{B}$ outputs

$$\left\langle X_{i,j}, \chi(\mathsf{att}_{i,j})\right\rangle,$$

as its own queries to $\mathsf{EMM}_V^\star$ and receives $\mathsf{itk}_{i,j}^\star$. It then sends to $\mathcal{A}$ the token

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

where $\mathsf{stk}_i^\star = (\mathsf{itk}_{i,1}^\star, \cdots, \mathsf{itk}_{i,p_i}^\star)$ for all $i \in [f]$, $\mathrm{e}_i \leftarrow \mathsf{Enc}_{K_1}(a_i)$. If $\mathsf{att}_i \in S \setminus I$, where $I = \left\{\mathsf{att} \in S : \mathsf{att} \in \bigcup_{j=1}^t \mathbb{S}(\mathsf{T}_{i_j})\right\}$ and $S \stackrel{def}{=} \{\mathsf{att}_1, \ldots, \mathsf{att}_h\}$,

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

Given $\mathrm{AccP}(\mathsf{att}_i)$, 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, $\mathsf{att}_i \in \mathbb{S}(\mathsf{VDB}) \bigcup I$, $\mathsf{ytk}_i := \mathsf{att}_i$. For all $i \in [d]$, if φ_i is of case-2, then $\mathsf{stk}_i = (\mathsf{dtk}_i, \mathsf{jtk}_i)$ such that $\mathsf{dtk}_i \leftarrow \Sigma_{\mathsf{DX}}.\mathsf{Token}_{K_D}\left(\chi(\mathsf{att}_{i,1})\right)$ and $\mathsf{jtk}_i \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}_{K_{\mathsf{att}_{i,1}}}\left(\left\langle \chi(\mathsf{att}_{i,1}), \chi(\mathsf{att}_{i,2})\right\rangle\right)$.

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

Whenever $\mathcal{A}$ outputs a heuristic normal form query q, $\mathcal{B}$ does the following. For all $i \in [d]$ such that φ_i is of case-2, $\mathcal{B}$ outputs $\chi(\mathsf{att}_{i,1})$ as its own query to $\mathsf{EDX}^\star$ and receives $\mathsf{dtk}_i^\star$. It then sends to $\mathcal{A}$ the token

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

where for all $i \in [f]$, $\mathrm{e}_i \leftarrow \mathsf{Enc}_{K_1}(a_i)$. If $\mathsf{att}_i \in S \setminus I$,

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

Given $\mathrm{AccP}(\mathsf{att}_i)$, 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, $\mathsf{att}_i \in \mathbb{S}(\mathsf{VDB}) \bigcup I$, $\mathsf{ytk}_i := \mathsf{att}_i$. For all $i \in [d]$, if φ_i is of case-1, $\mathsf{stk}_i = (\mathsf{itk}_{i,1}, \cdots, \mathsf{itk}_{i,p_i})$ such that for all $j \in [p_i]$

$$\mathsf{itk}_{i,j} := \mathbf{tk}_i \leftarrow \mathcal{S}_{\mathsf{MM}}\left(\left(\mathsf{rtk}_{\mathbf{r}}\right)_{\mathbf{r}\in\mathsf{DB}_{\mathsf{att}_{i,j}=X_{i,j}}}, \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\left(\mathsf{MM}_V, \chi(\mathsf{att}_{i,j})\right)\right), \text{ with } \mathsf{rtk}_{\mathbf{r}} \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}_{K_R}\left(\chi(\mathbf{r})\right)$$

for $\mathbf{r} \in \mathsf{DB}_{\mathsf{att}_{i,j}=X_{i,j}}$. If φ_i is of case-2, then $\mathsf{stk}_i = (\mathsf{dtk}^\star_i, \mathsf{jtk}_i)$ such that $\mathsf{jtk}_i \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}_{K_{\mathsf{att}_i}}\left(\left\langle \chi(\mathsf{att}_{i,1}), \chi(\mathsf{att}_{i,2})\right\rangle\right)$

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. 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 $\{\mathsf{MM}_{\mathbf{c}}\}_{\mathbf{c}\in\mathsf{DB}^{\mathsf{T}}}$, 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 heuristic normal form query q, $\mathcal{B}$ does the following. For all $i \in [d]$ such that φ_i is of case-2 with $\mathsf{att}_{i,1} = \mathrm{Loc}_\Psi[l]$, $\mathcal{B}$ outputs

$$\left\langle \chi(\mathrm{Loc}_\Psi[l]), \chi(\mathsf{att}_{i,2})\right\rangle$$

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

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

where for all $i \in [f]$, $\mathrm{e}_i \leftarrow \mathsf{Enc}_{K_1}(a_i)$. If $\mathsf{att}_i \in S \setminus I$,

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

Given $\mathrm{AccP}(\mathsf{att}_i)$, 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, $\mathsf{att}_i \in \mathbb{S}(\mathsf{VDB}) \bigcup I$, $\mathsf{ytk}_i := \mathsf{att}_i$. For all $i \in [d]$, if φ_i is of case-1, $\mathsf{stk}_i = (\mathsf{itk}_{i,1}, \cdots, \mathsf{itk}_{i,p_i})$ such that for all $j \in [p_i]$

$\mathsf{itk}_{i,j} := \mathbf{tk}_i \leftarrow \mathcal{S}_{\mathsf{MM}}\left(\left(\mathsf{rtk}_{\mathbf{r}}\right)_{\mathbf{r}\in\mathsf{DB}_{\mathsf{att}_{i,j}=X_{i,j}}}, \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\left(\mathsf{MM}_V, \chi(\mathsf{att}_{i,j})\right)\right)$, with $\mathsf{rtk}_{\mathbf{r}} \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}_{K_R}\left(\chi(\mathbf{r})\right)$ for $\mathbf{r} \in \mathsf{DB}_{\mathsf{att}_{i,j}=X_{i,j}}$. If φ_i is of case-2, then $\mathsf{stk}_i = (\mathsf{dtk}_i, \mathsf{jtk}^\star_i)$ equals:

- if $z = l$, $\mathsf{dtk}_i \leftarrow \mathcal{S}_{\mathsf{DX}}\left(\mathsf{EMM}^\star_{\mathrm{Loc}_\Psi[l]}, \mathcal{L}^{\mathsf{dx}}_{\mathsf{Q}}\left(\chi(\mathrm{Loc}_\Psi[l])\right)\right)$ and $\mathsf{jtk}_i = \mathsf{jtk}^\star_i$
- for all $z < l$, $\mathsf{dtk}_i \leftarrow \mathcal{S}_{\mathsf{DX}}\left(\mathsf{EMM}_{\mathrm{Loc}_\Psi[z]}, \mathrm{AccP}(\mathsf{EMM}_{\mathrm{Loc}_\Psi[z]}), \mathcal{L}^{\mathsf{dx}}_{\mathsf{Q}}\left(\chi(\mathrm{Loc}_\Psi[z])\right)\right)$ 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

$$\mathsf{jtk}_i \leftarrow \mathcal{S}_{\mathsf{MM}}\left(\left\{\mathsf{rtk}_{\mathbf{r}_1}, \mathsf{rtk}_{\mathbf{r}_2}\right\}_{(\mathbf{r}_1,\mathbf{r}_2)\in\mathsf{DB}_{\mathrm{Loc}_\Psi[z]=\mathsf{att}_{i,2}}}, \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\left(\mathsf{MM}_{\mathrm{Loc}_\Psi[z]}, \left\langle\chi(\mathrm{Loc}_\Psi[z]), \chi(\mathsf{att}_{i,2})\right\rangle\right)\right),$$

with $\mathsf{rtk}_{\mathbf{r}_1} \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}_{K_R}\left(\chi(\mathbf{r}_1)\right)$ and $\mathsf{rtk}_{\mathbf{r}_2} \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}_{K_R}\left(\chi(\mathbf{r}_2)\right)$.

- for all $z > l$, $\mathsf{dtk}_i \leftarrow \mathcal{S}_{\mathsf{DX}}\left(\mathsf{EMM}_{\mathrm{Loc}_\Psi[z]}, \mathrm{AccP}(\mathsf{EMM}_{\mathrm{Loc}_\Psi[z]}), \mathcal{L}^{\mathsf{dx}}_{\mathsf{Q}}\left(\chi(\mathrm{Loc}_\Psi[z])\right)\right)$ 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

$$\mathsf{jtk}_i \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}_{K_{\mathsf{att}_i}}\left(\left\langle\chi(\mathrm{Loc}_\Psi[z]), \chi(\mathsf{att}_{i,2})\right\rangle\right).$$

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_{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

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

is non-negligible, which is a contradiction.

□

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

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

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 MM_R. Upon receiving $\mathsf{EMM}_R^{\star}$—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

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

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

Whenever $\mathcal{A}$ outputs a heuristic normal form query q, $\mathcal{B}$ does the following. $\mathcal{B}$ outputs $\chi(\mathbf{r})$ as its own query to $\mathsf{EMM}_R^{\star}$ and receives $\mathsf{rtk}_{\mathbf{r}}^{\star}$. It then sends to $\mathcal{A}$ the token

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

where for all $i \in [f]$, $\mathrm{e}_i \leftarrow \mathsf{Enc}_{K_1}(a_i)$. If $\mathsf{att}_i \in S \setminus I$,

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

Given $\mathrm{AccP}(\mathsf{att}_i)$, 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, $\mathsf{att}_i \in \mathbb{S}(\mathsf{VDB}) \bigcup I$, $\mathsf{ytk}_i := \mathsf{att}_i$. For all $i \in [d]$, if φ_i is of case-1, $\mathsf{stk}_i = (\mathsf{itk}_{i,1}, \cdots, \mathsf{itk}_{i,p_i})$ such that for all $j \in [p_i]$ $\mathsf{itk}_{i,j} := \mathbf{tk}_i \leftarrow \mathcal{S}_{\mathsf{MM}}\left((\mathsf{rtk}_{\mathbf{r}}^{\star})_{\mathbf{r}\in\mathsf{DB}_{\mathsf{att}_{i,j}=X_{i,j}}}, \mathcal{L}_{\mathsf{Q}}^{\mathsf{mm}}\left(\mathsf{MM}_V, \chi(\mathsf{att}_{i,j})\right)\right)$. If φ_i is of case-2, then $\mathsf{stk}_i = (\mathsf{dtk}_i, \mathsf{jtk}_i)$ such that $\mathsf{dtk}_i \leftarrow \mathcal{S}_{\mathsf{DX}}\left(\mathsf{EMM}_{\mathsf{att}_{i,1}}, \mathrm{AccP}(\mathsf{EMM}_{\mathsf{att}_{i,1}}), \mathcal{L}_{\mathsf{Q}}^{\mathsf{dx}}\left(\chi(\mathsf{att}_{i,1})\right)\right)$ with $\mathrm{AccP}(\mathsf{EMM}_{\mathsf{att}_{i,1}})$ is a leakage that captures when and where $\mathsf{EMM}_{\mathsf{att}_i}$ was generated. Thus, it can be either equal to $\mathsf{EMM}_{\mathsf{att}_{i,1}} \leftarrow \mathcal{S}_{\mathsf{MM}}(\mathcal{L}_{\mathsf{S}}^{\mathsf{mm}}(\mathsf{MM}_{\mathsf{att}_{i,1}}))$, or, a previously simulated one, and

$$\mathsf{jtk}_i \leftarrow \mathcal{S}_{\mathsf{MM}}\left(\left\{\mathsf{rtk}_{\mathbf{r}_1}^{\star}, \mathsf{rtk}_{\mathbf{r}_2}^{\star}\right\}_{(\mathbf{r}_1,\mathbf{r}_2)\in\mathsf{DB}_{\mathsf{att}_{i,1}=\mathsf{att}_{i,2}}}, \mathcal{L}_{\mathsf{Q}}^{\mathsf{mm}}\left(\mathsf{MM}_{\mathsf{att}_{i,1}}, \left\langle\chi(\mathsf{att}_{i,1}), \chi(\mathsf{att}_{i,2})\right\rangle\right)\right).$$

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 $\mathsf{Game}_{3+\#\mathrm{Loc}_{\Psi}}$. 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

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

is non-negligible, which is a contradiction.

□

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

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

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 RCPA 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—that is either equal to $\mathrm{ct}^\star\leftarrow\mathsf{Enc}_{K_1}(c)$ or to $\mathrm{ct}^\star\xleftarrow{\$}\{0,1\}^k$—. $\mathcal{B}$ 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 heuristic normal form query q, $\mathcal{B}$ does the following. For all $i\in[f]$, $\mathcal{B}$ outputs a_i and receives $\mathrm{e}_i^\star$. It then sends to $\mathcal{A}$ the token

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

If $\mathsf{att}_i\in S\setminus I$,

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

Given $\mathrm{AccP}(\mathsf{att}_i)$, $\mathrm{ct}^\star_j$ 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, $\mathsf{att}_i\in\mathbb{S}(\mathsf{VDB})\bigcup I$, $\mathsf{ytk}_i:=\mathsf{att}_i$. For all $i\in[d]$, if φ_i is of case-1, $\mathsf{stk}_i=(\mathsf{itk}_{i,1},\cdots,\mathsf{itk}_{i,p_i})$ such that for all $j\in[p_i]$

$$\mathsf{itk}_{i,j}:=\mathbf{tk}_i\leftarrow\mathcal{S}_{\mathsf{MM}}\left((\mathsf{rtk}_{\mathbf{r}})_{\mathbf{r}\in\mathsf{DB}_{\mathsf{att}_{i,j}=X_{i,j}}},\mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\left(\mathsf{MM}_V,\chi(\mathsf{att}_{i,j})\right)\right),\text{ with }\mathsf{rtk}_{\mathbf{r}}\leftarrow\Sigma_{\mathsf{MM}}.\mathsf{Token}_{K_R}\left(\chi(\mathbf{r})\right)$$

for $\mathbf{r}\in\mathsf{DB}_{\mathsf{att}_{i,j}=X_{i,j}}$. If φ_i is of case-2, then $\mathsf{stk}_i=(\mathsf{dtk}_i,\mathsf{jtk}_i)$ $\mathsf{dtk}_i\leftarrow\mathcal{S}_{\mathsf{DX}}\left(\mathsf{EMM}_{\mathsf{att}_{i,1}},\mathrm{AccP}(\mathsf{EMM}_{\mathsf{att}_{i,1}}),\mathcal{L}^{\mathsf{dx}}_{\mathsf{Q}}\left(\chi(\mathsf{att}_{i,1})\right)\right.$ with $\mathrm{AccP}(\mathsf{EMM}_{\mathsf{att}_{i,1}})$ is a leakage that captures when and where $\mathsf{EMM}_{\mathsf{att}_{i,1}}$ was generated. Thus, it can be either equal to $\mathsf{EMM}_{\mathsf{att}_{i,1}}\leftarrow\mathcal{S}_{\mathsf{MM}}(\mathcal{L}^{\mathsf{mm}}_{\mathsf{S}}(\mathsf{MM}_{\mathsf{att}_{i,1}}))$, or, a previously simulated one, and

$$\mathsf{jtk}_i\leftarrow\mathcal{S}_{\mathsf{MM}}\left(\left\{\mathsf{rtk}_{\mathbf{r}_1},\mathsf{rtk}_{\mathbf{r}_2}\right\}_{(\mathbf{r}_1,\mathbf{r}_2)\in\mathsf{DB}_{\mathsf{att}_{i,1}=\mathsf{att}_{i,2}}},\mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\left(\mathsf{MM}_{\mathsf{att}_{i,1}},\left\langle\chi(\mathsf{att}_{i,1}),\chi(\mathsf{att}_{i,2})\right\rangle\right)\right),$$

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

$$\mathsf{rtk}_{\mathbf{r}}\leftarrow\mathcal{S}_{\mathsf{MM}}\left(\left\{\mathrm{ct}^\star_j\right\}_{j\in[\#\mathbf{r}]},\mathrm{AccP}(\mathbf{r}),\mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}\left(\mathsf{MM}_R,\chi(\mathbf{r})\right)\right),$$

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 $\mathcal{B}$ receives the encryption of the cell then, by construction, $\mathcal{A}$'s view is exactly its view in $\mathsf{Game}_{4+\#\mathrm{Loc}_\Psi}$. On the other hand, if $\mathcal{B}$ receives a random value in $\{0,1\}^k$, 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 for all distinguishers $\mathcal{D}$

$$\left|\Pr\left[\mathcal{D}[\mathrm{ct} \leftarrow \mathsf{Enc}_K(c)] = 1 \ : K \leftarrow \{0,1\}^k\right] - \Pr\left[\mathcal{D}[\mathrm{ct} \xleftarrow{\$} \{0,1\}^k] = 1\right]\right|$$

is non-negligible, which is a contradiction.

□

■