Boolean Searchable Symmetric Encryption with Worst-Case Sub-Linear Complexity

Seny Kamara[*] Brown University

Tarik Moataz[†] Brown University

Abstract

Recent work on searchable symmetric encryption (SSE) has focused on increasing its expressiveness. A notable example is the OXT construction (Cash et al., *CRYPTO '13*) which is the first SSE scheme to support conjunctive keyword queries with sub-linear search complexity. While OXT efficiently supports disjunctive and boolean queries that can be expressed in searchable normal form, it can only handle *arbitrary* disjunctive and boolean queries in linear time. This motivates the problem of designing expressive SSE schemes with *worst-case* sub-linear search; that is, schemes that remain highly efficient for *any* keyword query.

In this work, we address this problem and propose non-interactive highly efficient SSE schemes that handle *arbitrary* disjunctive and boolean queries with worst-case sub-linear search and optimal communication complexity. Our main construction, called IEX, makes black-box use of an underlying single keyword SSE scheme which we can instantiate in various ways. Our first instantiation, IEX-2Lev, makes use of the recent 2Lev construction (Cash et al., *NDSS '14*) and is optimized for search at the expense of storage overhead. Our second instantiation, IEX-ZMF, relies on a new single keyword SSE scheme we introduce called ZMF and is optimized for storage overhead at the expense of efficiency (while still achieving asymptotically sub-linear search). Our ZMF construction is the first adaptively-secure highly compact SSE scheme and may be of independent interest. At a very high level, it can be viewed as an encrypted version of a new Bloom filter variant we refer to as a Matryoshka filter. In addition, we show how to extend IEX to be dynamic and forward-secure.

To evaluate the practicality of our schemes, we designed and implemented a new encrypted search framework called *Clusion*. Our experimental results demonstrate the practicality of IEX and of its instantiations with respect to either search (for IEX-2Lev) and storage overhead (for IEX-ZMF).

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

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

Contents

1 Introduction **3**
1.1 Our Contributions and Techniques . . . 3

2 Related Work **6**

3 Preliminaries **7**
3.1 Cryptographic Primitives . . . 8

4 Definitions **9**

5 IEX: A Worst-Case Sub-Linear Disjunctive SSE Scheme **10**
5.1 Correctness and Efficiency . . . 11
5.2 Security . . . 14

6 Boolean Queries with IEX **15**

7 ZMF: A Compact and Adaptively-Secure SSE Scheme **17**
7.1 An Adaptively-Secure and Multi-Structure Set Encryption Scheme . . . 20
7.2 The ZMF Multi-Map Encryption Scheme . . . 22

8 Dynamic SSE with IEX **24**
8.1 Security Definitions . . . 24
8.2 DIEX: A Dynamic SSE Scheme . . . 25

9 Empirical Evaluation **28**
9.1 IEX-2Lev Evaluation . . . 29
9.2 IEX-ZMF Evaluation . . . 32

A Multi-Structure Structured Encryption **38**

B Proof of Theorem 5.1 **39**

C Proof of Theorem 7.1 **40**

D Proof of Theorem 7.2 **40**

1 Introduction

A structured encryption (STE) scheme encrypts a data structure in such a way that it can be privately queried. An STE scheme is secure if it reveals nothing about the structure and query beyond a well-specified and "reasonable" leakage profile [21, 19]. STE schemes come in two forms: response-revealing and response-hiding. The former reveals the query response in plaintext whereas the latter does not. An important special case of STE is searchable symmetric encryption (SSE) which encrypts search structures such as inverted indexes [21, 19, 29, 28, 17, 16] or search trees [25, 28]. Another example is graph encryption which encrypts various kinds of graphs [19, 32]. STE has received a lot of attention from Academia and Industry due to: (1) its potential applications to cloud storage and database security; and (2) the fact that, among a host of different encrypted search solutions (e.g., property-preserving encryption, fully-homomorphic encryption, oblivious RAM, functional encryption) it seems to provide the best tradeoffs between security and efficiency.

In recent years, much of the work on STE has focused on supporting more complex structures and queries. A notable example in the setting of SSE is the work of Cash et al. which proposed the first SSE scheme to support conjunctive queries in sub-linear time [17]. Their scheme, OXT, is also shown to support disjunctive and even boolean queries. Faber et al. later showed how to extend OXT to achieve even more complex queries including range, substring, wildcard and phrase queries. Another example is the BlindSeer project from Pappas et al. [35] and Fisch et al. [23] which present a solution that supports boolean and range queries as well as stemming in sub-linear time.

Naive solutions. Any boolean query $\phi(w_1, \ldots, w_q)$, where $w_1, \ldots, w_q$ are keywords and ϕ is a boolean formula, can be handled using a single-keyword SSE scheme in a naive way. In the case of response-revealing schemes it suffices to search for each keyword and have the server take the intersection and unions of the result sets appropriately. The issue with this approach, of course, is that the server learns more information than necessary: namely, it learns the result sets $\mathsf{DB}(w_1), \ldots, \mathsf{DB}(w_q)$ whereas it should only learn the set $\mathsf{DB}(\phi(w_1, \ldots, w_q))$. For response-hiding schemes, one can search for each keyword and compute the intersections and unions at the client. The problem with this approach is that the parties communicate more information than necessary: namely, the server sends elements within the intersections of the result sets multiple times. With this in mind, any boolean SSE solution should improve on one of the naive approaches depending on whether it is response-hiding or response-revealing.

Worst-case sub-linear search complexity. While OXT achieves sub-linear search complexity for conjunctive queries, its extension to disjunctive and arbitrary boolean queries does not. More precisely, OXT remains sub-linear only for queries in *searchable normal form* (SNF) which have the form $w_1 \wedge \phi(w_2, \ldots, w_q)$, where w_1 through w_q are keywords and ϕ is an arbitrary boolean formula. For non-SNF queries, OXT requires linear time in the number of documents. This motivates the following natural question: *can we design SSE schemes that support arbitrary disjunctive and arbitrary boolean queries with sub-linear search complexity?* In other words, can we design solutions for these queries that are efficient even in the worst-case?

1.1 Our Contributions and Techniques

In this work, we address this problem and propose efficient disjunctive and boolean SSE schemes with worst-case sub-linear search complexity and optimal communication overhead. Our schemes are non-interactive and, as far as we know, the first to achieve optimal communication complexity. To do this we make several contributions which we summarize below

Worst-case disjunctive search. Our first solution, which we call IEX, is a worst-case sub-linear disjunctive SSE scheme. While it leaks more than the naive response-hiding solution, we stress that it achieves *optimal* communication complexity which, for response-hiding schemes, is the main tradeoff we seek. In addition, it leaks *less* than OXT (when used for disjunctive queries) while achieving worst-case efficiency.

The underlying idea behind IEX's design is best expressed in set-theoretic terms where we view the result of a disjunctive query $w_1 \vee \cdots \vee w_q$ as the union of the results of each individual term. More precisely, if we denote by $\mathsf{DB}(w)$ the set of document identifiers that contain the query w, then $\mathsf{DB}(w_1 \vee \cdots \vee w_q) = \mathsf{DB}(w_1) \cup \cdots \cup \mathsf{DB}(w_q)$. Using the naive response-hiding approach, one could use a single-keyword response-hiding scheme to query each keyword and compute the union at the client but, as discussed above, this would incur poor communication complexity. Our approach is different and, intuitively speaking, makes use of the *inclusion-exclusion* principle as follows. Consider a three-term query $w_1 \vee w_2 \vee w_3$. Instead of searching for $\mathsf{DB}(w_1)$, $\mathsf{DB}(w_2)$, $\mathsf{DB}(w_3)$ and computing the union, we compute $\mathsf{DB}(w_1)$ and remove from it

$$\mathsf{DB}(w_1) \cap \mathsf{DB}(w_2) \quad \text{and} \quad \mathsf{DB}(w_1) \cap \mathsf{DB}(w_3).$$

We then compute $\mathsf{DB}(w_2)$ and remove from it $\mathsf{DB}(w_2) \cap \mathsf{DB}(w_3)$. Finally, we take the union of the remaining sets and add $\mathsf{DB}(w_3)$. It follows by the inclusion-exclusion principle that this results in exactly $\mathsf{DB}(w_1) \cup \mathsf{DB}(w_2) \cup \mathsf{DB}(w_3)$. If we could somehow support the intersection and removal operations at the server, then we could achieve optimal communication complexity. Note that this high-level approach is "purely disjunctive" in the sense that it does not rely on transforming the query into another form as done in OXT. The avoidance of SNF in particular is what enables us to achieve worst-case efficiency.

We stress that the intuition provided thus far is only a very high-level conceptual explanation of our approach and cannot be translated directly to work on encrypted data. The challenge is that no SSE scheme we are aware of directly supports the kind of set operations needed to implement this idea. Therefore, a major part of our contribution is in designing and analyzing such a scheme.

Boolean search. While IEX is naturally disjunctive, we show that it also supports boolean queries. Similarly to the disjunctive case, we explain our high-level approach in set-theoretic terms. First, recall that any boolean query can be written in conjunctive normal form (CNF) so it has the form $\Delta_1 \wedge \cdots \wedge \Delta_\ell$, where each $\Delta_i = w_{i,1} \vee \cdots \vee w_{i,q}$ is a disjunction. Given a response-hiding *disjunctive*-search scheme like IEX, a naive approach for CNF queries is to execute disjunctive searches for each disjunction $\Delta_1, \ldots, \Delta_\ell$ and have the client perform the intersection of the results. This approach is problematic, however, because it requires more communication than necessary. To avoid this we take the following alternative approach. We note that the result $\mathsf{DB}(\Delta_1 \wedge \cdots \wedge \Delta_\ell)$ is a subset of $\mathsf{DB}(\Delta_1)$ and that it can be computed by progressively keeping only the identifiers in $\mathsf{DB}(\Delta_1)$ that are also included in $\mathsf{DB}(\Delta_2)$ through $\mathsf{DB}(\Delta_\ell)$. Again, we stress that this description is only a high-level conceptual explanation of our approach and requires more work to instantiate over encrypted data.

The IEX structure. As mentioned above, a major challenge in this work is the design of an encrypted structure that supports the set-theoretic operations needed to implement the strategies discussed above. To achieve this, IEX makes use of a more complex structure than the traditional encrypted inverted index. In particular, IEX combines several instantiations of two kinds of structures: dictionaries and multi-maps. A dictionary (i.e., a key-value store) maps labels to values whereas a multi-map (i.e., an inverted index) maps labels to tuples of values. More precisely, the

IEX design consists of an encrypted *global* multi-map that maps every keyword w to its document identifiers $\mathsf{DB}(w)$ and an encrypted dictionary that maps every keyword to a *local* multi-map for w. The local multi-map of a keyword w maps all the keywords v that co-occur with w to the identifiers of the documents that contain both v and w. At a high-level, with the encrypted global multi-map we can recover $\mathsf{DB}(w_1)$. With the encrypted dictionary, we can recover the encrypted local multi-map for keywords w_2 through w_ℓ. And, finally, by querying the (encrypted) local multi-map of a keyword w with a keyword v, we can recover the identifiers of the documents that contain both w and v. With these basic operations, we can then execute a full disjunctive query as discussed above.

Instantiations. IEX is an abstract construction that makes black-box use of encrypted multi-maps and dictionaries which, in turn, can be instantiated with several concrete constructions, e.g., [21, 19, 28, 16]. [1] While its asymptotic complexity is not affected by how the building blocks are instantiated, its concrete efficiency is so we consider this choice carefully—especially how the local multi-maps are instantiated. We consider two instantiations. The first, IEX-2Lev, uses the 2Lev construction of Cash et al. [16] to encrypt the multi-maps (local and global). This particular instantiation is very efficient with respect to search time but produces large encrypted structures (e.g., 9.8GB for datasets of 34M keyword/id pairs).

To address this we propose a second instantiation called IEX-ZMF which trades off efficiency for compactness. In fact, we show that IEX-ZMF is an order of magnitude more compact than IEX-2Lev (e.g., producing 0.9GB EDBs for datasets with 34M keyword/id pairs). This compactness is achieved by encrypting IEX's local multi-maps with a new construction called ZMF which may be of independent interest and that we detail below.[2]

The ZMF scheme. ZMF is a multi-map encryption scheme that is inspired by and has similarities to the classic Z-IDX construction of Goh [25]. Its core design as well as its security are very different, however. While Z-IDX produces a collection of non-adaptively-secure *fixed*-size encrypted Bloom filters, ZMF produces a collection of *adaptively*-secure *variable*-sized encrypted Bloom filters. In addition, the hash functions used for each filter can all be derived from a fixed set of hash functions (even though the filters store a different number of elements). This last property is non-standard but is crucial for our approach to be practical as it allows us to generate constant-size tokens that can be used with every filter in the collection. We refer to such collections of Bloom filters as *matryoshka filters* and, as far as we know, they have not been considered in the past. As we detail in Section 7, encrypting matryoshka filters with adaptive security is quite challenging. For this, we rely on the random oracle model and on a non-standard use of online ciphers [10] which are streaming block ciphers in the sense that every ciphertext block depends only on the previous plaintext blocks. Note that like Z-IDX, ZMF has linear search time but we use it in our IEX construction only to encrypt the *local* multi-maps which guarantees that IEX-ZMF is still sub-linear.

Dynamism and forward-security. We extend IEX to be dynamic resulting in a new scheme DIEX. An important security property for dynamic SSE schemes is *forward security* which guarantees that updates to an encrypted structure cannot be correlated with previous queries. Forward security was introduced by Stefanov, Papamanthou and Shi [37] and recent work of Zhang, Katz and Papamanthou [39] has shown that it mitigates certain injection attacks on SSE schemes. One advantage of our DIEX construction is that it naturally inherits the forward-security of its underlying

[1]Other constructions such as [29, 17, 33, 37] could also be used but these are either dynamic or conjunctive which is not needed for the IEX.

[2]Multi-map encryption schemes are equivalent to SSE schemes so ZMF is an *adaptively*-secure compact SSE scheme with linear-time search.

encrypted multi-maps and dictionaries. That is, if the underlying structures are forward-secure then so is DIEX.

Reduced leakage. As we mentioned above, IEX leaks more than the naive response-hiding solution *while achieving optimal communication complexity*. We stress, however, that it leaks less than the naive response-revealing solution and than OXT. As an example, consider that if OXT is used to search for two conjunctions $\mathbf{w} = w_1 \wedge w_2$ and $\mathbf{w}' = w_3 \wedge w_2$ which share a common term, the server can recover the results for $\mathbf{w}'' = w_1 \wedge w_2 \wedge w_3$. In the case of disjunctions, OXT's leakage is equivalent to the naive response-revealing solution.

Experiments. To evaluate the efficiency of IEX and its instantiations we designed and built a new encrypted search framework called *Clusion* [5]. It is written in Java and leverages the Apache Lucene search library [2]. It also includes a Hadoop-based distributed parser and indexer we implemented to handle massive datasets. Our experiments show that IEX—specifically our IEX-2Lev instantiation—is very efficient and even achieves faster search times than those reported for a C++ implementation of OXT [17] on a comparable system. For example, for conjunctive, disjunctive and boolean queries with selectivity on the order of thousands, IEX-2Lev takes 12, 14.8 and 23.7ms, respectively. For the same conjunctive query, OXT is reported to take 200ms on a comparable system. Clearly, a C/C++ implementation of IEX would perform even better.

We also implemented IEX-ZMF to evaluate its efficiency and compactness. In our experiments, it produced EDBs of size 198MB and 0.9GB from datasets with 1.5M and 34M keyword/id pairs, respectively. This is highly compact in comparison to IEX-2Lev which produced 1.6GB, 9.8GB EDBs for 1.5M and 34M keyword/id pairs, respectively. We also evaluated the efficiency of IEX-ZMF and, as expected, its performance for setup, search and token size are worse than IEX-2Lev. For example, for a dataset with 34M keyword/id pairs, EDB setup takes 7.58 hours to process compared to 31 minutes for IEX-2Lev.

On a boolean query of the form $(w \vee x) \wedge (y \vee z)$, where the disjunctions had selectivity 2K and 10K, respectively, IEX-ZMF took 1610ms whereas IEX-2Lev took only 23.7ms. As expected due to its high degree of compactness, IEX-ZMF is slower than IEX-2Lev (this is the exact tradeoff we seek).

2 Related Work

SSE was first considered by Song, Wagner and Perrig [36]. Curtmola, Garay, Kamara and Ostrovsky [21] introduced the notion of adaptive-security for SSE and presented the first constructions that achieved optimal search time with a space-efficient index. STE was introduced by Chase and Kamara [19] who proposed constructions for two-dimensional arrays, graphs and web graphs.

In [25], Goh introduced the Z-IDX construction which has linear search complexity and produces highly compact indexes due to its use of Bloom filters. Here, we extract a general transformation implicitly used in the Z-IDX construction and use it in part to construct our ZMF scheme. Kamara, Papamanthou and Roeder gave the first optimal-time dynamic SSE scheme [29]. Cash et al. [17] proposed OXT; the first optimal-time conjunctive keyword search scheme. Faber et al. [22] extend OXT to handle range, substring, wildcard and phrase queries. Pappas et al. [35] and Fisch et al. [23] present solutions based on garbled circuits and Bloom filters that can support boolean formulas, ranges and stemming. In [35], the authors show how to build the first worst-case sub-linear time boolean encrypted search solution. Like Goh's Z-IDX construction and our ZMF scheme, the solution makes use of Bloom filters. In addition, it is the first adaptively-secure construction based on Bloom filters. For a disjunctive query $\mathbf{w}$, the scheme has search complexity $O(\log(n) \cdot C \cdot \mathsf{DB}(\mathbf{w}))$, where n is the number of documents and C is the cost of a 2-party secure function evaluation of a function that takes as input a Bloom filter of size $O(\#\mathrm{W})$ (i.e., the number of unique keywords in DB) and a q-term disjunctive query. We note that unlike IEX and OXT, it does not achieve optimal communication complexity. Also, while its search is sub-linear it involves multiple rounds of interactions.

Ishai, Kushilevitz, Lu and Ostrovsky propose a two-server SSE scheme that hides the access pattern and supports various complex queries including ranges, stemming and substring [27]. Cash et al. [16] design several I/O-efficient SSE schemes including the 2Lev construction which we use in one of our IEX instantiations. Kurosawa and Ohtaki [31] designed the first UC secure SSE scheme. Kurosawa [30] designed a linear-time construction that handles arbitrary boolean queries while not disclosing the structure of the boolean query itself. Forward Secrecy was first considered by Stefanov, Papamanthou and Shi [37]. In [15], Bost introduced an efficient forward secure construction. In [18], Cash and Tessaro give lower bounds on the locality of SSE by showing tradeoffs between locality, space overhead and read efficiency. Recently, Asharov, Naor, Segev and Shahaf gave SSE constructions with optimal locality, optimal space overhead and nearly-optimal read efficiency [9]. Encrypted search can also be achieved with other primitives like property-preserving encryption [11, 12], functional encryption [13, 14, 34], oblivious RAM [26], full-homomorphic encryption [24] and multi-party computation [38].

Online ciphers were introduced by Bellare, Boldyreva, Knudsen and Namprempre [10], where they propose several schemes including the HCB1 construction which we make use of in our ZMF implementation. More efficient constructions were later proposed by Andreeva et al. [8].

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 $\{1,\ldots,n\}$, and $2^{[n]}$ is the corresponding power set. We write $x \leftarrow \chi$ to represent an element x being sampled from a distribution χ, and $x \overset{\$}{\leftarrow} X$ 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 and s_i to its ith bit. $s^{|n}$ denotes the string s padded with $n-|s|$ 0's and $s_{|n}$ represents the first n bits of s. Given strings s and r, we refer to their concatenation as either $\langle s, r\rangle$ or $s\|r$. For an n-bit string s and for all nonnegative d, we denote by $s^{\|d}$ the string $\langle s_1^{|d}, \cdots, s_n^{|d}\rangle$. In this work, padding takes precedence over truncation; that is, $s_{|p}^{\|d} = (s^{\|d})_{|p}$.

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

Data structures. A *data structure* for a given data type is a representation in some computational model [3] of an object of the given type. Typically, the representation is optimized to support the type's query operation as efficiently as possible. For data types that support multiple queries, the representation is often optimized to efficiently support as many queries as possible. As a concrete

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

example, the dictionary *type* can be represented using various data structures depending on which queries one wants to support efficiently. Hash tables support Get and Put in expected $O(1)$ time whereas balanced binary search trees support both operations in worst-case $\log(n)$ time. For ease of understanding and to match colloquial usage, we will sometimes blur the distinction between data types and structures. So, for example, when referring to a *dictionary structure* or a *multi-map structure* what we are referring to is an unspecified instantiation of the dictionary or multi-map data type.

Basic structures. We make use of several basic data types including arrays, dictionaries and multi-maps which we recall here. An array A of capacity n stores n items at locations 1 through n and supports read and write operations. We write $v = \mathsf{A}[i]$ to denote reading the item at location i and $\mathsf{A}[i] = v$ the operation of storing an item at location i. A dictionary DX of capacity n is a collection of n label/value pairs $\{(\ell_i, v_i)\}_{i\leq n}$ and supports Get and Put operations. We write $v_i = \mathsf{DX}[\ell_i]$ to denote getting the value associated with label ℓ_i and $\mathsf{DX}[\ell_i] = v_i$ to denote the operation of associating the value v_i in DX with label ℓ_i. A multi-map MM with capacity n is a collection of n label/tuple pairs $\{(\ell_i, V_i)_i\}_{i\leq n}$ that supports Get and Put operations. Similarly to dictionaries, we write $V_i = \mathsf{MM}[\ell_i]$ to denote getting the tuple associated with label ℓ_i and $\mathsf{MM}[\ell_i] = V_i$ to denote operation of associating the tuple V_i to label ℓ_i. We sometimes write $\mathsf{MM}^{-1}[v]$ to refer to the set of labels in MM associated with tuples that include the value v. 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) [17, 16].

Document collections. A document collection is a set of documents $\mathbf{D} = (D_1, \ldots, D_n)$, each document consisting of a set of keywords from some universe W. We assume the universe of keywords is totally ordered (e.g., using lexicographic order) and denote by $\mathrm{W}[i]$ the ith keyword in W. We assume every document has an identifier that is independent of its contents and denote it $\mathsf{id}(D_i)$. We assume the existence of an efficient indexing algorithm that takes as input a data collection $\mathbf{D}$ and outputs a multi-map that maps every keyword w in W to the identifiers of the documents that contain w. In previous work, this multi-map is referred to as an inverted index or as a database. For consistency, we refer to any multi-map derived in this way from a document collection as a database and denote it DB. Given a keyword w, we denote by $\mathsf{co}_{\mathsf{DB}}(w) \subseteq \mathrm{W}$ the set of keywords in W that co-occur with w; that is, the keywords that are contained in documents that contain w. When DB is clear from the context we omit DB and write only $\mathsf{co}(w)$.

3.1 Cryptographic Primitives

Basic cryptographic primitives. A private-key encryption scheme is a set of three polynomial-time algorithms $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ such that Gen is a probabilistic algorithm that takes a security parameter k and returns a secret key K; Enc is a probabilistic algorithm takes a key K and a message m and returns a ciphertext c; Dec is a deterministic algorithm that takes a key K and a ciphertext c and returns m if K was the key under which c was produced. Informally, a private-key encryption scheme is secure against chosen-plaintext attacks (CPA) if the ciphertexts it outputs do not reveal any partial information about the plaintext even to an adversary that can adaptively query an encryption oracle. We say a scheme is random-ciphertext-secure against chosen-plaintext attacks (RCPA) if the ciphertexts it outputs are computationally indistinguishable from random even to an adversary that can adaptively query an encryption oracle.[4] In addition to encryption

[4]RCPA-secure encryption can be instantiated practically using either the standard PRF-based private-key encryption scheme or, e.g., AES in counter mode.

schemes, we also make use of pseudo-random functions (PRF) and permutations (PRP), which are polynomial-time computable functions that cannot be distinguished from random functions by any probabilistic polynomial-time adversary.

Online ciphers. An online cipher (OC) is a block cipher that can encrypt data streams. In particular, with an OC the encryption of the ith block in a stream depends only on the 1st through ith message blocks. OCs were introduced by Bellare, Boldyreva, Knudsen and Namprempre [10]. More formally, we say that a cipher $\mathsf{OC} : \{0,1\}^k \times \{0,1\}^{n\times B} \to \{0,1\}^{n\times B}$, where $B > 1$ is the block length, is B-online if there exists a function $X : \{0,1\}^k \times \{0,1\}^{n\times B} \to \{0,1\}^B$ such that for any $\mathbf{m} \in \{0,1\}^{n\times B}$,

$$\mathsf{OC}_K(\mathbf{m}) = \mathsf{OC}^1_K(\mathbf{m}) \| \ldots \| \mathsf{OC}^n_K(\mathbf{m}),$$

where $\mathsf{OC}^i_K(\mathbf{m}) = X(K, m_1, \ldots, m_i)$ for all $i \in [n]$ and where m_i is the ith block of $\mathbf{m}$. OCs cannot be pseudo-random permutations (see [10] for a simple distinguisher) but can satisfy the weaker requirement of being computationally indistinguishable from a random *online* permutation. An online permutation is simply a permutation on a domain $\{0,1\}^{n\times B}$ whose ith block depends only on the first i blocks of its input. We denote by $\mathsf{OPerm}_{n,B}$ the set of all online permutations over $\{0,1\}^{n\times B}$. Security for an online cipher $\mathsf{OC} : \{0,1\}^k \times \{0,1\}^{n\times B} \to \{0,1\}^{n\times B}$ then holds if for all PPT adversaries $\mathcal{A}$,

$$\left| \Pr\left[\mathcal{A}^{\mathsf{OC}_K(\cdot)} = 1 : K \xleftarrow{\$} \{0,1\}^k \right] - \Pr\left[\mathcal{A}^{f(\cdot)} = 1 : f \xleftarrow{\$} \mathsf{OPerm}_{n,B} \right] \right| \leq \mathsf{negl}(k).$$

4 Definitions

Structured encryption schemes encrypt data structures in such a way that they can be privately queried. There are several natural forms of structured encryption. The original definition of [19] considered schemes that encrypt both a structure and a set of associated data items (e.g., documents, emails, user profiles etc.). In [20], 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 former reveal the response to queries whereas the latter do not.

STE schemes are used as follows. During a setup phase, the client constructs an encrypted data structure EDS under a key K. 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 the key K. The server then uses the token tk to query EDS. If the scheme is response-revealing, it recovers a response r. On the other hand, if the scheme is response-hiding it recovers a message that it returns to the client who in turn decrypts it with a resolving algorithm.

Definition 4.1 (Structured encryption)**.** *A single-round response-hiding structured encryption scheme* $\Sigma_{\mathscr{T}} = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Query}, \mathsf{Resolve})$ *for data type* $\mathscr{T}$ *consists of four 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 *of type* $\mathscr{T}$ *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*.*
- $c \leftarrow \mathsf{Query}(\mathsf{EDS}, \mathsf{tk})$*: is a (possibly) probabilistic algorithm that takes as input an encrypted structure* EDS *and a token* tk *and outputs a message* c*.*

- $r \leftarrow \mathsf{Resolve}(K, c)$*: is a deterministic algorithm that takes as input a secret key K and a message c and outputs a response r.*

We say that a structured encryption scheme Σ is correct if for all $k \in \mathbb{N}$, for all $\mathsf{poly}(k)$-size structures DS of type $\mathscr{T}$, 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 $\mathsf{Token}(K, q_i)$, for all messages c output by $\mathsf{Query}(\mathsf{EDS}, \mathsf{tk}_i)$, $\mathsf{Resolve}(K, c)$ returns the correct response with all but negligible probability. The syntax of a response-revealing STE scheme can be recovered by omitting the $\mathsf{Resolve}$ *algorithm and having* Query *output the response.*

Security. The standard notion of security for STE 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 security. If, on the other hand, the operations are chosen adaptively, this leads to the stronger notion of adaptive security [21]. This notion of security was first formalized by Curtmola *et al.* in the context of searchable encryption [21] and later generalized to structured encryption in [19].

Definition 4.2 (Adaptive security [21, 19])**.** *Let $\Sigma_{\mathscr{T}} = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Query})$ be a structured encryption scheme for type $\mathscr{T}$ and consider the following probabilistic experiments where $\mathcal{A}$ is a stateful adversary, $\mathcal{S}$ is a stateful simulator, $\mathcal{L}_{\mathsf{S}}$ and $\mathcal{L}_{\mathsf{Q}}$ are leakage profiles and $z \in \{0,1\}^*$:*

$\mathbf{Real}_{\Sigma,\mathcal{A}}(k)$*: given z the adversary $\mathcal{A}$ outputs a structure DS of type $\mathscr{T}$ and 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 $\mathsf{tk}_i \leftarrow \mathsf{Token}(K, q_i)$. Finally, $\mathcal{A}$ outputs a bit b that is output by the experiment.*

$\mathbf{Ideal}_{\Sigma,\mathcal{A},\mathcal{S}}(k)$*: given z the adversary $\mathcal{A}$ generates a structure DS of type $\mathscr{T}$ 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 query leakage $\mathcal{L}_{\mathsf{Q}}(\mathsf{DS}, q_i)$ 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 $(\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{Q}})$-secure if for all PPT *adversaries $\mathcal{A}$, there exists a* PPT *simulator $\mathcal{S}$ such that for all $z \in \{0,1\}^*$,*

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

5 IEX: A Worst-Case Sub-Linear Disjunctive SSE Scheme

Our main construction, IEX, makes black-box use of a dictionary encryption scheme $\Sigma_{\mathsf{DX}} = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Get})$, a multi-map encryption scheme $\Sigma_{\mathsf{MM}} = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Get})$, a pseudo-random function F, and of a private-key encryption scheme $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. The details of the scheme are provided in Fig. 1. At a high-level, it works as follows.

Setup. The Setup algorithm takes as input a security parameter k and an index DB. It makes use of two data structures: a dictionary DX and a *global* multi-map MM_g. MM_g maps every keyword in $w \in \mathrm{W}$ to an encryption of the identifiers in $\mathsf{DB}(w)$. We refer to these encryptions as *tags* and they are computed by evaluating $\mathsf{SKE.Enc}$ using as coins the evaluation of F on keyword w and the identifier. The global multi-map MM_g is then encrypted using Σ_{MM}, resulting in EMM_g.

For each keyword $w \in \mathrm{W}$, the algorithm creates a *local* multi-map MM_w, that maps the keywords $v \in \mathsf{co}(w)$ to tags of identifiers in $\mathsf{DB}(v) \cap \mathsf{DB}(w)$. Intuitively, the purpose of the local multi-map MM_w is to quickly find out which documents contain both w and v, for any $v \neq w$. The local multi-maps MM_w are then encrypted with Σ_{MM}. This results in encrypted multi-maps EMM_w which are then stored in the dictionary DX such that $\mathsf{DX}[w] = \mathsf{EMM}_w$. In other words, it stores label/value pairs (w, MM_w) in DX. Finally, DX is encrypted with Σ_{DX}, resulting in an encrypted dictionary EDX. The output of Setup includes the encrypted structures $(\mathsf{EDX}, \mathsf{EMM}_g)$ as well as their keys.

There are several optimizations possible for Setup that we omit in our formal description for ease of exposition. The first is that the encrypted local multi-maps can be stored "by reference" in the encrypted dictionary EDX instead of "by value". More precisely, instead of storing the actual encrypted local multi-maps EMM_w in EDX one can just store a *pointer* to them. Another optimization is that, depending on how Σ_{MM} is designed, the keys for the local encrypted multi-maps could all be generated from a single key using a PRF (with a counter). This would reduce the size of K. This optimization can be easily applied to most known encrypted multi-map schemes including the ones from [21, 19, 28, 17, 16].

Token. The Token algorithm takes as input a key and a vector of keywords $\mathbf{w} = (w_1, \ldots, w_q)$. For all $i \in [q-1]$ it creates a "sub-token" $\mathbf{tk}_i = (\mathsf{dtk}_i, \mathsf{gtk}_i, \mathsf{ltk}_{i+1}, \ldots, \mathsf{ltk}_q)$ composed of a dictionary token dtk_i, a global token gtk_i for w_i and, for all keywords w_{i+1} through w_q in the disjunction, a local token ltk_j for w_j, with $i+1 \leq j \leq q$. Intuitively, the global token will allow the server to query the encrypted global multi-map EMM_g to recover tags of the ids in $\mathsf{DB}(w_i)$. The dictionary token for w_i will then allow the server to query the encrypted dictionary EDX to recover w_i's local multi-map EMM_i. Finally, the local tokens will allow the server to query w_i's encrypted local multi-map EMM_i to recover the tags of the ids of the documents that contain both w_i and w_{i+1}, w_i and w_{i+2}, etc. As we will see next, this information will be enough for the server to find the relevant documents. For the last keyword w_q in the disjunction, the algorithm only needs to create a global token.

Search. The Search algorithm takes as input $\mathsf{EDB} = (\mathsf{EDX}, \mathsf{EMM}_g)$ and a token $\mathsf{tk} = (\mathbf{tk}_1, \ldots, \mathbf{tk}_{q-1}, \mathsf{gtk}_q)$. For each sub-token $\mathbf{tk}_i = (\mathsf{dtk}_i, \mathsf{gtk}_i, \mathsf{ltk}_{i+1}, \ldots, \mathsf{ltk}_q)$, the server does the following. It first uses gtk_i to query the global multi-map EMM_g and recover a set of identifier tags T_i for $\mathsf{DB}(w_i)$. It then uses dtk_i to query the encrypted dictionary EDX to recover the local multi-map EMM_i for w_i and uses ltk_{i+1} to query EMM_i to recover the tags T' for identifiers of the documents that contain both w_i and w_{i+1}; that is, the tags for the set $I' = \mathsf{DB}(w_i) \cap \mathsf{DB}(w_{i+1})$. The server then removes T'_i from T_i. It then repeats this process for all local tokens ltk_{i+2} to ltk_q. Once it finishes processing all local tokens in $\mathbf{tk}_i$, it holds the set of tags for the set

$$\mathsf{DB}(w_i) \setminus \bigcup_{j=i}^{q-1} \Big(\mathsf{DB}(w_i) \bigcap \mathsf{DB}(w_{j+1}) \Big). \tag{1}$$

Once it finishes processing all the sub-tokens, the server holds tags T_1 through T_{q-1}. For gtk_q, the server just queries the global multi-map to recover T_q. Finally, it outputs the set

$$T = \bigcup_{i=1}^{q} T_i. \tag{2}$$

5.1 Correctness and Efficiency

We now analyze the correctness and efficiency of our construction. The correctness of IEX follows from Eqs. (1) and (2) and from the inclusion-exclusion principle. Given a disjunctive query $\mathbf{w} = (w_1, \ldots, w_q)$, by Eq. (2), IEX.$\mathsf{Search}(\mathsf{EDB}, \mathsf{Token}(K, \mathbf{w}))$ will output

$$T = \bigcup_{i=1}^{q} T_i$$

Let F be a pseudo-random function, $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a private-key encryption scheme, $\Sigma_{\mathsf{DX}} = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Get})$ be a dictionary encryption scheme and $\Sigma_{\mathsf{MM}} = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Get})$ be a multi-map encryption scheme. Consider the disjunctive SSE scheme IEX $= (\mathsf{Setup}, \mathsf{Token}, \mathsf{Search})$ defined as follows:

```
• $\mathsf{Setup}(1^k, \mathsf{DB})$:
    1. sample $K_1, K_2 \overset{\$}{\leftarrow} \{0,1\}^k$;
    2. initialize a dictionary $\mathsf{DX}$ and a multi-map $\mathsf{MM}_g$;
    3. for all $w \in \mathrm{W}$,
        (a) for all $\mathsf{id} \in \mathsf{DB}(w)$, let $\mathsf{tag}_{\mathsf{id}} := \mathsf{Enc}_{K_1}\big(\mathsf{id}; F_{K_2}(\mathsf{id}\|w)\big)$;
        (b) set $\mathsf{MM}_g[w] := \big(\mathsf{tag}_{\mathsf{id}}\big)_{\mathsf{id}\in\mathsf{DB}(w)}$;
        (c) initialize a multi-map $\mathsf{MM}_w$ of size $\#\mathsf{co}(w)$;
        (d) for all $v \in \mathsf{co}(w)$,
            i. for all $\mathsf{id} \in \mathsf{DB}(v) \cap \mathsf{DB}(w)$, let $\mathsf{tag}_{\mathsf{id}} := \mathsf{Enc}_{K_1}\big(\mathsf{id}; F_{K_2}(\mathsf{id}\|w)\big)$;
            ii. set $\mathsf{MM}_w[v] := \big(\mathsf{tag}_{\mathsf{id}}\big)_{\mathsf{id}\in\mathsf{DB}(v)\cap\mathsf{DB}(w)}$;
        (e) compute $(K_w, \mathsf{EMM}_w) \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Setup}\big(1^k, \mathsf{MM}_w\big)$;
        (f) set $\mathsf{DX}[w] := \mathsf{EMM}_w$;
    4. compute $(K_g, \mathsf{EMM}_g) \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Setup}(1^k, \mathsf{MM}_g)$;
    5. compute $(K_d, \mathsf{EDX}) \leftarrow \Sigma_{\mathsf{DX}}.\mathsf{Setup}(1^k, \mathsf{DX})$;
    6. set $K = \big(K_g, K_d, \{K_w\}_{w\in\mathrm{W}}\big)$ and $\mathsf{EDB} = (\mathsf{EMM}_g, \mathsf{EDX})$;
    7. output $(K, \mathsf{EDB})$.

• $\mathsf{Token}(K, \mathbf{w})$:
    1. parse $\mathbf{w}$ as $(w_1, \ldots, w_q)$;
    2. for all $i \in [q-1]$,
        (a) compute $\mathsf{gtk}_i \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}(K_g, w_i)$;
        (b) compute $\mathsf{dtk}_i \leftarrow \Sigma_{\mathsf{DX}}.\mathsf{Token}(K_d, w_i)$;
        (c) for all $i+1 \leq j \leq \#\mathbf{w}$, compute $\mathsf{ltk}_j \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}(K_{w_i}, w_j)$;
        (d) set $\mathbf{tk}_i = \big(\mathsf{dtk}_i, \mathsf{gtk}_i, \mathsf{ltk}_{i+1}, \ldots, \mathsf{ltk}_{\#\mathbf{w}}\big)$;
    3. compute $\mathsf{gtk}_q \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}(K_g, w_q)$;
    4. output $\mathsf{tk} = (\mathbf{tk}_1, \ldots, \mathbf{tk}_{q-1}, \mathsf{gtk}_q)$.

• $\mathsf{Search}(\mathsf{EDB}, \mathsf{tk})$:
    1. parse $\mathsf{EDB}$ as $(\mathsf{EMM}_g, \mathsf{EDX})$;
    2. parse $\mathsf{tk}$ as $(\mathbf{tk}_1, \ldots, \mathbf{tk}_{q-1}, \mathsf{gtk}_q)$;
    3. for all $i \in [q-1]$,
        (a) parse $\mathbf{tk}_i$ as $\big(\mathsf{dtk}_i, \mathsf{gtk}_i, \mathsf{ltk}_{i+1}, \ldots, \mathsf{ltk}_q\big)$;
        (b) compute $T_i \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_g, \mathsf{gtk}_i)$;
        (c) compute $\mathsf{EMM}_i \leftarrow \Sigma_{\mathsf{DX}}.\mathsf{Get}(\mathsf{EDX}, \mathsf{dtk}_i)$;
        (d) for all $i+1 \leq j \leq q$,
            i. compute $T' \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_i, \mathsf{ltk}_j)$;
            ii. set $T_i = T_i \setminus T'$;
    4. compute $T_q \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_g, \mathsf{gtk}_q)$;
    5. output $\bigcup_{i\in[q]} T_i$;
```

Figure 1: Our disjunctive SSE scheme IEX.

$$
\begin{aligned}
&= \Big(\bigcup_{i=1}^{q-1} T_i\Big) \bigcup T_q \\
&= \left(\bigcup_{i=1}^{q-1} \left(\mathsf{DB}(w_i) \setminus \bigcup_{j=i}^{q-1} \Big(\mathsf{DB}(w_i) \bigcap \mathsf{DB}(w_{j+1})\Big)\right)\right) \bigcup \mathsf{DB}(w_q) \\
&= \left(\bigcup_{i=1}^{q-2} \left(\mathsf{DB}(w_i) \setminus \bigcup_{j=i}^{q-1} \Big(\mathsf{DB}(w_i) \bigcap \mathsf{DB}(w_{j+1})\Big)\right)\right) \\
&\qquad\qquad \bigcup \underbrace{\Big(\mathsf{DB}(w_{q-1}) \setminus \Big(\mathsf{DB}(w_{q-1}) \bigcap \mathsf{DB}(w_q)\Big)\Big) \bigcup \mathsf{DB}(w_q)}_{U}
\end{aligned}
\tag{3}
$$

where the first and third equalities hold by Eqs. (2) and (1), respectively. Note, however, that U equals $\mathsf{DB}(w_{q-1}) \bigcup \mathsf{DB}(w_q)$:

$$
\begin{aligned}
U &= \mathsf{DB}(w_{q-1}) \bigcap \Big(\overline{\mathsf{DB}(w_{q-1})} \bigcup \overline{\mathsf{DB}(w_q)}\Big) \bigcup \mathsf{DB}(w_q) \\
&= \Big(\mathsf{DB}(w_{q-1}) \bigcap \overline{\mathsf{DB}(w_{q-1})}\Big) \bigcup \Big(\mathsf{DB}(w_{q-1}) \bigcap \overline{\mathsf{DB}(w_q)}\Big) \bigcup \mathsf{DB}(w_q) \\
&= \Big(\mathsf{DB}(w_{q-1}) \bigcap \overline{\mathsf{DB}(w_q)}\Big) \bigcup \mathsf{DB}(w_q) \\
&= \Big(\mathsf{DB}(w_{q-1}) \bigcup \mathsf{DB}(w_q)\Big) \bigcap \Big(\overline{\mathsf{DB}(w_q)} \bigcup \mathsf{DB}(w_q)\Big) \\
&= \mathsf{DB}(w_{q-1}) \bigcup \mathsf{DB}(w_q)
\end{aligned}
$$

Repeating the same argument for $q-2$, $q-3$ and so on and plugging into Eq. (3), we get that $T = \bigcup_{i=1}^{q} \mathsf{DB}(w_i)$.

Efficiency. The search complexity of IEX is $O(q^2 \cdot M)$, where $M = \max_{i\in[q]} \#\mathsf{DB}(w_i)$ and q is the number of terms in the disjunction. Tokens are of size $O(q)$. We also note that unlike BXT and OXT [17], IEX tokens are *selectivity-independent* in the sense that they do not depend on the size of the result. The IEX storage complexity is,

$$
O\bigg(\mathsf{strg}\Big(\sum_{w} \#\mathsf{DB}(w)\Big) + \sum_{w} \mathsf{strg}\Big(\sum_{v\in\mathsf{co}(w)} \#\mathsf{DB}(v) \cap \mathsf{DB}(w)\Big)\bigg),
$$

where strg is the storage complexity of the underlying encrypted multi-map encryption scheme Σ_{MM}.

A storage optimization. As we can see, the storage complexity of IEX can be large, especially if the underlying encrypted multi-maps are. This is indeed the case when they are instantiated with standard sub-linear constructions. We observe, however, that we can tradeoff storage complexity (and setup time) for the communication complexity of search as follows. When constructing a local multi-map EMM_w for a keyword w, we normally insert tags for the identifiers in $\mathsf{DB}(w) \cap \mathsf{DB}(v)$ for all $v \in \mathsf{co}(w)$. This is not necessary for correctness, however, so we can omit some of the co-occurring keywords from w's local multi-map. The tradeoff is that this will increase the communication complexity of IEX's search operation and, in particular, make it non-optimal.

To do this, we suggest using the following approach to decide whether to add a keyword $v \in \mathsf{co}(w)$ or not. Let $p < 1$ be a *filtering* parameter and let

$$
T_{w,v} \overset{def}{=} \frac{\#\mathsf{DB}(v) \cap \mathsf{DB}(w)}{\max(\#\mathsf{DB}(w), \#\mathsf{DB}(v))}.
$$

If $T_{w,v} > p$, then add v to EMM_w otherwise do not. With this filtering in place, the storage complexity of IEX is now

$$O\Bigg(\mathsf{strg}\Bigg(\sum_w \#\mathsf{DB}(w)\Bigg) + \sum_w \mathsf{strg}\Bigg(\sum_{\substack{v \in \mathsf{co}(w) \\ T_{w,v} > p}} \#\mathsf{DB}(v) \cap \mathsf{DB}(w)\Bigg)\Bigg).$$

In our experiments we set $p = 0.2$.

Remark. We note that when all the terms of the disjunctive query have selectivity $O(n)$, IEX has linear search complexity. This is, however, the best one can do. On the other hand, the communication complexity of IEX remains optimal independently of the selectivity of the terms. This similarly applies to OXT but not to BlindSeer since it induces a logarithmic (multiplicative) overhead.

5.2 Security

The setup leakage of IEX consists of the setup leakage of its underlying building blocks. In particular, this includes the setup leakage of the encrypted global multi-map and of the encrypted dictionary. Assuming the use of standard optimal-time multi-map and dictionary encryption schemes [21, 19, 28, 16], this reveals the size of the database DB as well as the total size of the local multi-maps stored in the dictionary. The query leakage of IEX for a query $\mathbf{w}$ includes, for each keyword $w_i \in \mathrm{W}$, the query leakage of the encrypted dictionary and of the encrypted global multi-map. It also includes the query leakage of every queried local multi-map as well as their setup leakage. Again if instantiated with standard constructions, this will consist of the search and access patterns which, respectively, capture whether or not the same query has been searched for and (in our case) the tags. Finally, the query leakage also includes the number of documents containing $\mathsf{DB}(w_i) \bigcap \mathsf{DB}(w_{j+1})$, for all $j \geq i$ and $i \in [q-1]$.

We now give a precise description of IEX's leakage profile and show that it is adaptively-secure with respect to it. Its setup leakage is

$$\mathcal{L}_{\mathsf{S}}^{\mathsf{iex}}(\mathsf{DB}) = \Big(\mathcal{L}_{\mathsf{S}}^{\mathsf{dx}}(\mathsf{DX}), \mathcal{L}_{\mathsf{S}}^{\mathsf{mm}}(\mathsf{MM}_g)\Big),$$

where $\mathcal{L}_{\mathsf{S}}^{\mathsf{dx}}(\mathsf{DX})$ and $\mathcal{L}_{\mathsf{S}}^{\mathsf{mm}}(\mathsf{MM}_g)$ are the setup leakages of the underlying dictionary and multi-map encryption schemes, respectively. Its query leakage is

$$\begin{aligned}\mathcal{L}_{\mathsf{Q}}^{\mathsf{iex}}(\mathsf{DB}, \mathbf{w}) = \Bigg(\Bigg(&\mathcal{L}_{\mathsf{Q}}^{\mathsf{dx}}(\mathsf{DX}, w_i), \mathcal{L}_{\mathsf{S}}^{\mathsf{mm}}(\mathsf{MM}_i), \\ &\mathcal{L}_{\mathsf{Q}}^{\mathsf{mm}}(\mathsf{MM}_g, w_i), \ldots, \mathcal{L}_{\mathsf{Q}}^{\mathsf{mm}}(\mathsf{MM}_i, w_q), \mathsf{TagPat}_i(\mathsf{DB}, \mathbf{w})\Bigg)_{i \in [q-1]}, \\ &\mathcal{L}_{\mathsf{Q}}^{\mathsf{mm}}(\mathsf{MM}_g, w_q), \mathsf{TagPat}_q(\mathsf{DB}, \mathbf{w})\Bigg),\end{aligned}$$

where, for all $i \in [q]$,

$$\mathsf{TagPat}_i(\mathsf{DB}, \mathbf{w}) = \Bigg(\Big(f_i(\mathsf{id})\Big)_{\mathsf{id} \in \mathsf{DB}(w_i) \cap \mathsf{DB}(w_{i+1})}, \ldots, \Big(f_i(\mathsf{id})\Big)_{\mathsf{id} \in \mathsf{DB}(w_i) \cap \mathsf{DB}(w_q)}\Bigg),$$

and f_i is a random function from $\{0,1\}^{|\mathsf{id}| + \log \#\mathrm{W}}$ to $\{0,1\}^k$.

Theorem 5.1. *If Σ_{DX} is adaptively $(\mathcal{L}_{\mathsf{S}}^{\mathsf{dx}}, \mathcal{L}_{\mathsf{Q}}^{\mathsf{dx}})$-secure, Σ_{MM} is adaptively $(\mathcal{L}_{\mathsf{S}}^{\mathsf{mm}}, \mathcal{L}_{\mathsf{Q}}^{\mathsf{mm}})$-secure,* SKE *is RCPA-secure and F is pseudo-random, then* IEX *is $(\mathcal{L}_{\mathsf{S}}^{\mathsf{iex}}, \mathcal{L}_{\mathsf{Q}}^{\mathsf{iex}})$-secure.*

The proof of Theorem 5.1 is in Appendix B.

6 Boolean Queries with IEX

While IEX is naturally disjunctive, it can also support boolean queries. The boolean variant is similar to IEX in that it uses the same encrypted structures (i.e., it has the same Setup algorithm) but different Token and Search algorithms. We refer to the boolean variant of IEX as BIEX. We now provide an overview of how BIEX works.

Overview of BIEX. Recall that any query can be written in conjunctive normal form (CNF) so it has the form $\Delta_1 \wedge \cdots \wedge \Delta_\ell$, where each $\Delta_i = w_{i,1} \vee \cdots \vee w_{i,q}$ is a disjunction. Note that the result $\mathsf{DB}(\Delta_1 \wedge \cdots \wedge \Delta_\ell)$ is the intersection of $\mathsf{DB}(\Delta_1)$ through $\mathsf{DB}(\Delta_\ell)$. But this intersection does not have to be computed "directly" by executing a naive intersection operation. A better alternative (from a leakage point of view) is to compute the intersection by starting with $\mathsf{DB}(w_1)$, keeping only the subset of identifiers of $\mathsf{DB}(w_1)$ that are also in $\mathsf{DB}(w_2)$, then keeping only the subset of identifiers that are also in $\mathsf{DB}(w_3)$ and so on. This alternative approach requires only information about $\mathsf{DB}(w_1)$ and the progressive subsets. Moreover, it uses operations that are already supported by the IEX structures.

How we do this exactly, is best explained through a concrete example. Suppose we have a CNF query with $\Delta_1 = w_1 \vee w_2$ and $\Delta_2 = w_3$. The first step would be to perform a disjunctive query for Δ_1, resulting in tags for the identifiers in $\mathsf{DB}(\Delta_1)$. In the second step, we want to filter out and keep the tags of identifiers in $\mathsf{DB}(\Delta_1) \cap \mathsf{DB}(w_3)$. To find these tags, it suffices to query the local multi-maps of w_1 and w_2 on w_3. In the first case, EMM_{w_1} will return tags for $\mathsf{DB}(w_1) \cap \mathsf{DB}(w_3)$ and in the second case EMM_{w_2} will return tags for $\mathsf{DB}(w_2) \cap \mathsf{DB}(w_3)$. Finally, we take the union of both of these intersections and perform a final intersection with $\mathsf{DB}(\Delta_1)$. The final result equals $\mathsf{DB}(\Delta_1) \cap \mathsf{DB}(w_3)$. Fig. 2 describes this process in more detail and for arbitrary boolean queries.

Correctness. To show correctness we need to show that, given a boolean query in CNF form $\Delta_1 \wedge \cdots \wedge \Delta_\ell$ such that $\Delta_i = w_{i,1} \vee \cdots \vee w_{i,q}$ (for simplicity we assume the disjunctions all have q terms), BIEX.Search outputs

$$\bigcap_{i\in[\ell]} \bigcup_{j\in[q]} \mathsf{DB}(w_{i,j}). \tag{4}$$

Looking at the description of BIEX.Search in Fig. 2, one can see that every time Step 4(d)i is invoked it outputs

$$\bigcup_{j\in[q]} \mathsf{DB}(w_{i,t}) \cap \mathsf{DB}(w_{1,j}),$$

for all $t \in [q]$ and $i \in [\ell]$. Note that this stems from the fact that $\Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_j, \mathsf{ltk}_{t,i,j})$ outputs $\mathsf{DB}(w_{i,t}) \cap \mathsf{DB}(w_{1,j})$ for every $j \in [q]$.

Also, based on the correctness of IEX we know that the search for the first disjunction will output $\bigcup_{j\in[q]} \mathsf{DB}(w_{1,j})$ (with no redundant identifiers). So we have the final result of the query

$$I_\ell = \bigcup_{j\in[q]} \mathsf{DB}(w_{1,j}) \bigcap \underbrace{\left(\bigcup_{j,l\in[q]} (\mathsf{DB}(w_{2,j}) \cap \mathsf{DB}(w_{1,l}))\right) \bigcap \cdots \left(\bigcup_{j,l\in[q]} (\mathsf{DB}(w_{\ell,j}) \cap \mathsf{DB}(w_{1,l}))\right)}_{\ell-1 \text{ terms}} \tag{5}$$

On the other hand, note that for all $i \in [\ell]$ we have by Morgan's laws that

$$\begin{aligned}\bigcup_{j\in[q]} \mathsf{DB}(w_{1,j}) \bigcap \bigcup_{j\in[q]} \mathsf{DB}(w_{i,j}) &= \bigcup_{j\in[q]} \mathsf{DB}(w_{1,j}) \bigcap \bigcup_{j\in[q]} \mathsf{DB}(w_{1,j}) \bigcap \bigcup_{j\in[q]} \mathsf{DB}(w_{i,j}) \\ &= \bigcup_{j\in[q]} \mathsf{DB}(w_{1,j}) \bigcap \Bigg(\bigcup_{j,l\in[q]} \left(\mathsf{DB}(w_{i,j}) \cap \mathsf{DB}(w_{1,l})\right)\Bigg)\end{aligned}$$

That is, we can recursively apply the above result on Eq. (5) for all $l \in [\ell]$ to obtain Eq. (4).

Efficiency. The storage complexity of BIEX is the same as IEX. Its search complexity is

$$O\Bigg(q^2 \cdot \Big(\max_{w\in\Delta_1} \#\mathsf{DB}(w) + \ell \cdot \#\mathsf{DB}(\Delta_1)\Big)\Bigg).$$

The term $q^2 \cdot \max_{w\in\Delta_1} \#\mathsf{DB}(w)$ is the time to search for the first disjunction and the second term $q^2 \cdot \ell \cdot \#\mathsf{DB}(\Delta_1)$ is the total number of local multi-map queries.

We can clearly see from the search complexity of BIEX that we can achieve better efficiency if the selectivity of the first disjunction is as small as possible. In practice, therefore, the first disjunction should be the one with the smallest selectivity; similarly to how the first keyword is chosen in OXT. Note that if the first disjunction in the CNF form of the boolean query matches the entire database then the search complexity of BIEX will be linear while the optimal complexity might be sub-linear (the communication complexity of BIEX will remain optimal, however). It is not obvious to us how to improve this without pre-computing every possible query as it seems almost inherent to the query itself. With this in mind, it follows that BIEX has a sub-linear worst-case search complexity when the first disjunction's selectivity is sub-linear.

The communication complexity of BIEX is optimal since the final set I_ℓ does not contain any redundant identifiers. Finally, note that it is non-interactive and token size is independent of the query's selectivity.

Security. The setup leakage of BIEX is the same as IEX's. Its query leakage includes the query leakage of IEX on the first disjunction and the query leakage of the encrypted local multi-maps when queried on all the terms of disjunctions $\Delta_2, \ldots, \Delta_\ell$. Finally, it also includes the number of documents that match the terms of the first disjunction and the terms of remaining disjunctions.

We now give a precise description of the leakage profile of BIEX and show that it is adaptively-secure with respect to it. The setup leakage is

$$\mathcal{L}_{\mathsf{S}}^{\mathsf{iexb}}(\mathsf{DB}) = \mathcal{L}_{\mathsf{S}}^{\mathsf{iex}}(\mathsf{DB}),$$

where $\mathcal{L}_{\mathsf{S}}^{\mathsf{iex}}(\mathsf{DB})$ is the setup leakages of IEX. Given a CNF query $\Delta_1 \wedge \cdots \wedge \Delta_\ell$, the query leakage is

$$\begin{aligned}\mathcal{L}_{\mathsf{Q}}^{\mathsf{iexb}}\Big(\mathsf{DB}, \bigwedge_{i=1}^{\ell} \Delta_i\Big) = \Bigg(&\mathcal{L}_{\mathsf{Q}}^{\mathsf{iex}}(\mathsf{DB}, \Delta_1), \Big(\mathcal{L}_{\mathsf{Q}}^{\mathsf{mm}}(\mathsf{MM}_i, w_{l,1}), \cdots, \mathcal{L}_{\mathsf{Q}}^{\mathsf{mm}}(\mathsf{MM}_i, w_{l,q}), \\ &\mathsf{TagPat}_{i,l}\Big(\mathsf{DB}, \bigwedge_{i=1}^{\ell} \Delta_i\Big)\Big)_{\substack{i\in[q] \\ l\in[2,\cdots,\ell]}}\Bigg).\end{aligned}$$

where,

$$\mathsf{TagPat}_{i,l}\Big(\mathsf{DB}, \bigwedge_{i=1}^{\ell} \Delta_i\Big) = \Big(\Big(f_i(\mathsf{id})\Big)_{\mathsf{DB}(w_{1,i})\cap\mathsf{DB}(w_{l,1})}, \ldots, \Big(f_i(\mathsf{id})\Big)_{\mathsf{DB}(w_{1,i})\cap\mathsf{DB}_{(w_{l,q})}}\Big)$$

and f_i is a random function from $\{0,1\}^{n+\log \#\mathrm{W}}$ to $\{0,1\}^k$.

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

The proof of Theorem 6.1 is similar (at a high-level) to the proof of Theorem 5.1.

7 ZMF: A Compact and Adaptively-Secure SSE Scheme

The main limitation of IEX is its storage complexity of

$$O\Big(\mathsf{strg}_g(\mathsf{MM}_g) + \sum_w \mathsf{strg}_\ell(\mathsf{MM}_w)\Big),$$

where strg_g and strg_ℓ are the storage complexity of the global and local EMMs, respectively. If the latter are instantiated with standard sub-linear-time constructions such as [21, 29, 17, 16], we have

$$O\Big(\sum_w \#\mathsf{DB}(w) + \sum_w \sum_{v\in\mathsf{co}(w)} \#\mathsf{DB}(v)\cap\mathsf{DB}(w)\Big), \tag{6}$$

which does not compare favorably to standard single-keyword search solutions which require only $O(\sum_w \#\mathsf{DB}(w))$, or to the OXT construction of [17] which requires

$$O\Big(\sum_w \#\mathsf{DB}(w) + \log\Big(\frac{1}{\varepsilon}\Big)\cdot\sum_w \#\mathsf{DB}(w)\Big)$$

when XSet is instantiated with a Bloom filter with a false positive rate of ε. In particular, note that the second term in the asymptotic expression above hides a constant of 1, which makes OXT reasonably compact.

Our approach. The main storage inefficiency in IEX comes from the local EMMs which contribute the second term in Eq. (6). Ideally, we could improve things if we could use more compact local EMMs. Unfortunately, all known sub-linear constructions require $O(\sum_w \#\mathsf{DB}(w))$ storage. We observe, however, that for local EMMs sub-linear search is not necessary since in practice the number of label/tuple pairs they store is small in comparison to the total number of documents n. So, for our purposes, a linear-time construction would work as long as it was compact. In [25], Goh proposed a very compact construction called Z-IDX based on Bloom filters. Specifically, it needs only

$$O\Big(\log\Big(\frac{1}{\varepsilon}\Big)\cdot\sum_{v\in\mathbf{V}} \#\mathsf{MM}^{-1}[v]\Big)$$

bits of storage, where $\mathbf{V}$ is the value space of the multi-map and ε is the false positive rate. If we could encrypt the local EMMs of IEX with Z-IDX, the former's storage would be

$$O\Big(\sum_w \#\mathsf{DB}(w) + \log\Big(\frac{1}{\varepsilon}\Big)\cdot\sum_w \#\mathsf{co}(w)\Big),$$

Let IEX $=$ (Setup, Token, Search) be the IEX scheme described in Figure 1 and let $\Sigma_{\mathsf{DX}} =$ (Setup, Token, Get) and $\Sigma_{\mathsf{MM}} =$ (Setup, Token, Get) be its underlying dictionary and multi-map encryption schemes, respectively. Consider the boolean SSE encryption scheme BIEX = (Setup, Token, Search) defined as follows:

```
• Setup($1^k$, DB): output $(K, \mathsf{EDB}) \leftarrow$ IEX.Setup($1^k$, DB).

• Token($K, \mathbf{w}$):
    1. parse $K$ as $(K_g, K_d, \{K_w\}_{w \in \mathrm{W}})$;
    2. parse $\mathbf{w}$ as $\left(\Delta_1 \bigwedge \cdots \bigwedge \Delta_\ell\right)$ where for all $i \in [\ell]$, $\Delta_i = \left(w_{i,1} \bigvee \cdots \bigvee w_{i,d}\right)$;
    3. compute $\mathbf{tk}_1 \leftarrow \mathrm{IEX}.\mathsf{Token}_K(\Delta_1)$;
    4. for all $2 \leq i \leq \ell$ and all $j \in [q]$,
        (a) for all $1 \leq s \leq q$, compute $\mathsf{ltk}_{s,i,j} \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}(K_{w_{1,s}}, w_{i,j})$;
        (b) set $\mathbf{tk}_{i,j} = \big(\mathsf{ltk}_{1,i,j}, \ldots, \mathsf{ltk}_{q,i,j}\big)$;
        (c) set $\mathbf{tk}_i = \big(\mathbf{tk}_{i,1}, \ldots, \mathbf{tk}_{i,q}\big)$;
    5. output $\mathsf{tk} = (\mathbf{tk}_1, \ldots, \mathbf{tk}_\ell)$.

• Search(EDB, tk):
    1. parse EDB as $(\mathsf{EMM}_g, \mathsf{EDX})$;
    2. parse tk as $(\mathbf{tk}_1, \ldots, \mathbf{tk}_\ell)$;
    3. compute $I_1 \leftarrow \mathrm{IEX}.\mathsf{Search}(\mathsf{EDB}, \mathbf{tk}_1)$;
    4. for all $2 \leq i \leq \ell$,
        – instantiate an empty set $I_i$;
        – parse $\mathbf{tk}_i = \big(\mathbf{tk}_{i,1}, \ldots, \mathbf{tk}_{i,q}\big)$;
        – for $j \in [q]$,
            (a) get $\mathsf{dtk}_j$ from $\mathbf{tk}_1$;
            (b) compute $\mathsf{EMM}_j \leftarrow \Sigma_{\mathsf{DX}}.\mathsf{Get}(\mathsf{EDX}, \mathsf{dtk}_j)$;
            (c) parse $\mathbf{tk}_{i,j} = \big(\mathsf{ltk}_{1,i,j}, \ldots, \mathsf{ltk}_{q,i,j}\big)$;
            (d) for $s \in [q]$,
                i. compute $I' \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_j, \mathsf{ltk}_{s,i,j})$;
                ii. compute $I_i = I_i \bigcup \big(I_{i-1} \bigcap I'\big)$;
    5. output $I_\ell$;
```

Figure 2: The scheme BIEX.

Let $\Sigma_{\mathsf{SET}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Token}, \mathsf{Test})$ be a multi-structure set encryption scheme and consider the multi-map encryption scheme $\Sigma_{\mathsf{MM}} = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Get})$ defined as follows:

```
• $\mathsf{Setup}(1^k, \mathsf{MM})$:
    1. compute $K \leftarrow \mathsf{Gen}(1^k)$;
    2. let $\mathbf{V}$ be the range of $\mathsf{MM}$;
    3. for all $v \in \mathbf{V}$,
        (a) let $S_v = \mathsf{MM}^{-1}(v)$;
        (b) compute $\mathsf{ESET}_v \leftarrow \Sigma_{\mathsf{SET}}.\mathsf{Enc}(K, S_v)$;
    4. output $\mathsf{EMM} = (\mathsf{ESET}_v)_{v \in \mathbf{V}}$.
• $\mathsf{Token}(K, \ell)$: output $\mathsf{tk} \leftarrow \Sigma_{\mathsf{SET}}.\mathsf{Token}(K, \ell)$
• $\mathsf{Get}(\mathsf{EMM}, \mathsf{tk})$:
    1. let $I = \emptyset$;
    2. for all $v \in \mathbf{V}$,
        (a) if $\Sigma_{\mathsf{SET}}.\mathsf{Test}(\mathsf{ESET}_v, \mathsf{tk})$ outputs 1, set $I = I \cup \{v\}$;
    3. output $I$.
```

Figure 3: The Z-IDX transformation.

which is much more competitive with OXT (note that the second term here also has a constant of 1). Unfortunately, this approach does not work because Z-IDX is not adaptively secure. Nevertheless, we show how to construct a highly compact scheme that is. In the following, we first recall how Z-IDX works.

Goh's Z-IDX scheme. Like any SSE scheme, Z-IDX can be viewed as a STE scheme and, in particular, as a multi-map encryption scheme. Conceptually, we observe that Z-IDX can be abstracted into two parts: (1) a compiler that transforms an underlying set encryption scheme into a multi-map encryption scheme; and (2) a concrete set encryption scheme based on Bloom filters and PRFs. We refer to the former as the Z-IDX transformation and describe it in detail in Fig. 3. Given a set encryption scheme Σ_{SET}, it produces a multi-map encryption scheme Σ_{MM} that works as follows. The $\Sigma_{\mathsf{MM}}.\mathsf{Setup}$ algorithm takes as input a multi-map MM that maps labels to tuples of values from $\mathbf{V}$. It creates $\#\mathbf{V}$ sets $(S_v)_{v\in\mathbf{V}}$ such that S_v holds the labels in MM that map to v. It then encrypts each set S_v with Σ_{SET} resulting in an encrypted set ESET_v. The encrypted multi-map EMM is simply the collection of encrypted sets $(\mathsf{ESET}_v)_{v\in\mathbf{V}}$. A Σ_{MM} token for a label ℓ is a Σ_{SET} token for ℓ and $\Sigma_{\mathsf{MM}}.\mathsf{Get}$ uses the token to test each set in $\mathsf{EMM} = (\mathsf{ESET}_v)_{v\in\mathbf{V}}$ and outputs v if the test succeeds.

Note that for Σ_{MM} to work, Σ_{SET} must satisfy a stronger STE form than what is described in Definition 4.1. In particular, it must be what we call *multi-structure* in the sense that the tokens produced with a key K can be used to query all the structures encrypted under K. We provide formal syntax and security definitions of multi-structure STE schemes in Appendix A. The main difference between standard and multi-structure STE schemes are that in the latter the Setup algorithm is replaced with a key generation algorithm $\mathsf{Gen}(1^k)$ that takes as input a security parameter and outputs a secret key K; and an encryption algorithm $\mathsf{Enc}(K, \mathsf{DS})$ that takes as input a secret key K and a data structure DS and outputs an encrypted structure EDS.

Adaptive security. From our abstract perspective, the reason Z-IDX is not adaptively-secure is because the Z-IDX transformation is (implicitly) applied to a set encryption scheme that is not adaptively-secure. We show in Theorem 7.1 below, however, that if the transformation is applied to an adaptively-secure set encryption scheme then the result is adaptively-secure as well. More precisely, we show that if the set encryption scheme is adaptively $(\mathcal{L}_{\mathsf{S}}^{\mathsf{set}}, \mathcal{L}_{\mathsf{Q}}^{\mathsf{set}})$-secure then the Z-IDX transformation yields a multi-map encryption scheme with the following leakage profile:

$$\mathcal{L}_{\mathsf{S}}^{\mathsf{mm}}(\mathsf{MM}) = \left(\left(\mathcal{L}_{\mathsf{S}}^{\mathsf{set}}(\mathsf{MM}^{-1}[v])\right)_{v\in\mathbf{V}}, \#\mathbf{V}\right),$$

and

$$\mathcal{L}_{\mathsf{Q}}^{\mathsf{mm}}(\mathsf{MM}, q) = \mathcal{L}_{\mathsf{Q}}^{\mathsf{set}}\left(\left(\mathsf{MM}^{-1}[v]\right)_{v\in\mathbf{V}}, q\right).$$

Theorem 7.1. *If Σ_{SET} is adaptively $(\mathcal{L}_{\mathsf{S}}^{\mathsf{set}}, \mathcal{L}_{\mathsf{Q}}^{\mathsf{set}})$-secure then the scheme Σ_{MM} that results from applying the* Z-IDX *transformation to it is adaptively $(\mathcal{L}_{\mathsf{S}}^{\mathsf{mm}}, \mathcal{L}_{\mathsf{Q}}^{\mathsf{mm}})$-secure.*

The proof of Theorem 7.1 is in Appendix C.

7.1 An Adaptively-Secure and Multi-Structure Set Encryption Scheme

In this Section, we construct an adaptively-secure, highly-compact and multi-structure set encryption scheme. Then, by applying the Z-IDX transformation to it we get an adaptively-secure and highly-compact multi-map encryption scheme which we then use in IEX.

Adaptive security. The main difficulty in designing adaptively-secure encrypted structures is supporting *equivocation* during simulation. Roughly speaking, the issue is that during the $\mathbf{Ideal}(k)$ experiment the simulator first needs to simulate an encrypted structure for the adversary and later needs to be able to simulate tokens that work correctly with the simulated structure produced in the first step. The challenge in supporting equivocation is that at the time the encrypted structure is simulated, the simulator has no information about the adversary's queries so it is not clear how to simulate the structure in a way that will work correctly at query time. So to handle equivocation, the construction needs to be carefully designed and, typically, needs expensive cryptographic primitives. Fortunately, as first shown by Chase and Kamara [19], in the setting of symmetric STE, equivocation can be achieved very efficiently based only on XOR and PRF operations.

Our base scheme. One possible way to design an encrypted Bloom filter is as follows. Let $\mathbf{U}$ be a universe of elements. Given a set $S \subseteq \mathbf{U}$, insert the value $F_K(a)$, for all $a \in S$, in a standard Bloom filter, where F is a pseudo-random function. The token for an element $a \in \mathbf{U}$ is $\mathsf{tk} = F_K(a)$ and the Bloom filter can be queried by doing a standard Bloom filter test on tk.

The main problems with this construction are that: (1) it reveals information about the size of S; and (2) it is not adaptively-secure. To achieve adaptive security, we can encrypt the Bloom filter by XORing each of its bits with a pad generated from another pseudo-random function G. This encryption step both hides the size of S and allows for equivocation. Now the token tk for an element $a \in \mathbf{U}$ includes $F_{K_1}(a)$ and the pads for locations $H_1(F_{K_1}(a))$ through $H_\lambda(F_{K_1}(a))$, where $(H_1, \ldots, H_\lambda)$ are the hash functions used for the Bloom filter.

For this to work, however, the pads have to be designed carefully. More precisely, correctness requires that the pads only depend on the locations that they mask otherwise two (or more) elements a_1 and a_2 that collide under one of the hash functions will produce different masks for the same location. To get such *location-dependent* pads we compute them as $G_{K_2}(\ell)$, where ℓ is the ℓth bit of the filter. Now, a token for element a is set to

$$\mathsf{tk} = \left(F_{K_1}(a), G_{K_2}\left(H_1(F_{K_1}(a))\right), \ldots, G_{K_2}\left(H_\lambda(F_{K_1}(a))\right)\right).$$

The base construction described so far is *compact* and *adaptively-secure* but not multi-structure.

Reusability. Recall that a multi-structure STE scheme can produce *multiple* encrypted structures $(\mathsf{EDS}_1, \ldots, \mathsf{EDS}_n)$ under a single key K in such a way that a *single* (constant-size) token tk can be used to query all the structures generated under key K. So to make our base scheme multi-structure, the pads have to be filter-dependent in addition to being location-dependent so that different pads are used for different filters *even if they mask the same location*. We do this by setting the pads to be the output of a random oracle applied to the pair $(G_{K_2}(\ell), \mathsf{id})$ where id is the identifier of the filter. The purpose of the random oracle here is twofold. First, it enables the extraction of n (random) pads from pairs $(G_{K_2}(\ell), \mathsf{id}_1)$ through $(G_{K_2}(\ell), \mathsf{id}_n)$ without relying on n secret keys. This, in turn, means the tokens can be of size independent of n. Second, it allows the simulator to equivocate on the pads while, again, keeping the tokens independent of n.

While the base scheme is now compact, adaptively-secure and multi-structure, it produces very large tokens. The problem is that if two sets S_1 and S_2 have different sizes, then the parameters of their Bloom filters (i.e., the array sizes, number of hash functions and hash function ranges) have to be different. The consequence is that in our encrypted set scheme, we will need different sets of hash functions for *each* filter which, in turn, means the tokens will have to include multiple pads for *every filter*.

Matryoshka filters. We solve this problem as follows. Instead of encrypting a set of standard Bloom filters as in our base construction, we encrypt a new filter-based structure we refer to as *matryoshka filters* (MF).[5] MFs are essentially a set of *nested* Bloom filters of varying sizes whose hash functions are all derived from a fixed set of hash functions. More precisely, consider a sequence of sets $S_1, \ldots, S_n \subseteq \mathbf{U}$ not necessarily of the same size. We assume for simplicity that the sets have size a multiple of 2. For some false negative rate $2^{-\lambda}$, choose λ independent and ideal random hash functions $(H_1, \ldots, H_\lambda)$ from $\mathbf{U}$ to $[(\lambda/\ln 2) \cdot \max_i \#S_i]$. We refer to these functions as the *maximal* hash functions and to their associated filter as the *maximal* filter. For every set S_i, construct a Bloom filter of size $[\lceil(\lambda/\ln 2) \cdot \#S_i\rceil]$ with hash functions $(H^i_1, \ldots, H^i_h)$ where, for all $j \in [\lambda]$, $H^i_j(a) = H_j(a)_{\|p_i}$ with $p_i = \lceil \log\left((\lambda/\ln 2) \cdot \#S_i\right)\rceil$. We refer to these hash functions as the *derived* functions and to their associated filters as the *derived* filters. Note that if the maximal hash functions are ideal random functions then so are the derived functions so the standard Bloom filter analysis holds.

Encrypting matryoshka filters. As mentioned above, our final solution consists of adapting our base scheme to encrypt matryoshka filters instead of standard Bloom filters. In other words, we XOR each bit of each matryoshka filter with location- and filter-dependent pads. The main difference with the base scheme is that here the pads also need to be nested; that is, given a pad for the maximal filter we need to be able to construct the pads for the derived filters. To support this, we make use of the properties of online ciphers; namely, that given an n-bit string s and a B-online cipher OC, the following equality holds:

$$\mathsf{OC}_K\left(s^{\|B}_{|p\times B}\right) = \mathsf{OC}_K\left(s^{\|B}\right)_{|p\times B}, \tag{7}$$

where $p < n$. This can be derived as follows. From the correctness property of online ciphers, we have

$$\mathsf{OC}_K\left(s^{\|B}_{|p\times B}\right) = \mathsf{OC}^1_K\left(s^{\|B}_{|p\times B}\right)\| \cdots \|\mathsf{OC}^p_K\left(s^{\|B}_{|p\times B}\right)$$

[5]The term matryoshka here refers to Russian nested dolls which are called matryoshka dolls.

$$= X\left(K, s^{\|B}_{|B}\right)\|\cdots\|X\left(K, s^{\|B}_{|p\times B}\right),$$

and

$$\mathsf{OC}_K\left(s^{\|B}\right) = \mathsf{OC}^1_K\left(s^{\|B}\right)\|\cdots\|\mathsf{OC}^n_K\left(s^{\|B}\right)$$
$$= X\left(K, s^{\|B}_{|B}\right)\|\cdots\|X\left(K, s^{\|B}_{|n\times B}\right),$$

for some function X. It follows then that

$$\mathsf{OC}_K\left(s^{\|B}\right)_{|p\times B} = X\left(K, s^{\|B}_{|B}\right)\|\cdots\|X\left(K, s^{\|B}_{|p\times B}\right)$$
$$= \mathsf{OC}_K\left(s^{\|B}_{|p\times B}\right).$$

Now, to encrypt the ℓth bit of a matryoshka filter, we use a pad constructed as

$$\mathsf{R}\left(\mathsf{OC}_K\left(\ell^{\|B}_{|p\times B}\right), \mathsf{id}(S)\right),$$

where R is a random oracle and $\mathsf{id}(S)$ is the identifier of the filter. Note that the pad is both filter- and location-dependent. In addition, if the server is provided the value $\mathsf{OC}_K(\ell^{\|B})$ for the maximal filter, it follows by Eq. (7) that it can derive the above pad as

$$\mathsf{R}\left(\mathsf{OC}_K(\ell^{\|B})_{|p\times B}, \mathsf{id}(S)\right).$$

The detailed description of our set encryption scheme is given in Fig. 4. In the Theorem below we show that it is adaptively-secure with the following leakage profile:

$$\mathcal{L}^{\mathsf{est}}_{\mathsf{S}}(S) = \#S \quad \text{and} \quad \mathcal{L}^{\mathsf{set}}_{\mathsf{Q}}\left(S_1, \ldots, S_n, q\right) = \left(b_1, \ldots, b_n, \mathsf{SP}(q)\right),$$

where SP is the *search pattern*; that is, if and when two queries are the same. More formally, if t queries have been made, $\mathsf{SP}(q)$ outputs a t-bit string with a 1 at location i if q is equal to the ith query.

Theorem 7.2. *If* OC *is secure, then the multi-structure set encryption scheme described in Fig. 4 is adaptively* $(\mathcal{L}^{\mathsf{set}}_{\mathsf{S}}, \mathcal{L}^{\mathsf{set}}_{\mathsf{Q}})$*-secure in the random oracle model.*

The proof of Theorem 7.2 is in Appendix D.

7.2 The ZMF Multi-Map Encryption Scheme

By applying the Z-IDX transformation to our multi-structure set encryption scheme from Fig. 4, we get a new adaptively-secure multi-map encryption scheme we call ZMF. We state its security formally in the following Corollary of Theorems 7.1 and 7.2. Its leakage profile is,

$$\mathcal{L}^{\mathsf{zmf}}_{\mathsf{S}}(\mathsf{MM}) = \left(\left(\#\mathsf{MM}^{-1}[v]\right)_{v\in\mathbf{V}}, \#\mathbf{V}\right) \quad \text{and} \quad \mathcal{L}^{\mathsf{zmf}}_{\mathsf{Q}}(\mathsf{MM}, q) = \left(b_1, \ldots, b_{\#\mathbf{V}}, \mathsf{SP}(q)\right)$$

where b_i is 1 if $q \in \mathsf{MM}^{-1}[v_i]$ and 0 otherwise, and v_i is the ith value in $\mathbf{V}$.

Corollary 7.3. *The* ZMF *multi-map encryption scheme which results from applying the* Z-IDX *transformation to the set encryption scheme of Fig. 4 is* $(\mathcal{L}^{\mathsf{zmf}}_{\mathsf{S}}, \mathcal{L}^{\mathsf{zmf}}_{\mathsf{Q}})$*-adaptively secure.*

Let F be a pseudo-random function family, $\mathcal{H} : \{0,1\}^* \to [\sigma]$ be a family of hash functions modeled as random oracles where σ is a public upper bound, and $\mathsf{R} : \{0,1\}^* \to \{0,1\}$ be a random oracle. Let $\mathsf{OC} : \{0,1\}^{g(k)} \times \{0,1\}^{\gamma \times B} \to \{0,1\}^{\gamma \times B}$ a B-online cipher with $\gamma = \log \sigma$ blocks. Let $\varepsilon \in [0,1]$ be a false positive rate that is hardcoded in each algorithm. Set $\lambda = \log(1/\varepsilon)$ and set $\mathsf{H}_1, \ldots, \mathsf{H}_\lambda \leftarrow \mathcal{H}$. Consider the set encryption scheme $\Sigma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Token}, \mathsf{Test})$ defined as follows:

```
• $\mathsf{Gen}(1^k)$:
    1. sample $K_1 \xleftarrow{\$} \{0,1\}^k$ and $K_2 \xleftarrow{\$} \{0,1\}^{g(k)}$;
    2. output $K = (K_1, K_2)$;
• $\mathsf{Enc}(K, S)$:
    1. let $\mathsf{A}$ be a binary array of size $m = \lceil \lambda \cdot \#S / \ln 2 \rceil$ initialized to all 0's;
    2. for all items $a \in S$ and all $i \in [\lambda]$,
        (a) compute $T = F_{K_1}(a)$;
        (b) compute $\ell = \mathsf{H}_i(T)$;
        (c) compute $s = \mathsf{OC}_{K_2}\big(\ell^{\|B}_{|\log m \times B}\big)$;
        (d) set $\mathsf{A}\big[\ell_{|\log m}\big] = 1 \oplus \mathsf{R}\big(s, \mathsf{id}(S)\big)$;
    3. for all $i \in [m]$ such that $\mathsf{A}[i] = 0$,
        (a) compute $s = \mathsf{OC}_{K_2}\big(i^{\|B}_{|\log m \times B}\big)$;
        (b) set $\mathsf{A}\big[i\big] = 0 \oplus \mathsf{R}\big(s, \mathsf{id}(S)\big)$;
    4. set $\mathsf{ESET} = \mathsf{A}$;
    5. output $\mathsf{ESET}$.
• $\mathsf{Token}(K, a)$:
    1. compute $T = F_{K_1}(a)$;
    2. for all $i \in [\lambda]$,
        (a) compute $\ell = \mathsf{H}_i(T)$;
        (b) compute $s_i = \mathsf{OC}_{K_2}\big(\ell^{\|B}\big)$;
    3. output $\mathsf{tk} = \big(T, s_1, \ldots, s_\lambda\big)$.
• $\mathsf{Test}(\mathsf{ESET}, \mathsf{tk})$:
    1. parse $\mathsf{tk}$ as $(T, s_1, \ldots, s_\lambda)$;
    2. parse $\mathsf{ESET}$ as $\mathsf{A}$;
    3. set $m = |\mathsf{A}|$;
    4. for all $i \in [\lambda]$,
        (a) compute $b_i = \mathsf{A}\big[\mathsf{H}_i(T)_{|\log m}\big] \oplus \mathsf{R}\Big(\big(s_i\big)_{|\log m \times B}, \mathsf{id}(\mathsf{ESET})\Big)$;
    5. if, for all $i \in [\lambda]$, $b_i = 1$ output 1, otherwise output 0.
```

Figure 4: An adaptively-secure multi-structure set encryption scheme.

8 Dynamic SSE with IEX

Here, we describe a dynamic version of IEX. We first recall the syntax and security definitions for dynamic STE and detail our construction in Section 8.2.

8.1 Security Definitions

An STE scheme is dynamic if it supports updates on the encrypted structure without requiring a complete re-construction. Dynamic STE schemes are used in the same manner as static schemes except that when the client wants to update the structure it generates an update token utk that it sends to the server. Given utk and the encrypted structured EDS, the server generates an updated structure EDS'. Here, we will consider dynamic schemes that are *stateful*.

Definition 8.1 (Dynamic structured encryption)**.** *A single-round response-hiding dynamic structured encryption scheme* $\Sigma_{\mathscr{T}} = (\mathsf{Setup}, \mathsf{Token}^{\mathsf{sr}}, \mathsf{Query}, \mathsf{Token}^{\mathsf{up}}, \mathsf{Update}, \mathsf{Resolve})$ *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*.*
- $c \leftarrow \mathsf{Query}(\mathsf{EDS}, \mathsf{tk})$*: is a (possibly) probabilistic algorithm that takes as input an encrypted structure* EDS *and a token* tk *and outputs a message* c*.*
- $\mathsf{utk} \leftarrow \mathsf{Token}^{\mathsf{up}}(k, st, u)$*: is a (possibly) probabilistic algorithm that takes as input a secret key* K*, the state* st*, an update* u *and returns 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 encrypted structure* EDS*.*
- $r \leftarrow \mathsf{Resolve}(K, c)$*: is a deterministic algorithm that takes as input a secret key* K *and a message* c *and outputs a response* r*.*

We say that a dynamic structured encryption scheme $\Sigma_{\mathscr{T}}$ *is correct if for all* $k \in \mathbb{N}$*, for all* $\mathsf{poly}(k)$*-size structures* DS *of type* $\mathscr{T}$*, for all* (K, st, EDS) *output by* $\mathsf{Setup}(1^k, \mathsf{DS})$ *and all sequences of* $m = \mathsf{poly}(k)$ *operations* $o_1, \ldots, o_m$ *each of which can be a query* q_i *or an update* u_i*, for all tokens* tk_i *output by* $\mathsf{Token}^{\mathsf{sr}}(K, q_i)$ *or* utk_i *output by* $\mathsf{Token}^{\mathsf{up}}(K, u_i)$*, for all encrypted structures* EDS *output by* $\mathsf{Update}(\mathsf{EDS}, \mathsf{utk}_i)$*, for all messages* c *output by* $\mathsf{Query}(\mathsf{EDS}, \mathsf{tk}_i)$*,* $\mathsf{Resolve}(K, c)$ *returns the correct response with all but negligible probability.*

The syntax of a response-revealing dynamic STE scheme can be recovered by omitting the $\mathsf{Resolve}$ algorithm and having Query output the response.

Security. We formalize the security of dynamic STE schemes similarly to the security of static STE schemes. The main difference is that we also include the $\mathsf{Token}^{\mathsf{up}}$ and Update operations and the potential update leakage $\mathcal{L}_{\mathsf{U}}$. Intuitively, we require that a 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, the queries or the updates beyond the query and update leakage $\mathcal{L}_{\mathsf{Q}}$ and $\mathcal{L}_{\mathsf{U}}$, respectively.

Definition 8.2 (Adaptive security [21, 19])**.** *Let* $\Sigma_{\mathscr{T}} = (\mathsf{Setup}, \mathsf{Token}^{\mathsf{sr}}, \mathsf{Query}, \mathsf{Token}^{\mathsf{up}}, \mathsf{Update})$ *be a dynamic structured encryption scheme for type* $\mathscr{T}$ *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* DS *of type* $\mathscr{T}$ *and receives* EDS *from the challenger, where* $(K, st, \mathsf{EDS}) \leftarrow \mathsf{Setup}(1^k, \mathsf{DS})$*. The adversary then* adaptively *chooses a polynomial number of operations $o_1, \ldots, o_m$ such that o_i is either a query q_i or an update u_i. For all $i \in [m]$, the adversary receives* $\mathsf{tk}_i \leftarrow \mathsf{Token}(K, st, q_i)$ *of $o_i = q_i$ or* $\mathsf{utk}_i \leftarrow \mathsf{Token}^{\mathsf{up}}(K, st, u_i)$ *if $o_i = u_i$. 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* DS *of type* $\mathscr{T}$ *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 structure* EDS *to $\mathcal{A}$. The adversary then* adaptively *chooses a polynomial number of operations $o_1, \ldots, o_m$ such that o_i is either a query q_i or an update u_i. For all $i \in [m]$, the simulator receives either query leakage* $\mathcal{L}_{\mathsf{Q}}(\mathsf{DS}, q_i)$ *or update leakage* $\mathcal{L}_{\mathsf{U}}(\mathsf{DS}, u_i)$*. In the former case, it returns a query token* tk_i *to $\mathcal{A}$ and in the latter it returns an update 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 $z \in \{0,1\}^*$,*

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

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 [15] in the context of SSE. The definition, however, does not seem to be strong enough to capture the intuition described above. The approach of [15] is to define a forward-secure SSE scheme as one whose update leakage $\mathcal{L}_{\mathsf{U}}(\mathsf{DB}, u)$, where $u = \{(\mathsf{id}_i, \mathrm{W}_i)\}_i$, 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 $t = \#\mathrm{W}_i$, 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. This is formalized simply by requiring that the scheme be $(\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{Q}}, \mathcal{L}_{\mathsf{U}})$-secure where $\mathcal{L}_{\mathsf{U}}(\mathsf{DB}, u) = 1^k$ 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 [15] seems to satisfy this stronger property.

8.2 DIEX: A Dynamic SSE Scheme

We describe our dynamic SSE construction DIEX. As far as we know, it is the first adaptively-secure dynamic SSE scheme that is forward-secure and supports Boolean search queries in sub-linear time.

In particular, it supports the addition, deletion and editing of files.

Overview. As a starting point, we describe a dynamic version of IEX that is *not* forward-secure. For this, we make two changes to our static construction. First, we replace the encrypted dictionary EDX and the global encrypted multi-map EMM_g with a dynamic encrypted dictionary EDX^+ and a dynamic global encrypted multi-map EMM_g^+. The encrypted local multi-maps remain static. Second, we require that these new structures be *response-hiding.* We provide a high level description of our construction which is described in detail in Fig. 5.

The DIEX.Setup algorithm is the same as the IEX.Setup with the exception that it uses a dynamic encrypted dictionary and a dynamic encrypted multi-map and outputs state information st. The DIEX.$\mathsf{Token}^{\mathsf{sr}}$ algorithm is similar to IEX.Token with the exception that it is stateful. Here, the state is just used to generate tokens for the underlying dynamic dictionary and global multi-map encrypted structures. The DIEX.$\mathsf{Token}^{\mathsf{up}}$ algorithm works as follows. It takes as inputs the key K, the state st and an update $u = (\mathsf{op}, \mathsf{id}, \mathrm{W}_{\mathsf{id}})$ that consists of an operation $\mathsf{op} \in \{\mathsf{edit}^+, \mathsf{edit}^-\}$, the document identifier id being edited and a set of keywords W_{id} to add or delete based on op. We have the following cases:

- if $u = (\mathsf{edit}^+, \mathsf{id}, \mathrm{W}_{\mathsf{id}})$, the client will update the global multi-map EMM_g with pairs $(w, \mathsf{tag}_{\mathsf{id}})$ for all $w \in \mathrm{W}_{\mathsf{id}}$. Here, $\mathsf{tag}_{\mathsf{id}} := \mathsf{Enc}_{K_1}(\mathsf{id}; F_{K_2}(\mathsf{id}\|w))$ as in IEX. This is done by generating update tokens $(\mathsf{utk}_g^w)_{w\in\mathrm{W}_{\mathsf{id}}}$ for EMM_g using $\Sigma_{\mathsf{MM}}.\mathsf{Token}^{\mathsf{up}}$. For all $w \in \mathrm{W}_{\mathsf{id}}$, the client generates a new local multi-map MM_w that maps all $v \in \mathrm{W}_{\mathsf{id}} \setminus \{w\}$ to $\mathsf{tag}_{\mathsf{id}}$. It encrypts all these local multi-maps $(\mathsf{MM}_w)_{w\in\mathrm{W}_{\mathsf{id}}}$ with $\Sigma_{\mathsf{DX}}.\mathsf{Setup}$, resulting in $(\mathsf{EMM}_w)_{w\in\mathrm{W}_{\mathsf{id}}}$ and creates update tokens $(\mathsf{utk}_d^w)_{w\in\mathrm{W}_{\mathsf{id}}}$ for EDX. The algorithm outputs an update token
$$\mathsf{utk} = \Big(\mathsf{op}, (\mathsf{utk}_d^w)_{w\in\mathrm{W}_{\mathsf{id}}}, (\mathsf{utk}_g^w)_{w\in\mathrm{W}_{\mathsf{id}}}\Big),$$
and $st = (st_d, st_g)$, where the former is the state maintained by Σ_{DX} and the latter is the state maintained by Σ_{MM}.
- if $u = (\mathsf{edit}^-, \mathsf{id}, \mathrm{W}_{\mathsf{id}})$, the client only updates EMM_g. Specifically, it removes all pairs $(w, \mathsf{tag}_{\mathsf{id}})$ for $w \in \mathrm{W}_{\mathsf{id}}$. This can be done by computing tags as above and generating update tokens $(\mathsf{utk}_g^w)_{w\in\mathrm{W}_{\mathsf{id}}}$ using $\Sigma_{\mathsf{MM}}.\mathsf{Token}^{\mathsf{up}}$. The algorithm outputs the update token
$$\mathsf{utk} = \Big(\mathsf{op}, (\mathsf{utk}_g^w)_{w\in\mathrm{W}_{\mathsf{id}}}\Big),$$
and $st = st_g$ where st_g is the state maintained by Σ_{MM}.

The Update algorithm takes as input EDB and an update token utk and outputs EDB'. If $\mathsf{op} = \mathsf{edit}^+$, it uses the sub-tokens in utk to update EMM_g and EDX. If $\mathsf{op} = \mathsf{edit}^-$, it only updates EMM_g. The Search algorithm is the same as IEX.Search. Recall that we do not update the local multi-maps already in EDX. This is not necessary to for correctness because, during search, the server will take the intersection of the tags returned from the global multi-map EMM_g and from the appropriate local multi-maps. However, because EMM_g is properly updated, the intersection operation will filter out the old/stale tags from the local multi-map.

Forward security. We note that DIEX is forward secure if its underlying structures are. Specifically, if Σ_{MM} and Σ_{DX} are forward secure then so is DIEX. This is easy to see from the fact the DIEX tokens only consist of Σ_{DX} and Σ_{MM} tokens so if the former can be simulated from the security parameter, then the latter can. As a possible instantiation of a forward secure multi-map and dictionary encryption scheme, one can use the Sophos scheme of Bost [15].

Let $\Sigma_{\mathsf{DX}}^{+} = (\mathsf{Setup}, \mathsf{Token}^{\mathsf{sr}}, \mathsf{Get}, \mathsf{Token}^{\mathsf{up}}, \mathsf{Update})$ and $\Sigma_{\mathsf{MM}}^{+} = (\mathsf{Setup}, \mathsf{Token}^{\mathsf{sr}}, \mathsf{Get}, \mathsf{Token}^{\mathsf{up}}, \mathsf{Update})$ be dynamic dictionary and multi-map encryption schemes, respectively. Let $\mathrm{IEX}^{+} = (\mathsf{Setup}^{+}, \mathsf{Token}^{+}, \mathsf{Search}^{+})$ be the IEX scheme described in Fig. 1 with Σ_{MM} and Σ_{DX} replaced with Σ_{MM}^{+} and Σ_{DX}^{+}, respectively, and let $\Sigma_{\mathsf{MM}} = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Query})$ be the static multi-map encryption scheme used to encrypt the local multi-maps. Consider the dynamic disjunctive SSE scheme $\mathrm{DIEX} = (\mathsf{Setup}, \mathsf{Token}^{\mathsf{sr}}, \mathsf{Search}, \mathsf{Token}^{\mathsf{up}}, \mathsf{Update})$ defined as follows:

```
• $\mathsf{Setup}(1^k, \mathsf{DB})$: output $(K, st, \mathsf{EDB}) \leftarrow \mathrm{IEX}^{+}.\mathsf{Setup}(1^k, \mathsf{DB})$;
• $\mathsf{Token}^{\mathsf{sr}}(K, \mathbf{w})$: output $\mathsf{tk} \leftarrow \mathrm{IEX}^{+}.\mathsf{Token}(K, st, \mathbf{w})$;
• $\mathsf{Token}^{\mathsf{up}}(K, st, u)$
    1. parse $u$ as $(\mathsf{op}, \mathsf{id}, \mathrm{W}_{\mathsf{id}})$ and $st$ as $(st_g, st_d)$
    2. if $\mathsf{op} = \mathsf{edit}^{+}$,
        (a) for all $w \in \mathrm{W}_{\mathsf{id}}$,
            i. let $\mathsf{tag}_{\mathsf{id}} := \mathsf{Enc}_{K_1}\big(\mathsf{id}; F_{K_2}(\mathsf{id}\|w)\big)$;
            ii. compute $(\mathsf{utk}_g^w, st_g) \leftarrow \Sigma_{\mathsf{MM}}^{+}.\mathsf{Token}^{\mathsf{up}}(K, st_g, (\mathsf{op}, w, \mathsf{tag}_{\mathsf{id}}))$;
            iii. initialize a multi-map $\mathsf{MM}_w$ of size $\#\mathrm{W}_{\mathsf{id}}$;
            iv. for all $v \in \mathrm{W}_{\mathsf{id}} \setminus \{w\}$, set $\mathsf{MM}_w[v] = \mathsf{tag}_{\mathsf{id}}$;
            v. compute $(K_w, \mathsf{EMM}_w) \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Setup}\big(1^k, \mathsf{MM}_w\big)$;
            vi. compute $(\mathsf{utk}_d^w, st_d) \leftarrow \Sigma_{\mathsf{MM}}^{+}.\mathsf{Token}^{\mathsf{up}}(K, st_d, (\mathsf{op}, w, \mathsf{EMM}_w))$;
        (b) output $\mathsf{utk} = \big(\mathsf{op}, (\mathsf{utk}_d^w)_{w\in\mathrm{W}_{\mathsf{id}}}, (\mathsf{utk}_g^w)_{w\in\mathrm{W}_{\mathsf{id}}}\big)$;
    3. if $\mathsf{op} = \mathsf{edit}^{-}$,
        (a) for all $w \in \mathrm{W}_{id}$
            i. let $\mathsf{tag}_{\mathsf{id}} := \mathsf{Enc}_{K_1}\big(\mathsf{id}; F_{K_2}(\mathsf{id}\|w)\big)$;
            ii. compute $(\mathsf{utk}_g^w, st_g) \leftarrow \Sigma_{\mathsf{MM}}^{+}.\mathsf{Token}^{\mathsf{up}}(K, st_g, (\mathsf{op}, \mathrm{W}_{\mathsf{id}}, \mathsf{tag}_{\mathsf{id}}))$;
        (b) output $\mathsf{utk} = \big(\mathsf{op}, (\mathsf{utk}_g^w)_{w\in\mathrm{W}_{\mathsf{id}}}\big)$ and the updated state $st = (st_g, st_d)$;
• $\mathsf{Update}(\mathsf{EDB}, \mathsf{utk})$
    1. parse $\mathsf{utk}$ as $\big(\mathsf{op}, (\mathsf{utk}_i)_{i\in[\#\mathsf{tk}]}\big)$ and $\mathsf{EDB} = (\mathsf{EDX}, \mathsf{EMM}_g)$;
    2. if $\mathsf{op} = \mathsf{edit}^{-}$, then for all $i \in [\#\mathsf{utk}]$ compute $\mathsf{EMM}_g \leftarrow \Sigma_{\mathsf{MM}}^{+}.\mathsf{Update}(\mathsf{EMM}_g, \mathsf{utk}_i, \mathsf{op})$;
    3. if $\mathsf{op} = \mathsf{edit}^{+}$, then for all $i \in [\#\mathsf{utk}/2]$, compute $\mathsf{EMM}_g \leftarrow \Sigma_{\mathsf{MM}}^{+}.\mathsf{Update}(\mathsf{EMM}_g, \mathsf{utk}_i, \mathsf{op})$
       and $\mathsf{EDX} \leftarrow \Sigma_{\mathsf{DX}}^{+}.\mathsf{Update}(\mathsf{EDX}, \mathsf{utk}_{i+\#\mathsf{utk}/2+1}, \mathsf{op})$;
    4. output $\mathsf{EDB} = (\mathsf{EDX}, \mathsf{EMM}_g)$;
• $\mathsf{Search}(\mathsf{EDB}, \mathsf{utk})$: output $\bigcup_{i\in[q]} T_i \leftarrow \mathrm{IEX}^{+}.\mathsf{Search}(\mathsf{EDB}, \mathsf{tk})$.
```

Figure 5: The scheme DIEX.

Efficiency. The efficiency of DIEX depends on the underlying multi-map and dictionary encryption schemes. Using optimal constructions, the search complexity of DIEX is the same as IEX; that is, $O(q^2 \cdot M)$, where $M = \max_{i\in[q]} \#\mathsf{DB}(w_i)$ and q is the number of terms in the disjunction.

Security. We show that DIEX is adaptively secure with respect to the following well-defined leakage profile. The setup and query leakages are the same as IEX so we only describe the update leakage. For an update $u = (\mathsf{edit}^+, \mathsf{id}, \mathrm{W}_{\mathsf{id}})$,

$$\mathcal{L}_{\mathsf{U}}\Big(\mathsf{DB}, u\Big) = \Big(\mathcal{L}^{\mathsf{mm}}_{\mathsf{U}}(\mathsf{MM}_g, (\mathsf{op}, w, \mathsf{id})), \mathcal{L}^{\mathsf{dx}}_{\mathsf{U}}(\mathsf{DX}, (\mathsf{op}, w, \mathsf{id})), \mathcal{L}^{\mathsf{mm}}_{\mathsf{S}}(\mathsf{MM}_w)\Big)_{w\in \mathrm{W}_{\mathsf{id}}}.$$

If $u = (\mathsf{edit}^-, \mathsf{id}, \mathrm{W}_{\mathsf{id}})$: $\mathcal{L}^{\mathsf{diex}}_{\mathsf{U}}\Big(\mathsf{DB}, u\Big) = \Big(\mathcal{L}^{\mathsf{mm}}_{\mathsf{U}}(\mathsf{MM}_g, (\mathsf{op}, w, \mathsf{id}))\Big)_{w\in \mathrm{W}_{\mathsf{id}}}.$

Theorem 8.3. *If Σ_{DX} is adaptively $(\mathcal{L}^{\mathsf{dx}}_{\mathsf{S}}, \mathcal{L}^{\mathsf{dx}}_{\mathsf{Q}}, \mathcal{L}^{\mathsf{dx}}_{\mathsf{U}})$-semantically secure and Σ_{MM} is adaptively $(\mathcal{L}^{\mathsf{mm}}_{\mathsf{S}}, \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}, \mathcal{L}^{\mathsf{mm}}_{\mathsf{U}})$-secure, then* DIEX *is adaptively $(\mathcal{L}^{\mathsf{diex}}_{\mathsf{S}}, \mathcal{L}^{\mathsf{diex}}_{\mathsf{Q}}, \mathcal{L}^{\mathsf{diex}}_{\mathsf{U}})$-secure.*

The proof of Theorem 8.3 is similar to the proof of Theorem 5.1.

9 Empirical Evaluation

To evaluate the practicality of our constructions, we designed and implemented a new SSE framework called *Clusion*. It is implemented in Java and consists of 3445 lines of codes calculated using CLOC [4].

Parsing and indexing. Our framework uses the Lucene parser [2] to parse and index data. Lucene is an open-source high-performance text search engine written in Java maintained by the Apache project. We modify the Lucene parser to handle several types of files including `pdf` files, Microsoft Office files (`doc, docx, pptx, xlsx`) and basic `txt` files. For media files, such as pictures and movies, our framework only indexes the file name. Future work could include adding an image processing step that extracts keywords from media files (e.g., using AlchemyAPI [1]). For massive datasets, we also implemented a distributed Lucene-based parser and indexer using Hadoop [6]. Currently, the distributed parser/indexer only handles text files (e.g., `txt, html` etc.).

Cryptographic primitives. Cryptography in framework uses the Bouncy Castle library [3]. We instantiate PRFs with HMAC-SHA256 and symmetric encryption with AES in the CTR mode with a 128/256 bit key. For online ciphers, we implemented the HBC1 construction of Bellare et al. [10]. Because we never need to decrypt the OC-encrypted values, we replaced the encryption operation in HBC1 with a PRF evaluation. [6]

Experimental setup. We ran our experiments on Amazon EC2 instance c3.8xlarge an Intel Xeon E5-2680 v2 (Ivy Bridge) Processors with 32 vCPU and 60 GiB of RAM running Ubuntu Linux. All our benchmarks use the Enron email dataset which consists of 444,482 files and has total size 1.3GB uncompressed. All our time measurements are based on 50 executions and we include (when required) the minimum, maximum and average times.

Experiment overview. To evaluate IEX, we implemented two of its possible instantiations. The first is IEX-2Lev and is optimized for search efficiency. It uses the 2Lev scheme from Cash et al. [16] to encrypt its global and local multi-maps. Our second instantiation is IEX-ZMF and is optimized for compactness. It uses 2Lev to encrypt its global multi-map and ZMF to encrypt its local multi-maps.

[6]We verified that the proof of HBC1 follows as-is with a PRF

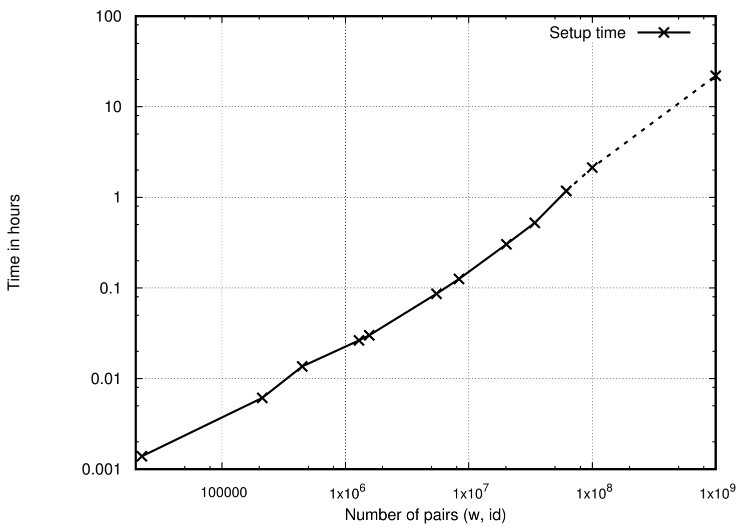

Figure 6: IEX-2Lev.Setup

9.1 IEX-2Lev Evaluation

In our experiments we want to evaluate several characteristics of our constructions. The first is setup time; that is, how long it takes to produce the encrypted data structures needed for search. The second is search time, i.e., the time needed to search for different types of queries. In particular, we consider conjunctive, disjunctive and boolean queries all on an EDB with 2M keyword/id pairs generated from a subset of the Enron dataset. The third is token size for different types of queries.

Setup time. To evaluate setup we measured the time it takes to produce EDB from DB. Note that IEX and BIEX have the same setup algorithm so our empirical results apply to both schemes. We stress that our measurements do not reflect the time it takes to parse and index the input dataset. The results of our experiments for IEX-2Lev with a filtering parameter of $p = 0.2$ are in Fig. 6. The Figure gives the setup time in seconds as a function of the number of keyword/id pairs in the dataset. We ran experiments with number of keyword/id pairs ranging from $22,460$ to $61,573,757$ and then extrapolated up to 10^9 under the assumption (validated by our theoretical analysis) that setup time increases linearly in the number of keyword/id pairs with a filtering parameter $p = 0.2$. Fig. 6 shows that IEX-2Lev requires 1.17 hours to process 61.5M pairs which is approximately $68\mu s$ per pair. We extrapolate that 1B pairs would require 21.97 hours.

The setup time of IEX-2Lev is about three times *slower* than the setup time of OXT-2Lev as reported in [16] (cf., the 2Lev graph in Fig 10) which is of $19.8\mu s$ per pair. Our implementation, however, is in Java whereas the implementation of [16] is in C. [7]

Conjunctive search time. Our first set of search experiments focus on conjunctive keyword searches and are summarized in Fig. 7. The queries have the form $(w \wedge x)$ and $(x \wedge w)$, where the selectivity of w is set to either 15 or 2K and the selectivity of x is increased up to 10K and extrapolated to 1M. As can be seen, IEX-2Lev is very efficient for conjunctive queries. For example,

[7] On our evaluation machine, a single PRF operation using Bouncy Castle's HMAC-SHA256 implementation in Java took $65\mu s$ whereas a PRF operation using OpenSSL's HMAC-SHA256 implementation in C took $8.6\mu s$ on 1024 bytes. Given that PRF operations are the dominant operation in both constructions we can roughly say that the setup time can greatly decrease if IEX was implemented in C.

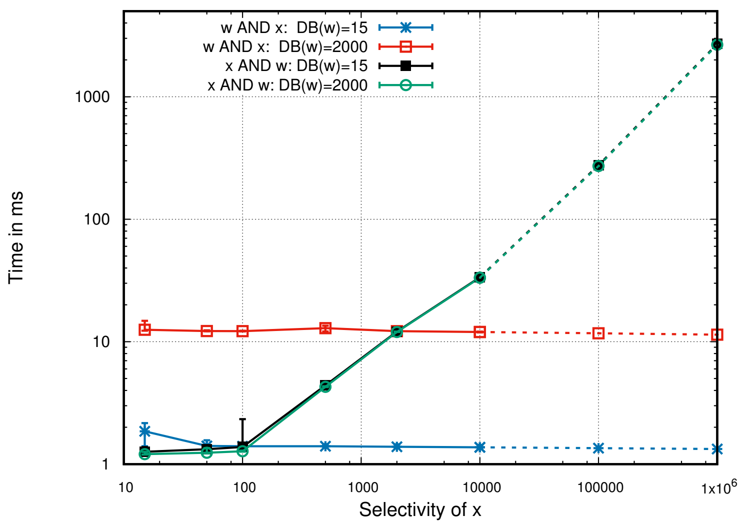

Figure 7: IEX-2Lev conjunctive keyword search

search time for $q_1 = (w \wedge x)$ with $\mathsf{DB}(w) = 15$ is about 1.2ms (independent of $\mathsf{DB}(x)$) which compares favorably—especially considering our implementation is in Java—to the time of 5ms reported for OXT on the same query. Similarly, for $q_2 = (w \wedge x)$ with $\mathsf{DB}(w) = 2000$, IEX-2Lev takes 12ms whereas OXT takes 200ms for a selectivity of x equal to 10K. The most noticeable difference, however, occurs for $q_3 = (x \wedge w)$ with $\mathsf{DB}(w) = 15$. If, for example, we set $\mathsf{DB}(x) = 10\text{K}$, IEX-2Lev takes just above 30ms whereas OXT takes just under 1s, a $30\times$ improvement.

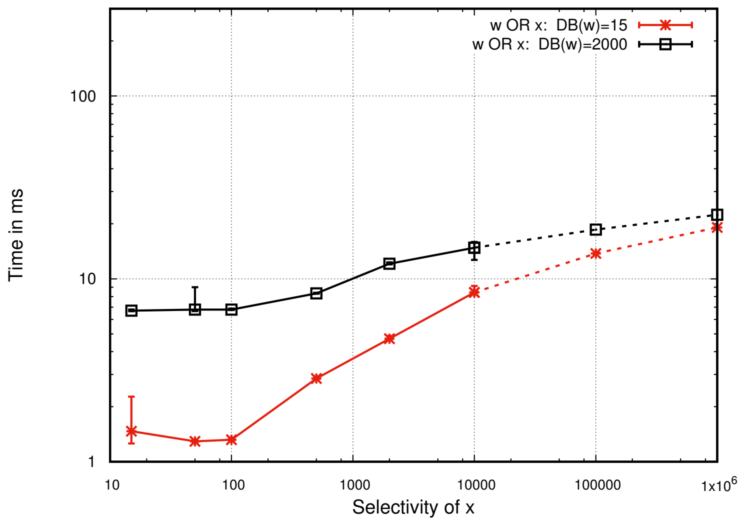

Figure 8: IEX-2Lev disjunctive keyword search

Disjunctive search time. In our second set of search experiments, summarized in Fig. 8, we measure IEX-2Lev's search time for two queries of the form $(w \vee x)$ with $\mathsf{DB}(w) = 15$ and $\mathsf{DB}(w) = 2\text{K}$, respectively, and varying $\mathsf{DB}(x)$ up to 10K and extrapolating to 1M. Our results

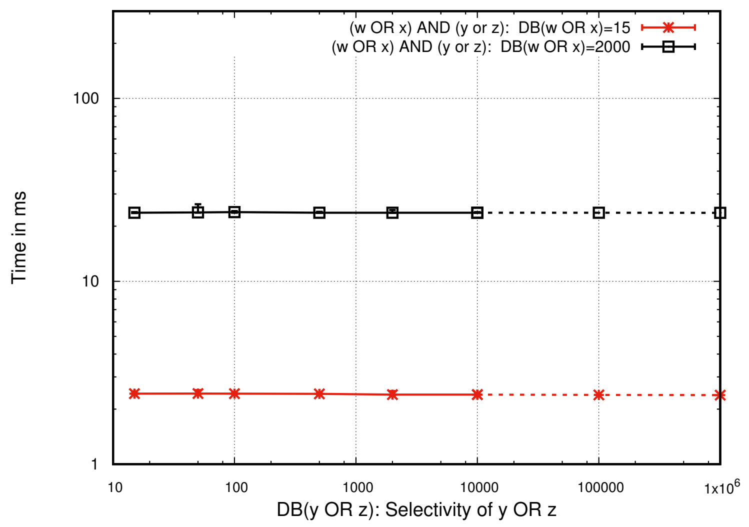

Figure 9: IEX-2Lev boolean keyword search

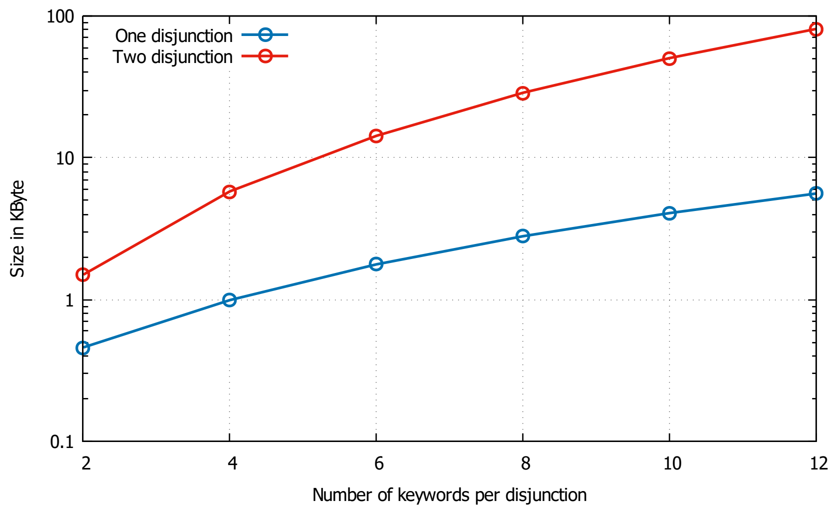

Figure 10: IEX-2Lev token Size

clearly show that IEX-2Lev is also very efficient for disjunctive queries as it takes 8.4 and 14.8ms for $\mathsf{DB}(w) = 15$ and $\mathsf{DB}(w) = 2\text{K}$, respectively, when $\mathsf{DB}(x) = 10\text{K}$.

Boolean search time. Our final set of search experiments are for boolean queries of the form $\big((w \vee x) \wedge (y \vee z)\big)$. We consider two cases: namely, when $\mathsf{DB}(w \vee x) = 15$ and $\mathsf{DB}(w \vee x) = 2K$ and, in each case, with $\mathsf{DB}(y \vee z)$ varied up to 10K and extrapolated to 1M. The results are summarized in Fig. 9 and again show that IEX-2Lev performs very efficiently as it takes, e.g., only 2.4ms and 23.7ms when $\mathsf{DB}(y \vee z) = 10\text{K}$.

Token size. To evaluate the token size of IEX-2Lev, we considered two types of queries. The first has the form $q_1^m = (w_1 \vee \cdots \vee w_m)$ consisting of a single disjunction, and the second has the form $q_2^m = (w_1 \vee \cdots \vee w_m) \wedge (x_1 \vee \cdots \vee x_m)$. We measured the size of the tokens when varying m from 2

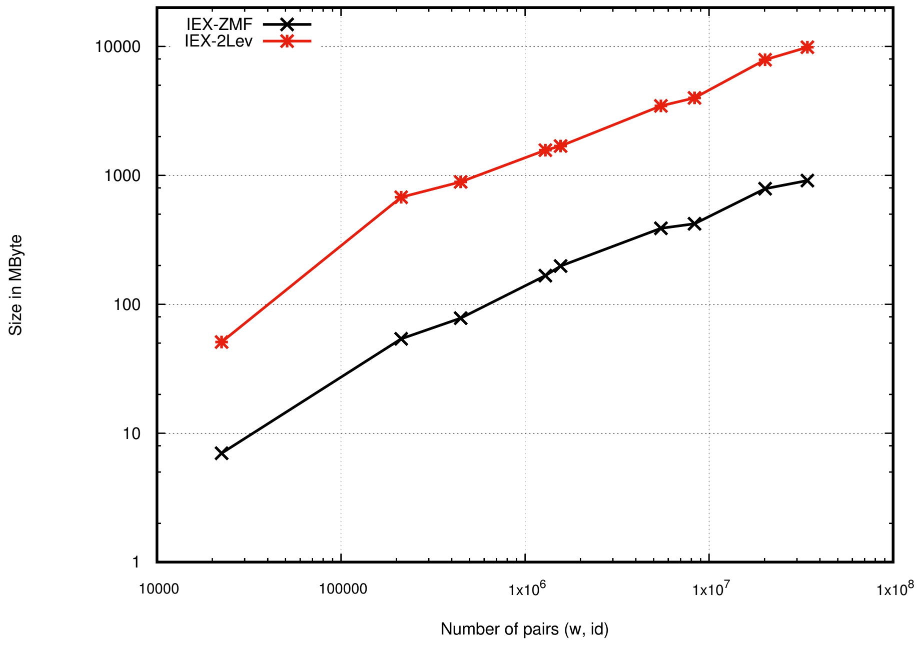

Figure 11: IEX-ZMF Storage Size

to 12. The results are summarized in Fig. 10. We can see that token size are very compact. For example, for q_2^{12}, the token size is only 81KB and for q_1^{12} it is only 5.6KB. Recall that, unlike OXT, the token size (and therefore the communication complexity) of IEX is independent of the query's selectivity.

9.2 IEX-ZMF Evaluation

We now turn our attention to IEX-ZMF and evaluate its setup time, conjunctive, disjunctive and boolean search time, its token size and its storage size.

Storage size. We report our results for storage size in Fig. 11. We measured the size of the data structures using SizeOF [7] Java agent which is based on serialization. Our experimental results confirm our theoretical findings and show that IEX-ZMF is *much* more compact than IEX-2Lev. For example, for $34,220,587$ keyword/id pairs IEX-ZMF produced an EDB of 911MB whereas IEX-2Lev produced an EDB of 9.8GB. In addition, we believe that IEX-ZMF's compactness can be further enhanced by improving the underlying data representations used in our implementation.

Setup time. Similarly to the setup experiment above, we measured the time it takes for IEX-ZMF to produce its encrypted search structure. Again, recall that our measurements do not reflect the time it takes to parse and index the input dataset. The results for IEX-ZMF are with a filtering parameter of $p = 0.2$. We ran experiments with the number of keyword/id pairs ranging from $22,460$ to $34,220,587$ and then extrapolated up to 1B. The results show that IEX-ZMF requires 7.58 hours to process 34M keyword/id pairs which is approximately $797\mu s$ per pair. We extrapolate that 1B keyword/id pairs would require 217 hours. This represents a 1 order of magnitude slowdown over IEX-2Lev and $40\times$ slower than OXT-2Lev as reported in [16].

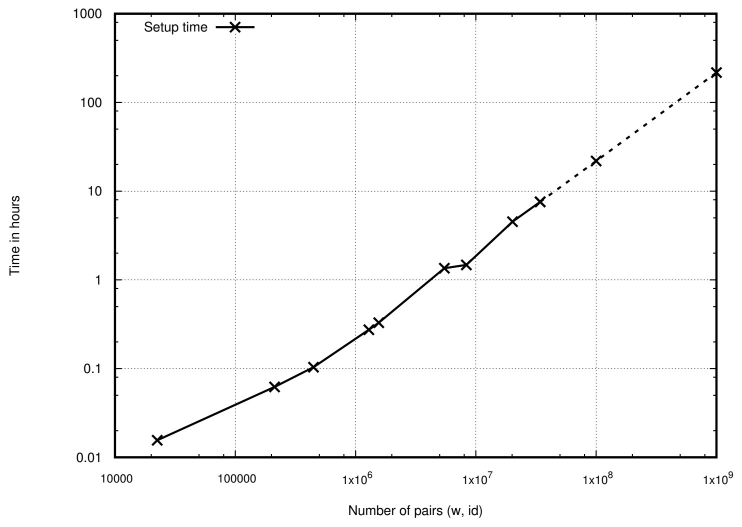

Figure 12: IEX-ZMF.Setup

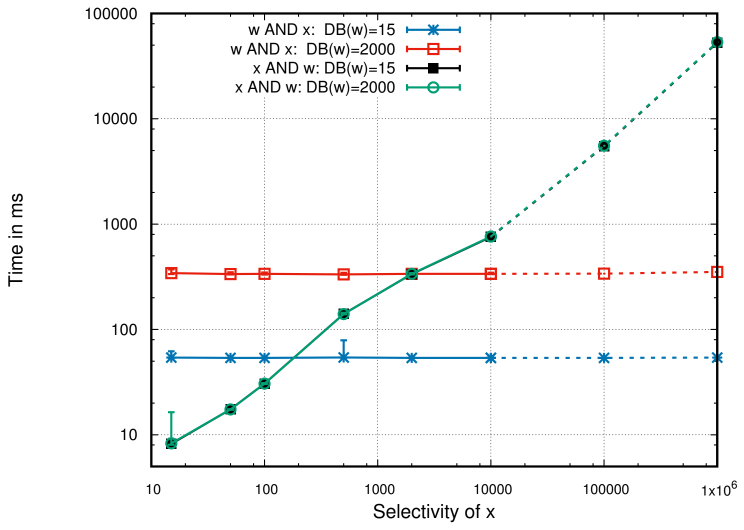

Figure 13: IEX-ZMF conjunctive keyword search

Conjunctive search time. We evaluated IEX-ZMF on the same conjunctive queries as IEX-2Lev. The results are in Fig. 13 and show that IEX-ZMF is slower than IEX-2Lev, as expected. For example, for $(w \wedge x)$ with $\mathsf{DB}(w) = 15$, IEX-ZMF takes around 54ms whereas IEX-2Lev only takes 1.5ms; a 30$\times$ slowdown. Similarly, for $(w \wedge x)$ with $\mathsf{DB}(w) = 2\text{K}$, IEX-ZMF takes 336ms whereas IEX-2Lev takes, e.g., 23ms when $\mathsf{DB}(x) = 10\text{K}$.

Disjunctive search time. Our disjunctive search experiments, which are the same as for IEX-2Lev, are summarized in Fig. 14. Here, search time is constant, requiring around 59.5 and 322ms for $\mathsf{DB}(w) = 15$ and $\mathsf{DB}(w) = 2\text{K}$, respectively. In both cases, IEX-ZMF is considerably slower than IEX-2Lev which requires 8.4 and 14.8ms for $\mathsf{DB}(w) = 15$ and $\mathsf{DB}(w) = 2\text{K}$ even for

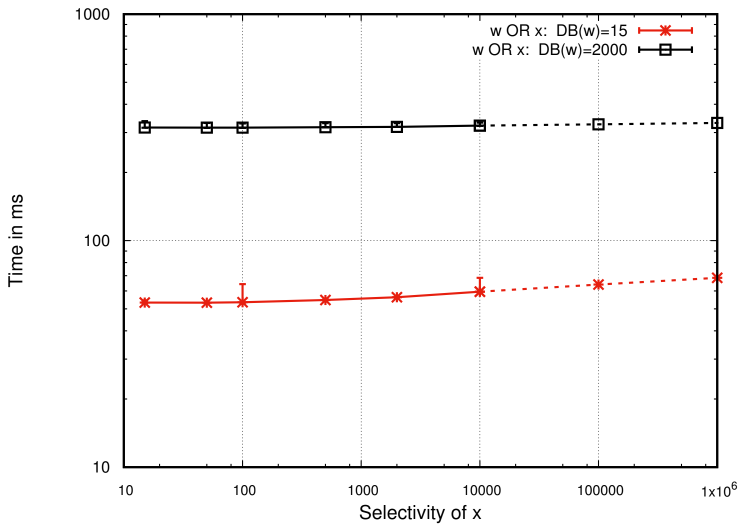

Figure 14: IEX-ZMF disjunctive keyword search

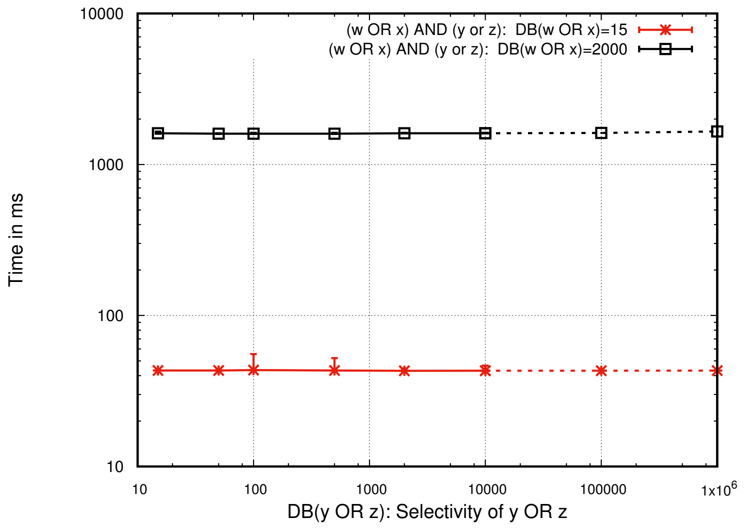

Figure 15: IEX-ZMF boolean keyword search

$\mathsf{DB}(x) = 10\text{K}$.

Boolean search time. The results of our boolean search experiments are given in Fig. 15 and again show that, as expected, IEX-ZMF is slower than IEX-2Lev. While its search time is constant, it requires about 59.5ms and 1610ms, respectively, for the two boolean queries, whereas IEX-2Lev requires only 2.4ms and 23.7ms even when $\mathsf{DB}(y \vee z) = 10\text{K}$.

Token size. The token sizes of IEX-ZMF are reported in Fig. 16. They are considerably larger than the tokens of IEX-2Lev. In fact, whereas the former produces tokens of size 1.7MB and 25MB for q_1^{12} and q_2^{12}, IEX-2Lev tokens are only of size 5.6KB and 81KB for the same queries. This is a consequence of two reasons: first, we select a false positive rate equal to 20, this implies that we need

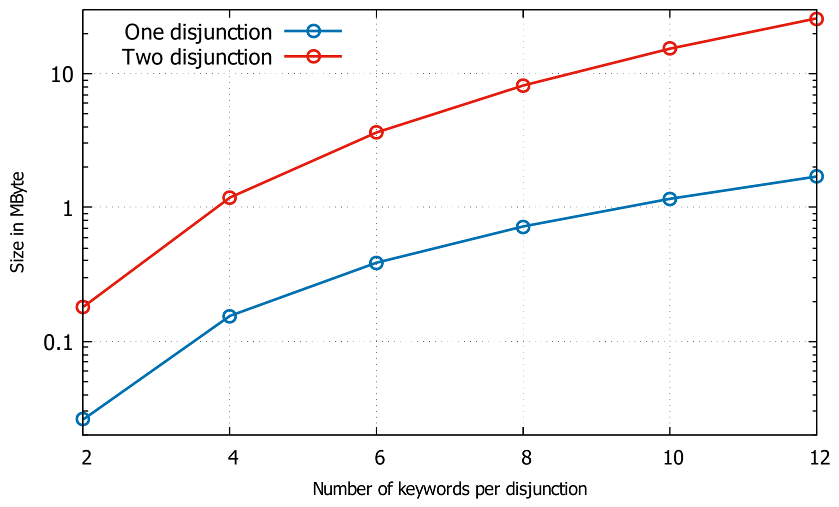

Figure 16: IEX-ZMF token Size

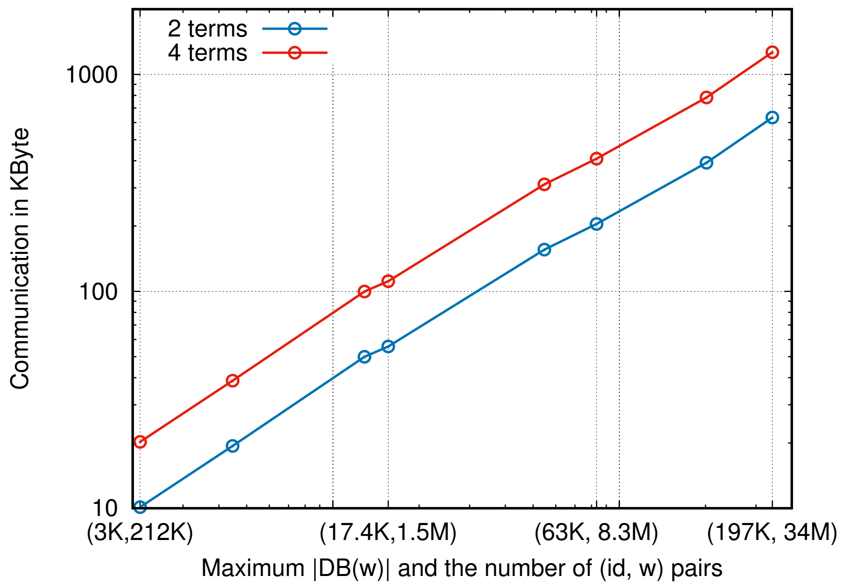

Figure 17: IEX-ZMF and IEX-2Lev communication upper bounds with p=0.2

20× larger local multi-map token compared to IEX-2Lev. Second, as we consider the upper-bound when generating the OC tokens, this also makes our tokens much larger.

Communication overhead. The filtering parameter reduces the storage overhead at the cost of loosing communication optimality. We set the filtering parameter to 0.2. The worst case communication complexity for a k-terms disjunctive queries (or any arbitrary boolean query in the CNF form with a k-terms first disjunction) equals: $O(k \cdot p \cdot \max_{w \in W} |\mathsf{DB}(w)|)$. The communication upper bound for $k = 2, 4$ is reported in Fig. 17. The communication upper bound for a number of pairs equal to 34M is 632KB as $\max_{w \in W} |\mathsf{DB}(w)| = 197,712$.

References

[1] Alchemyvision api. https://www.ibm.com/watson/developercloud/alchemyvision/api/v1/#introduction.

[2] Apache lucene. http://lucene.apache.org.

[3] Bouncy castle. http://www.bouncycastle.org.

[4] Cloc. http://www.cloc.sourceforge.net.

[5] Clusion. https://github.com/orochi89/Clusion.

[6] Powered by hadoop. See http://wiki.apache.org/hadoop/PoweredBy.

[7] Sizeof. http://http://sizeof.sourceforge.net.

[8] E. Andreeva, A. Bogdanov, A. Luykx, B. Mennink, E. Tischhauser, and K. Yasuda. Parallelizable and authenticated online ciphers. In *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part I*, pages 424–443, 2013.

[9] 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.

[10] M. Bellare, A. Boldyreva, L. R. Knudsen, and C. Namprempre. On-line ciphers and the hash-cbc constructions. *IACR Cryptology ePrint Archive*, 2007:197, 2007.

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

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

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

[14] D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. In *Theory of Cryptography Conference (TCC '11)*, volume 6597 of *Lecture Notes in Computer Science*, pages 253–273. Springer, 2011.

[15] R. Bost. Sophos - forward secure searchable encryption. In *ACM Conference on Computer and Communications Security (CCS '16)*, 20016.

[16] 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.

[17] 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.

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

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

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

[21] 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.

[22] S. Faber, S. Jarecki, H. Krawczyk, Q. Nguyen, M. Rosu, and M. Steiner. Rich queries on encrypted data: Beyond exact matches. In *Computer Security - ESORICS 2015 - 20th European Symposium on Research in Computer Security, Vienna, Austria, September 21-25, 2015, Proceedings, Part II*, pages 123–145, 2015.

[23] B. A. Fisch, B. Vo, F. Krell, A. Kumarasubramanian, V. Kolesnikov, T. Malkin, and S. M. Bellovin. Malicious-client security in blind seer: A scalable private dbms. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 395–410. IEEE, 2015.

[24] C. Gentry. Fully homomorphic encryption using ideal lattices. In *ACM Symposium on Theory of Computing (STOC '09)*, pages 169–178. ACM Press, 2009.

[25] E.-J. Goh. Secure indexes. Technical Report 2003/216, IACR ePrint Cryptography Archive, 2003. See http://eprint.iacr.org/2003/216.

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

[27] Y. Ishai, E. Kushilevitz, S. Lu, and R. Ostrovsky. Private large-scale databases with distributed searchable symmetric encryption. In *Topics in Cryptology - CT-RSA 2016 - The Cryptographers' Track at the RSA Conference 2016, San Francisco, CA, USA, February 29 - March 4, 2016, Proceedings*, pages 90–107.

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

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

[30] K. Kurosawa. Garbled searchable symmetric encryption. In *Financial Cryptography and Data Security (FC '14)*, 2014.

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

[32] X. Meng, S. Kamara, K. Nissim, and G. Kollios. GRECS: graph encryption for approximate shortest distance queries. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, pages 504–517, 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] A. O'Neill. Definitional issues in functional encryption, 2010. Cryptology ePrint Archive, Report 2010/556.

[35] 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.

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

[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] A. Yao. Protocols for secure computations. In *IEEE Symposium on Foundations of Computer Science (FOCS '82)*, pages 160–164. IEEE Computer Society, 1982.

[39] 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 Multi-Structure Structured Encryption

Multi-structure STE schemes are less restricted than standard schemes in the sense that their tokens can be used with more than a single encrypted structure (generated under the same key). The syntax and security definitions of multi-structure STE schemes is given below.

Definition A.1 (Multi-structure STE)**.** *Let $\mathscr{T}$ be an abstract data type. A multi-structure structured encryption scheme $\Sigma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Token}, \mathsf{Query})$ for $\mathscr{T}$ consists of four polynomial-time algorithms that work as follows:*

- $K \leftarrow \mathsf{Gen}(1^k)$*: is a probabilistic algorithm that takes as input a security parameter 1^k and outputs a key K.*
- $\mathsf{EDS} \leftarrow \mathsf{Enc}(K, \mathsf{DS})$*: is a probabilistic algorithm that takes as input a key K and a structure* DS *of type $\mathscr{T}$ and outputs 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 (possibly) probabilistic algorithm that takes as input an encrypted structure* EDS *and a token* tk *and outputs either $\perp$ or a response r.*

We say that a multi-structure STE scheme Σ is correct if for all $k \in \mathbb{N}$, for all K output by $\mathsf{Gen}(1^k)$*, for all* DS *of type $\mathscr{T}$, for all* EDS *output by* $\mathsf{Enc}(K, \mathsf{DS})$*, and for all sequences of $m = \mathsf{poly}(k)$ queries $q_1, \ldots, q_m \in \mathbf{Q}_{\mathsf{DS}}$, for all tokens tk_i output by* $\mathsf{Token}(K, q_i)$*,* $\mathsf{Query}(\mathsf{EDS}, \mathsf{tk}_i)$ *returns the correct response with all but negligible probability.*

Ideally, we would like multi-structure STE schemes to provide the same notion of security as standard schemes as formalized in Definition 8.2. Intuitively speaking, this is the guarantee that an encrypted structure EDS leaks nothing beyond a well-specified leakage function $\mathcal{L}_{\mathsf{S}}(\mathsf{DS})$ of the plaintext structure and that the encrypted structure EDS and a token tk for some query q leaks nothing beyond a well-specified leakage function $\mathcal{L}_{\mathsf{Q}}(\mathsf{DS}, q)$ of the plaintext structure and query. Of course, we want to this to hold even against adaptive adversaries.

For multi-structure schemes, however, this intuition needs to be augmented to capture the fact that the tokens are used with *all* the structures generated under some key K. In particular, this means that the scheme should guarantee that, given a token for a query q, nothing is leaked beyond a well-specified leakage function $\mathcal{L}_{\mathsf{Q}}(\mathsf{DS}_1, \ldots, \mathsf{DS}_n, q)$, where $\mathsf{DS}_1, \ldots, \mathsf{DS}_n$ are the structures encrypted under the key K.

Definition A.2 (Multi-structure adaptive semantic security)**.** *Let $\Sigma_{\mathscr{T}} = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Query})$ be a multi-structure STE scheme for type $\mathscr{T}$ 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$_{\Sigma,\mathcal{A}}(k)$*: the challenger generates a key $K \leftarrow \mathsf{Gen}(1^k)$. Given z the adversary $\mathcal{A}$* adaptively *sends polynomially-many structures $\mathsf{DS}_1, \ldots, \mathsf{DS}_n$ to the challenger who returns their encryptions $\mathsf{EDS}_1, \ldots, \mathsf{EDS}_n$. The adversary adaptively sends polynomially-many queries $q_1, \ldots, q_m$ and receives tokens $\mathsf{tk}_1, \ldots, \mathsf{tk}_m$. Finally, $\mathcal{A}$ outputs a bit b that is output by the experiment.*

Ideal$_{\Sigma,\mathcal{A},\mathcal{S}}(k)$*: given z the adversary $\mathcal{A}$ adaptively sends polynomially-many structures $\mathsf{DS}_1, \ldots, \mathsf{DS}_n$ to the challenger. For all $1 \leq i \leq n$, given leakage $\mathcal{L}_{\mathsf{S}}(\mathsf{DS}_i)$, the simulator returns EDS_i to the adversary. The adversary adaptively sends polynomially-many queries $q_1, \ldots, q_m$ to the challenger. For all $1 \leq i \leq m$, given leakage $\mathcal{L}_{\mathsf{Q}}(\mathsf{DS}_1, \ldots, \mathsf{DS}_n, q_i)$, the simulator returns tk_i to the adversary. Finally, $\mathcal{A}$ outputs a bit b that is output by the experiment.*

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

$\mathcal{S}$ *such that for all* $z \in \{0,1\}^*$, *the following expression is negligible in* k*:*

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

B Proof of Theorem 5.1

Theorem 5.1. *If* Σ_{DX} *is adaptively* $(\mathcal{L}^{\mathsf{dx}}_{\mathsf{S}}, \mathcal{L}^{\mathsf{dx}}_{\mathsf{Q}})$*-secure,* Σ_{MM} *is adaptively* $(\mathcal{L}^{\mathsf{mm}}_{\mathsf{S}}, \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}})$*-secure,* SKE *is RCPA-secure and* F *is pseudo-random, then* IEX *is* $(\mathcal{L}^{\mathsf{iex}}_{\mathsf{S}}, \mathcal{L}^{\mathsf{iex}}_{\mathsf{Q}})$*-secure.*

Proof. Let $\mathcal{S}_{\mathsf{DX}}$ and $\mathcal{S}_{\mathsf{MM}}$ be the simulators guaranteed to exist from the adaptive semantic security of Σ_{DX} and Σ_{MM}. Consider the simulator $\mathcal{S}$ for IEX that works as follows. To simulate EDB, it takes as input the setup leakage

$$\mathcal{L}^{\mathsf{iex}}_{\mathsf{S}} = \Big(\mathcal{L}^{\mathsf{dx}}_{\mathsf{S}}(\mathsf{DX}), \mathcal{L}^{\mathsf{mm}}_{\mathsf{S}}(\mathsf{MM}_g)\Big),$$

computes $\mathsf{EDX} \leftarrow \mathcal{S}_{\mathsf{DX}}(\mathcal{L}^{\mathsf{dx}}_{\mathsf{S}}(\mathsf{DX}))$, $\mathsf{EMM}_g \leftarrow \mathcal{S}_{\mathsf{MM}}(\mathcal{L}_{\mathsf{S}}(\mathsf{MM}_g))$ and outputs $\mathsf{EDB} = (\mathsf{EDX}, \mathsf{EMM}_g)$.

To simulate a token, it takes as input the query leakage

$$\begin{aligned}
&\mathcal{L}^{\mathsf{iex}}_{\mathsf{Q}}(\mathsf{EDB}, \mathbf{w}) = \\
&\quad\Bigg(\bigg(\mathcal{L}^{\mathsf{dx}}_{\mathsf{Q}}(\mathsf{DX}, w_i), \mathcal{L}^{\mathsf{mm}}_{\mathsf{S}}(\mathsf{MM}_i), \\
&\qquad\quad \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}(\mathsf{MM}_g, w_i), \\
&\qquad\quad \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}(\mathsf{MM}_i, w_{i+1}), \\
&\qquad\quad \ldots, \\
&\qquad\quad \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}(\mathsf{MM}_i, w_q), \mathsf{TagPat_i}(\mathsf{EDB}, \mathbf{w}), \bigg)_{i \in [q-1]}, \\
&\qquad\quad \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}(\mathsf{MM}_g, w_q), \mathsf{TagPat_q}(\mathsf{EDB}, \mathbf{w})\Bigg),
\end{aligned}$$

and constructs a token $\mathsf{tk} = (\mathbf{tk}_1, \ldots, \mathbf{tk}_{q-1}, \mathsf{tk}_q)$ as follows. For all $i \in [q-1]$, it sets

$$\mathbf{tk}_i = \big(\mathsf{dtk}_i, \mathsf{gtk}_i, \mathsf{ltk}_{i+1}, \ldots, \mathsf{ltk}_q\big).$$

Here dtk_i is simulated as $\mathsf{dtk}_i \leftarrow \mathcal{S}_{\mathsf{DX}}(\mathcal{L}^{\mathsf{DX}}_{\mathsf{Q}}(\mathsf{DX}, w_i), \mathsf{EMM}_i)$, where $\mathsf{EMM}_i \leftarrow \mathcal{S}_{\mathsf{MM}}(\mathcal{L}^{\mathsf{mm}}_{\mathsf{S}}(\mathsf{MM}_i))$. gtk_i is simulated as $\mathsf{gtk}_i \leftarrow \mathcal{S}_{\mathsf{MM}}(\mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}(\mathsf{MM}_g, w_i), (\mathsf{tag}_{\mathsf{id}})_{\mathsf{id} \in \mathsf{DB}(w_i)})$. For all $i+1 \leq j \leq q$, ltk_j is simulated as

$$\mathsf{ltk}_j \leftarrow \mathcal{S}_{\mathsf{MM}}\Big(\mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}(\mathsf{MM}_i, w_j), (\mathsf{tag}_{\mathsf{id}})_{\mathsf{id} \in \mathsf{DB}(w_i) \cap \mathsf{DB}(w_j)}\Big).$$

It remains to show that for all probabilistic polynomial-time adversaries $\mathcal{A}$, the probability that $\mathbf{Real}(k)$ outputs 1 is negligibly-close to the probability that $\mathbf{Ideal}(k)$ outputs 1. This can be shown with a standard sequence of games argument that shows that the simulated EDB and tk are indistinguishable from the real ones due to the adaptive security of Σ_{MM} and Σ_{DX}, the RCPA-security of SKE and the pseudo-randomness of F. In particular, the last two properties are used to show that the random tags from the leakage are indistinguishable from the encrypted identifiers in the $\mathbf{Real}(k)$ experiment.

■

C Proof of Theorem 7.1

Theorem 7.1 *If Σ_{SET} is adaptively $(\mathcal{L}_{\mathsf{S}}^{\mathsf{set}}, \mathcal{L}_{\mathsf{Q}}^{\mathsf{set}})$-secure then the scheme Σ_{MM} that results from applying the* Z-IDX *transformation to it is adaptively $(\mathcal{L}_{\mathsf{S}}^{\mathsf{mm}}, \mathcal{L}_{\mathsf{Q}}^{\mathsf{mm}})$-secure.*

Proof. Let $\mathcal{S}_{\mathsf{SET}}$ be the simulator guaranteed to exist by the adaptive (multi-structure) security of Σ_{SET} and consider the simulator $\mathcal{S}$ for Σ_{MM} that works as follows. To simulate EMM, it takes as input the setup leakage

$$\mathcal{L}_{\mathsf{S}}^{\mathsf{mm}}(\mathsf{MM}) = \left(\left(\mathcal{L}_{\mathsf{S}}^{\mathsf{set}}(\mathsf{MM}^{-1}[v]) \right)_{v \in \mathbf{V}}, \#\mathbf{V} \right)$$

and simulates $\#\mathbf{V}$ encrypted sets as follows. For $1 \leq i \leq \#\mathbf{V}$, it computes

$$\mathsf{ESET}_i \leftarrow \mathcal{S}_{\mathsf{SET}}(\mathcal{L}_i),$$

where $\mathcal{L}_i$ is the ith element in $\mathcal{L}_{\mathsf{S}}^{\mathsf{mm}}(\mathsf{MM})$. It then outputs $\mathsf{EMM} = (\mathsf{ESET}_1, \ldots, \mathsf{ESET}_{\#\mathbf{V}})$.

To simulate a token, $\mathcal{S}$ takes as input the query leakage

$$\mathcal{L}_{\mathsf{Q}}^{\mathsf{mm}}(\mathsf{MM}, q) = \mathcal{L}_{\mathsf{Q}}^{\mathsf{set}} \left(\left(\mathsf{MM}^{-1}[v] \right)_{v \in \mathbf{V}}, q \right)$$

and computes

$$\mathsf{tk} \leftarrow \mathcal{S}_{\mathsf{SET}} \left(\mathcal{L}_{\mathsf{Q}}^{\mathsf{set}} \left(\left(\mathsf{MM}^{-1}[v] \right)_{v \in \mathbf{V}}, q \right) \right).$$

It remains to show that for all probabilistic polynomial-time adversaries $\mathcal{A}$, the probability that $\mathbf{Real}(k)$ outputs 1 is negligibly-close to the probability that $\mathbf{Ideal}(k)$ outputs 1. This can be done using a standard sequence of games argument that shows that the simulated EMM and tk are indistinguishable from the real ones due to the adaptive (multi-structure) security of Σ_{SET}.

■

D Proof of Theorem 7.2

Theorem 7.2 *If* OC *is secure, the multi-structure set encryption scheme described in Fig. 4 is adaptively $(\mathcal{L}_{\mathsf{S}}^{\mathsf{set}}, \mathcal{L}_{\mathsf{Q}}^{\mathsf{set}})$-secure in the random oracle model.*

Proof. Here, we describe in detail a simulator for the proof and defer its full analysis to an ulterior version of this work. Consider the simulator $\mathcal{S}$ that works as follows. To simulate an encrypted set ESET_i, for $i \in [n]$, it takes as input the setup leakage $\mathcal{L}_{\mathsf{S}}^{\mathsf{set}}(S_i) = \#S_i$ and initializes a binary array A_i of size $m = \lceil \log(1/\varepsilon) \cdot \#S_i / \ln 2 \rceil$, where ε is the false positive rate. For all $1 \leq j \leq m$, it then sets $\mathsf{A}_i[j] \xleftarrow{\$} \{0,1\}$. Finally, it outputs $\mathsf{ESET}_i = \mathsf{A}_i$. For clarity, we assume that each A_i has size 2^i.

To simulate a token, $\mathcal{S}$ takes as input the query leakage

$$\mathcal{L}_{\mathsf{Q}}^{\mathsf{set}} \Big(S_1, \ldots, S_n, q \Big) = \Big(b_1, \ldots, b_n, \mathsf{SP}(q) \Big),$$

where, for all $1 \leq i \leq n$, b_i is 1 if $q \in S_i$ and 0 otherwise, and where $\mathsf{SP}(q)$ is the search pattern as defined in Section 7.1.

Note that in the $\mathbf{Real}^m(k)$ experiment the adversary learns the size of the encrypted Bloom filters and that from this it can infer information about the number of elements it contains and, therefore, about its number of 1s and 0s. Indeed, it can be shown by a standard argument that, if X_i^0 is the random variable that counts the number of 0s in A_i, $E[X_i^0] = m \cdot (1 - 1/m)^{\#S_i \cdot \log(1/\varepsilon)}$. Moreover, it can be shown that X_i^0 is concentrated around its expected value such that for all $i \in [n]$ and all $\lambda > 0$

$$\Pr[|X_i^0 - E[X_i^0]| \geq \lambda] \leq 2e^{-\frac{2\lambda^2}{m}}.$$

In other words, the adversary has some knowledge about the plaintext distribution of 1s and 0s in the encrypted sets that the simulator has to respect throughout its simulation. In the following, we denote by $\#S_1^0, \cdots, \#S_n^0$ the expected number of 0s associated to $\mathsf{ESET}_1, \cdots, \mathsf{ESET}_n$.

We first provide a high level description of the simulator before getting into the details. The difficult part of simulating multiple encrypted matryoshka filters is to make sure that the number of 0s and 1s will *always* verify the publicly available bound detailed above, *for every* possible sequence of adaptive queries. Throughout the query simulation, the simulator has therefore to construct a set of *location sets* that will help it to keep track of all locations that have been unmasked. The main difficulty of this process is the inherent dependence between the nested matryoshka filters. When a position gets unmasked in the smallest filter, determining the positions in the largest filters becomes dependent and cannot anymore chosen arbitrary. This is a consequence of using the online cipher property. For example, if an element exists in more than one filter, the unmasked positions in the smallest filter have to verify the following properties: (1) there are available positions that can be set to 1 in larger filters containing the element, and (2) the number of positions opened to be 1 has to verify the publicly known bounds of the Bloom Filters detailed above. In the following, we keep track of the opened positions by creating three sets Loc_i, Zeroes_i and Ones_i for all $i \in n$. In this simulation, we have to program the random oracles that we use in the construction. For this, we use $\lambda + 1$ dictionaries RDX and $\mathsf{RDX}^1, \cdots, \mathsf{RDX}^\lambda$ respectively for the hash used in the masking phase, and λ used as the hash functions of the Bloom filters. Finally, we use two dictionaries DX_1 and DX_2 to simulate the behavior of a random function and an online permutation, respectively. We provide the details of the simulation below:

First, $\mathcal{S}$ it initializes three dictionaries DX_1, DX_2 and RDX, λ dictionaries $\mathsf{RDX}^1, \cdots, \mathsf{RDX}^\lambda$, n empty sets Ones_i and Zeroes_i, and one empty set R. To simulate the token, we can have two cases for all $S \in (S_1, \ldots, S_n)$ such that:

- (first case) If q is a repeated query, then set $T = \mathsf{DX}_1[q]$ and for all $j \in [\lambda]$, compute $s_j = \mathsf{DX}_2[\mathsf{RDX}^j[T]]$
- (second case) otherwise, set $T \overset{\$}{\leftarrow} \{0,1\}^k \setminus R$. Update the set R and dictionary DX_1 such that $R = R \cup \{T\}$ and $\mathsf{DX}_1[q] = T$
- for all $j \in [\lambda]$
    1. initialize n sets Loc_i for all $i \in [n]$. For $i \in [n]$, if $b_i = 0$, then set $\mathsf{Loc}_i = \{1, \cdots, 2^i\}$, otherwise if $b_i = 1$ and $\#\mathsf{Ones}_i < (2^i - \#S_i^0)$, then set $\mathsf{Loc}_i = [2^i] \setminus \mathsf{Zeroes}_i$, otherwise if $b_i = 1$ and $\#\mathsf{Ones}_i = (2^i - \#S_i^0)$, then set $\mathsf{Loc}_i = \mathsf{Ones}_i$
    2. set γ to be the smallest integer in $[n]$ such that $b_\gamma = 1$, for all $i \in [n]$, compute $\mathsf{Loc}_\gamma = \bigcap_{\substack{i \in [n] \\ b_i = 1}} \mathsf{Loc}_i^\gamma$, where Loc_i^γ contains all elements in Loc_i truncated at position γ.
    3. generate the position $pos_\gamma \overset{\$}{\leftarrow} \mathsf{Loc}_\gamma$. For all $i < \gamma$, set $pos_i = (pos_\gamma)_{|i}$. For all $i > \gamma$, if $b_i = 0$, then set $pos_i = pos_{i-1} \| b$ where $b \overset{\$}{\leftarrow} \{0,1\}$, otherwise if $b_i = 1$, then set $pos_i = pos_{i-1} \| b$, where $b \overset{\$}{\leftarrow} \{0,1\}$, if $pos_i \notin \mathsf{Loc}_i$, then flip the bit b such that $pos_i = pos_{i-1} \| (1 - b)$

4. set $\mathsf{RDX}^j[T] = pos_n$. For all $i \in [n]$, if $b_i = 1$, then add pos_i to Ones_i, otherwise if $b_i = 0$ and $pos_i \notin \mathsf{Ones}_i \cup \mathsf{Zeroes}_i$, then toss a coin distributed following the Bernoulli distribution with parameter $p = \#\mathsf{Ones}_i/2^i$, if the coin toss equals a head then add pos_i to Ones_i, and to Zeroes_i otherwise.
5. compute $s = \mathsf{GCP}(pos_n, \mathsf{DX}_2.\mathsf{labels})$, where GCP denotes the greatest common prefix between the string pos_n and all strings in $\mathsf{DX}_2.\mathsf{labels}$. $\mathcal{S}$ computes $\kappa = s \| r$, where $r \xleftarrow{\$} \{0,1\}^{\log m \times B - |s|}$.
6. finally, $\mathcal{S}$ updates the two dictionaries DX_2 and RDX such that $\mathsf{DX}_2[pos_n] = \kappa$ and $\mathsf{RDX}[\kappa \| i] = 1 \oplus \mathsf{A}[pos_n]$ if $b_i = 1$ and $\mathsf{RDX}[\kappa \| i] = \mathsf{A}[pos_n]$ otherwise.

For programming the random oracle R, H_i, for $i \in [\lambda]$, we perform the following:

$\mathsf{R}(s)$: if $s \in \mathsf{RDX}.\mathsf{labels}$, then $\mathsf{R}[s] = \mathsf{RDX}[s]$. Otherwise, compute $r \xleftarrow{\$} \{0,1\}$, set $\mathsf{R}[s] = r$, and set $\mathsf{RDX}[s] = r$.

$\mathsf{H}_i(s)$: if $s \in \mathsf{RDX}^i.\mathsf{labels}$, then $\mathsf{H}_i[s] = \mathsf{RDX}^i[s]$. Otherwise, compute $r \xleftarrow{\$} \{0,1\}^k$, set $\mathsf{H}_i[s] = r$, and set $\mathsf{RDX}^i[s] = r$.

Now, we have to show that for PPT adversaries $\mathcal{A}$, the output of the real experiment $\mathbf{Real}_{\Sigma_{\mathsf{ESET}},\mathcal{A}}(k)$ and ideal experiment $\mathbf{Ideal}_{\Sigma_{\mathsf{ESET}},\mathcal{A},\mathcal{S}}(k)$ are indistinguishable. For this, we define one reduction defined in the following sequence of games.

- Game_0: is the same as the real experiment.
- Game_1: is the same as Game_0 except that we do not generate the key K_1 and the PRF calls in line 2.(a) are replaced by a random function f generated uniformly at random from $\{\{0,1\}^k \leftarrow [\sigma]\}$
- Game_2: is exactly the same as Game_1 except that we instantiate the random function f as follows. First, define R, an empty set and, DX_1 an empty dictionary. Fill it as follows, for all $i \in [\sum_i^n \#S_i]$, compute $r_i \leftarrow \{0,1\}^k$ and then set $R \leftarrow r_i$. Then, whenever, there is a call to the random function evaluation of an element $a \in S_i$ for $i \in [n]$, the call is either replaced by randomly sampling an element T from R if a was not queried before, or by $\mathsf{DX}_1[a]$ otherwise. We then update R such that $R = R \setminus \{T\}$ and $\mathsf{DX}_1[a] = T$.
- Game_3: is exactly the same as Game_2 except that we program the random oracle H_i, for all $i \in [\lambda]$, as follows: instantiate two empty sets Ones_i and Zeroes_i, for $i \in [n]$. Instantiate λ empty dictionaries RDX^1 to RDX^λ. For all $i \in [n]$ and $j \in [\lambda]$, if $a \in S_i$, determine $\gamma \leq i$ such that $a \in S_\gamma$ and $a \notin S_l$ for all $\gamma + 1 \leq l \leq i-1$, otherwise set $\gamma = \perp$. There are three cases:
  - if $\gamma = \perp$ and $\#\mathsf{Ones}_i < 2^i - \#S_i^0$, then set $pos \xleftarrow{\$} \{1, \cdots, 2^i\} \setminus \mathsf{Zeroes}_i$, update $\mathsf{Ones}_i \leftarrow \{pos\}$ and then set $\mathsf{RDX}^j[\mathsf{DX}_1[a]] = pos$,
  - if $\gamma = \perp$ and $\#\mathsf{Ones}_i = 2^i - \#S_i^0$, then set $pos \xleftarrow{\$} \mathsf{Ones}_i$ and then update $\mathsf{RDX}^j[\mathsf{DX}_1[a]] = pos$,
  - otherwise set $pos = \mathsf{RDX}^j[\mathsf{DX}_1[a]]$, update $\mathsf{RDX}^j[\mathsf{DX}_1[a]] = pos \| r_1 \| r_2 \| \cdots \| r_{i-\gamma}$ where $r_l \xleftarrow{\$} \{0,1\}$, then for all $l \in \{\gamma, \cdots, i\}$, toss a coin distributed following Bernoulli distribution with parameter $p = \#\mathsf{Ones}_i/2^i$,
    - if it is head add $pos \| r_1 \| \cdots \| r_l$ to $\mathsf{Ones}_{\gamma+l}$,
    - and to $\mathsf{Zeroes}_{\gamma+l}$ otherwise.
- Game_4: is exactly the same as Game_3 except that we do not generate the key K_2, and the online cipher OC evaluations in line 2.(c) are replaced by a valid random online permutation in $\mathsf{Operm}_{\log m, B}$.

- Game_5: is exactly the same as Game_4 except that we implement the random online permutation from $\mathsf{Operm}_{\log m,B}$ as follows. First, instantiate an empty dictionary DX_2. Second, we verify whether there is an online permutation that was previously generated and that has a common prefix with the computed position such that: if for $j \in [\lambda]$, $pos_j = \mathsf{RDX}^j[T]$, then the common prefix s equals to $s = \mathsf{GCP}(pos_j, \mathsf{DX}_2.\mathsf{labels})$. Finally, to generate the online permutation, we pad s with random bits to reach the required bit length $\log m \times B$, and finally update $\mathsf{DX}_2[pos_j] = s$.
- Game_6: is exactly the same as Game_5 except that the evaluations of the random oracles R, H_1, ..., H_λ are replaced by a random bits sampled uniformly at random.

■