Rich Queries on Encrypted Data: Beyond Exact Matches*

Sky Faber** Stanislaw Jarecki*** Hugo Krawczyk†
Quan Nguyen‡ Marcel Rosu§ Michael Steiner¶

Abstract. We extend the searchable symmetric encryption (SSE) protocol of [Cash et al., Crypto'13] adding support for range, substring, wildcard, and phrase queries, in addition to the Boolean queries supported in the original protocol. Our techniques apply to the basic single-client scenario underlying the common SSE setting as well as to the more complex Multi-Client and Outsourced Symmetric PIR extensions of [Jarecki et al., CCS'13]. We provide performance information based on our prototype implementation, showing the practicality and scalability of our techniques to very large databases, thus extending the performance results of [Cash et al., NDSS'14] to these rich and comprehensive query types.

1 Introduction

Searchable symmetric encryption (SSE) addresses a setting where a client outsources an encrypted database (or document/file collection) to a remote server $\mathcal{E}$ such that the client, which only stores a cryptographic key, can later search the collection at $\mathcal{E}$ while hiding information about the database and queries from $\mathcal{E}$. Leakage to $\mathcal{E}$ is to be confined to well-defined forms of data-access and query patterns while preventing disclosure of explicit data and query plaintext values. SSE has been extensively studied [25,12,7,10,8,18,15,17,14,6,13,5,20,19], particularly in last years due to the popularity of clouds and data outsourcing, focusing almost exclusively on single-keyword search.

Recently, Cash et al. [6] and Pappas et al. [20] presented the first SSE solutions that go well beyond single-keyword search by supporting Boolean queries on multiple keywords in sublinear time. In particular, [6,5] build a very scalable system with demonstrated practical performance with databases containing indexes in the order of tens of billions document-keyword pairs. In this work we extend the search capabilities of the system from [6] (*referred to as the OXT protocol*) by supporting range queries (e.g., return all records of people born between two given dates), substring queries (e.g., return records with textual information containing a given pattern, say 'crypt'), wildcard queries (combining substrings with one or more single-character wildcards), and phrase queries (return records that contain the phrase "searchable encryption"). Moreover, by preserving the overall system design and optimized data structures of [5], we can run any of these new queries in combination with Boolean-search capabilities (e.g., combining a range and/or substring query with a conjunction of additional keywords/ranges/substrings) and we can do so while preserving the scalability of the system and additional properties such as support for *dynamic data*.

We also show how to extend our techniques to the more involved multi-client SSE scenarios studied by Jarecki et al. [13]. In the first scenario, denoted MC-SSE, the owner of the data, $\mathcal{D}$, outsources its data to a remote server $\mathcal{E}$ in encrypted form and later allows multiple clients to access the data via search queries and according to an authorization policy managed by $\mathcal{D}$. The system is intended to limit the information learned

* Preliminary version published at ESORICS 2015 [11]
** U. California Irvine. Email: fabers@uci.edu.
*** U. California Irvine. Email: stasio@ics.uci.edu.
† IBM Research. Email: hugo@ee.technion.ac.il.
‡ Google, Inc. Email: quannguyen@google.com.
§ Bloomberg. Email: marcelrosu@gmail.com.
¶ IBM Research. Email: steiner@acm.org.

by clients beyond the result sets returned by authorized queries while also limiting information leakage to server $\mathcal{E}$. A second scenario, OSPIR-SSE or just OSPIR (for Outsourced Symmetric PIR), addresses the multi-client setting but adds a requirement that $\mathcal{D}$ can authorize queries to clients following a given policy, but without $\mathcal{D}$ learning the specific values being queried. That is, $\mathcal{D}$ learns minimal information needed to enforce policy, e.g., the query type or the field to which the keyword belongs, say `last name`, but not the actual last name being searched.

We present our solution for range queries in Section 3, showing how to reduce any such query to a *disjunction of exact keywords*, hence leveraging the Boolean query capabilities of the OXT protocol and its remarkable performance. In the OSPIR setting, we show how $\mathcal{D}$ can authorize range queries based on the total size of the queried range without learning the actual endpoints of the range. This is useful for authorization policies that limit the size of a range as a way of preventing a client from obtaining a large fraction of the database. Thus, $\mathcal{D}$ may learn that a query on a data field spans 7 days but not *which* 7 days the query is about. Achieving privacy from both $\mathcal{D}$ and $\mathcal{E}$ while ensuring that the authorized search interval does not exceed a size limit enforced by $\mathcal{D}$, is challenging. We propose solutions based on the notion of *universal tree covers* for which we present different instantiations trading performance and security depending on the SSE model that is being addressed.

The other queries we support, i.e. substrings, wildcards and phrases, are all derived from a novel technique that allows us to search on the basis of positioning information (where the data and the position information are encrypted). This technique can be used to implement any query type that can be reduced to Boolean formulas on queries of the form "are two data elements at distance Δ?". For example, in the case of substring queries, the substring is tokenized (i.e., subdivided) into a sequence of possibly-overlapping k-grams (strings of k characters) and the search is performed as a conjunction of such k-grams. However, to avoid false positives, i.e., returning documents where the k-grams appear but not at the right distances from each other, we use the relative positions of the tokens to ensure that the combined k-grams represent the searched substring. Wildcard queries are processed similarly, because t consecutive wildcard positions (i.e., positions that can be occupied by any character) can be implemented by setting the distance between the two k-grams that bracket the string of t wildcards to $k + t$. Phrase queries are handled similarly, by storing whole words together with their encrypted positions in the text.

The crux of this technique is a homomorphic computation on encrypted position information that gives rise to a very efficient SSE protocol between client $\mathcal{C}$ and server $\mathcal{E}$ for computing relative distances between data elements while concealing this information from $\mathcal{E}$. This protocol meshes naturally with the homomorphic properties of OXT but in its general form it requires an additional round of interaction between client and server. In the SSE setting, the resulting protocol preserves most of the excellent performance of the OXT protocol (with the extra round incurring a moderate increase in query processing latency). For the OSPIR setting we resort to bilinear groups for some homomorphic operations, hence impacting performance in a more noticeable way which we are currently investigating.

We prove the security of our protocols in the SSE model of [10,8,6], and the extensions to the MC-SSE and OSPIR settings of [13], where security is defined in the real-vs-ideal model and is parametrized by a specified leakage function $\mathcal{L}(\mathsf{DB}, \mathbf{q})$. A protocol is said to be secure with leakage profile $\mathcal{L}(\mathsf{DB}, \mathbf{q})$ against adversary $\mathcal{A}$ if the actions of $\mathcal{A}$ on adversarially-chosen input DB and query set $\mathbf{q}$ can be simulated with access to the leakage information $\mathcal{L}(\mathsf{DB}, \mathbf{q})$ only (and not to DB or $\mathbf{q}$). This allows modeling and bounding the partial leakage incurred by SSE protocols. It means that even an adversary that has full information about the database and queries, or even chooses them at will, does not learn anything from the protocol execution other than what can be derived solely from the defined leakage profile. We achieve provable *adaptive* security against adversarial servers $\mathcal{E}$ and $\mathcal{D}$, and against malicious clients. Servers $\mathcal{E}$ and $\mathcal{D}$ are assumed to return correct results (e.g., server $\mathcal{E}$ returns all documents specified by the protocol) but can otherwise behave maliciously. However, in the OSPIR setting, query privacy from $\mathcal{D}$ is achieved as long as $\mathcal{D}$ does not collude with $\mathcal{E}$.

Practicality of our techniques was validated by a comprehensive implementation of: (i) the SSE protocols for range, substring and wildcard queries, and their combination with Boolean functions on exact keywords, and (ii) the OSPIR-SSE protocol for range queries. These implementations (extending those of [6,13,5]) were tested by an independent evaluator on DB's of varying size, up to 10 Terabytes with 100 million records and 25.6 billion record-keyword pairs. Performance was compared to MariaDB's (an open-source fork of MySQL) performance on the same databases running on *plaintext data and plaintext queries*. Due to the highly optimized protocols and careful I/O management, the performance of our protocols matched and often exceeded the performance of the plaintext system. These results are presented in Section 6.

Related Work. The only work we are aware of that addresses substring search on symmetrically encrypted data is the work of Chase and Shen [9]. Their method, based on suffix trees, is very different than ours and the leakage profiles seem incomparable. This is a promising direction, although the applicability to (sublinear) search on large databases, and the integration with other query types, needs to be investigated. Its potential generalization to the multi-client or OSPIR settings is another interesting open question. Range and Boolean queries are supported, also for the OSPIR setting, by Pappas et al. [20] (building on the work of Raykova et al [22]). Their design is similar to ours in reducing range queries to disjunctions (with similar data expansion cost) but their techniques are very different offering an alternative (and incomparable) leakage profile for the parties. The main advantages of our system are the support of the additional query types presented here and its scalability. The scalability of [20] is limited by their crucial reliance on Bloom filters that requires database sizes whose resultant Bloom filters can fit in RAM. A technique that has been suggested for resolving range queries in the SSE setting is *order-preserving encryption* (e.g., it is used in the CryptDB system [21]). However, it carries a significant intrinsic loss of privacy as the ordering of ciphertexts is visible to the holding server (and the encryption is deterministic). Range queries are supported in the multi-writer public key setting by Boneh-Waters [4] and Shi et al. [24] but at a significantly higher computational cost.

2 Preliminaries

Our work concerns itself with databases in a very general sense, including relational databases (with data arranged in "rows" and "columns"), document collections, textual data, etc. We use interchangeably the word 'document' and 'record'. We think of keywords as (attribute,value) pairs. The attribute can be structured data, such as name, age, SSN, etc., or it can refer to a textual field. We sometimes refer explicitly to the keyword's attribute but most of the time it remains implicit. We denote by m the number of distinct attributes and use $I(w)$ to denote the attribute of keyword w.

SSE protocols and formal setting (following [6]). Let τ be a security parameter. A database $\mathsf{DB} = (\mathsf{ind}_i, \mathsf{W}_i)_{i=1}^d$ is a list of identifier and keyword-set pairs, where $\mathsf{ind}_i \in \{0,1\}^\tau$ is a document identifier and $\mathsf{W}_i = \mathsf{DB}[\mathsf{ind}_i]$ is a list of its keywords. Let $\mathsf{W} = \bigcup_{i=1}^d \mathsf{W}_i$. A *query* ψ is a predicate on W_i where $\mathsf{DB}(\psi)$ is the set of identifiers of document that satisfy ψ. E.g. for a single-keyword query we have $\mathsf{DB}(w) = \{\mathsf{ind}$ s.t. $w \in \mathsf{DB}[\mathsf{ind}]\}$.

A *searchable symmetric encryption (SSE) scheme* Π consists of an algorithm Setup and a protocol Search fitting the following syntax. Setup takes as input a database DB and a list of document (or record) decryption keys RDK, and outputs a secret key K along with an encrypted database EDB. The search protocol Search proceeds between a *client* $\mathcal{C}$ and *server* $\mathcal{E}$, where $\mathcal{C}$ takes as input the secret key K and a query ψ and $\mathcal{E}$ takes as input EDB. At the end of the protocol, $\mathcal{C}$ outputs a set of $(\mathsf{ind}, \mathsf{rdk})$ pairs while $\mathcal{E}$ has no output. We say that an SSE scheme is *correct* for a family of queries Ψ if for all $\mathsf{DB}, \mathsf{RDK}$ and all queries $\psi \in \Psi$, for $(K, \mathsf{EDB}) \leftarrow \mathsf{Setup}(\mathsf{DB}, \mathsf{RDK})$, after running Search with client input (K, ψ) and server input EDB, the client outputs $\mathsf{DB}(\psi)$ and $\mathsf{RDK}[\mathsf{DB}(\psi)]$ where $\mathsf{RDK}[S]$ denotes $\{\mathsf{RDK}[\mathsf{ind}] \mid \mathsf{ind} \in S\}$. Correctness can be statistical (allowing a negligible probability of error) or computational (ensured only against computationally bounded attackers - see [6]).

Note (retrieval of matching encrypted records). Above we define the output of the SSE protocol as the set of identifiers ind pointing to the encrypted documents matching the query (together with the set of associated record decryption keys rdk). The retrieval of the document payloads, which can be done in a variety of ways, is thus decoupled from the storage and processing of the metadata which is the focus of the SSE protocols.

Multi-Client SSE Setting [13]. The MC-SSE formalism extends the SSE syntax by an algorithm $\mathsf{GenToken}$, which generates a search-enabling value token from the secret key K generated by the data owner $\mathcal{D}$ in Setup, and query ψ submitted by client $\mathcal{C}$. Protocol Search is then executed between server $\mathcal{E}$ and client $\mathcal{C}$ on resp. inputs EDB and token, and the protocol must assure that $\mathcal{C}$ outputs sets $\mathsf{DB}(\psi)$ and $\mathsf{RDK}[\mathsf{DB}(\psi)]$.

OSPIR SSE Setting [13]. An OSPIR-SSE scheme replaces the $\mathsf{GenToken}$ procedure, which in MC-SSE is executed by the data owner $\mathcal{D}$ on the cleartext client's query q, with a two-party protocol between $\mathcal{C}$ and $\mathcal{D}$ that allows $\mathcal{C}$ to compute the search-enabling token without $\mathcal{D}$ learning ψ. However, $\mathcal{D}$ should be able to enforce a query-authorization policy on $\mathcal{C}$'s query. We consider attribute-based policies, where queries are authorized based on the attributes associated to keywords in the query (e.g., a client may be authorized to run a range query on attribute 'age' but not on 'income', or perform a substring query on the 'address' field but not on the 'name' field, etc.). Later, we will consider extensions where the policy can define further constraints, e.g., the total size of an allowed interval in a range query, or the minimal size of a pattern in a substring query. An attribute-based policy for any query type is represented by a set of attribute-sequences P s.t. a query ψ involving keywords (or substrings, ranges, etc) $(w_1, ..., w_n)$ is *allowed by policy* P if and only if the sequence of attributes $\mathsf{av}(\psi) = (I(w_1), ..., I(w_n)) \in \mathsf{P}$. Using this notation, the goal of the $\mathsf{GenToken}$ protocol is to let $\mathcal{C}$ compute token corresponding to its query on ψ only if $\mathsf{av}(\bar{w}) \in \mathsf{P}$. Note that different query types will have different entries in P. Reflecting these goals, an OSPIR-SSE scheme is a tuple $\Sigma = (\mathsf{Setup}, \mathsf{GenToken}, \mathsf{Search})$ where Setup and Search are as in MC-SSE, but $\mathsf{GenToken}$ is a protocol run by $\mathcal{C}$ on input ψ and by $\mathcal{D}$ on input (P, K), with $\mathcal{C}$ outputting token if $\mathsf{av}(\psi) \in \mathsf{P}$, or $\perp$ otherwise, and $\mathcal{D}$ outputting $\mathsf{av}(\psi)$.

3 Range Queries

Our solution for performing range queries on encrypted data reduces these queries to a disjunction of exact keywords and therefore can be integrated with SSE solutions that support such disjunctions. In particular, we use this solution to add range query support to the OXT protocol from [6,13] while keeping all the other properties of OXT intact. This includes OXT's remarkable scalability, its support for different models (SSE, MC, OSPIR), and its boolean search capability. Thus, we obtain a protocol where range queries can be run in isolation or in combination with boolean expressions on other terms, including conjunctive ranges such as $30 \leq \text{AGE} \leq 39$ and $50{,}000 \leq \text{INCOME} \leq 99{,}999$.

Range queries can be applied to any ordered set of elements; our description focuses on integer ranges for simplicity. We denote range queries with input an interval $[a, b]$, for integers $a \leq b$, by $\mathsf{RQ}(a, b)$. We refer to a and b as the *endpoints* and to the number $b - a + 1$ as the *size of the range*. Inequality queries of the form $x \geq a$ are represented by the range $[a, b]$ where b is an upper bound on all applicable values for the searched attribute; queries of the form $x \leq b$ are handled similarly.

We now describe the extensions to the OXT protocol (and its OSPIR version) for supporting range queries. Thanks to our generic reduction of range queries to disjunctions of exact keywords, our range-query presentation does not require a detailed knowledge of the OXT protocol and basic familiarity with OXT suffices (the interested reader can find more details on OXT in the above papers and also in Section 4.1).

Pre-Processing (Setup). For concreteness, consider a database table with an attribute (or column) A over which range queries are enabled. The values in the column are mapped to integer values between 0 and $2^t - 1$ for some number t. To support range queries on attribute A we augment the given cleartext database DB

with t *virtual*[1] columns which are populated at Setup as follows. Consider a full binary tree with $t+1$ levels and 2^t leaves. Each node in the tree is labeled with a binary string describing the path from the root to the node: The root is labeled with the empty string, its children with strings 0 and 1, its grandchildren with 00, 01, 10, 11, and so on. A node at depth d is labeled with a string of length d, and the leaves are labeled with t-long strings that correspond to the binary representation of the integer value in that leaf, i.e. a t-bit binary representation padded with leading zeros.

Each of the t added columns correspond to a level in the tree, denoted $A'(1), A'(2), \ldots, A'(t)$ (A' indicates that this is a "virtual attribute" derived from attribute A). A record (or row) whose value for attribute A has binary representation $v_{t-1}, \ldots, v_1, v_0$ will have the string $(v_{t-1}, \ldots, v_1, v_0)$ in column $A'(t)$, the string $(v_{t-1}, \ldots v_1)$ in column $A'(t-1)$, and so on till column $A'(1)$ which will have the string v_{t-1}. Once the above plaintext columns $A'(1), \ldots, A'(t-1)$ are added to DB (note that $A'(t)$ is identical to the original attribute A), they are processed by the regular OXT pre-processing as any other original DB column, but they will be used exclusively for processing range queries.

Client processing. To query for a range $\mathsf{RQ}(a, b)$, the client selects a set of nodes in the tree that form a *cover* of the required range, namely, a set of tree nodes for which the set of descendant leaves corresponds exactly to all elements in the range $[a, b]$ (e.g. a cover for range 3 to 9 in a tree of depth 4 will contain cover nodes 0011, 01, 100). Let $c_1, \ldots, c_\ell$ be the string representation of the nodes in the cover and assume these nodes are at depths $d_1, \ldots, d_\ell$, respectively (not all depths have to be different). The query then is formed as a *disjunction of the ℓ exact-match queries "column $A'(d_i)$ has value c_i", for $i = 1, \ldots, \ell$*. Note that this simple reduction to a disjunction of exact terms allows us to reuse the full OXT functionality (and implementation). In particular, we can combine range queries with Boolean expressions on other terms, etc. Also note that we assume that the client knows how nodes in the tree are represented; in particular it needs to know the total depth of the tree. We stress that *this reduction to a disjunctive query works with* any *strategy for selecting the cover set*. This is important since different covers present different trade-offs between performance and leakage. Moreover, since the pre-processing of data is independent of the choice of cover, one can allow multiple cover strategies to co-exist to suit different leakage-performance trade-offs. Later, we will describe specific strategies for cover selection.

Interaction of client $\mathcal{C}$ with server $\mathcal{E}$. The search at $\mathcal{E}$ is carried exactly as in the Search phase of OXT as with any other disjunction. In particular, $\mathcal{E}$ does not need to know whether this disjunction comes from a range query.

Server $\mathcal{D}'s$ token generation and authorization. For the case of single-client and multi-client) SSE, token generation and authorization work as with any disjunction in the original OXT protocol. However, in the OSPIR setting, $\mathcal{D}$ needs to authorize the query without learning the queried values. Specifically, in the scenario addressed by our implementation, authorization of range queries is based on the searched attribute (e.g., age) and the total size of the range (i.e., policy attaches to each client an upper bound on the size of a range the client is allowed to query for the given attribute). To enforce this policy, we allow $\mathcal{D}$ to learn the searched attribute and the total size of the range, i.e., $b - a + 1$, but not the actual end-point values a, b. This is accomplished as follows.

Client $\mathcal{C}$ computes a cover corresponding to his range query and maps each node in the cover to a keyword (d, c), where d is the depth of the node in the tree and c the corresponding string. It then generates a disjunction of the resultant keywords $(d_i, c_i), i = 1, \ldots, \ell$, where ℓ is the size of the cover, d_i acts as the keyword's attribute and c_i as its value. $\mathcal{C}$ provides $\mathcal{D}$ with the attributes $d_1, \ldots, d_\ell$ thus allowing $\mathcal{D}$ to provide the required search tokens to $\mathcal{C}$ as specified by the OXT protocol for the OSPIR setting [13] (OXT requires the keyword attribute to generate such token). However, before providing these tokens, $\mathcal{D}$ needs to verify that the total size of the range is under the bound that $\mathcal{C}$ is authorized for. $\mathcal{D}$ computes this size using her knowledge of the depths $d_1, \ldots, d_\ell$ by the formula $\sum_{i=1}^{\ell} 2^{t-d_i}$ which gives the number of leaves covered

[1] The original DB is not changed, only the inverted indexes (TSet's) corresponding to these virtual columns are generated.

by these depths. Note that this ensures the total size of the range to be under a given bound but the range can be formed of non-consecutive intervals. Importantly, note that this authorization approach works with any cover selection strategy used by the client.

Cover Selection. There remains one *crucial* element to take care of: Making sure that the knowledge of the cover depths $d_1, \ldots, d_\ell$ does not reveal to $\mathcal{D}$ any information other than the total size of the range. Note that the way clients select covers is essentially independent of the mechanisms for processing of range queries described above. Here we analyze some choices for cover selection. The considerations for these choices are both *performance* (e.g. size of the cover) and *privacy*. Privacy-wise the goal is to limit the leakage to server $\mathcal{E}$ and, in the OSPIR case, also to $\mathcal{D}$. In the latter case, the goal is to avoid leakage beyond the size of the range that $\mathcal{D}$ needs to learn in order to check policy compliance. These goals raise general questions regarding *canonical covers* and *minimal over-covers* which we outline below.

A natural cover selection for a given range is one that minimizes the number of nodes in the cover (hence minimizes the number of disjuncts in the search expression). Unfortunately, such cover leaks information beyond the size of a range, namely, it allows to distinguish between ranges of the same size. E.g., ranges $[0, 3]$ and $[1, 4]$ are both of size 4 but the first has a single node as its minimal cover while the latter requires 3 nodes. Clearly, if $\mathcal{C}$ uses such a cover, $\mathcal{D}$ (and possibly $\mathcal{E}$) will be able to distinguish between the two cases.

Canonical Profiles and Universal Covers. The above example raises the following question: Given that authorization allows $\mathcal{D}$ to learn the depths of nodes in a cover, is there a way of choosing a cover that only discloses the total size of the range (i.e., does not allow to distinguish between two different ranges of the same size even when the depths are disclosed)? In other words, we want a procedure that given a range produces a cover with a number of nodes and depths that is the same for any two ranges of the same size. We call such covers universal. The existence of universal covers is demonstrated by the cover that uses each leaf in the range as a singleton node in the cover. Can we have a *minimal universal cover*? Next, we answer this question in the affirmative.

Definition 1. *The* profile *of a range cover is the multi-set of integers representing the heights of the nodes in the cover. (The* height *of a tree node is its distance from a leaf, i.e., leaves have height 0, their parents height 1, and so on up to the root which has height $t - 1$.) A profile for a range of size n is* universal *if any range of size n has a cover with this profile. A* universal cover *is one whose profile is universal. A universal profile for n is* minimal *if there is a range of size n for which all covers have that profile. (For example, for $n > 2$ the all-leaves cover is universal but not minimal.)*

Definition 2 (Canonical profile). *A profile for ranges of size n is called* canonical *if it is composed of the heights $0, 1, 2, \ldots, L - 1$, where $L = \lfloor \log(n + 1) \rfloor$, plus the set of powers ('1' positions) in the binary representation of $n' = n - 2^L + 1$. A* canonical cover *is one whose profile is canonical.*

Example: for $n = 20$ we have $L = 4, n' = 5$, and the canonical profile is $\{0, 1, 2, 3, 0, 2\}$ where the last 0, 2 correspond to the binary representation 101 of 5 (note that $20 = 2^0 + 2^1 + 2^2 + 2^3 + 2^0 + 2^2$).

Theorem 1. *For every integer $n > 0$ the canonical profile of ranges of size n is universal and minimal (and the only such profile).*

The proof of this lemma is elementary but somewhat lengthy and is presented in Appendix A where we also present a procedure for computing canonical covers. (We note that a notion similar to canonical covers has been used, independently and in a different context, in [16].)

3-node universal over-covers. The canonical cover has the important property of not leaking any information to $\mathcal{D}$ beyond the size of the range (that $\mathcal{D}$ needs to learn anyway to authorize a query). However, the number of nodes in a canonical cover can leak information on the range size to server $\mathcal{E}$ (assuming that $\mathcal{E}$ knows that a given disjunction corresponds to a range query). Another drawback is that canonical covers

may include $2 \log n$ nodes. Ideally, we would like to use covers with a small and fixed number of nodes that also have *universal profiles*, i.e., any two ranges of a given size will always be represented by covers with the same depths profile. While we show this to be impossible for exact covers, we obtain covers with the above properties by allowing false-positives, i.e., covers that may include elements outside the requested range, hence we call them *over-covers*.

We instantiate this approach by showing an example of 3-node universal over-cover which works on all ranges, and its covered range may be up to a 66% larger than the original range (and it is 40% larger on average). Note that leakage-wise, false positives (visible to $\mathcal{C}$) are not a problem in the single-client SSE setting while in the multi-client case one would require the over-cover to fully fall under the range sizes authorized for $\mathcal{C}$. Importantly, these over-covers reduce leakage to $\mathcal{E}$ by fixing the number of disjuncts regardless of range. Choosing a cover strategy can depend on a specific query and the particular attribute where the range query is applied to. *Fortunately a client can choose its own strategy for cover selection on an individual query basis.* All other parts of the protocol are independent of the client's choice of particular covers.

We define a 3-node universal over-cover profile for a given range size n as follows. Let $n = 2^L + 2^s + n' + 2$ where $s < L, n' + 2 < 2^s$, i.e., L and s are the two leftmost positions of 1's in the binary representation of $n-2$ (for $n = 2^L + 2$ define $s = 0$). The 3-node profile contains heights L, s and $\max\{s+1,\, L-1\}$ (i.e., the third element is L if $s = L-1$ and $L-1$ otherwise).

Theorem 2. *Every range of size $n = 2^L + 2^s + n' + 2$, $s < L, n' + 2 < 2^s$, has a 3-node over-cover with profile heights L, s and $\max\{s+1,\, L-1\}$.*

Proof. In any range of size $n = 2^L + 2^s + n' + 2$ as above there must be a 2^L boundary (i.e., a number in the range of the form $k \cdot 2^L$ for some $k \geq 0$). We consider two cases depending on whether there is a full 2^L block inside the range (i.e. a sub-range $[k \cdot 2^L, (k+1) \cdot 2^L - 1]$ for some $k \geq 0$) or not.

In the first case (full 2^L block) consider the following sub-range lengths (where commas correspond to boundaries between 2^L blocks):

1. $(2^{s-1} + 1,\, 2^L,\, 2^{s-1} + 1)$ which requires a L-height node and two s-height nodes
2. $(1,\, 2^L,\, 2^s + 1)$ which requires a L-height node and a $(s+1)$-height node.

For the case where no full 2^L block is included in the range, the range will have two parts across a 2^L-boundary. One of the two parts (the larger) can be covered by L-height node (which case 1 shows to be needed). Thus we need two nodes to cover the second part. The worst case is when this second part is as large as possible which happens at $\lfloor n/2 \rfloor = 2^{L-1} + 2^{s-1} + \lfloor n' \rfloor + 1$. Since two $(L-1)$-height nodes are insufficient to cover it then we need at least one L-height node for this; the remaining range of size $2^{s-1} + \lfloor n' \rfloor + 1$ then requires a s-height node. Thus, this case requires in total a $(L-1)$-height node and a s-height node.

Summarizing, we need all of the following three combinations of heights: $\{L,\, s,\, s\}, \{L,\, s+1\}, \{L,\, L-1,\, s\}$. The 3-node profile defined in the theorem is the minimal to satisfy all these combinations.

Here are a few examples to ilustrate the Theorem. To calculate the profile of a range of size $n = 15$, we write the binary representation of $n-2$, which is 13=1101, i.e., $L = 3, s = 2$, so the 3-node profile is $(3, 3, 2)$, and the size of the over-cover is 20, with an overhead of 5 leaves. For $n = 20$ we have 18=10010, hence the profile is $(4, 3, 1)$ and overhead is 6. For $n = 100$, we have 98=1100010, hence the profile is $(6, 6, 5)$ and overhead is 60.

Choosing a 3-node universal over-cover for a given interval can be implemented with a simple algorithm (just follow the constructive argument in the proof). The drawback of choosing an over-cover is that it contains values outside the given range. We call the number of leaves covered by an over-cover its volume, so if V is the volume of the over-cover of a range of size n, the overhead is $V/n - 1$. The worst case overhead is for range sizes n of the form $2^L + 2^{L-1} + 2$ for which $V/n - 1 = (2 \cdot 2^L + 2^{L-1})/(2^L + 2^{L-1} + 2) \approx 5/3$, i.e.,

a 66% overhead in the worst case. We can also calculate the average overhead for all ranges whose 3-node universal profile starts with a given L, i.e., numbers n with $n-2$ between 2^L and $2^{L+1}-1$. This is done by computing the sum, over these n values, of the volumes of the corresponding 3-node over-covers. For every L the average overhead comes very close to 7/18-th, i.e. about 40%.

Finally, we note that using 2-node over-covers is not recommended as their volume can grow up to three times the size of the original range.

4 Substring Queries

Our substring-search capable SSE scheme is based on the conjunctive-search SSE protocol OXT of [6], and it extends that protocol as follows: Whereas the OXT scheme of [6] supported efficient retrieval of records containing several required keywords at once (i.e. satisfying a *conjunction* of several keyword-equality search terms), our extension supports efficient retrieval of records containing the required keywords *at required relative positions to one another*. This extension of conjunctive search with positional distance criteria allows us to handle several query types common in text-based information retrieval. To simplify the description, and using the notation from Section 2, consider a database $\mathsf{DB} = (\mathsf{ind}_i, T_i)$ containing records with just one free text attribute, i.e. where each record T_i is a text string. We support the following types of queries q:

Substring Query. Here q is a text string, and $\mathsf{DB}(q)$ returns all ind_i s.t. T_i contains q as a substring.

Wildcard Query. Here q is a text string which can contain wildcard characters $'?'$ (matching any single character), and $\mathsf{DB}(q)$ returns all ind_i s.t. T_i contains a substring q' s.t. for all j from 1 to $|q|$, $q_j =' ?' \vee q_j = q'_j$, where q_j and q'_j denote j-th characters in strings q and q'. If the query should match only prefixes (suffixes) of T_i, the query can be prefixed (suffixed) with a $'\hat{}'$ ($'\$'$).

Phrase Query. Here q is a sequence of words, i.e. text strings, $q = (q^1, \ldots, q^l)$, where each q^i can equal to a wildcard character $'?'$. Records T_i in DB are also represented as sequences of words, $T_i = (T_i^1, \ldots, T_i^n)$. $\mathsf{DB}(q)$ returns all ind_i s.t. for some k and for all j from 1 to l, it holds that $q^j =' ?' \vee q^j = T_i^{k+j}$. (Note that phrase queries allow a match of a single wildcard with a whole word of any size, while in a wildcard query a single wildcard can match only a single character.)

All these query types utilize the same crypto machinery that we describe next for the substring case. In Section 4.2 we explain briefly how to adapt the techniques to these queries too.

4.1 Basic SSE Substring Search

Here we present protocol SUB-SSE-OXT that supports substring search in the basic SSE model (i.e., a single client $\mathcal{C}$ outsources its encrypted database to server $\mathcal{E}$) and where the query consists of a single substring. This simpler case allows us to explain and highlight the basic ideas that we also use for addressing the general case of boolean expressions that admit substrings as the expression terms as well as for extending these solutions to the more involved MC and OSPIR settings.

Figure 1 describes the protocol where shadowed text highlights the changes with respect to the original OXT protocol from [6] for resolving conjunctive queries in the SSE model (the reader can visualize the underlying OXT protocol by omitting the shadowed text). We first explain the basic rationale and functioning of the conjunctive-search OXT protocol, and then we explain how we extend it by imposing additional constraints on *relative positions* of the searched terms, and how this translates into support for substring-search SSE.

The Conjunctive SSE Scheme OXT. Let $q = (w_1, \ldots, w_n)$ be a conjunctive query where $\mathsf{DB}(q) = \cap_{i=1}^n \mathsf{DB}(w_i)$. Let F_G be a Pseudorandom Function (PRF) with key K_G. (This PRF will map onto a cyclic group G, hence the name.) Let the setup algorithm create as metadata a set of (keyed) hashes XSet, named for "cross-check set", containg the hash values $\mathsf{xtag}_{w,\mathsf{ind}} = F_G(K_G, (w, \mathsf{ind}))$ for all keywords $w \in \mathsf{W}$ and records

$\mathsf{ind} \in \mathsf{DB}(w)$. Let the setup also create the matadata needed to quickly retrieve the set of record indexes $\mathsf{DB}(w)$ matching any given *single* keyword $w \in \mathsf{W}$. The OXT protocol is based on a simple conjunctive *plaintext* search algorithm which identifies all records corresponding to a conjunctive query $q = (w_1, \ldots, w_n)$ as follows: It first identifies the set of indexes $\mathsf{DB}(w_1)$ satisfying the first term w_1, called an *s-term*, and then for each $\mathsf{ind} \in \mathsf{DB}(w_1)$ it returns ind as part of $\mathsf{DB}(q)$ if and only if hash value $\mathsf{xtag}_{w_i,\mathsf{ind}} = F_G(K_G, (w_i, \mathsf{ind}))$ is in XSet for all *x-terms* (i.e. "cross-check terms") $w_2, \ldots, w_n$. If group G is sufficiently large then except for negligible collision probability, if $\mathsf{xtag}_{w_i,\mathsf{ind}} \in \mathsf{XSet}$ for $i \geq 2$ then $\mathsf{ind} \in \cap_{i=2}^{n} \mathsf{DB}(w_i)$, and since ind was taken from $\mathsf{DB}(w_1)$ it follows that $\mathsf{ind} \in \mathsf{DB}(q)$. Since this algorithm runs in $O(|\mathsf{DB}(w_1)|)$ time w_1 should be chosen as the least frequent keyword in q.

To implement the above protocol over *encrypted* data the OXT protocol modifies it in three ways: First, the metadata supporting retrieval of $\mathsf{DB}(w)$ is implemented using single-keyword SSE techniques, specifically the *Oblivious Storage* data structure TSet [6,5], named for "tuples set", which reveals to server $\mathcal{E}$ only the total number of keyword occurrences in the database, $\sum_{w \in \mathsf{W}} |\mathsf{DB}(w)|$, but hides all other information about individual sets $\mathsf{DB}(w)$ except those actually retrieved during search. (A TSet can be implemented very efficiently as a hash table using PRF F whose key K_T is held by client $\mathcal{C}$, see [6,5].) Secondly, the information stored for each w in the TSet datastructure, denoted $\mathsf{TSet}(w)$, which $\mathcal{E}$ can recover from TSet given $F(K_T, w)$, is not the plaintext set of indexes $\mathsf{DB}(w)$ but the encrypted version of these indexes using a special-purpose encryption. Namely, a tuple corresponding to the c-th index ind_c in $\mathsf{DB}(w)$ (arbitrarily ordered) contains value $y_c = F_p(K_I, \mathsf{ind}_c) \cdot F_p(K_z, c)^{-1}$, an element in a prime-order group Z_p where F_p is a PRF onto Z_p, and K_I, K_z are two PRF keys where K_I is global and K_z is specific to keyword w (derived e.g. via another PRF on input w). This encryption enables fast secure computation of hash $\mathsf{xtag}_{w_i,\mathsf{ind}_c}$ between client $\mathcal{C}$ and server $\mathcal{E}$, where $\mathcal{E}$ holds ciphertext $y_c = F_p(K_I, \mathsf{ind}_c) \cdot F_p(K_z, c)^{-1}$ of c-th index ind_c taken from $\mathsf{TSet}(w_1)$ and $\mathcal{C}$ holds keyword w_i and keys K_I, K_z. Let $F_G(K_G, (w, \mathsf{ind})) = g^{F_p(K_X,w) \cdot F_p(K_I,\mathsf{ind})}$ where g generates group G and $K_G = (K_X, K_I)$ where K_X is a PRF key. $\mathcal{C}$ then sends to $\mathcal{E}$:

$$\mathsf{xtoken}[c, i] = g^{F_p(K_X, w_i) \cdot F_p(K_z, c)}$$

for $i = 2, \ldots, h$ and $c = 1, \ldots, |\mathsf{TSet}(w_1)|$, and $\mathcal{E}$ computes $F_G(K_G, (w_i, \mathsf{ind}_c))$ for each c, i as:

$$(\mathsf{xtoken}[c, i])^{y_c} = (\mathsf{xtoken}[c, i])^{F_p(K_I, \mathsf{ind}_c) \cdot F_p(K_z, c)^{-1}}$$

Since K_z is specific to w_1 mask $z_c = F_p(K_z, c)$ applied to ind_c in y_c is a one-time pad, hence this protocol reveals only the intended values $F_G(K_G, (w_i, \mathsf{ind}_c))$ for all $\mathsf{ind}_c \in \mathsf{DB}(w_1)$ and $w_2, \ldots, w_n$.

Extending OXT to Substring SSE. The basic idea for supporting substring search is first to represent a substring query as a conjunction of k-grams (strings of length k) at given relative distances from each other (e.g., a substring query *'yptosys'* can be represented as a conjunction of a 3-gram *'tos'* and 3-grams *'ypt'* and *'sys'* at relative distances -2 and 2 from the first 3-gram, respectively), and then to extend the conjunctive search protocol OXT of [6] so that it verifies not only whether the conjunctive terms all occur within the same document, but also that they occur at positions whose relative distances are specified by the query terms. We call representation of a substring q as a set of k-grams with relative distances a *tokenization* of q. We denote the *tokenizer* algorithm as T, and we denote its results as $T(q) = (\mathsf{kg}_1, (\Delta_2, \mathsf{kg}_2), \ldots, (\Delta_\mathsf{h}, \mathsf{kg}_\mathsf{h}))$ where Δ_i are any non-zero integer values, including negatives, e.g. $T(\text{'}yptosys\text{'})$ can output $(\text{'}tos\text{'}, (-2, \text{'}ypt\text{'}), (2, \text{'}sys\text{'}))$, but many other tokenizations of the same string are possible. We call k-gram kg_1 an *s-gram* and the remaining k-grams *x-grams*, in parallel to the s-term and x-term terminology of OXT, and as in OXT the s-gram should be chosen as the least frequent k-gram in the tokenization of q. Let KG be a list of k-grams which occur in DB. Let $\mathsf{DB}(\mathsf{kg})$ be the set of $(\mathsf{ind}, \mathsf{pos})$ pairs s.t. $\mathsf{DB}[\mathsf{ind}]$ contains k-gram kg at position pos, and let $\mathsf{DB}(\mathsf{ind}, \mathsf{kg})$ be the set of pos's s.t. $(\mathsf{ind}, \mathsf{pos}) \in \mathsf{DB}(\mathsf{kg})$.

The basic idea of the above conjuctive-search protocol to handling substrings is that the hashes xtag inserted into the XSet will use PRF F_G applied to a *triple* $(\mathsf{kg}, \mathsf{ind}, \mathsf{pos})$ for each $\mathsf{kg} \in \mathsf{KG}$ and $(\mathsf{ind}, \mathsf{pos}) \in \mathsf{DB}(\mathsf{kg})$, and when processing search query q where $T(q) = (\mathsf{kg}_1, (\Delta_2, \mathsf{kg}_2), \ldots, (\Delta_\mathsf{h}, \mathsf{kg}_\mathsf{h}))$, server $\mathcal{E}$ will return (encrypted)

Setup($\mathsf{DB}, \mathsf{RDK}$)

- Select keys K_S, K_T for PRF F_τ and K_I, K_X for PRF F_p, and parse DB as $(\mathsf{ind}_i, \mathsf{pos}_i, \mathsf{kg}_i)_{i=1}^{d}$. (PRF F_τ maps onto $\{0,1\}^\tau$ and F_p onto Z_p.)
- Initialize $\mathbf{T}$ to an empty array and XSet to an empty set. For each k-gram $\mathsf{kg} \in \mathsf{KG}$ do the following:
  - Set $\mathsf{strap} \leftarrow F_\tau(K_S, \mathsf{kg})$, $(K_z, K_e, K_u) \leftarrow (F_\tau(\mathsf{strap}, 1), F_\tau(\mathsf{strap}, 2), F_\tau(\mathsf{strap}, 3))$.
  - For $c = 1, \ldots, |\mathsf{DB}(\mathsf{kg})|$, for $(\mathsf{ind}, \mathsf{pos})$ a c-th tuple in $\mathsf{DB}(\mathsf{kg})$ (randomly permuted) do:
    - Set $\mathsf{rdk} \leftarrow \mathsf{RDK}(\mathsf{ind})$, $\mathsf{e} \leftarrow \mathsf{Enc}(K_e, (\mathsf{ind}|\mathsf{rdk}))$, $\mathsf{xind} \leftarrow F_p(K_I, \mathsf{ind})$.
    - Set $\mathsf{xtag} \leftarrow g^{F_p(K_X, \mathsf{kg}) \cdot \mathsf{xind}^{\mathsf{pos}}}$ and add xtag to XSet.
    - Set $z \leftarrow F_p(K_z, c)$, $u \leftarrow F_p(K_u, c)$, $y \leftarrow \mathsf{xind} \cdot z^{-1}$, $v \leftarrow \mathsf{xind}^{\mathsf{pos}} \cdot u^{-1}$.
    - Append (e, y, v) to $\mathbf{T}[\mathsf{kg}]$.
- Set $\mathsf{TSet} \leftarrow \mathsf{TSetSetup}(\mathbf{T}, \langle F_\tau \rangle, K_T)$. Output $K = (K_S, K_X, K_T)$ and $\mathsf{EDB} = (\mathsf{TSet}, \mathsf{XSet})$.

Search protocol

Client $\mathcal{C}$, on input $K = (K_S, K_X, K_T)$ defined above and query q s.t. $T(q) = (\mathsf{kg}_1, (\Delta_2, \mathsf{kg}_2), \ldots, (\Delta_\mathsf{h}, \mathsf{kg}_\mathsf{h}))$:

- Set $\mathsf{stag} \leftarrow F_\tau(K_T, \mathsf{kg}_1)$, $\mathsf{strap} \leftarrow F_\tau(K_S, \mathsf{kg}_1)$.
- $(K_z, K_e, K_u) \leftarrow (F_\tau(\mathsf{strap}, 1), F_\tau(\mathsf{strap}, 2), F_\tau(\mathsf{strap}, 3))$, and $\{\mathsf{xtrap}_i \leftarrow g^{F_p(K_X, \mathsf{kg}_i)}\}_{i=2}^{\mathsf{h}}$.
- Send $(\mathsf{stag}, \Delta_2, \ldots, \Delta_\mathsf{h})$ to $\mathcal{E}$, and for $c = 1, 2, \ldots$, until $\mathcal{E}$ sends stop, do the following:
  - Set $z_c \leftarrow F_p(K_z, c)$, $u_c \leftarrow F_p(K_u, c)$, and $\{\mathsf{xtoken}[c, i] \leftarrow (\mathsf{xtrap}_i)^{((z_c)^{\Delta_i} \cdot (u_c))}\}_{i=2}^{\mathsf{h}}$.
  - Send $\mathsf{xtoken}[c] = (\mathsf{xtoken}[c, 2], \ldots, \mathsf{xtoken}[c, \mathsf{h}])$ to $\mathcal{E}$.

Server $\mathcal{E}$, on input $\mathsf{EDB} = (\mathsf{TSet}, \mathsf{XSet})$, responds with a set ESet formed as follows:

- On message $(\mathsf{stag}, \Delta_2, \ldots, \Delta_n)$ from $\mathcal{C}$, retrieve $\mathbf{t} \leftarrow \mathsf{TSetRetrieve}(\mathsf{TSet}, \mathsf{stag})$ from TSet.
- For $c = 1, ..., |\mathbf{t}|$, retrieve c-th tuple (e, y, v) in $\mathbf{t}$.
- On $\mathsf{xtoken}[c]$ from $\mathcal{C}$, add e to ESet if $\forall i = 2, \ldots, \mathsf{h} : (\mathsf{xtoken}[c, i])^{(y^{\Delta_i} \cdot v)} \in \mathsf{XSet}$. When $c = |\mathbf{t}|$ send stop to $\mathcal{C}$.

Client $\mathcal{C}$ computes $(\mathsf{ind}|\mathsf{rdk}) \leftarrow \mathsf{Dec}(K_e, \mathsf{e})$ for each e in ESet and adds $(\mathsf{ind}, \mathsf{rdk})$ to its output.

Fig. 1. SUB-SSE-OXT: SSE Protocol for Substring Search (shadowed text indicates additions to the basic OXT protocol for supporting substring queries)

index ind corresponding to some $(\mathsf{ind}_c, \mathsf{pos}_c)$ *pair* in $\mathsf{DB}(\mathsf{kg}_1)$ if and only if

$$F_G(K_G, (\mathsf{kg}_i, \mathsf{ind}_c, \mathsf{pos}_c + \Delta_i)) \in \mathsf{XSet} \text{ for } i = 2, \ldots, \mathsf{h}$$

To support this modified search over encrypted data the setup procedure $\mathsf{Setup}(\mathsf{DB}, \mathsf{RDK})$ forms EDB as a pair of data structures TSet and XSet as in OXT, except that keywords are replaced by k-grams and both the encrypted tuples in TSet and the hashes xtag in XSet will be modified by the position-related information as follows. First, the tuple corresponding to the c-th (index,position) pair $(\mathsf{ind}_c, \mathsf{pos}_c)$ in $\mathsf{DB}(\mathsf{kg})$ will contain value $y_c = F_p(K_I, \mathsf{ind}_c) \cdot F_p(K_z, c)^{-1}$ together with a new position-related value $v_c = F_p(K_I, \mathsf{ind}_c)^{\mathsf{pos}_c} \cdot F_p(K_u, c)^{-1}$, where K_z, K_u are independent PRF keys specific to kg. Secondly, XSet will contain values computed as:

$$F_G((K_X, K_I), (\mathsf{kg}, \mathsf{ind}, \mathsf{pos})) = g^{F_p(K_X, \mathsf{kg}) \cdot F_p(K_I, \mathsf{ind})^{\mathsf{pos}}} \quad (1)$$

In the Search protocol, client $\mathcal{C}$ will tokenize its query q as $T(q) = (\mathsf{kg}_1, (\Delta_2, \mathsf{kg}_2), \ldots, (\Delta_\mathsf{h}, \mathsf{kg}_\mathsf{h}))$, send $\mathsf{stag}_{\mathsf{kg}_1} = F_T(K_T, \mathsf{kg}_1)$ to server $\mathcal{E}$, who uses it to retrieve $\mathsf{TSet}(\mathsf{kg}_1)$ from TSet, send the position-shift

vectors $(\Delta_2, \ldots, \Delta_{\mathsf{h}})$ to $\mathcal{E}$, and then, in order for $\mathcal{E}$ to compute $F_G(K_G, (\mathsf{kg}_i, \mathsf{ind}_c, \mathsf{pos}_c + \Delta_i))$ for all c, i pairs, client $\mathcal{C}$ sends to $\mathcal{E}$:

$$\mathsf{xtoken}[c, i] = g^{F_p(K_X, \mathsf{kg}_i) \cdot (F_p(K_z, c))^{\Delta_i} \cdot F_p(K_u, c)}$$

which lets $\mathcal{E}$ compute $F_G(\mathsf{kg}_i, \mathsf{ind}_c, \mathsf{pos}_c + \Delta_i)$ as $(\mathsf{xtoken}[c, i])$ exponentiated to power $(y_c)^{\Delta_i} \cdot v_c$ for (y_c, v_c) in the c-th tuple in $\mathsf{TSet}(\mathsf{kg}_1)$, which computes correctly because

$$y_c^{\Delta_i} \cdot v_c = F_p(K_I, \mathsf{ind}_c)^{\Delta_i + \mathsf{pos}_c} \cdot F_p(K_z, c)^{-\Delta_i} \cdot F_p(K_u, c)^{-1}$$

4.2 Wildcards and Phrase Queries

Any sequence of single character wildcards within regular substring queries can be handled by changing tokenization to allow gaps in the query string covered by the computed tokens, e.g. $T('ypt??yst')$ would output $('ypt', (5, 'yst'))$.

In addition to support *wildcard queries* matching prefixes and/or suffixes, we add special "anchor" tokens at the beginning ($'\hat{}'$) and end ($'\$'$) of every record to mark the text boundaries. These anchors are then added during tokenization. This allows searching for substrings at fixed positions within a record. For these queries $T('ypt??yst')$ would output $('\hat{}yp', (1, 'ypt'), (6, 'yst'), (7, 'st\$'))$

Still, this simple change limits us to queries which contain k consecutive characters in-between every substring of wildcards. However, we can remove this restriction if we add to the the XSet all unigrams (i.e. $k = 1$) occurring in a text in addition to the original k-grams.

Adding support for phrase queries is another simple change to the way we parse DB. Instead of parsing by (k-gram,position) pairs, we parse each record by (word,position). Tokenization of q then becomes splitting q into its component words and relative position of each word to the s-term word. As with substrings, wildcards in q result in a gap in the returned Δ's.

4.3 Query Flexibility

While many queries can be formed by using substring or wildcard queries independently, many queries are not computable. We can greatly increase the number of available queries by combining the two query types. This allows us to answer any query q s.t. all non-wildcard characters in q are part of at least one k length substring containing no wildcards and q starts and ends with a non-wildcard character. This may require a sufficiently large k (a performance benefit) but limit the type of queries supported. To further increase flexibility we can index fields with multiple values for k or with a different k for each data structure: k_x for XSet and k_s for TSet. The result is a very flexible policy that we can support any query q that meets the following: (1) there exists at least one consecutive k_s length sequence of non-wildcards in q, (2) all non-wildcard characters in q are part of at least one k_x length substring containing no wildcards, and (3) q starts and ends with a non-wildcard character.

Condition (3) above can be avoided if we have an index for $k = 1$ by exploiting OXT's general support of boolean expressions including negation: To handle queries q with n leading (resp. trailing) wildcards, we take the tokenization t of the query string stripped of the leading (resp. trailing) wildcards q' and search for q' but make sure to exclude matches which would be a distance less than n from an anchor. Formally we query: $t \wedge \neg \vee_{i=1}^{n} (-i, \hat{})$ (resp. $t \wedge \neg \vee_{i=1}^{n} (\delta_{\max} + i, \$)$) with $\hat{}$ and $\hat{}$ the anchors and $\delta_{\max}$ the relative position of the right "edge" of q'. The OXT support for general boolean expressions can also be used to support some subset of regular expressions: besides the already implicitly used conjunctions, we could support queries containg: disjunctions to handle alternative sub-patterns such as "Court (Road|Street)", negations to explicitly exclude sub-patterns such as "Michael!(a)" and combinations thereof.

4.4 Substring Protocol Extensions

Firstly, we generalize the substring-search protocol SUB-SSE-OXT to support any Boolean query where atomic terms can be formed by any number of substring search terms and/or exact keyword terms.

We note that in above protocol substring/wildcard terms have to be s-terms which restricts the task to handle general queries. Nevertheless, we still can trivially extend above to handle cases where there is at most one substring or wildcard term per conjunction at the "top-level" of the query expression (tree): we just append any additional top-level conjuncts to the substring/wild-card term as x-terms and compose them as in OXT with any other top-level disjuncts.

Furthermore, we note that a pattern matching only a single k-gram (*singleton*) can be treated as a normal equality-match term and for many environments the set of lengths of queried sub-sequences can be fairly small, e.g., in one government sponsored project it was 3. To exploit this at only moderate cost, we can add a k-gram index for all k values in the set of queryable sub-sequence lengths and include in the index position information as normal and also as information for equality-matching (i.e. we add two tags to the XSet). During query-processing we always try to select a k for a sub-sequence term which results in a singleton and correspondingly compute the equality-term tag rather than the k-gram-tag in such a case. With this strategy, we can handle multiple substring/wildcard terms as long as all but one are singletons per top-level conjuncts. These additional indexes have the added benifit of increased query-time performance.

However, to allow for arbitrary queries we have to extend our protocols. We call the resulting protocol MIXED-SSE-OXT, so named because it freely *mixes* substring and exact keyword search terms, and present it in Appendix B.1. The ability to handle Boolean formulas on exact keywords together with substring terms comes from the similarities between substring-handling SUB-SSE-OXT and Boolean-formula-handling OXT of [6]. However, one significant adjustment needed to put the two together is to disassociate the position-related information v_c in the tuples in $\mathsf{TSet}(\mathsf{kg})$ from the index-related information y_c in these tuples. This is because when all k-gram terms are x-terms (as would be the the case e.g. when an exact keyword is chosen as an s-term) then $\mathcal{E}$ must identify the position-related information pertaining to a particular $(\mathsf{kg}, \mathsf{ind})$ pair given the $(\mathsf{kg}, \mathsf{ind})$-related xtoken value. Our MIXED-SSE-OXT protocol supports this by adding another oblivious TSet-like datastructure which uses $\mathsf{xtag}_{\mathsf{kg},\mathsf{ind}}$ to retrieve the position-related information, i.e. the v_c's, for all $\mathsf{pos} \in \mathsf{DB}(\mathsf{ind}, \mathsf{kg})$.

A second extension generalizes the SUB-SSE-OXT protocol to the OSPIR setting [13] where $\mathcal{D}$ can *obliviously* enable third-party clients $\mathcal{C}$ to compute the search-enabling tokens (see Section 2). The main ingredient in this extension is the usage of Oblivious PRF (OPRF) evaluation for several PRF functions used in MIXED-SSE-OXT for computing search tokens. Another important component is a novel protocol which securely computes the $\mathsf{xtag}_{\mathsf{kg},\mathsf{ind},\mathsf{pos}}$ values given these obliviously-generated trapdoors, in a way which avoids leaking any partial-match information to $\mathcal{C}$. This protocol, named MIXED-OSPIR-OXT and presented in Appendix B.2, uses bilinear maps which results in a significant slowdown compared to the MIXED-SSE-OXT in the (single client) SSE setting.

Future work. We are investigating more efficient variants of the MIXED-OSPIR-OXT protocol that would require less pairing operations and would reduce the communication between clients and server $\mathcal{E}$. In particular, we can show that in the Multi-Client (MC) setting where the third-party clients' queries are not hidden from the database owner $\mathcal{D}$, one can simplify the xtag-computation protocol, in particular eliminating the usage of bilinear maps and making the resulting protocol almost equal in cost to the MIXED-SSE-OXT protocol.

5 Security Analysis

Privacy of an SSE scheme, in the SSE, Multi-Client, or OSPIR settings, is quantified by a *leakage profile* $\mathcal{L}$, which is a function of the database DB and the sequence of client's queries $\mathbf{q}$. We call an SSE scheme

$\mathcal{L}$-*semantically-secure* against party P (which can be $\mathcal{C}$, $\mathcal{E}$, or $\mathcal{D}$) if for all DB and $\mathbf{q}$, the entirety of P's view of an execution of the SSE scheme on database DB and $\mathcal{C}$'s sequence of queries $\mathbf{q}$ is efficiently *simulatable* given only $\mathcal{L}(\mathsf{DB}, \mathbf{q})$. We say that the scheme is *adaptively* secure if the queries in $\mathbf{q}$ can be set adaptively by the adversary based on their current view of the protocol execution. An efficient simulation of a party's view in the protocol means that everything that the protocol exposes to this party carries no more information than what is revealed by the $\mathcal{L}(\mathsf{DB}, \mathbf{q})$ function. Therefore specification of the $\mathcal{L}$ function fully characterizes the privacy quality of the solution: What it reveals about data DB and queries $\mathbf{q}$, and thus also what it hides. (See [6,13] for a more formal exposition.)

5.1 Security of Range Queries

Below we state the security of the range query protocol for stand-alone range queries and we informally comment on the case of range queries that are parts of composite (e.g., Boolean) queries. We consider adaptive security against honest-but-curious and non-colluding servers $\mathcal{E}, \mathcal{D}$, and against fully malicious clients. For query $q^j = \mathsf{RQ}(a^j, b^j)$, let $((d_1^j, c_1^j), \ldots, (d_t^j, c_t^j))$ be the tree cover of interval $[a^j, b^j]$ and let $w_i^j = (d_i^j, c_i^j)$. We define three leakage functions for $\mathcal{D}, \mathcal{E}, \mathcal{C}$, respectively:
• $\mathcal{L}_{\mathcal{D}}(\mathsf{DB}, (q^1, \ldots, q^m))$ includes the query type ("range" in this case), the attribute to which q^j pertains, and the size of the range $b^j - a^j + 1$, for each q^j.
• $\mathcal{L}_{\mathcal{E}}(\mathsf{DB}, (q^1, \ldots, q^m)) = \mathcal{L}_{\mathsf{OXT}}(\mathsf{DB}, (w_1^1, \ldots, w_t^m))$ where the latter function represents the leakage to server $\mathcal{E}$ in the OXT protocol for a query series that includes all w_i^j's. By the analysis of [6], this leakage contains the TSet leakage (which in our TSet implementation is just the total number of document-keyword pairs in DB), the sequence $\{(|\mathsf{DB}(w_i^j)| : (i,j) = (1,1), \ldots, (t,m)\}$, i.e., the number of elements in each $\mathsf{DB}(w_i^j)$, and the result set returned by the query (in the form of encrypted records).
• $\mathcal{L}_{\mathcal{C}}(\mathsf{DB}, (q^1, \ldots, q^m)) = \emptyset$.

Theorem 3. *The range protocol from Sec. 3 is secure in the OSPIR model with respect to $\mathcal{D}, \mathcal{E}, \mathcal{C}$ with leakage profiles $\mathcal{L}_{\mathcal{D}}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{C}}$, respectively.*

The leakage functions for $\mathcal{D}$ and $\mathcal{C}$ are as good as possible: $\mathcal{D}$ only learns the information needed to enforce authorization, namely the attribute and size of the range, while there is no leakage at all to the client. The only non-trivial leakage is $\mathcal{E}$'s which leaks the number of documents matching each disjunct or, equivalently, the size of each sub-range in the range cover. The leakage to $\mathcal{D}$ remains the same also when the range query is part of a composite query. For the client this is also the case except that when the range query is the s-term of a Boolean expression, the client also learns an upper bound on the sizes $|\mathsf{DB}(w_i^j)|$ for all i, j. For $\mathcal{E}$, a composite query having range as its s-term is equivalent to tm separate expressions w_i^j as in [6] (with reduced leakage due to disjoint s-terms), and if the range term is an x-term in a composite query then w_i^j's leak the same as if they were x-terms in a conjunction.

5.2 Security of Substring Queries

Here we prove the security of protocol SUB-SSE-OXT against server $\mathcal{E}$. Our security arguments are based on the following assumptions: the T-set implementation is secure against adaptive adversaries [6,5]; F_p and F_τ are secure pseudorandom functions; the hash function H is modeled as a random oracle; and the q-DDH assumption [1] (see Appendix C) holds in the group G.[2]

Security Against Server $\mathcal{E}$. We first describe the leakage function corresponding to server $\mathcal{E}$. It is an adaptation of the leakage for the conjunctive protocol from [6] to our setting. To simplify presentation

[2] The extension to the OSPIR model also assumes the One-More Gap Diffie-Hellman assumption and assumes bilinear groups where the linear DH assumption [2,23] holds.

(avoiding complex notation) and focus on the important aspects of this leakage function, our description assumes that substring queries contain a single substring tokenized into two k-grams, i.e., one s-term k-gram and one x-term k-gram. The extension to the general case is similar to the extension from two-term conjunctions to general conjunctions in [6].

Leakage to Server $\mathcal{E}$. We represent a sequence of Q non-adaptive substring queries by $\mathbf{q} = (\mathbf{s}, \mathbf{x}, \mathbf{\Delta})$ s.t. $(\mathbf{s}[i], (\mathbf{x}[i], \mathbf{\Delta}[i]))$ is the tokenization $T(\mathbf{q}[i])$ of the i-th substring query $\mathbf{q}[i]$, where $\mathbf{s}[i], \mathbf{x}[i]$ are k-grams, and $\mathbf{\Delta}[i]$ is an integer between $-k+1$ and $k-1$. For notation simplicity we assume that vector $\mathbf{q}$ does not contain repeated queries, although $\mathcal{E}$ would learn that a repeated query has been made. Function $\mathcal{L}_{\mathcal{E}}(\mathsf{DB}, \mathbf{q})$ which specifies leakage to $\mathcal{E}$ outputs $(N, \bar{\mathbf{s}}, \mathsf{SP}, \mathsf{RP}, \mathsf{DP}, \mathsf{IP})$ defined as follows:

• The $(N, \bar{\mathbf{s}}, \mathsf{SP}, \mathsf{RP})$ part of this leakage is exactly the same as in the conjunctive SSE protocol SSE-OXT of [6] on which our substring-search SUB-SSE-OXT protocol is based. $N = \sum_{i=1}^{d} |\mathsf{W}_i|$ is the total number of appearances of all k-grams in all the documents, and it is revealed simply by the size of the EDB metadata. $\bar{\mathbf{s}} \in [m]^Q$ is the *equality pattern* of $\mathbf{s} \in \mathsf{KG}^Q$ indicating which queries have the equal s-terms. For example, if $\mathbf{s} = (abc, abc, xyz, pqr, abc, pqr, def, xyz, pqr)$ then $\bar{\mathbf{s}} = (1, 1, 2, 3, 1, 3, 4, 2, 3)$. SP is the *s-term support size* which is the number of occurrences of the s-term k-gram in the database, i.e. $\mathsf{SP}[i] = |\mathsf{DB}(\mathbf{s}[i])|$. Finally, RP is the *results pattern*, i.e. $\mathsf{RP}[i]$ is the set of $(\mathsf{ind}, \mathsf{pos})$ pairs where ind is an identifier of document which matches the query q, and pos is a position of the s-term k-gram $\mathbf{s}[i]$ in that document.

• DP is the *Delta pattern* $\mathbf{\Delta}[i]$ of the queries, i.e. the shifts between k-grams in a query which result from the tokenization of the queries.

• IP is the *conditional intersection pattern*, which is a Q by Q table IP defined as follows: $\mathsf{IP}[i, j] = \emptyset$ if $i = j$ or $\mathbf{x}[i] \neq \mathbf{x}[j]$. Otherwise, $\mathsf{IP}[i, j]$ is the set of all triples $(\mathsf{ind}, \mathsf{pos}, \mathsf{pos}')$ (possibly empty) s.t. $(\mathsf{ind}, \mathsf{pos}) \in \mathsf{DB}(\mathbf{s}[i])$, $(\mathsf{ind}, \mathsf{pos}') \in \mathsf{DB}(\mathbf{s}[j])$, and $\mathsf{pos}' = \mathsf{pos} + (\mathbf{\Delta}[i] - \mathbf{\Delta}[j])$.

Understanding Leakage Components. Parameter N is the size of the meta-data, and leaking such a bound is unavoidable. The equality pattern $\bar{\mathbf{s}}$, which leaks repetitions in the s-term k-gram of different substring queries, and the s-term support size SP, which leaks the total number of occurrences of this s-term in the database, are both a consequence of the optimized search that singles out the s-term in the query, which we adopt from the conjunctive SSE search solution of [6]. RP is the result of the query and therefore no real leakage in the context of SSE. Note also that the RP over-estimates the information $\mathcal{E}$ observes, because $\mathcal{E}$ observes only a pointer to the encrypted document, and a pointer to the encrypted tuple storing a unique $(\mathsf{ind}, \mathsf{pos})$ pair, but not the pair $(\mathsf{ind}, \mathsf{pos})$ itself. DP reflects the fact that our protocols leak the relative shifts Δ between k-grams which result from tokenization of the searched string. If tokenization was canonical, and divided a substring into k-grams based only on the substring length, the shifts Δ would reveal only the substring length. (Otherwise, see below for how Δ's can be hidden from $\mathcal{E}$.)

The IP component is the most subtle. It is a consequence of the fact that when processing the $\mathbf{q}[i]$ query $\mathcal{E}$ computes the (pseudo)random function $F_G(\mathbf{x}[i], \mathsf{ind}, \mathsf{pos} + \mathbf{\Delta}[i])$ for all $(\mathsf{ind}, \mathsf{pos}) \in \mathsf{DB}(\mathbf{s}[i])$, and hence can see collisions in it. Consequently, if two queries $\mathbf{q}[i]$ and $\mathbf{q}[j]$ have the same x-gram then for any document ind which contains the s-grams $\mathbf{s}[i]$ and $\mathbf{s}[j]$ in positions, respectively, pos and $\mathsf{pos}' = \mathsf{pos} + (\mathbf{\Delta}[i] - \mathbf{\Delta}[j])$, server $\mathcal{E}$ can observe a collision in F_G and triple $(\mathsf{ind}, \mathsf{pos}, \mathsf{pos}')$ will be included in the IP leakage. Note, however, that $\mathsf{IP}[i, j]$ defined above overstates this leakage, because $\mathcal{E}$ does not learn the $\mathsf{ind}, \mathsf{pos}, \mathsf{pos}'$ values themselves, but only establishes a link between two *encrypted* tuples, one containing $(\mathsf{ind}, \mathsf{pos})$ in $\mathsf{TSet}(\mathbf{s}[i])$ and one containing $(\mathsf{ind}, \mathsf{pos}')$ in $\mathsf{TSet}(\mathbf{s}[j])$. To visualize the type of queries which will trigger this leakage, take $k = 3$, $\mathbf{q}[i] =$ *MOTHER*, $\mathbf{q}[j] =$ *OTHER*, and let $\mathbf{q}[i]$ and $\mathbf{q}[j]$ tokenize with a common x-gram, e.g. $T(\mathbf{q}[i]) = (\text{MOT}, (\text{HER}, 3))$ and $T(\mathbf{q}[j]) = (\text{OTH}, (\text{HER}, 2))$. The $\mathsf{IP}[i, j]$ leakage will contain tuple $(\mathsf{ind}, \mathsf{pos}, \mathsf{pos}')$ for $\mathsf{pos}' = \mathsf{pos} + (\mathbf{\Delta}[i] - \mathbf{\Delta}[j]) = \mathsf{pos} + 1$ iff record $\mathsf{DB}[\mathsf{ind}]$ contains 3-gram $\mathbf{s}[i] = \text{MOT}$ at position pos and 3-gram $\mathbf{s}[j] = \text{OTH}$ at position $\mathsf{pos} + 1$, i.e. iff it contains substring MOTH.

Theorem 4. *Protocol SUB-SSE-OXT (restricted to substrings which tokenize into two k-grams) is adaptively $\mathcal{L}_{\mathcal{E}}$-semantically-secure against malicious server $\mathcal{E}$, assuming the security of the PRF's, the encryption scheme*

Enc, *and the* TSet *scheme, the random oracle model for hash functions, and the q-DDH assumption on the group G of prime order.*

The proof of Theorem 4 is included in Appendix C.

Hiding Deltas. Since the tokenizer T should pick the least frequent k-gram as an s-gram, the information on which k-gram was chosen, which is visible from the vector of Δ's, can leak some sensitive statistics about the substring term. For example, if the tokenizer chooses the s-gram based on the k-gram frequency statistics, but then determines all the x-grams in a canonical way, then there are $n-k+1$ ways of tokenizing an n-character substring, hence $\mathcal{E}$ learns to which of the $n-k+1$ partitions the client's substring term belongs. If this moderate information leakage is unacceptable, it can be eliminated entirely at a moderate cost incurred by a Δ-hiding variant of the Search protocol. This can be done by relying on a multiplicative homomorphism of either ElGamal or linear encryption to create a multiplicative sharing of xind^{Δ} without revealing Δ to $\mathcal{E}$, and then combine it with the multiplicative sharing of $\mathsf{xind}^{\mathsf{pos}}$ in the xtag computation.

6 Implementation and Performance

Here we provide testing and performance information for our prototype implementation of the range and SUB-SSE-OXT protocols described in Sections 3 and 4.1. The results confirm the scalability of our solutions to very large databases and complex queries. The prototype is an extension of the OXT implementation of [5]. Both the description of the changes and performance information are limited, to the extent possible, to the protocols introduced in this paper. An extensive evaluation of the prototype is outside of the scope of this paper as it would be highly dependent on previous work.

Prototype Summary. The three components of our system are the preprocessor, the server, and the client. The preprocessor generates the encrypted database from the cleartext data. The client, which implements a representative set of SQL commands, 'encrypts' end-user requests and 'decrypts' server responses. The server uses the encrypted database to answer client SELECT-type queries or expands the encrypted database on UPDATE, INSERT, and (even) DELETE queries [5].

To support range queries (see Section 3) the Boolean-query OXT prototype was augmented with generation of range-specific TSet's at pre-processing, and with range-specific authorization and range-cover computation at the client. Support for substring and wildcard queries required redesigning pre-processing to take into account the k-gram position information, adding support for 'k-gram'-based record tokenization to the client, and changing the Search protocol to support *position-enhanced* computation (see Section 4) and authorization. A few other changes were necessary in order to continue handling UPDATE, INSERT and DELETE queries. These extensions largely follow the update mechanics outlined in [5], with the addition of a new PSet^{+} data structure.

To match the SQL standard, our implementation uses the `LIKE` operator syntax for substring and wildcard queries: `'_'` (`'%'`) represent single-character (variable-length) wildcards and the query must match the complete field, i.e, unless a query must match the prefix (suffix) of fields, it should begin (end) with a `'%'`.

Experimental Platform. The experiments described in the remainder of this section were run on two Dell PowerEdge R710 systems, each one of them equipped with two Intel Xeon X5650 processors, 96GB RAM (12x8 1066MHz), an embedded Broadcom 1GB Ethernet with TOE and a PERC H700 RAID controller with a 1GB Non-Volatile Cache and 1 or 2 daisy-chained MD1200 disk controllers each with 12 2TB 7.2k RPM Near-Line SAS hard drives configured for Raid 6 (19TB and 38TB total storage per machine).

An automated test harness, written by an independent evaluator [26], drives the evaluation, including the set of queries and the dataset used in the experiments.

Dataset. The synthetic dataset used in the reported experiments is a US census-like table with twenty one columns of standard personal information, such as name (first, last), address (street, city, state, zipcode),

SSN, etc. The values in each column are generated according to the distributions in the most recent US census. In addition, the table has one XML column with at most 10000 characters, four text columns with varying average lengths (a total of at most 12300 characters or $\approx$ 2000 words), and a binary column (payload) with a maximum size of 100KB. Our system can perform structured queries on data in all but the XML and binary columns. The size of (number of records in) the table is a parameter of the dataset generator. We tested on a wide variety of database sizes, but we focus our results on a table with 100 million records or 10TBytes.

Experimental Methodology. In the initial step, the encrypted database is created from the cleartext data stored in a MariaDB (a variant of open-source MySQL RDBMS) table. Then, a per-protocol collection of SQL queries, generated by the harness to test its features, is run against the MariaDB sever and against our system. The queries are issued sequentially by the harness, which also records the results and the execution times of each query. Finally, the harness validates the test results by comparing the result sets from our system and from the MariaDB server. Not only does this step validate the correctness of our system, it also ensures our system meets our theoretical false positive threshold over large, automatically generated, collections of queries.

Encrypted Index. We built a searchable index on all personal information columns (twenty one) in the plaintext database but we only use a small subset of these indexes for the following experiments. Note that we support substring and wildcard queries simultaneously over a given column using a single shared index. We built a substring-wildcard index for four columns (average length of 12 characters) and a range index for five columns of varying types (one 64 bit integer, one date, one 32 bit integer, and one enum). Each substring-wildcard index was constructed with a single k value of 4. Each range index has a granularity of one. For the date type, this equates to a day. We support date queries between 0-01-01 and 9999-12-31, and integer queries between 0 and integer max ($2^{32}-1$ or $2^{64}-1$).

On average each record generates 256.6 document-keyword pairs (tuples) among all indexes. This equates to a total encrypted index for our largest database of $\approx$ 20TB. We back our XSet by an in memory Bloom filter with a false positive rate of 2^{-12}; this allows us to save unnecessary disk accesses and it does not influence the false positive rate of the system.

Performance Costs by Query Type. Our complex query types have both increased storage overhead and query time costs as compared to the keyword only implementation of [5]. In order to support substring and wildcard queries on a column, we must store additional tuples: for a record of length l (for the indexed field) we must store $(l-k)+3$ tuples. Note that we must pay this cost for each k we chose to create the index for. The choice of k also affects query time performance. For a query q, it's performance is linearly dependent on the number of tokens generated by the tokenization $T(q)$. A smaller k results in a larger number of tokens. Specifically for subsequence queries there will be $\lceil |q|/k \rceil$-1 xtokens[3]. k also impacts the number of matching documents returned by the s-term. A larger k results in a higher entropy s-term. The choice of k is a careful trade-off between efficiency and flexibility.

Range queries incur storage costs linear in their bit depth. Specifically, $log_2(max\_value)$ tuples are stored for a record for each range field. Notably for date fields this value is 22. In addition we implemented the *canonical cover* from Section 3, which results in up to $2 * log_2(max\_value)$ disjunctions.

Phrase queries incur storage costs linear in the total number of words in a column. Specifically for every record with n free-text words, the index stores n tuples. Although phrase queries and free-text queries can be supported via the same index, we have to pay the marginally higher price of the phrase index in which we must store even repeated words.

Encrypted Search Performance. We illustrate the performance of our system using the latency (i.e., total time from query issuing to completion) of a large number of representative `SELECT` queries. The independent evaluator selected a representative set of queries to test the correctness and performance of the range,

[3] Wildcard queries pay a similar overhead, related to the size of each contiguous substring within the query.

substring and wildcard queries (phrase queries were not implemented). The two leftmost columns in Table 1 show how many unique queries were selected for each query type. The third, fourth and fifth columns characterize the 95% fastest queries of each type. Finally, the rightmost column shows the percentage of queries that complete in less than two minutes.

All queries follow the pattern `SELECT id FROM CensusTable WHERE ...`, with each query having a specific WHERE clause. Range-type queries use the `BETWEEN` operator to implement two-sided comparison on numerical fields as well as date and enum fields. Specific queries were chosen to assess the performance effect of differing result set sizes and range covers. In particular, in order to assess the effect of cover size, queries with moderate result sets (of size under 10,000) were chosen while the size of cover sets range from a handful to several dozens. The results show relatively homogeneous latencies (all under 0.8 seconds) in spite of the large variations in cover size, highlighting the moderate effect of cover sizes.

Our instantiation of SUB-SSE-OXT includes extensions for supporting substring and wildcard searches simultaneously. However, to evaluate the effects of each specific extension we measure them individually. Both query types use the `LIKE` operator in the `WHERE` clause.

Substring queries use the variable-length wildcard '%' at the beginning, at the end, or at both ends of the `LIKE` operand, as in `WHERE city LIKE '%ttle Falls%'`. Wildcard queries use the single-character wildcard ('_') anywhere in the `LIKE` operand, provided the query criteria dictated by k is still met.

In addition, we noticed that the choice of s-gram dominates the latency of the substring queries. Our analysis shows that low performing queries can often be tied to high-frequency s-terms (e.g., "ing " or "gton "), which are associated with large Tsets. By default, the current implementation uses the first k characters in the pattern string as s-gram. Thus, implementing a tokenization strategy guided by the text statistics (which we leave for future work) can significantly reduce query latency for many of the slow performers. To estimate the potential benefits of such a strategy, we added the `STARTAT 'n'` option to the `LIKE 'pattern'` operator, where `'n'` is the starting position of the s-gram. Experiments using the `'%gton Colle%'` pattern show latency improvements of up to 32 times when the s-gram starts at the third or fourth character in the pattern string.

Query type	# of queries	fastest 95% avg	min	max	% ≤ 120 secs
range	197	.37	.19	.61	100
substring	939	40	0.22	166	93
wildcard	511	31.22	6.7	224	93

Table 1. Latency (in secs) for 10 TByte DB, 100M records, 25.6 billion record-keyword pairs

Comparison to Cleartext Search. Here we include the most relevant aspects of the performance comparison between our prototype and MariaDB. In the case of the 100 million record database, for $\approx 45\%$ of the range queries, the two systems have very similar performance. For the remaining 55%, our system is increasingly (up to 500 times!) faster. The large variations in MariaDB performance seem to arise from its reliance on data (and index) caching, which is hindered by large DBs. In contrast, our system issues between $\log_2 s$ and $2\log_2 s$ disk accesses *in parallel* (where s is the size of the cover). On smaller census databases (with fewer records) that fit in RAM, MariaDB outperforms our system, sometimes by more than one order of magnitude, although in this case all query latencies (ours and MariaDB's) are under a second. Additionally, for substring and wildcard queries and the largest, 100 million records, database our system always outperforms MariaDB, admittedly due to MariaDB's lack of support for a specialized-index based substring search. Instead, it often scans the dataset to resolve queries involving the `LIKE` operator.

7 Conclusion

This work presents a significant advance in the ability to run truly complex queries on encrypted data in a variety of operational and trust models. Specifically, we augmented the capabilities of the OXT protocol from the works of [6,13,5] to support substring, wildcard, phrase and range queries, and to allow any combination of these query types under boolean expressions. By leveraging and expanding the underlying machinery of OXT we are able to build on the impressive scalability of the protocol, and while the new query types carry costs in performance and storage, we demonstrated their practicality through a prototype implementation tested under large scale databases by an independent evaluator. One important conclusion is that searching on outsourced encrypted data with significant functionality and privacy-preserving properties is practical today even for large databases. Hopefully, we will see the actual use of these technologies in the near future.

Acknowledgment

An earlier version of this paper was published as [11]. The research described was conducted while the authors were affiliated with IBM Research and the University of California, Irvine and were supported by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior National Business Center (DoI/NBC) contract number D11PC20201. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoI/NBC, or the U.S. Government.

References

1. Boneh, D., Boyen, X.: Efficient selective-id secure identity-based encryption without random oracles. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT. Lecture Notes in Computer Science, vol. 3027, pp. 223–238. Springer (2004) 13
2. Boneh, D., Boyen, X., Shacham, H.: Short group signatures. In: Advances in Cryptology–CRYPTO 2004. pp. 41–55. Springer (2004) 13
3. Boneh, D., Boyen, X., Shacham, H.: Short group signatures. In: Franklin, M. (ed.) Advances in Cryptology – CRYPTO 2004. Lecture Notes in Computer Science, vol. 3152, pp. 41–55. Springer, Berlin, Germany, Santa Barbara, CA, USA (Aug 15–19, 2004) 25
4. Boneh, D., Waters, B.: Conjunctive, subset, and range queries on encrypted data. In: Theory of cryptography, pp. 535–554. Springer (2007) 3
5. Cash, D., Jaeger, J., Jarecki, S., Jutla, C., Krawczyk, H., Rosu, M.C., Steiner, M.: Dynamic searchable encryption in very large databases: Data structures and implementation. In: Symposium on Network and Distributed Systems Security (NDSS 2014) (2014) 1, 3, 9, 13, 15, 16, 18
6. Cash, D., Jarecki, S., Jutla, C., Krawczyk, H., Roşu, M.C., Steiner, M.: Highly-scalable searchable symmetric encryption with support for boolean queries. In: Advances in Cryptology–CRYPTO 2013, pp. 353–373. Springer (2013) 1, 2, 3, 4, 8, 9, 12, 13, 14, 18, 24, 27
7. Chang, Y.C., Mitzenmacher, M.: Privacy preserving keyword searches on remote encrypted data. In: Ioannidis, J., Keromytis, A., Yung, M. (eds.) ACNS 05: 3rd International Conference on Applied Cryptography and Network Security. Lecture Notes in Computer Science, vol. 3531, pp. 442–455. Springer, Berlin, Germany, New York, NY, USA (Jun 7–10, 2005) 1
8. Chase, M., Kamara, S.: Structured encryption and controlled disclosure. In: Abe, M. (ed.) Advances in Cryptology – ASIACRYPT 2010. Lecture Notes in Computer Science, vol. 6477, pp. 577–594. Springer, Berlin, Germany, Singapore (Dec 5–9, 2010) 1, 2
9. Chase, M., Shen, E.: Pattern matching encryption. Cryptology ePrint Archive, Report 2014/638 (2014), http://eprint.iacr.org/ 3

10. Curtmola, R., Garay, J.A., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. In: Juels, A., Wright, R.N., Vimercati, S. (eds.) ACM CCS 06: 13th Conference on Computer and Communications Security. pp. 79–88. ACM Press, Alexandria, Virginia, USA (Oct 30 – Nov 3, 2006) 1, 2
11. Faber, S., Jarecki, S., Krawczyk, H., Nguyen, Q., Rosu, M., Steiner, M.: Rich queries on encrypted data: Beyond exact matches. In: Proceedings of the Twentieth European Symposium on Research in Computer Security (ESORICS). Lecture Notes in Computer Science, vol. 9327, pp. 123–145. Springer-Verlag, Berlin Germany (2015), Part II 1, 18
12. Goh, E.J.: Secure indexes. Cryptology ePrint Archive, Report 2003/216 (2003), http://eprint.iacr.org/ 1
13. Jarecki, S., Jutla, C., Krawczyk, H., Rosu, M., Steiner, M.: Outsourced symmetric private information retrieval. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. pp. 875–888. ACM (2013) 1, 2, 3, 4, 5, 12, 13, 18, 24, 25, 26, 28
14. Kamara, S., Papamanthou, C.: Parallel and dynamic searchable symmetric encryption. In: Sadeghi, A.R. (ed.) FC 2013: 17th International Conference on Financial Cryptography and Data Security. Lecture Notes in Computer Science, vol. 7859, pp. 258–274. Springer, Berlin, Germany, Okinawa, Japan (Apr 1–5, 2013) 1
15. Kamara, S., Papamanthou, C., Roeder, T.: Dynamic searchable symmetric encryption. In: Yu, T., Danezis, G., Gligor, V.D. (eds.) ACM CCS 12: 19th Conference on Computer and Communications Security. pp. 965–976. ACM Press, Raleigh, NC, USA (Oct 16–18, 2012) 1
16. Kiayias, A., Tang, Q.: How to keep a secret: leakage deterring public-key cryptosystems. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. pp. 943–954. ACM (2013) 6
17. Kurosawa, K., Ohtaki, Y.: UC-secure searchable symmetric encryption. In: Keromytis, A.D. (ed.) FC 2012: 16th International Conference on Financial Cryptography and Data Security. Lecture Notes in Computer Science, vol. 7397, pp. 285–298. Springer, Berlin, Germany, Kralendijk, Bonaire (Feb 27 – Mar 2, 2012) 1
18. van Liesdonk, P., Sedhi, S., Doumen, J., Hartel, P.H., Jonker, W.: Computationally efficient searchable symmetric encryption. In: Proc. Workshop on Secure Data Management (SDM). pp. 87–100 (2010) 1
19. Naveed, M., Prabhakaran, M., Gunter, C.A.: Dynamic searchable encryption via blind storage. In: 35th IEEE Symposium on Security and Privacy, 2014. pp. 639–654. IEEE Computer Society Press (2014) 1
20. Pappas, V., Vo, B., Krell, F., Choi, S., Kolesnikov, V., Keromytis, A., Malkin, T.: Blind Seer: A scalable private DBMS. In: 35th IEEE Symposium on Security and Privacy, 2014. pp. 359–374. IEEE Computer Society Press (2014) 1, 3
21. Popa, R.A., Redfield, C.M.S., Zeldovich, N., Balakrishnan, H.: CryptDB: Protecting confidentiality with encrypted query processing. In: Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP'11). ACM (Oct 2011) 3
22. Raykova, M., Vo, B., Bellovin, S.M., Malkin, T.: Secure anonymous database search. In: Proceedings of the 2009 ACM workshop on Cloud computing security. pp. 115–126. ACM (2009) 3
23. Shacham, H.: A cramer-shoup encryption scheme from the linear assumption and from progressively weaker linear variants. Cryptology ePrint Archive, Report 2007/074 (2007), http://eprint.iacr.org/ 13
24. Shi, E., Bethencourt, J., Chan, T.H., Song, D., Perrig, A.: Multi-dimensional range query over encrypted data. In: Security and Privacy, 2007. SP'07. IEEE Symposium on. pp. 350–364. IEEE (2007) 3
25. Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: 2000 IEEE Symposium on Security and Privacy. pp. 44–55. IEEE Computer Society Press, Oakland, California, USA (May 2000) 1
26. Varia, M., Price, B., Hwang, N., Hamlin, A., Herzog, J., Poland, J., Reschly, M., Yakoubov, S., Cunningham, R.K.: Automated assesment of secure search systems. Operating Systems Review 49(1), 22–30 (2015) 15

A On Canonical Covers

Here we present a proof of Theorem 1 as well as a procedure for computing canonical covers.

A.1 Computing a canonical cover

We first describe a simple procedure for generating a minimal (in the number of nodes) cover which we then use for computing a canonical cover.

Given a t-depth binary tree and a range $[a, b]$ included in $[0, 2^t - 1]$ we say that a tree node N is the next-max node for $[a, b]$ if N covers the maximal range (i.e., largest set of leaves) that includes the endpoint a and is fully contained in $[a, b]$.

The next-max algorithm. The algorithm starts with the given range $[a, b]$, then iteratively chooses the next-max node for the current range, adds the node to the cover, removes from the range the covered prefix, and goes on to choose the next-max node for the remaining suffix of the range. The next-max node can be selected by traversing the path from the leaf a to the root until a node N is reached whose cover exceeds b. The node added to the cover is the last node in the a path before reaching N (more efficient implementations, e.g., based on the binary representation of nodes, are possible).

We will call the cover generated by the next-max algorithm a next-max cover.

Lemma 1. *The next-max cover is minimal in the number of nodes.*

Proof. If a given cover C' has a smaller number of nodes than the next-max cover C, there must be two consecutive leaves in the range that are covered by different nodes N_1, N_2 in C but are under the same node N in C'. Thus, N is an ancestor of both N_1, N_2 whose covered leaves are all in range so the next-max algorithm should have chosen N before choosing N_1.

The canonical next-max algorithm. The canonical cover of a given range $[a, b]$ can be found as follows. We first compute the canonical profile for $n = b - a + 1$, then we apply the next-max algorithm from Section 2 except that we restrict the cover nodes to have heights as given by the canonical profile. That is, at each iteration we keep a set U of unused profile heights and we choose the next node as one that provides a maximal cover among nodes whose heights are in U.

A canonical cover for a range $[a, b]$ can be found by a simple procedure. Define the cover $c(v)$ of a tree node v as the interval of leaves under the subtree rooted at v. We say that v covers a prefix of $[a, b]$ if $c(v) = [a, b'], b' \leq b$. The procedure first computes the canonical profile for $n = b - a + 1$, sets U as the multi-set of heights defined by the profile, and sets R to the interval $[a, b]$. Then, from all nodes v that cover a prefix of R one selects the one with the largest cover $c(v)$ and such that $height(v) \in U$. Then, $height(v)$ is removed from U and the above procedure is repeated with interval R defined as $[a, b] \setminus \{c(v)\}$. When R becomes empty the selected nodes form a canonical cover.

Example: In a tree with 32 leaves, the next-max algorithm computes a minimal cover for range $[0, 19]$ as the nodes with label 0 (covers 0-15) and node 100 (covers 16-19), i.e., with profile $\{4, 2\}$. On the other hand, applying the next-max algorithm subject to the canonical profile $\{0, 0, 1, 2, 2, 3\}$ for $n = 20$ results in a cover with nodes $\{00, 010, 011, 1000, 10010, 10011\}$. While this profile is suboptimal for the range $[0, 19]$ we prove below that it is necessary for range $[1, 20]$.

Note: The ordering of nodes selected by the above procedure can leak information on the range's endpoints. Thus, a client choosing a canonical cover will present the cover heights to $\mathcal{D}$ in an independent order (e.g., sorted by decreasing heights).

Note that while the profile of a cover generated by the above algorithm for a given range size is independent of the ordering in which the cover nodes are found, this ordering may be different for different ranges of the same size. Therefore, it is important that when the client generates the range disjunction for authorization, it does so using an order that is independent of the order in which these nodes were found by the above algorithm (it can order them randomly, lexicographically, by depth, etc.).

A.2 Proof of Theorem 1

Definition 3. *A cover is called* two-bounded *if no height in its profile repeats more than twice. A cover* $\{c_1, \ldots, c_\ell\}$ *(with nodes ordered from left to right) is called* convex *if for any* $i < j < k$*, if* $height(c_i) \geq$

height(c_k) *then* *height*$(c_i) \geq$ *height*$(c_j) \geq$ *height*(c_k) *(i.e., a convex range will have a profile with a non-decreasing prefix followed by a non-increasing suffix).*

We will make use of the following:

Observation. After placing a node of height i in a cover, one can select the next node to be of height j for any $j \leq i$ (assuming the uncovered range is of size at least 2^j). Indeed, the node of height i defines a subtree of size 2^i after which another subtree of size 2^i starts, but then also a subtree of size 2^j starts at that boundary for any $j \leq i$.

Lemma 2. *A next-max cover is two-bounded and convex.*

Proof. Every node in a next-max cover (i.e., a cover produced by the next-max algorithm) has the property that its sibling is not in the cover (otherwise one can add their parent), but if one had three nodes with same height, the middle one would have a sibling in the cover. Thus, the cover is two-bounded.

Consider any three values i, j, k in the profile of a next-max cover, such that a node of height j was placed between a node of height i and a node of height k. We need to prove that if $j \leq i$ then $k \leq j$. If we had $k > j$ and $k \leq i$, then by the above observation the next-max algorithm would have placed the node of height k before the one of height j. If $k > j$ and $k > i$, we consider two sub-cases: $i > j$ and $i = j$. In the first case, a subtree of size 2^i is followed by a subtree of size $2^j, j < i$, which does not end in a 2^i boundary, hence not in a 2^k boundary ($k > i$), so it can't be followed by a node of height k. In the second case, we have a 2^i subtree followed by another 2^i subtree and then a 2^k one. The second 2^i subtree ends at a 2^k boundary hence also at a 2^{i+1} boundary (since $k \geq i+1$), so the next-max algorithm would have chosen a $i+1$-height (or larger) node instead of the first i-height node.

Hereafter, we focus our attention on two-bounded covers (which the above lemma shows to exist for any range and which also produce universal covers as we will see below).

Lemma 3. *For any $L > 0$, ranges of size $n = 2^L - 1$ have a unique two-bounded cover and its profile is canonical, i.e., $\{0, 1, 2, \ldots, L-1\}$.*

Proof. Let C be a two-bounded cover of a given range of size $n = 2^L - 1$ with profile P. Let P_1, P_2 be sets defined, respectively, as the set of all the values in P without repetitions and the set of values in P that repeat (we assume for contradiction that P_2 is not empty). Let n_1 be the sum of 2^i over all i in P_1 and let n_2 be the sum of 2^i over all i in P_2. We have that $n = n_1 + n_2$. By construction the binary representations of n_1 and n_2 have (at least) one bit in common (corresponding to the repeating values). Let i be the least significant of these common bits. Then, bit i is 0 in $n = n_1 + n_2$, contradicting the fact that in the binary representation of n all bits (0 to $L-1$) are 1's. Thus P contains no repetitions and therefore it must contain exactly the values $\{0, 1, 2, \ldots, L-1\}$.

Lemma 4. *Any range has a convex canonical cover.*

Proof. We prove the lemma by induction on the number of repeated elements in the canonical profile.

Base step: If no value in the canonical profile repeats then the profile is $\{0, 1, 2, \ldots, L-1\}$ which means $n = 2^L - 1$ for which we know that a canonical cover exists and is convex (convexity follows from Lemma 3 that shows that the canonical cover is the only two-bounded cover for $n = 2^L - 1$ – hence it is also a next-max cover – and from Lemma 2 that shows that such cover is convex).

Induction step: If a value in the profile repeats, consider the least value that repeats, say i. The profile for $n - 2^i$ has one less repeated value, hence by induction the range $[a, b - 2^i]$ has a convex canonical range. The idea is to add a i-height node to the $[a, b - 2^i]$ cover. But where does 2^i go? Choose the last (rightest)

node, call it N, in the $[a, b-2^i]$ cover that has a height at least i, then the claim is that we can add the new i-height node after N. This is possible by the observation preceding Lemma 2. Also note that by the choice of N, the height of nodes after N in the $[a, b-2^i]$ cover are all of height less than i, thus by convexity they form a non-increasing sequence. Hence the new node will fit immediately after N. Note that the obtained cover is canonical and convex, proving the claim.

Corollary 1. *The canonical cover can be computed by the canonical next-max algorithm described above.*

Proof. The proof follows the inductive argument from the proof of Lemma 4. For the base case of $n = 2^L - 1$ we know that the canonical cover is obtained via the next-max algorithm using the canonical profile while in the inductive step we always choose the least available height and place it as to the right as possible.

Lemma 5. *For all n, the only two-bounded cover for range $[1, n]$ is the canonical cover (hence the canonical cover guaranteed by Lemma 4 is also minimal – and unique among universal two-bounded covers).*

Proof. Let $L = \lfloor \log(n+1) \rfloor$. Let $n_1 = 2^L - 1, n_2 = n - n_1$. Consider any cover C of $[1, n]$ and look at the nodes in C that are needed to cover $[1, n_1]$. In principle, these nodes could cover beyond n_1, however this is not possible as it would require a node that covers both leaves n_1 and $n_1 + 1$. However the smallest subtree covering these two leaves is of size $2^{L+1} > n$. Thus, $[1, n_1]$ must be covered exactly by a prefix C_1 of C and the range $[n_1 + 1, n]$ must be covered exactly by remaining suffix C_2 of C. The profile of C_1 must be $\{0, 1, 2, \ldots, L-1\}$ by Lemma 3 while the profile P_2 of C_2 is a subset of $\{0, 1, 2, \ldots, L-1\}$ without repetitions (otherwise we'd have an element in C repeating more than twice). n_2 is the sum of 2^i over i in P_2 and since these i's do not repeat then we get that P_2 is the set of '1' positions in the binary representation of n_2.

Lemmas 4 and 5 prove Theorem 1.

B Substring SSE Extensions

B.1 MIXED-SSE-OXT: Substring Terms in General Boolean Formula Queries

We show how to generalize the SUB-SSE-OXT protocol so that it supports conjunctions (or indeed, any Boolean formula) whose atomic terms can be formed by any number of substring search terms and/or exact keyword terms, with flexible choice of s-term as either one of the exact keyword terms or a k-gram in one of the substring terms. The resulting protocol, shown in Figure 2, is called MIXED-SSE-OXT because it freely *mixes* substring and exact keyword search terms.

Protocol SUB-SSE-OXT stores in each tuple of $\mathbf{T}[\mathsf{kg}]$ the (encrypted) values of xind and $\mathsf{xind}^{\mathsf{pos}}$. In MIXED-SSE-OXT we decouple these two values: We store in $\mathbf{T}[\mathsf{kg}]$ only the xind values, and we create a separate data structure for the (encrypted) position-related values $\mathsf{xind}^{\mathsf{pos}}$. By shifting the encrypted $\mathsf{xind}^{\mathsf{pos}}$ values to another data structure, we can combine the k-gram and keyword indexes together so that for all $a \in (\mathsf{KG} \cup \mathsf{W})$ and all $\mathsf{ind} \in \mathsf{DB}(a)$, list $\mathbf{T}[a]$ will include an entry (y, e) at some position c s.t. $\mathsf{e} = \mathsf{Enc}(K_e, (\mathsf{ind}|\mathsf{rdk}))$ and $y = \mathsf{xind}/z$, for $\mathsf{xind} = F_p(K_I, \mathsf{ind})$, $\mathsf{rdk} = \mathsf{RDK}[\mathsf{ind}]$, $z = F_p(K_z, c)$, and keys K_z, K_e derived from $\mathsf{strap} = F_\tau(K_S, a)$ similarly as in the SUB-SSE-OXT. Treating the k-gram and exact keywords in this uniform way allows us to compose the substring search capability of SUB-SSE-OXT with the Boolean search on exact keywords of SSE-OXT.

For storing the position-related information for resolving subsequence queries, namely the encrypted $\mathsf{xind}^{\mathsf{pos}}$ values, we use a separate look-up table $\mathbf{P}$. Let F'_G denote a "truncated" version of function F_G from equation (1), namely

$$F'_G((K_X, K_I), (\mathsf{kg}, \mathsf{ind})) = g^{F_p(K_X, \mathsf{kg}) \cdot F_p(K_I, \mathsf{ind})} \tag{2}$$

$\underline{\mathsf{Setup}}(\mathsf{DB}, \mathsf{RDK})$

- Pick keys K_S, K_T for PRF F_τ, K_I, K_X for PRF F_p, parse DB as $(\mathsf{ind}_i, \mathsf{W}_i)_{i=1}^{d}$ and $(\mathsf{ind}_i, \mathsf{pos}_i, \mathsf{kg}_i)_{i=1}^{d'}$.
- Initialize $\mathbf{T}$ and $\mathbf{P}$ to empty arrays and XSet to an empty set. For each $w \in \mathsf{W} \cup \mathsf{KG}$ do the following:
  - Set $\mathsf{strap} \leftarrow F_\tau(K_S, w)$ and $(K_z, K_e) \leftarrow (F_\tau(\mathsf{strap}, 1), F_\tau(\mathsf{strap}, 2))$.
  - For $c = 1, \ldots, |\mathsf{DB}(w)|$, for ind a c-th element in $\mathsf{DB}(w)$ (randomly permuted) do:
    - Set $\mathsf{rdk} \leftarrow \mathsf{RDK}(\mathsf{ind})$, $\mathsf{e} \leftarrow \mathsf{Enc}(K_e, (\mathsf{ind}|\mathsf{rdk}))$, $\mathsf{xind} \leftarrow F_p(K_I, \mathsf{ind})$.
    - Set $z \leftarrow F_p(K_z, c)$, $y \leftarrow \mathsf{xind} \cdot z^{-1}$, and append (e, y) to $\mathbf{T}[w]$.
    - If $w \in \mathsf{W}$ then set $\mathsf{xtag} \leftarrow g^{F_p(K_X, w) \cdot \mathsf{xind}}$ and add xtag to XSet.
    - If $w \in \mathsf{KG}$ then for $c' = 1, \ldots, |\mathsf{DB}(\mathsf{ind}, w)|$, for pos a c'-th element in $\mathsf{DB}(\mathsf{ind}, w)$ do:
      - Set $\mathsf{ptag} \leftarrow g^{F_p(K_X, w) \cdot \mathsf{xind}}$, $K_u \leftarrow F_\tau(\mathsf{strap}, 3, \mathsf{ptag})$, $u \leftarrow F_p(K_u, c')$, $v \leftarrow \mathsf{xind}^{\mathsf{pos}} \cdot u^{-1}$, append v to $\mathbf{P}[(w, \mathsf{ind})]$.
      - Set $\mathsf{xtag} \leftarrow g^{F_p(K_X, w) \cdot F_p(K_I, \mathsf{ind})^{\mathsf{pos}}}$ and add xtag to XSet.
- Set $\mathsf{TSet} \leftarrow \mathsf{TSetSetup}(\mathbf{T}, \langle F_\tau \rangle, K_T)$ and $\mathsf{PSet} \leftarrow \mathsf{TSetSetup}(\mathbf{P}, (K_X, K_I))$.
- Output $K = (K_S, K_X, K_T)$, $\mathsf{EDB} = (\mathsf{TSet}, \mathsf{XSet}, \mathsf{PSet})$.

$\underline{\mathsf{Search}}$ protocol

- Client $\mathcal{C}$, on input key $K = (K_S, K_X, K_T)$ and conjunctive query $\bar{w}$ consisting of a single (for simplicity) substring-search term q, tokenized as $(\mathsf{kg}_1, (\mathsf{kg}_2, \Delta_2), \ldots, (\mathsf{kg}_\mathsf{h}, \Delta_\mathsf{h}))$, and exact-keyword terms $w_1, \ldots, w_n$, with w_1 as the s-term of the query:
  - Set $\mathsf{stag} \leftarrow F_\tau(K_T, w_1)$, $\mathsf{strap} \leftarrow F_\tau(K_S, w_1)$, $(K_z, K_e) \leftarrow (F_\tau(\mathsf{strap}, 1), F_\tau(\mathsf{strap}, 2))$.
  - Set $\{\mathsf{xtrap}_i \leftarrow g^{F_p(K_X, w_i)}\}_{i=2}^{n}$, $\{\mathsf{xtrap}_{\mathsf{kg}_i} \leftarrow g^{F_p(K_X, \mathsf{kg}_i)}\}_{i=1}^{\mathsf{h}}$, and $\mathsf{strap}_{\mathsf{kg}_1} \leftarrow F_\tau(K_S, \mathsf{kg}_1)$.
  - Send $(\mathsf{stag}, \Delta_2, \ldots, \Delta_\mathsf{h})$ to $\mathcal{E}$.
  - For $c = 1, 2, \ldots$, until $\mathcal{E}$ sends stop_c do:
    - Set $z_c \leftarrow F_p(K_z, c)$; Set $\{\mathsf{xtoken}[c, i] \leftarrow (\mathsf{xtrap}_i)^{z_c}\}_{i=2}^{n}$ and $\mathsf{ptoken}[c] \leftarrow (\mathsf{xtrap}_{\mathsf{kg}_1})^{z_c}$.
    - Send $\mathsf{xtoken}[c] = (\mathsf{xtoken}[c, 2], \ldots, \mathsf{xtoken}[c, n], \mathsf{ptoken}[c])$ to $\mathcal{E}$.
    - On $\mathsf{ptag}[c]$ from $\mathcal{E}$, set $K_u \leftarrow F_\tau(\mathsf{strap}_{\mathsf{kg}_1}, 3, \mathsf{ptag}[c])$, and for $c' = 1, 2, \ldots$, until $\mathcal{E}$ sends $\mathsf{stop}_{c,c'}$ do:
      - Set $u_{c'} \leftarrow F_p(K_u, c')$ and $\left\{\mathsf{xtoken}_{\mathsf{kg}}[c, c', i] \leftarrow (\mathsf{xtrap}_{\mathsf{kg}_i})^{(z_c)^{\Delta_i} \cdot (u_{c'})}\right\}_{i=2}^{\mathsf{h}}$.
      - Send $\mathsf{xtoken}_{\mathsf{kg}}[c, c'] = (\mathsf{xtoken}_{\mathsf{kg}}[c, c', 2], \ldots \mathsf{xtoken}_{\mathsf{kg}}[c, c', \mathsf{h}])$ to $\mathcal{E}$.
- Server $\mathcal{E}$, on input $\mathsf{EDB} = (\mathsf{TSet}, \mathsf{XSet}, \mathsf{PSet})$, responds with a set ESet formed as follows:
  - On $(\mathsf{stag}, \Delta_2, \ldots, \Delta_\mathsf{h})$ from $\mathcal{C}$, retrieve $\mathbf{t} \leftarrow \mathsf{TSetRetrieve}(\mathsf{TSet}, \mathsf{stag})$.
  - For $c = 1, ..., |\mathbf{t}|$, on $\mathsf{xtoken}[c]$ from $\mathcal{C}$, retrieve c-th tuple (e, y) in $\mathbf{t}$, set $(\mathsf{OK}_c^1, \mathsf{OK}_c^2) \leftarrow (0, 0)$.
    - Set $\mathsf{OK}_c^1 \leftarrow 1$ if $\forall i = 2, \ldots, n : (\mathsf{xtoken}[c, i])^y \in \mathsf{XSet}$.
    - Set $\mathsf{ptag}[c] \leftarrow \mathsf{ptoken}[c]^y$, retrieve $\mathbf{p} \leftarrow \mathsf{PSet}[\mathsf{ptag}[c]]$, send $\mathsf{ptag}[c]$ to $\mathcal{C}$.
    - If $|\mathbf{p}| = 0$ send $\mathsf{stop}_{c,c'}$ to $\mathcal{C}$ and continue (to next c). Otherwise, for $c' = 1, \ldots, |\mathbf{p}|$ do:
      - On $\mathsf{xtoken}[c, c']$ from $\mathcal{C}$, retrieve c'-th element v from $\mathbf{p}$;
      - Set $\mathsf{OK}_c^2 \leftarrow 1$ if $\forall i = 2, \ldots, \mathsf{h} : (\mathsf{xtoken}_{\mathsf{kg}}[c, c', i])^{y^{\Delta_i} \cdot v} \in \mathsf{XSet}$; Send $\mathsf{stop}_{c,c'}$ to $\mathcal{C}$ when $c' = |\mathbf{p}|$.
    - If $(\mathsf{OK}_c^1, \mathsf{OK}_c^2) = (1, 1)$ then add e to ESet. When $c = |\mathbf{t}|$ send stop_c to $\mathcal{C}$.
- Client $\mathcal{C}$ computes $(\mathsf{ind}|\mathsf{rdk}) \leftarrow \mathsf{Dec}(K_e, \mathsf{e})$ for each e in ESet, and adds $(\mathsf{ind}, \mathsf{rdk})$ to its output.

Fig. 2. MIXED-SSE-OXT: SSE for Conjunctions of Multiple Substring and Exact Keyword Terms

For every $(\mathsf{kg}, \mathsf{ind})$ s.t. k-gram kg appears in $\mathsf{DB}[\mathsf{ind}]$, $\mathbf{P}[(\mathsf{kg}, \mathsf{ind})]$ stores a list of values $v = \mathsf{xind}^{\mathsf{pos}}/u$, where $\mathsf{xind} = F_p(K_I, \mathsf{ind})$, for each pos s.t. $(\mathsf{ind}, \mathsf{pos}) \in \mathsf{DB}(\mathsf{kg})$. The value u that masks c-th value $\mathsf{xind}^{\mathsf{pos}}$ is computed as $u = F_p(K_u, c)$ where $K_u = F_\tau(\mathsf{strap}_{\mathsf{kg}}, \mathsf{ptag}_{(\mathsf{kg},\mathsf{ind})})$ and $\mathsf{ptag}_{(\mathsf{kg},\mathsf{ind})} = F'_G((K_X, K_I), (\mathsf{kg}, \mathsf{ind}))$ (ptag serves a similar purpose as stag but for positioning information). Finally, we store $\mathbf{P}$ in another instance of the TSet data structure, called PSet, where a handle for identification and decryption of a list $\mathbf{P}[(\mathsf{kg}, \mathsf{ind})]$ is an output of $F'_G((K_X, K_I), \cdot)$ on $(\mathsf{kg}, \mathsf{ind})$, i.e. $\mathsf{ptag}_{(\mathsf{kg},ind)}$.

Lastly, the data-structure XSet will store the xtag values for both exact keywords and for (k-gram,position) pairs, i.e. it will contain values $F_G((K_X, K_I), (\mathsf{kg}, \mathsf{ind}, \mathsf{pos}))$ for all $\mathsf{kg} \in \mathsf{KG}$ and $(\mathsf{ind}, \mathsf{pos}) \in \mathsf{DB}(\mathsf{kg})$ and values $F'_G((K_X, K_I)(w, \mathsf{ind})$ for all $w \in \mathsf{W}$ and $\mathsf{ind} \in \mathsf{DB}(w)$.

The three data structures $(\mathsf{TSet}, \mathsf{PSet}, \mathsf{XSet})$ are used in Search to combine the substring processing in SUB-SSE-OXT with the Boolean search (on exact keywords) of the original SSE-OXT. Assume that $\mathcal{C}$'s query q is a conjunction of n exact query terms $w_1, \ldots, w_n$ and a single substring search term q' tokenized as $T(q') = (\mathsf{kg}_1, (\Delta_2, \mathsf{kg}_2), \ldots, (\Delta_\mathsf{h}, \mathsf{kg}_\mathsf{h}))$. Assume also that the exact keyword w_1 is chosen as an s-term. All these assumptions are not necessary and are used solely to simplify the protocol description below. Client $\mathcal{C}$ sends $\mathsf{stag}_{w_1} = F_T(K_T, w_1)$ to $\mathcal{E}$, who uses it to retrieve $\mathbf{t} = \mathbf{T}[w_1]$ from TSet. For each (encrypted) ind in $\mathbf{t}$, $\mathcal{C}$ and $\mathcal{E}$ perform the following: First they compute (in parallel, and following the combined operations of SSE-OXT and SUB-SSE-OXT) values $\mathsf{ptag}_{(\mathsf{kg}_1,\mathsf{ind})} = F'_G((K_X, K_I), (\mathsf{kg}_1, \mathsf{ind}))$ and $\mathsf{xtag}_{(w_i,\mathsf{ind})} = F'_G((K_X, K_I), (w_i, \mathsf{ind}))$ for $i = 2, \ldots, n$. If $\mathsf{xtag}_{(w_i,\mathsf{ind})} \notin \mathsf{XSet}$ for any $i = 2, \ldots, n$ or if the list $\mathbf{p} = \mathbf{P}[\mathsf{ptag}_{(\mathsf{kg}_1,\mathsf{ind})}]$ is empty, we can conclude that $\mathsf{ind} \notin \mathsf{DB}(q)$, and so $\mathcal{E}$ moves on to the next (encrypted) ind in $\mathbf{t}$. Otherwise, $\mathcal{E}$ sends back $\mathsf{ptag}_{(\mathsf{kg}_1,\mathsf{ind})}$ to $\mathcal{C}$, who uses it to derive the key K_u, and then for each (encrypted) $\mathsf{ind}^{\mathsf{pos}}$ value in $\mathbf{p}$, $\mathcal{C}$ and $\mathcal{E}$ jointly compute $\mathsf{xtag}_{(\mathsf{kg}_i,\mathsf{ind},\mathsf{pos}+\Delta_i)} = F_G((K_X, K_I), (\mathsf{kg}_i, \mathsf{ind}, \mathsf{pos} + \Delta_i))$ for $i = 2, \ldots, \mathsf{h}$. The latter computation is similar to the one described for SUB-SSE-OXT above, except that the (encrypted) ind value comes from list $\mathbf{t}$ while (encrypted) $\mathsf{ind}^{\mathsf{pos}}$ value comes from list $\mathbf{p}$. If $\mathsf{xtag}_{(\mathsf{kg}_i,\mathsf{ind},\mathsf{pos}+\Delta_i)} \in \mathsf{XSet}$ for *some* pos in the $\mathbf{p}$ list and all $i = 2, \ldots, \mathsf{h}$, we can conclude (except for probability of collision in F_G) that substring q' appears in $\mathsf{DB}[\mathsf{ind}]$ at position pos, and hence that $\mathsf{ind} \in \mathsf{DB}(q)$. Therefore in that case $\mathcal{E}$ sends ciphertext e corresponding to this ind to $\mathcal{C}$, which allows $\mathcal{C}$ to retrieve and decrypt record $\mathsf{DB}[\mathsf{ind}]$.

If query q involves more substring terms, each of them is processed as the substring term q' above. Since $\mathbf{T}[w]$ for $w \in \mathsf{W}$ and $\mathbf{T}[\mathsf{kg}]$ for $\mathsf{kg} \in \mathsf{KG}$ are implemented in the same way, an s-gram kg_1 from any substring search term can play the role of the s-term. Finally, the protocol can be easily modified to support any query expressed as $w \wedge \Phi(w_1, \ldots, w_n, q'_1, \ldots, q'_k)$, where w is either an exact keyword term or a substring term, $w_1, \ldots, w_n$ are exact keyword terms, $q'_1, \ldots, q'_k$ are substring terms, and Φ is any Boolean formula. The protocol cost is upper-bounded by $(n + \mathsf{h}_1 + \ldots + \mathsf{h}_k)$ exponentiations per party per each tuple in $\mathbf{T}[w]$, where h_i is the number of k-grams in the tokenization of q'_i.

Note that MIXED-SSE-OXT adds an extra communication round compared to SUB-SSE-OXT. However, $\mathcal{E}$ can generate its responses $\mathsf{ptag}_{(\mathsf{kg}_1,\mathsf{ind})}$ (one for each $c = 1, \ldots, |\mathbf{t}|$ and each substring term q_i, $i = 1, \ldots, k$) without retrieving list $\mathsf{PSet}[(\mathsf{kg}_1, \mathsf{ind})]$ from the disk (except for a small probability of a false positive error) if $\mathcal{E}$ keeps a Bloom filter which is small enough to fit in the memory and which allows $\mathcal{E}$ to check if any ptag value corresponds to a non-empty list in PSet.

B.2 MIXED-OSPIR-OXT: Substring and Keyword Search in OSPIR Setting

The MIXED-SSE-OXT protocol extends to the Multi-Client and OSPIR settings. Because of the similarity between MIXED-SSE-OXT with the SSE-OXT protocol of [6], we can re-use all the techniques of [13], which adopted protocol SSE-OXT to the Multi-Client and OSPIR settings. Here we recall these techniques briefly: First, we modify several PRF's used by $\mathcal{C}$ in MIXED-SSE-OXT so that they can be efficiently computed via an Oblivious PRF (OPRF) protocol between $\mathcal{C}$ and the data owner $\mathcal{D}$. (In particular we replace g^{xtrap} for $\mathsf{xtrap} = F_p(K_I, \mathsf{kg})$ with $\mathsf{xtrap} = H(\mathsf{kg})^{K_I}$, so e.g. $F_G(\mathsf{kg}, \mathsf{ind}, \mathsf{pos})$ becomes $\mathsf{xtrap}_{\mathsf{kg}}^{F_p(K_I,ind)^{\mathsf{pos}}}$.) The security of the OPRF protocol implies that all the individual terms in $\mathcal{C}$'s query are hidden from $\mathcal{D}$. However, just like

in the OSPIR-OXT protocol of [13], we ask client $\mathcal{C}$ to reveal the attributes of every term (exact keyword or k-gram) in its query, which allows $\mathcal{D}$ to apply an attribute-based access control policy. To make this policy enforcement effective, we replace PRF keys involved in these OPRF instances with an array of keys, one for each database attribute. Third, to prevent malicious $\mathcal{C}$ from mixing and matching the trapdoors received for different query terms, and thus potentially violate $\mathcal{D}$'s access control policy, we use the same technique as [13], i.e. $\mathcal{D}$ blinds each trapdoor it obliviously computes, for each term w_i or kg_i, by a random blinding factor ρ_i used in the exponent, e.g. $\mathcal{C}$ computes $\mathsf{xtrap}_i{}^{\rho_i}$ instead of xtrap_i. $\mathcal{D}$ then puts the vector of these ρ_i factors in an authenticated envelope encrypted under a symmetric key shared by $\mathcal{D}$ and $\mathcal{E}$. During the Search protocol, $\mathcal{E}$ receives this envelope from $\mathcal{C}$, authenticates it, decrypts it, and then adds factor ρ_i^{-1} in the exponent to de-blind the xtag (or ptag) value it computes jointly with $\mathcal{C}$, where $\mathcal{C}$ enters a trapdoor blinded by the corresponding factor ρ_i. In this way ρ_i and ρ_i^{-1} factors cancel each other out in the exponent.

However, while all the above mentioned methods carry over from the OSPIR-OXT protocol of [13], protocol MIXED-SSE-OXT differs fundamentally from SSE-OXT in one aspect for which we need new techniques. Namely, MIXED-SSE-OXT contains an extra round of interaction in which $\mathcal{C}$ learns the $\mathsf{ptag}_{(\mathsf{kg}_1,\mathsf{ind})}$ values, potentially for each ind encrypted in $\mathbf{T}[w_1]$. Consider two queries $q^{(i)}$ and $q^{(j)}$ whose s-terms are different, i.e. $w_1{}^{(i)} \neq w_1{}^{(j)}$, but their substring queries have the same s-gram i.e. $\mathsf{kg}_1{}^{(i)} = \mathsf{kg}_1{}^{(j)}$. Since $\mathsf{ptag}_{(\mathsf{kg}_1,\mathsf{ind})}$ is a deterministic function of $(\mathsf{kg}_1, \mathsf{ind})$, the ptag values leak the number of common ind's in $\mathsf{DB}(w_1{}^{(i)})$ and $\mathsf{DB}(w_1{}^{(j)})$, regardless of what information the client $\mathcal{C}$ legitimately gets in $\mathsf{DB}(q^{(i)})$ and $\mathsf{DB}(q^{(j)})$.

We address this problem by modifying the function F_G and the way position information $\mathsf{ind}^{\mathsf{pos}}$ is encrypted in the $\mathbf{P}[(\mathsf{kg}_1, \mathsf{ind})]$ list, which in turn allows us to modify the two-party computation of xtag's $F_G(K_G, (\mathsf{kg}_i, \mathsf{ind}, \mathsf{pos} + \Delta_i))$, for $i = 2, \ldots, \mathsf{h}$, in a way that prevents leakage of any information to $\mathcal{C}$ and at the same time assures that $\mathcal{D}$ learns only the final output of F_G on these inputs. We have several ways of doing this, relying on different computational assumptions and resulting in different pre-computation/online efficiency trade-offs. One solution comes from using an elliptic curve group G with a bilinear map $e : G \times G \to G_T$, where F_G can be defined as $F_G((K_X, K_I), (\mathsf{kg}, \mathsf{ind}, \mathsf{pos})) = e(\mathsf{xtrap}_{\mathsf{kg}}, h)^{\mathsf{xind}^{\mathsf{pos}}}$, and using a variant of ElGamal encryption based on Linear Diffie-Hellman (LDH) assumption on G to jointly compute F_G given the position-related information in $\mathbf{P}$ in the form of encrypted $h^{\mathsf{ind}^{\mathsf{pos}}}$ values.

MIXED-OSPIR-SSE using Bilinear Maps. We provide a more detailed description of the MIXED-OSPIR-SSE variant which uses a group with a bilinear map to allow for a practical two-party computation of xtag's, i.e. of function $F_G((K_X, K_I), (\cdot, \cdot, \cdot))$. Let G be a group of a prime order p with a bilinear map $e : G \times G \to G_T$. Assume that the Linear Diffie-Hellman (LDH) assumption holds on G [3]. Consider function F_G modified as $F_G((K_X, K_I), (\mathsf{kg}, \mathsf{ind}, \mathsf{pos})) = e(\mathsf{xtrap}_{\mathsf{kg}}, h)^{\mathsf{xind}^{\mathsf{pos}}}$, where $\mathsf{xtrap}_{\mathsf{kg}}$ and xind are defined as before, i.e. $\mathsf{xtrap}_{\mathsf{kg}} = H(\mathsf{kg})^{K_X[I(\mathsf{kg})]}$ for H mapping onto G, and $\mathsf{xind} = F_p(K_I, \mathsf{ind})$. We also change the way values $\mathsf{ind}^{\mathsf{pos}}$ are encrypted in list $\mathbf{P}[(\mathsf{kg}, \mathsf{ind})]$, namely for every pos at which kg appears in $\mathsf{DB}[\mathsf{ind}]$, $\mathbf{P}[(\mathsf{kg}, \mathsf{ind})]$ contains a *Linear Encryption* (LE) ciphertext $(a, b, c) = \mathsf{Enc}_{(x_1,x_2)}(h^{\mathsf{ind}^{\mathsf{pos}}})$ where h is a generator of G and the encryption key $(x_1, x_2) \in Z_p \times Z_p$ is set as $(F_p(\mathsf{strap}_{\mathsf{kg}}, \mathsf{n}_1), F_p(\mathsf{strap}_{\mathsf{kg}}, \mathsf{n}_2))$. The encryption $\mathsf{Enc}_{(x_1,x_2)}(m)$ on message $m \in G$ picks random r, s in Z_p, and outputs $(a, b, c) = (h^s, h^r, m \cdot h^{x_1 \cdot s + x_2 \cdot r})$. The decryption $\mathsf{Dec}_{(x_1,x_2)}(a, b, c)$ outputs $a^{-x_1} \cdot b^{-x_2} \cdot c$.

In the Search protocol, when $\mathcal{E}$ identifies a non-empty list $\mathbf{P}[(\mathsf{kg}, \mathsf{ind})]$, then for each ciphertext $\mathsf{Enc}_{(x_1,x_2)}(h^{\mathsf{ind}^{\mathsf{pos}}})$ in this list, and each x-gram kg_i in the tokenization of $\mathcal{C}$'s substring search term, the two parties perform a sub-protocol whose goal is for $\mathcal{E}$ to compute $\mathsf{xtag}_i = e(\mathsf{xtrap}_{\mathsf{kg}_i}, h)^{\mathsf{xind}^{\mathsf{pos}+\Delta_i}}$. Recall that for each x-gram kg_i, $\mathcal{C}$ holds the shift Δ_i corresponding to kg_i and a blinded trapdoor $(\mathsf{xtrap}_{\mathsf{kg}_i})^{\rho_i}$, while $\mathcal{E}$ holds the corresponding de-blinding factor ρ_i^{-1}. Recall also that $\mathcal{C}$ and $\mathcal{E}$ hold a multiplicative sharing, z and y s.t. $z \cdot y = \mathsf{xind}$, hence $z^{\Delta_i} \cdot y^{\Delta_i} = \mathsf{xind}^{\Delta_i}$. Let $B = ((\mathsf{xtrap}_{\mathsf{kg}_i})^{\rho_i})^{z^{\Delta_i}}$, let $v = y^{\Delta_i} \cdot \rho^{-1}$, and denote $h^{\mathsf{ind}^{\mathsf{pos}}}$ encrypted in (a, b, c) as m. The goal of the sub-protocol therefore reduces to computing $e(B, m)^v$, on $\mathcal{E}$'s input $(a, b, c) = \mathsf{Enc}_{(x_1,x_2)}(m)$ and v, and $\mathcal{C}$'s input (x_1, x_2) and B. Note that $e(B, m)^v = e(\mathsf{xtrap}_{\mathsf{kg}_i}, h)^t$ for $t = (\rho_i \cdot z^{\Delta_i}) \cdot \mathsf{ind}^{\mathsf{pos}} \cdot (y^{\Delta_i} \cdot \rho^{-1}) = \mathsf{ind}^{\Delta_i + \mathsf{pos}}$. An additional input into this computation is $\mathcal{E}$'s LE private

key (k_1, k_2) and the corresponding public key $(K_1, K_2) = (h^{k_1}, h^{k_2})$ held by $\mathcal{C}$. The computation proceeds as follows:

(1) $\mathcal{E}$ sends the following three tuples to $\mathcal{C}$:

$$(\alpha_a, \beta_a, \gamma_a) \leftarrow \mathsf{Enc}_{(k_1,k_2)}(a)$$
$$(\alpha_b, \beta_b, \gamma_b) \leftarrow \mathsf{Enc}_{(k_1,k_2)}(b)$$
$$(\alpha_c, \beta_c, \gamma_c) \leftarrow \mathsf{Enc}_{(k_1,k_2)}(c)$$

(2) $\mathcal{C}$ picks r_δ, s_δ at random in Z_p, computes

$$\bar{\alpha} = (\alpha_a)^{-x_1} \cdot (\alpha_b)^{-x_2} \cdot \alpha_c \cdot h^{r_\delta}$$
$$\bar{\beta} = (\beta_a)^{-x_1} \cdot (\beta_b)^{-x_2} \cdot \beta_c \cdot h^{s_\delta}$$
$$\bar{\gamma} = (\gamma_a)^{-x_1} \cdot (\gamma_b)^{-x_2} \cdot \gamma_c \cdot (K_1)^{r_\delta} \cdot (K_2)^{s_\delta}$$

and sends $(\alpha, \beta, \gamma) = (e(B, \bar{\alpha}), e(B, \bar{\beta}), e(B, \bar{\gamma}))$ to $\mathcal{E}$.

(3) $\mathcal{E}$ outputs $\mathsf{xtag} = (\alpha^{-k_1} \cdot \beta^{-k_2} \cdot \gamma)^v [= e(B, m)^v]$.

The crucial point is that when $\mathcal{C}$ uses the decryption key (x_1, x_2) on the twice-encrypted values – first under (x_1, x_2) and then under (k_1, k_2) – then by exponentiation commutativity the result $(\bar{\alpha}, \bar{\beta}, \bar{\gamma})$ is an encryption under key (k_1, k_2) of the same plaintext m which was encrypted under key (x_1, x_2) in (a, b, c). (Terms h^{r_δ}, h^{s_δ}, and $(K_1)^{r_\delta}(K_2)^{s_\delta}$ randomize this re-encryption.)

We note that only the computation of the xtag's corresponding to k-gram positions, i.e. to $(\mathsf{ind}, \mathsf{kg}, \mathsf{pos})$ triples, will be computed using the above approach, while the xtag's corresponding to exact keywords, i.e. to $(\mathsf{ind}, \mathsf{kg})$ pairs, will still be computed as in MIXED-SSE-OXT. This modification does not add new rounds to MIXED-SSE-OXT: Instead of ptag, $\mathcal{E}$ will now send the three tuples computed as in step (1) above for each pos encrypted in $\mathbf{p} = \mathsf{PSet}[\mathsf{ptag}]$. Moreover, for large databases $\mathcal{E}$ will now have to access the disk to retrieve $\mathbf{p}$ from the disk before it can send its response to $\mathcal{C}$. As for pre-computation, the computational cost increase incurred by this method will be moderate, because the bilinear map operation is done only once per each kg in KG, and the linear encryption operations involve fixed-base exponentiations. However, the on-line procedure cost will be dominated by three pairings per each x-gram kg_i for $i = 2, \ldots, \mathsf{h}$ in the search term and each $\mathsf{ind}^{\mathsf{pos}}$ s.t. $(\mathsf{ind}, \mathsf{pos}) \in \mathsf{DB}[\mathsf{kg}_1]$ (and s.t. $\mathsf{ind} \in \mathsf{DB}[w_1]$). We are currently investigating the exact effects of these changes on the overall performance of the protocol.

Security in the OSPIR Setting. The privacy profile of the MIXED-OSPIR-OXT protocol against malicious data owner $\mathcal{D}$, malicious clients $\mathcal{C}$, and honest but curious server $\mathcal{E}$, are very similar to those of the OSPIR-OXT protocol of [13] for the case of exact-keyword conjunctions. Privacy profile against $\mathcal{D}$ is similar because we use the same mechanisms for adapting our MIXED-SSE-OXT protocol to the OSPIR setting as [13], namely oblivious computation of the PRF's. However, in addition to revealing the vector of attributes pertaining to the query terms, which enables attribute-based access policy control by $\mathcal{D}$, the MIXED-OSPIR-OXT protocol additionally reveals the number of k-grams in each substring term and their relative positions $\Delta_2, \ldots, \Delta_{\mathsf{h}}$. Formally, if $\mathbf{q}$ is a vector of queries of the form $q = (w_1, \ldots, w_n, q'_1, \ldots, q'_k)$ where each w_i is an exact query term, with w_1 chosen as the s-term (to simplify the presentation), and each q'_i is a substring term s.t. $T(q'_i) = (\mathsf{kg}_{i,1}, (\mathsf{kg}_{i,2}, \Delta_{i,2}), \ldots, (\mathsf{kg}_{i,\mathsf{h}_i}, \Delta_{i,\mathsf{h}_i}))$, then $\mathcal{L}_{\mathcal{D}}(\mathsf{DB}, \mathbf{q})$ consists of DB (since $\mathcal{D}$ is the owner of the database DB) and the following information for each q in $\mathbf{q}$: a vector of attributes $(I(w_1), \ldots, I(w_n), I(q'_1), \ldots, I(q'_k))$, and the vectors of shifts $(\Delta_{i,2}, \ldots, \Delta_{i,\mathsf{h}_i})$ for each substring term q'_i in q. We note that if the moderate leakage of information on the substring terms leaked in the Δ vectors is unacceptable, it can be eliminated by a Δ-hiding variant of our protocol.

The privacy profile to the malicious $\mathcal{C}$ is also very similar to the OSPIR-OXT protocol. As in there, the client learns the size of the TSet list for the s-term keyword or k-gram in the search query. However, since we handle position-related information by the PSet's, one for every substring term in the search query, $\mathcal{C}$ also learns the sizes of these PSet's. Formally, if $\mathbf{q}$ is a vector of queries of the form $q = (w_1, \ldots, w_n, q'_1, \ldots, q'_k)$

where each w_i is an exact query terms, and w_1 is the s-term, and each q'_i is a substring term, and if $T(q'_i) = (\mathsf{kg}_{i,1}, (\mathsf{kg}_{i,2}, \Delta_{i,2}), \ldots, (\mathsf{kg}_{i,\mathsf{h}_i}, \Delta_{i,\mathsf{h}_i}))$, then $\mathcal{L}_{\mathcal{C}}(\mathsf{DB}, \mathbf{q})$ consists of $|\mathsf{DB}(w_1)|$ for s-term w_1 in each query q in $\mathbf{q}$, and $|\mathsf{DB}(\mathsf{kg}_{i,1})|$ for s-term k-gram $\mathsf{kg}_{i,1}$ in each substring term q_i in each query q in $\mathbf{q}$.

The privacy profile to the honest-but-curious server $\mathcal{E}$ is similar as that specified for the SUB-SSE-OXT protocol in Section 5.2, but it contains some new elements. For simplicity of notation we will assume that each query has n exact terms and k substring terms, and that each substring search term tokenizes to h k-grams. Building on the above notation, denote the i-th query as $q^{(i)} = ({w_1}^{(i)}, \ldots, {w_n}^{(i)}, {q'_1}^{(i)}, \ldots, {q'_k}^{(i)})$ where ${w_1}^{(i)}$ is an s-term, and let $T({q'_j}^{(i)}) = \left({\mathsf{kg}_{j,1}}^{(i)}, (\mathsf{kg}^{(i)}_{j,2}, {\Delta_{j,2}}^{(i)}), \ldots, (\mathsf{kg}_{j,\mathsf{h}}{}^{(i)}, {\Delta_{j,\mathsf{h}}}^{(i)})\right)$. Define function $\mathcal{L}_{\mathcal{E}}(\mathsf{DB}, \mathbf{q})$ which specifies leakage to $\mathcal{E}$ as a vector $(N, \overline{\mathsf{s}}, \mathsf{SP}, \mathsf{RP}, \mathsf{DP}, \mathsf{IP}, \mathsf{PSP})$. Leakage elements $N, \overline{\mathsf{s}}, \mathsf{SP}, \mathsf{RP}$ are defined exactly the same as in Section 5.2, or indeed as in the underlying OXT protocol of [6]. The delta-pattern DP is defined as in Section 5.2, except that it is generalized to k substring terms with h k-grams each. (Formally, $\mathsf{DP}[i]$ is the sequence of vectors $\{({\Delta_{j,2}}^{(i)}, \ldots, {\Delta_{j,\mathsf{h}}}^{(i)})\}$ for $j = 2, \ldots, k$.)

The *conditional intersection pattern* IP in $\mathcal{E}$'s leakage function $\mathcal{L}_{\mathcal{E}}$ contains the leakage due to exact keyword terms and the s-grams of the substring terms, which is the same as in the OXT protocol of [6], and the leakage due to the remaining k-grams in the substring terms, which is the generalization of the IP leakage described in Section 5.2 to the case of multiple substring terms each with multiple k-grams. Formally, we define IP as a tuple $(\mathsf{IPw}, \mathsf{IPs}, \mathsf{IPk})$. The IPw part contains the leakage due to the exact keyword x-terms, exactly as in the OXT protocol of [6]. The IPs part contains the leakage due to the s-grams, i.e. the s-term k-grams in each substring term, which is similar to the leakage IPw because in the MIXED-OSPIR-OXT protocol $\mathcal{E}$ computes a ptag for each s-gram in the same way as it computes an xtag for each exact keyword x-term, namely as a PRF of the (keyword,record-index) pair, and thus both have the same value whenever the (keyword,record-index) pair repeats. Formally, IPw is a $Q \times Q \times n \times n$ table (where Q is the number of queries) where $\mathsf{IPw}[i_1, i_2, j_1, j_2]$ is non-zero only if $i_1 \neq i_2$, i.e. if this entry relates to two different queries, and if ${w_{j_1}}^{(i_1)} = {w_{j_2}}^{(i_2)}$ and $2 \leq j_1, j_2 \leq n$, i.e. if the j_1-th keyword in i_1-th query is the same as the j_2-th keyword in the i_2-th query (with both keywords being x-terms), in which case $\mathsf{IPw}[i_1, i_2, j_1, j_2]$ contains all indexes ind in $\mathsf{DB}({w_1}^{(i_1)}) \cap \mathsf{DB}({w_1}^{(i_2)})$, i.e. indexes of records that contain the s-terms of both i_1-th and i_2-th queries. IPs is a $Q \times Q \times k \times k$ table where $\mathsf{IPs}[i_1, i_2, j_1, j_2]$ is non-zero only if $i_1 \neq i_2$ and ${\mathsf{kg}_{j_1,1}}^{(i_1)} = {\mathsf{kg}_{j_2,1}}^{(i_2)}$, i.e. if the s-gram in the j_1-th substring term in i_1-th query is the same as the s-gram in the j_2-th substring term in the i_2-th query, in which case $\mathsf{IPs}[i_1, i_2, j_1, j_2]$ contains all indexes ind in $\mathsf{DB}({w_1}^{(i_1)}) \cap \mathsf{DB}({w_1}^{(i_2)})$, exactly as in the case of IPw leakge above. The third part of IP leakage is IPk, the leakage due to $\mathcal{E}$'s computation of xtag's for each (k-gram,position,record-index) tuple. This leakage is exactly the same as the IP leakage in the SUB-SSE-OXT protocol described in Section 5.2, but generalized to multiple substring terms and k-grams. Formally, IPk is a $Q \times Q \times k \times k \times \mathsf{h} \times \mathsf{h}$ table, where $\mathsf{IPk}[i_1, i_2, j_1, j_2, \ell_1, \ell_j]$ is non-zero only if $i_1 \neq i_2$ and ${\mathsf{kg}_{j_1,\ell_1}}^{(i_1)} = {\mathsf{kg}_{j_2,\ell_2}}^{(i_2)}$, i.e. if the ℓ_1-th k-gram in the j_1-th substring term in i_1-th query is the same as the ℓ_2-th k-gram in the j_2-th substring term in the i_2-th query, in which case $\mathsf{IP}[i_1, i_2, j_1, j_2, \ell_1, \ell_j]$ contains the set of all triples $(\mathsf{ind}, \mathsf{pos}_1, \mathsf{pos}_2)$ (possibly empty) s.t. $(\mathsf{ind}, \mathsf{pos}_1) \in \mathsf{DB}({w_1}^{(i_1)})$, $(\mathsf{ind}, \mathsf{pos}_2) \in \mathsf{DB}({w_1}^{(i_2)})$, and $\mathsf{pos}_2 = \mathsf{pos}_1 + ({\Delta_{j_1,\ell_1}}^{(i_1)} - {\Delta_{j_2,\ell_2}}^{(i_2)})$, i.e. indexes of records which contain the s-terms of the i_1-th and the i_2-th queries, at positions whose relative distance matches the difference between the Δ's associated with the above two k-grams in the tokenization of the corresponding queries.

The last component of $\mathcal{E}$'s leakage function is a *PSet size pattern* PSP, which is a $Q \times k$ table whose entry $\mathsf{PSP}[i, j]$ contains a sequence of integers $(s_1, s_2, \ldots, s_m)$ where $m = |\mathsf{DB}({w_1}^{(i)})|$, s.t. s_c is the number of occurrences of ${\mathsf{kg}_{j,1}}^{(i)}$, i.e. the s-gram in the j-th substring term in the i-th query, in record $\mathsf{DB}[\mathsf{ind}_c]$, where ind_c is the c-th record index in $\mathsf{DB}({w_1}^{(i)})$. This leakage comes from the fact that for each s-gram in each query $\mathcal{E}$ retrieves the (possibly empty) PSet associated with $\mathsf{ptag}[c]$ for $c = 1, \ldots, m$, and the size of this PSet reflects the number of occurrences of k-gram ${\mathsf{kg}_{j,1}}^{(i)}$ in the c-th record in $\mathsf{DB}({w_1}^{(i)})$.

Note: We stress that that above formal specification of $\mathcal{E}$'s leakage is in many ways an overstatement. Most importantly, the real information $\mathcal{E}$ learns due to the IP leakage does not contain the indexes (even

randomized) of the records which satisfy the conditioned formed by the two queries, but only the fact that the corresponding TSet entries contain the same index ind.

The proof of theorem 5 below is very simple, while the proofs of theorem 6 and 7, although more complex, are similar to the proof of security against the client and the server of the OSPIR-OXT protocol of [13]. All proofs are omitted.

Theorem 5. *Protocol MIXED-OSPIR-OXT is $\mathcal{L}_{\mathcal{D}}$-semantically-secure against malicious data owner $\mathcal{D}$.*

Theorem 6. *Protocol MIXED-OSPIR-OXT is $\mathcal{L}_{\mathcal{C}}$-semantically-secure against a malicious client $\mathcal{C}$, assuming the security of the encryption* Enc*, the authenticated encryption, the* TSet *implementation, the PRF's F_p and F_τ, assuming the random oracle model for hash functions, the One-More GDH and the LDH assumptions on the group G with a bilinear map, the q-DDH assumption on its target group G_T, and the One-More GDH assumption on a standard prime-order group.*

Theorem 7. *Protocol MIXED-OSPIR-OXT is $\mathcal{L}_{\mathcal{E}}$-semantically-secure against honest-but-curious server $\mathcal{E}$, assuming the security of the encryption* Enc*, the* TSet *implementation, the PRF's F_p and F_τ, the random oracle model for hash functions, and the LDH assumptions on group G.*

C Security Proof for Substring Search SSE

Here we present the proof of Theorem 4 stated in Section 5, which describes the security property of protocol SUB-SSE-OXT, the basic substring search SSE protocol shown in Figure 1. We simplify notation by focusing on substrings which tokenize into two k-grams. The extension to any number of k-grams is straightforward.

Hardness assumptions. We recall the q-DDH assumption (we assume familiarity with the DDH assumption). Let G be a prime order cyclic group of order p generated by g. We say that the *q-decision Diffie-Hellman (q-DDH) assumption* holds in G if $\mathbf{Adv}^{\mathrm{q-ddh}}_{G,A}$ is negligible for any generator g and all efficient adversaries A,

$$\begin{aligned}\mathbf{Adv}^{\mathrm{q-ddh}}_{G,A} &= \Pr[A(g, g^a, g^{a^2}, \ldots, g^{a^{q-1}}, g^{a^q}) = 1] \\ &\quad - \Pr[A(g, g^a, g^{a^2}, \ldots, g^{a^{q-1}}, g^b) = 1]\end{aligned}$$

where the probability is over the randomness of A and uniformly chosen a, b from Z_p^*.

Note that the q-DDH assumption implies the DDH assumption. We will use the following lemma in the argument below. Let α, β be integers, let $\mathbf{a} \in (Z_p^*)^\alpha$, $\mathbf{b} \in (Z_p^*)^\beta$, and let $\mathbf{q} = (1, 2, \ldots, q)$. Let $\mathbf{a} \cdot \mathbf{b^q}$ be the $(\alpha \times \beta \times q)$ array M s.t. $\mathrm{M}[i,j,k] = \mathbf{a}[i] \cdot \mathbf{b}[j]^k$, and let $g^{\mathbf{a}\cdot\mathbf{b^q}}$ be the $(\alpha \times \beta \times q)$ array M_G s.t. $\mathrm{M}_G[i,j,k] = g^{\mathrm{M}[i,j,k]}$ where $\mathrm{M} = \mathbf{a} \cdot \mathbf{b^q}$.

Lemma 6. *If the q-DDH assumption holds in G then for any integers α, β (polynomial in $|p|$) and any efficient adversary A, we have that $\mathbf{Adv}^A_G$ is negligble, where*

$$\mathbf{Adv}^A_G = \Pr[A(g, g^{\mathbf{a}\cdot\mathbf{b^q}}) = 1] - \Pr[A(g, \mathrm{M}_G) = 1]$$

where $\mathbf{a}$ is uniform over $(Z_p^*)^\alpha$, $\mathbf{b}$ is uniform over $(Z_p^*)^\beta$, and M_G is uniform over $G^{\alpha\times\beta\times q}$.

Let K, X, Y be sets, and let $F : K \times X \to Y$ be a family of keyed functions. We say that F is a *pseudorandom function (PRF)* if for all efficient adversaries A, $\mathbf{Adv}^{\mathrm{prf}}_{F,A}$ is negligible, where

$$\mathbf{Adv}^{\mathrm{prf}}_{F,A} = \Pr[A^{F(k,\cdot)} = 1] - \Pr[A^{f(\cdot)} = 1]$$

where the probability is over the randomness of A, $k \xleftarrow{\$} K$, and $f \xleftarrow{\$} \mathsf{Fun}(X, Y)$.

As a corollary of lemma 6 we get the following, where $[q]$ stands for the set of integers $\{1, \ldots, q\}$:

Corollary 2. *If the q-DDH assumption holds in G, if $F_G : K_1 \times X \to G$ and $F_p : K_2 \times Y \to Z_p^*$ are PRF's then $F : (K_1 \times K_2) \times (X \times Y \times [q]) \to G$ where $F((k_1, k_2)(x, y, i)) = (F_G(k_1, x))^{(F_p(k_2,y))^i}$ is a non-adaptive PRF.*

We will also use the following well-known fact:

Lemma 7. *Under the DDH assumption on G, for any set X, if H is a hash function mapping X to G then under the DDH assumption function $F : Z_p^* \times X \to G$ defined as $F(k, x) = H(x)^k$ for k uniform in Z_p^*, is a PRF in the random oracle model (ROM) for H.*

Proof of Theorem 4 from Section 5. Let DB be any text strings database and $\mathbf{q}$ be any sequence of Q queries to it as described above. Let $(N, \overline{\mathsf{s}}, \mathsf{SP}, \mathsf{DP}, \mathsf{RP}, \mathsf{IP}) \leftarrow \mathcal{L}_{\mathcal{E}}(\mathsf{DB}, \mathbf{q})$ (note that algorithm $\mathcal{L}_{\mathcal{E}}$ is deterministic). Let A be an efficient algorithm which plays the role of $\mathcal{E}$, i.e. receives the $(\mathsf{TSet}, \mathsf{XSet})$ input generated by $\mathsf{Setup}(\mathsf{DB})$ and input $(\mathsf{stag}, \Delta_1, \mathsf{xtoken}[1], \mathsf{xtoken}[2], \ldots)$ generated by the client $\mathcal{C}$ on input $\mathbf{q}[\tau]$ (and $K = (K_S, K_X, K_T)$ generated in the same $\mathsf{Setup}(\mathsf{DB})$ procedure). We will first make several modifications in the way we look at this information, at some point involving the simulator SIM_T for the underlying T-set implementation, arguing that A's view remains indistinguishable between each consecutive modification. Finally we show a simulator which generates this modified view given only input $(N, \overline{\mathsf{s}}, \mathsf{SP}, \mathsf{DP}, \mathsf{RP}, \mathsf{IP})$, which will complete the proof.

(1) First, we replace $\mathsf{strap} = H(w)^s$ values generated in Setup and $\mathsf{GenToken}$ with random elements in group G. This modification results in an indistinguishable change in A's view because by Lemma 7, $H(w)^s$ is a PRF (since q-DDH implies DDH), and key s is never exposed to A. We will denote strap and stag values generated for keyword w_1 as strap_{w_1} and stag_{w_1}.

(2) Secondly, we replace each (K_z, K_e, K_u) triple generated for a given strap_{w_1} with random τ-bit strings. This modification results in an indistinguishable change in A's view because F_τ is a PRF and strap_{w_1} values are never exposed to A. We will denote the key triple generated for a particular strap_{w_1} as $(K_{w_1,z}, K_{w_1,e}, K_{w_1,u})$.

(3) Third, we replace each (z_c, u_c) pair generated for a given $(K_{w_1,z}, K_{w_1,u})$ key and counter c, with random values in Z_p^*. This modification results in an indistinguishable change in A's view because $(K_{w_1,z}, K_{w_1,u})$ keys are not exposed to A, and F_p is a PRF. Let us denote the tuple (z_c, u_c) generated for counter c from keys $(K_{w_1,z}, K_{w_1,u})$ as $(z_{w_1,c}, u_{w_1,c})$.

(4) Next, we replace generation of ciphertext e in a $(e, y, [u])$ tuple with $\mathsf{e} \leftarrow \mathsf{Enc}(K_{w_1,e}(0^{2\lambda}))$ instead of $\mathsf{Enc}(K_{w_1,e}, (\mathsf{ind}|\mathsf{rdk}))$ (since ind and rdk are both λ-long bit strings). This modification results in an indistinguishable change in A's view because $(\mathsf{Enc}, \mathsf{Dec})$ is a CPA secure encryption, and the keys $K_{w_1,e}$ are never exposed to A. We will denote the tuple (e, y, u) generated for s-term w_1 and counter c as $(\mathsf{e}_{w_1,c}, y_{w_1,c}, u_{w_1,c})$. We will also designate $\mathsf{ind}, \mathsf{pos}, \mathsf{xind}$ values corresponding to the c-th position in $\mathbf{T}(\mathsf{stag}_{w_1})$ as $\mathsf{ind}_{w_1,c}, \mathsf{pos}_{w_1,c}, \mathsf{xind}_{w_1,c}$.

For convenience of notation, we will denote the pair of keys (K_X, K_I) as K_{XI}. Define function $F_{\mathsf{xtag}} : (((Z_p^*)^m \times \{0,1\}^\lambda) \times (\{0,1\}^\lambda \times \{0,1\}^\lambda \times [q])) \to G$ as $F_{\mathsf{xtag}}(K_{XI}, (w, \mathsf{ind}, \mathsf{pos})) = (F_G(K_X, w))^{(F_p(K_I, \mathsf{ind}))^{\mathsf{pos}}}$. Note that XSet consists of values $F_{\mathsf{xtag}}(K_{XI}, (w, \mathsf{ind}, \mathsf{pos}))$ computed for every $(w, \mathsf{ind}, \mathsf{pos})$ tuple s.t. $(\mathsf{ind}, \mathsf{pos}) \in \mathsf{DB}(w)$. Observe that values $\mathsf{xtoken}_{\mathsf{kg}}[c, i]$ sent by $\mathcal{C}$ in the Search are of the form

$$\mathsf{xtoken}_{\mathsf{kg}}[c, i] = \mathsf{xtag}^{\left((y_{w_1,c})^{(\Delta_i)} \cdot v_{w_1,c}\right)^{-1}} \tag{3}$$

for $\mathsf{xtag} = F_{\mathsf{xtag}}(K_{XI}, (w_i, \mathsf{ind}_{w_1,c}, \mathsf{pos}_{w_1,c} + \Delta_i))$.

(5) Since the value satisfying this equation is unique, we will have Charlie generate the $\mathsf{xtoken}_{\mathsf{kg}}[c, i]$ values by equation (3).

(6) The next modification is that we change the way Setup generates $y_{w_1,c}$ and $v_{w_1,c}$ elements in each $\mathbf{T}[\mathsf{stag}_{w_1}]$ tuple, by choosing both of them at random in Z_p^*, and then defining $z_{w_1,c}$ and $u_{w_1,c}$ generated by

$\mathcal{C}$ in Search as $\mathsf{xind}/y_{w_1,c}$ and $\mathsf{xind}^{\mathsf{pos}}/v_{w_1,c}$, respectively. This modification does not change A's view because either way $y_{w_1,c}, v_{w_1,c}$ are random elements in Z_p^*. Note that after the above modification $\mathsf{xtoken}_{\mathsf{kg}}[c,i]$ elements $\mathcal{C}$ sends depend only on $y_{w_1,c}, v_{w_1,c}$, and no longer on $z_{w_1,c}, u_{w_1,c}$.

(7) Therefore, after this modification $\mathcal{C}$ will skip generating $z_{w_1,c}, u_{w_1,c}$ in the Search procedure. In fact, these values will not be generated anywhere in the game.

Let us look closer now at where the Setup procedure, at this point in our series of modifications, needs the key $K_{XI} = (K_X, K_I)$ and the $\mathsf{ind}, \mathsf{xind}, \mathsf{pos}$ values. Note that Setup no longer uses ind, xind, and $\mathsf{xind}^{\mathsf{pos}}$ to generate the $(\mathsf{e}_{w_1,c}, y_{w_1,c}, v_{w_1,c})$ tuples in $\mathbf{T}[\mathsf{stag}_{w_1}]$, and therefore in particular it does not use the K_I key at this point either. The only place key $K_{XI} = (K_X, K_I)$ (and value xind) is used is in the generation of the xtag value inserted into XSet, i.e. in the generation of $F_{\mathsf{xtag}}(K_{XI}, (w, \mathsf{ind}, \mathsf{pos}))$. Note that by Lemma 7, function $F_G : Z_p^* \times \{0,1\}^\lambda \to G$ where $F_G(K_X, w) = H(w)^{e_i}$ is a PRF, for $i = I(w)$ and $K_X = (e_1, \ldots, e_m)$ is chosen uniformly in $(Z_p^*)^m$. Therefore, by Corollary 2, assuming q-DDH, ROM, and the fact that F_p is a PRF, function F_{xtag} is a PRF.

(8) Consequently, in the next modification we replace $F_{\mathsf{xtag}}(K_{XI}, \cdot)$ with a random function $F_{\mathsf{xtag}}(\cdot)$, which assigns a random element in G to every $(w, \mathsf{ind}, \mathsf{pos})$ triple. Since, as we discussed above, key $K_{XI} = (K_X, K_I)$ is not used anywhere else at this point except in the computation of F_{xtag}, and A's view can be generated using black-box access to F_{xtag}, this modification results in an indistinguishable change in A's view.

Let us reassess what A's view consists of at this point. First, recall that by Lemma 7, function $F_G : Z_p^* \times \{0,1\}^\lambda \to G$ where $F_G(K_T, w) = H(w)^{k_i}$ is a PRF, for $i = I(w)$ and $K_T = (k_1, \ldots, k_m)$ is chosen uniformly in $(Z_p^*)^m$. Using this notation, T-set tuples (e, y, v) are formed as an encryption of $0^{2\lambda}$ (e) and random Z_p^* nonces (y and v), and inserted into TSet with an stag handle computed as $\mathsf{stag}_w = F_G(K_T, w)$, while X-set is populated with values of $F_{\mathsf{xtag}}(\mathsf{ind}, w, \mathsf{pos})$ for all $w \in \mathsf{KG}$ and all $(\mathsf{ind}, \mathsf{pos}) \in \mathsf{DB}(w)$. Then, for each query $\mathbf{q}[i]$, tokenized as $(\mathbf{s}[i], (\mathbf{x}[i], \boldsymbol{\Delta}[i]))$, procedure Search sends to A a singleton $(\Delta_2[i])$ (since we assume that each query tokenizes into two k-grams, we have $\mathsf{h} = 2$), value $\mathsf{stag}_{\mathbf{S}[i]} = F_G(K_T, \mathbf{s}[i])$, and a stream of $\mathsf{xtoken}_{\mathsf{kg}}[c, 2]$ values (again, recall that $\mathsf{h} = 2$) for $c = 1, 2, \ldots$. In the i-th query we will denote these values as $\mathsf{xtoken}_{\mathsf{kg}}[c, 2][i]$. These values are computed as in equation (3), but modified by replacing $F_{\mathsf{xtag}}(K_{XI}, \cdot)$ with $F_{\mathsf{xtag}}(\cdot)$, and with (w_1, w_2, Δ_2) terms set to the corresponding values in query $\mathbf{q}[i]$, i.e., they are computed as:

$$\mathsf{xtoken}_{\mathsf{kg}}[c, 2][i] = \mathsf{xtag}^{\left((y_{\mathbf{S}[i],c})^{(\boldsymbol{\Delta}[i])} \cdot v_{\mathbf{S}[i],c}\right)^{-1}} \tag{4}$$

for $\mathsf{xtag} = F_{\mathsf{xtag}}(K_{XI}, (\mathbf{x}[i], \mathsf{ind}_{\mathbf{S}[i],c}, \mathsf{pos}_{\mathbf{S}[i],c} + \boldsymbol{\Delta}[i]))$.

Therefore, since function $F_G(K_T, \cdot)$ is a PRF and the T-set implementation is secure, in the next change **(9)** we will use simulator SIM_T instead of the T-set implementation. In other words, we compute $\mathbf{T}[\mathbf{s}[i]]$ for each $i = 1, \ldots, Q$ as the set of $\mathsf{SP}[i] = |\mathsf{DB}(\mathbf{s}[i])|$ triples (e, y, v) computed as above, and we run $\mathsf{SIM}_T(N, \mathbf{T})$ to generate the TSet datastructure and the search handles $\mathsf{stag}_{\mathbf{S}[i]}$ for $i = 1, \ldots, Q$. By the security of the T-set implementation, this modification results in an indistinguishable change in A's view.

(10) Finally, we change the generation of the xtag values in XSet and the $\mathsf{xtoken}_{\mathsf{kg}}$ generated in Search as follows: To generate xtag's in XSet we simply chose N random elements in G. We also keep a table XT indexed by $(w_2, \mathsf{ind}, \mathsf{pos})$ triples to which we assign some elements in G as the game progresses. Then, we generate $\mathsf{xtoken}_{\mathsf{kg}}[c, 2][i]$ as follows.

1. First we check if $XT(\mathbf{x}[i], \mathsf{ind}_{\mathbf{S}[i],c}, \mathsf{pos}_{\mathbf{S}[i],c} + \boldsymbol{\Delta}[i])$ is already defined. If it is, we move to the second step, but if it is not then we first define this entry in the XT table as follows:
   (a) If $(\mathsf{ind}_{\mathbf{S}[i],c}, \mathsf{pos}_{\mathbf{S}[i],c})$ is in $\mathsf{DB}(w)$ then we assign to this entry in the XT table to a random un-used value in XSet (i.e. to a random value which is not yet assigned to any other entry in the XT table).
   (b) Otherwise, i.e. if $(\mathsf{ind}_{\mathbf{S}[i],c}, \mathsf{pos}_{\mathbf{S}[i],c})$ is not in $\mathsf{DB}(w)$ then we assign a random element in G to this entry in the XT table.
2. Secondly, we take $\mathsf{xtag} \leftarrow XT(\mathbf{x}[i], \mathsf{ind}_{\mathbf{S}[i],c}, \mathsf{pos}_{\mathbf{S}[i],c} + \boldsymbol{\Delta}[i])$ and we compute $\mathsf{xtoken}_{\mathsf{kg}}[c, 2][i]$ as xtag exponentiated to $\left((y_{\mathbf{S}[i],c})^{(\boldsymbol{\Delta}[i])} \cdot v_{\mathbf{S}[i],c}\right)^{-1}$.

It follows by the randomness of F_{xtag} and by equation (4) that the above modification does not change A's view: The xtag values remain random and the only thing that A can observe is whether or not the xtag's computed for each $[c, 2, i]$ hit some previously observed xtag value, and whether this value is in XSet or not.

We will argue that the above view can be generated given only leakage $(N, \bar{\mathbf{s}}, \mathsf{SP}, \mathsf{DP}, \mathsf{RP}, \mathsf{IP})$ generated by $(\mathsf{DB}, \mathbf{q})$. By the security of the T-set implementation A's views of TSet, of the $\mathsf{stag}_{\mathbf{S}[i]}$ values, and hence also of the $\mathbf{T}[\mathbf{s}[i]]$ vectors of (e, y, v) tuples retrieved from TSet via $\mathsf{stag}_{\mathbf{S}[i]}$'s, is simulated correctly on input $(N, \bar{\mathbf{s}}, \mathsf{SP})$. Leakage $\mathsf{DP}[i]$ is used directly as $\boldsymbol{\Delta}[i]$. The result pattern $\mathsf{RP}[i]$ is used to assign some random positions c for each $\mathbf{T}[\mathbf{s}[i]]$ to the $(\mathsf{ind}, \mathsf{pos})$ values in $\mathsf{RP}[i]$, and to decide whether the xtag computed for this position should be from XSet or not. Finally, the IP leakage is used to detect repetitions in the xtag values, i.e. to simulate the view from step (10) above without the XT table. (Note that the XT table keeps all the xtag's which A sees/computes during Search not only for values $(\mathbf{x}[i], \mathsf{ind}, \mathsf{pos} + \Delta)$ corresponding to $(\mathsf{ind}, \mathsf{pos} + \Delta)$ in $\mathsf{DB}(\mathbf{x}[i])$ and $(\mathsf{ind}, \mathsf{pos})$ in $\mathsf{DB}(\mathbf{s}[i])$, which the simulator can simulate using $\mathsf{RP}[i]$, but also for values which are not in DB. Here is where IP table is necessary: For every $\mathbf{q}[i], \mathbf{q}[j]$ pair, IP gives to the simulator the set of ind's with the corresponding positions $\mathsf{pos}_i, \mathsf{pos}_j$, s.t. $\mathbf{x}[i] = \mathbf{x}[j]$ and $\mathsf{pos}_i + \boldsymbol{\Delta}[i] = \mathsf{pos}_j + \boldsymbol{\Delta}[j]$, which is precisely the information needed to detect when some xtag value (i.e. some entry in the XT table) should repeat. Since this simulated view is identical to the view in step (10), the theorem follows.