Outsourced Symmetric Private Information Retrieval

Stanislaw Jarecki* Charanjit Jutla† Hugo Krawczyk‡ Marcel Rosu§ Michael Steiner¶

Abstract

In the setting of searchable symmetric encryption (SSE), a data owner $\mathcal{D}$ outsources a database (or document/file collection) to a remote server $\mathcal{E}$ in encrypted form such that $\mathcal{D}$ 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. Recently, Cash et al. presented a protocol, OXT, which can run arbitrary boolean queries in the SSE setting and which is remarkably efficient even for very large databases.

In this paper we investigate a richer setting in which the data owner $\mathcal{D}$ outsources its data to a server $\mathcal{E}$ but $\mathcal{D}$ is now interested to allow clients (third parties) to search the database such that clients learn the information $\mathcal{D}$ authorizes them to learn but nothing else while $\mathcal{E}$ still does not learn about the data or queried values as in the basic SSE setting. Furthermore, motivated by a wide range of applications, we extend this model and requirements to a setting where, similarly to private information retrieval, the client's queried values need to be hidden also from the data owner $\mathcal{D}$ even though the latter still needs to authorize the query. Finally, we consider the scenario in which authorization can be enforced by the data owner $\mathcal{D}$ without $\mathcal{D}$ learning the policy, a setting that arises in court-issued search warrants.

We extend the OXT protocol of Cash et al. to support arbitrary boolean queries in all of the above models while withstanding adversarial non-colluding servers ($\mathcal{D}$ and $\mathcal{E}$) and arbitrarily malicious clients, and while preserving the remarkable performance of the protocol.

1 Introduction

Consider a database DB composed of collection of documents or records and an application that needs to search DB based on the keywords contained in these records. For example, DB

*U. California Irvine. Email: stasio@ics.uci.edu.
†IBM Research. Email: csjutla@us.ibm.com
‡IBM Research. Email: hugo@ee.technion.ac.il
§U. California Irvine. Email: marcelrosu@gmail.com
¶IBM Research. Email: msteiner@us.ibm.com

can be a medical relational database with records indexed by a set of attributes (e.g., name, zipcode, medical condition, etc.), an email repository indexed by English words and/or envelope information (date, sender, receivers, etc.), a collection of webpages indexed by text and metadata, etc. A search query consists of a boolean expression on keywords that returns all documents whose associated keywords satisfy that expression. In this paper we are concerned with applications where the database is outsourced to an external server $\mathcal{E}$ and search is performed at $\mathcal{E}$ *privately.* That is, $\mathcal{E}$ stores an encrypted version of the original database DB (plus some metadata) and answers encrypted queries from clients such that the client obtains the documents matching his query without $\mathcal{E}$ learning plaintext information about the data and queries.

The most basic setting of private data outsourcing as described above is where the owner of the data itself, $\mathcal{D}$, is the party performing the search at $\mathcal{E}$. In this setting, $\mathcal{D}$ initially processes DB into an encrypted database EDB and sends it to $\mathcal{E}$. $\mathcal{D}$ only keeps a set of cryptographic keys that allows her to later run encrypted searches on $\mathcal{E}$ and decrypt the matching documents returned by $\mathcal{E}$. This setting is known as *searchable symmetric encryption (SSE)* and has been studied extensively [28, 15, 16, 10, 13, 11, 23]. While most of the research has focused on single-keyword searches (i.e., return all documents containing a given keyword), recently Cash et al. [9] provided the first SSE solution, the OXT protocol, that can support in a practical and private way arbitrary boolean queries on sets of keywords and in very large DBs. The leakage to $\mathcal{E}$, which is formally specified and proven in [9], is in the form of data-access and query patterns, never as direct exposure of plaintext data or searched values.

In this work we are concerned with richer outsourcing scenarios where multiple third parties (clients) access the data at $\mathcal{E}$ but only through queries authorized by the data owner $\mathcal{D}$. For example, consider a hospital outsourcing an (encrypted) database to an external service $\mathcal{E}$ such that clients (doctors, administrators, insurance companies, etc.) can search the database but only via queries authorized according to the hospital's policy and without these clients learning information on non-matching documents. As before, $\mathcal{E}$ should learn as little as possible about data and queries.

In this multi-client scenario (to which we refer as *MC-SSE*), $\mathcal{D}$ provides search tokens to clients based on their queries and according to a given authorization policy. Security considers multiple clients acting maliciously and possibly colluding with each other (trying to gain information beyond what they are authorized for) and a semi-trusted server $\mathcal{E}$ which acts as "honest-but-curious" but does not collude with clients. Extending SSE solutions to the multi-client scenario is straightforward when (a) search tokens are fully determined by the query and (b) the SSE protocol does not return false positives (returning false positives, i.e. documents that do not match a query, is allowed in SSE since the recipient in that case is the owner of the data but not in the multi-client setting where clients are not allowed to learn data they were not authorized for). In such cases, $\mathcal{D}$ would receive the client's query, generate the corresponding SSE search tokens as if $\mathcal{D}$ herself was searching the database, and provide the tokens to the client together with a signature that $\mathcal{E}$ can check before processing

the search. However, for enabling general boolean queries, the SSE OXT protocol of [9] requires a number of tokens that is not known a-priori (it depends on the searched data, not only on the query) and therefore the above immediate adaptation does not work.

Our first contribution is in extending the OXT protocol from [9] to the MC-SSE setting while preserving its full boolean-query capabilities and performance. In this extension, $\mathcal{D}$ provides the client $\mathcal{C}$ with a set of query-specific trapdoors which the client can then transform into search tokens as required by OXT. The set of trapdoors given to $\mathcal{C}$ is fully determined by the query and independent of the searched data. An additional subtle technical challenge posed by OXT is how to allow $\mathcal{E}$ to verify that the search tokens presented by $\mathcal{C}$ are authorized by $\mathcal{D}$. The simple solution is for $\mathcal{D}$ to sign the trapdoors, however in OXT these trapdoors need to be hidden from $\mathcal{E}$ (otherwise $\mathcal{E}$ can learn information about unauthorized searches) so a simple signature on them cannot be verified by $\mathcal{E}$. Our solution uses a *homomorphic signature* by $\mathcal{D}$ on the trapdoors that $\mathcal{C}$ can then transform homomorphically into signatures on the search tokens. We show that forging the tokens or their signatures is infeasible even by fully malicious clients.

The resulting MC-OXT protocol preserves the full functional properties of OXT, namely support for arbitrary boolean queries, the same level of privacy (i.e., same leakage profile) with respect to $\mathcal{E}$, and the same remarkable performance. Privacy with respect to clients is near-optimal (see Section 3.1 for why such leakage may be inevitable) with leakage confined only to information on the number of documents matching one of the query terms (typically, the least-frequent term).

Next, we extend the MC-OXT protocol to an even more challenging setting we call Outsourced Symmetric Private Information Retrieval (OSPIR), where on top of the MC-SSE requirements, one asks that *client queries be hidden from* $\mathcal{D}$ - similarly to the Private Information Retrieval (PIR) primitive. This requirement arises in important outsourcing scenarios. In the medical database example mentioned above, the hospital authorizes doctors or other parties to search the medical database according to certain policy; however, in some cases the actual query values are to be kept secret from the hospital itself (due to privacy, liability and regulatory requirements). Only the minimal information for determining the compliance of a query to the policy should be disclosed to the hospital. For example, the policy may indicate that only conjunctions with a minimal number of terms are allowed or that the query needs to include at least three of a set of attributes, etc. In such a case, there is no need for the hospital to learn the particular values being searched (such as a specific last name or medical condition). In other cases, as in outsourced patent or financial information search, the provider $\mathcal{D}$ may want to enforce that a client $\mathcal{C}$ pays for the type of query it is interested in but $\mathcal{C}$ wants to keep his query hidden from both $\mathcal{D}$ and $\mathcal{E}$. Applications to intelligence scenarios are discussed in [18] (see also [31]).

Thus, we relax the query privacy requirement with respect to $\mathcal{D}$ to allow for minimal information needed for $\mathcal{D}$ to determine policy compliance. Specifically, we consider the case where keywords are formed by pairs of attribute-values. For example, in a relational database, attributes are defined by the database columns (e.g., SSN, name, citizenship,

etc.), while in an email repository attributes can refer to envelope information such as sender and receivers or to the message body (in which case the values are, say, English words). In this case, a policy defines the class of boolean expressions allowed for a given client and the attributes that may be used as inputs into these expressions. In order to enforce the policy, $\mathcal{D}$ learns the boolean expression and attributes but nothing about the searched values. For policies defined via classes of attributes (e.g. allowing any attribute from the set of attributes {name, city, zipcode}) leakage to $\mathcal{D}$ can be further reduced by revealing the class and not the specific attributes in the query.

Our most advanced result is extending the OXT protocol to the above OSPIR setting. The resultant protocol, OSPIR-OXT, adds some crucial new ingredients to OXT: It uses *oblivious PRFs (OPRF)* for hiding the query values from $\mathcal{D}$, uses attribute-specific keys for enforcing policy compliance, and uses homomorphic signatures (or the more general abstraction of *shared OPRFs*) for query verification by $\mathcal{E}$. A further extension of the protocol accommodates an external policy manager, e.g., a judge in a warrant-based search, who checks policy compliance and allows server $\mathcal{D}$ to enforce the policy without learning the attributes being searched.

Performance-wise our extensions to OXT preserve the protocol's performance in both pre-processing (creating EDB) and search phases. $\mathsf{OSPIR\text{-}OXT}$ adds to the computational cost by adding a few exponentiations but these are generally inexpensive relative to the I/O cost (especially thanks to common-base exponentiation optimizations). The protocols we provide for MC-SSE and OSPIR models support encrypted search over database containing tens of billions record-keyword pairs, for example a full snapshot of English Wikipedia or a 10-TByte, 100M-record US-census database (see Sections 4.3 and 4.4).

We achieve provable security against adaptive adversarial honest-but-curious server $\mathcal{E}$, against arbitrarily malicious (but non-colluding[1] with $\mathcal{E}$) server $\mathcal{D}$, and against arbitrarily malicious clients. Our security models extend the SSE model [13, 11, 9] to the more complex settings of MC-SSE and OSPIR. In all cases 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 queries 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 partial leakage allowed by SSE protocols. It means that even an adversary that has full information about a database, or even chooses it, does not learn anything from the protocol execution other than what can be derived solely from the defined leakage profile.

Related work: Searchable symmetric encryption (SSE) has been extensively studied [28, 15, 16, 10, 13, 11, 23] (see [13, 11] for more on related work). Most SSE research focused on single-keyword search, and after several solutions with complexity linear in the database size, Curtmola et al. [13] present the first solution for single-keyword search whose complexity is linear in the number of matching documents. They also improve on previous security

[1] See Section 5.2.

models, in particular by providing an adaptive security definition and a solution in this model.

Extending single-keyword SSE to search by conjunctions of keywords was considered in [16, 7, 2], but all these schemes had $O(|\mathsf{DB}|)$ search complexity. The first SSE which can handle very large DBs and supports conjunctive queries is the OXT protocol discussed above, given by Cash et al. [9]. The MC-SSE and OSPIR schemes we present are based on this protocol and they preserve its performance and privacy characteristics.

Extension of the two-party client-server model of SSE to the multi-client setting was considered by Curtmola et al, [13], but their model disallowed per-query interaction between the data owner and the client, leading to a relatively inefficient implementation based on broadcast encryption. Multi-client SSE setting which allows such interaction was considered by Chase and Kamara [11] as SSE with "controlled disclosure", and by Kamara and Lauter [22], as "virtual private storage", but both considered only single-keyword queries and did not support query privacy from the data owner. De Cristofaro et al. [12] extended multi-client SSE to the OSPIR setting, which supports query privacy, but only for the case of single-keyword queries. In recent independent work, Pappas et al. [26] provide support for boolean queries in a setting similar to our OSPIR setting (but with honest-but-curious clients).

SSE schemes which support efficient updates of the encrypted database appeared in [29, 23] for single-keyword SSE. The OXT SSE scheme of [9] which supports arbitrary boolean queries, has been extended to the dynamic case in [8], and the same techniques apply to the MC-SSE and OSPIR schemes presented in this paper.

Recently Islam et al. [19] showed that frequency analysis revealed by access control patterns in SSE schemes can be used to predict single-keyword queries. Such attacks, although harder to stage, are possible for conjunctive queries as well, but the general masking and padding countermeasures suggested in [19] are applicable to the MC-OXT and OSPIR-OXT protocols.

In other directions, SSE was extended to the *public key* setting, allowing any party to encrypt into the database, first for single-keyword search [5, 30, 1, 3, 6, 27], and later for conjunctive queries as well [6], but all these PKSE schemes have $O(|\mathsf{DB}|)$ search complexity. Universally composable SSE was introduced by [24], also with $O(|\mathsf{DB}|)$ search complexity.

Multi-client SSE and OSPIR models are related to the work on multi-client ORAM, e.g. see the recent work of Huang and Goldberg [17], which aims for stronger privacy protection of client's queries from server $\mathcal{E}$, but multi-client ORAM supports DB lookups by (single) indexes instead of (boolean formulas on) keywords, and they can currently support much smaller DB sizes.

Paper organization. We first present our protocols for the case of conjunctive queries: in Section 2 we recall the basic OXT protocol [9], suitably reformulated for our generalizations, in Section 3 we address the multi-client SSE model, and in Section 4 we handle the OSPIR

model. In Section 4.2 we explain how to extend support for general boolean queries. Security models and claims are presented in Section 5. While the main implementation details and performance analysis is deferred to a companion paper [8], we provide some information on computational cost and performance measurements from our implementation in Section 4.3 and 4.4, respectively.

2 SSE and the OXT Protocol

We first recall the SSE OXT protocol from [9] that forms the basis for our solution to searchable encryption in the more advanced MC and OSPIR models.

SSE protocols and formal setting [9]. Let λ 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 $\mathsf{ind}_i \in \{0,1\}^\lambda$ is a document identifier and $\mathsf{W}_i \subseteq \{0,1\}^*$ is a list of keywords in that document. We set W to $\bigcup_{i=1}^d \mathsf{W}_i$. A *query* $\psi(\bar{w})$ is specified by a tuple of keywords $\bar{w} \in \mathsf{W}^*$ and a boolean formula ψ on $\bar{w}$. We write $\mathsf{DB}(\psi(\bar{w}))$ for the set of identifiers of documents that "satisfy" $\psi(\bar{w})$. Formally, this means that $\mathsf{ind}_i \in \mathsf{DB}(\psi(\bar{w}))$ iff the formula $\psi(\bar{w})$ evaluates to true when we replace each keyword w_j with true or false depending on if $w_j \in \mathsf{W}_i$ or not (in particular $\mathsf{DB}(w) = \{\mathsf{ind}_i \text{ s.t. } w \in \mathsf{W}_i\}$). Below we let d denote the number of records in DB, $m = |\mathsf{W}|$, and $N = \sum_{w \in \mathsf{W}} |\mathsf{DB}(w)|$.

A *searchable symmetric encryption (SSE) scheme* Π consists of an algorithm $\mathsf{EDBSetup}$ and a protocol Search fitting the following syntax. $\mathsf{EDBSetup}$ takes as input a database DB and a list of document decryption keys RDK, and outputs a secret key K along with an encrypted database EDB. The search protocol proceeds between a *client* $\mathcal{C}$ and *server* $\mathcal{E}$, where $\mathcal{C}$ takes as input the secret key K and a query $\psi(\bar{w})$ 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* if for all $\mathsf{DB}, \mathsf{RDK}$ and all queries $\psi(\bar{w})$, for $(K, \mathsf{EDB}) \leftarrow \mathsf{EDBSetup}(\mathsf{DB}, \mathsf{RDK})$, after running Search with client input $(K, \phi(\bar{w}))$ and server input EDB, the client outputs $\mathsf{DB}(\phi(\bar{w}))$ and $\mathsf{RDK}[\mathsf{DB}(\phi(\bar{w}))]$ where $\mathsf{RDK}[S]$ denotes $\{\mathsf{RDK}[\mathsf{ind}] \,|\, \mathsf{ind} \in S\}$. Correctness can be statistical (allowing a negligible probability of error) or computational (ensured only against computationally bounded attackers - see [9]).

Note (conjunctive vs. Boolean queries). Throughout the paper we present our protocols for the case of conjunctive queries where a query consists of n keywords $\bar{w} = (w_1, \ldots, w_n)$ and it returns all documents containing all these keywords. The adaptation to the case of boolean queries is described in Section 4.2.

Note (the array RDK*).* Our SSE syntax, and the OXT description in Figure 1, includes as input to $\mathsf{EDBSetup}$ an array RDK that contains, for each document in DB, a key rdk used to encrypt that document. When a client retrieves the index ind of a document matching its query, it also retrieves the *record-decrypting key* rdk needed to decrypt that record. This mechanism is not strictly needed in the SSE setting (where rdk could be derived from ind

using a PRF with a secret key known to $\mathcal{C}$) and it is not part of the original OXT in [9], but it is needed in the more advanced models considered later. This extension does not change the functionality and security properties of OXT as analyzed in [9]).

Note (retrieval of matching encrypted records). Our formalism defines the output of the SSE protocol as the set of ind identifiers pointing to the encrypted records matching the query, together with the associated record decryption key. For the sake of generality, we do not model the processing and retrieval of encrypted records. This allows us to decouple the storage and processing of document payloads (which can be done in a variety of ways, with varying types of leakage) from the storage and processing of the metadata, which is the focus of our protocols.

SSE Security. The SSE setting considers security w.r.t. an adversarial server $\mathcal{E}$, hence security is parametrized via a leakage function capturing information learned by $\mathcal{E}$ from the interaction with $\mathcal{C}$. See Section 5.

The TSet Datastructure for Inverted Index. The SSE protocol OXT, on which our MC-SSE and OSPIR-SSE schemes are based, uses a datastructure TSet which abstracts an inverted index (a list) in a way which minimizes leakage of information to the server $\mathcal{E}$ on which this index is stored. We adopt the API for this datastructure given by [9]. Specifically, the abstract TSetSetup operation receives a collection $\mathbf{T}$ of lists $\mathbf{T}[w]$ for each $w \in \mathsf{W}$ and builds the TSet data structure out of these lists; it returns TSet and a key K_T. Then, for any w, procedure $\mathsf{TSetGetTag}(K_T, w)$ (typically a PRF) generates a search handle, denoted stag, which allows for retrieval of $\mathbf{T}[w]$ from TSet, via procedure $\mathsf{TSetRetrieve}(\mathsf{TSet}, \mathsf{stag})$. By convention we define $\mathbf{T}[w]$ as an empty list for every bitstring $w \notin \mathsf{W}$. The elements in the $\mathbf{T}$ lists are called *tuples*, hence the name "TSet" which stands for a "tuple set", and their exact contents are defined by the OXT protocol. A TSet implementation as defined in [9] must provide *privacy*, in that the TSet datastructure does not reveal anything about the tuple lits in $\mathbf{T}$ except their total size $\sum_w |\mathbf{T}[w]|$; and *correctness*, in that $\mathsf{TSetRetrieve}(\mathsf{TSet}, \mathsf{stag})$ returns $\mathbf{T}[w]$ for $\mathsf{stag} = \mathsf{TSetGetTag}(K_T, w)$. However, here we extend the above correctness property, because unlike the two-party SSE settings of [9], in the MC-SSE or OSPIR-SSE settings we must require TSet implementation to have no false positives in addition to no false negatives. We formally specify this extended notion of TSet correctness in Section 5.1.

2.1 The OXT Protocol

The OXT protocol [9] is presented in Figure 1; see [9] for full design rationale and analysis. Here we provide a high level description as needed for the extensions to this protocol we introduce in the following sections.

The basis of OXT is the following simple search algorithm over *unencrypted* databases. The algorithm uses two types of data structures. First, for every keyword w there is an inverted index (a list) I_w pointing to the indices ind of all documents that contain w. Then, for every document ind there is a list L_{ind} of all keywords contained in document ind. To

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

Key Generation. $\mathcal{D}$ picks key K_S for PRF F_τ and keys K_T, K_X, K_I for PRF F_p. F_τ and F_p are PRF's which output strings in respectively $\{0,1\}^\tau$ and Z_p^*, and τ is a security parameter.

- Initialize XSet to an empty set, and initialize $\mathbf{T}$ to an empty array indexed by keywords from W.
- For each $w \in \mathsf{W}$, build the tuple list $\mathbf{t}$ and insert elements into set XSet as follows:
  - Initialize $\mathbf{t}$ to be an empty list.
  - Set $\mathsf{strap} \leftarrow F_\tau(K_S, w)$ and $(K_z, K_e) \leftarrow (F_\tau(\mathsf{strap}, 1), F_\tau(\mathsf{strap}, 2))$.
  - Initialize $c \leftarrow 0$; then for all ind in $\mathsf{DB}(w)$ in random order:
    - 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 $c \leftarrow c + 1$, $z_c \leftarrow F_p(K_z, c)$, $y \leftarrow \mathsf{xind} \cdot z_c^{-1}$. Append (e, y) to $\mathbf{t}$.
    - Set $\mathsf{xtag} \leftarrow g^{F_p(K_X, w) \cdot \mathsf{xind}}$ and add xtag to XSet.
  - $\mathbf{T}[w] \leftarrow \mathbf{t}$.
- Create $(\mathsf{TSet}, K_T) \leftarrow \mathsf{TSetSetup}(\mathbf{T})$, and output key $K = (K_S, K_X, K_T, K_I)$ and $\mathsf{EDB} = (\mathsf{TSet}, \mathsf{XSet})$.

Search protocol

Client $\mathcal{C}$ on input K defined as above and a conjunctive query $\bar{w} = (w_1, \ldots, w_n)$:

- Sets $\mathsf{stag} \leftarrow \mathsf{TSetGetTag}(K_T, w_1)$; $\mathsf{strap} \leftarrow F_\tau(K_S, w_1)$, $(K_z, K_e) \leftarrow (F_\tau(\mathsf{strap}, 1), F_\tau(\mathsf{strap}, 2))$.
- Sends to the server $\mathcal{E}$ the message $(\mathsf{stag}, \mathsf{xtoken}[1], \mathsf{xtoken}[2], \ldots)$ where $\mathsf{xtoken}[\cdot]$ are defined as:
  - For $c = 1, 2, \ldots$, until $\mathcal{E}$ sends stop:
    - Set $z_c \leftarrow F_p(K_z, c)$ and set $\mathsf{xtoken}[c, i] \leftarrow g^{F_p(K_X, w_i) \cdot z_c}$ for $i = 2, \ldots, n$.
    - Set $\mathsf{xtoken}[c] \leftarrow (\mathsf{xtoken}[c, 2], \ldots, \mathsf{xtoken}[c, n])$.

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

- Retrieve $\mathbf{t} \leftarrow \mathsf{TSetRetrieve}(\mathsf{TSet}, \mathsf{stag})$ from TSet.
- For $c = 1, \ldots, |\mathbf{t}|$ do:
  - Retrieve (e, y) from the c-th tuple in $\mathbf{t}$
  - If $\forall i = 2, \ldots, n : \mathsf{xtoken}[c, i]^y \in \mathsf{XSet}$ then send e to client $\mathcal{C}$.
- When last tuple in $\mathbf{t}$ is reached, sends stop to $\mathcal{C}$ and halt.

For each received e client $\mathcal{C}$ computes $(\mathsf{ind}|\mathsf{rdk}) \leftarrow \mathsf{Dec}(K_e, \mathsf{e})$ and outputs $(\mathsf{ind}, \mathsf{rdk})$.

Figure 1: OXT: Oblivious Cross-Tags SSE Scheme

search for a conjunction $\bar{w} = (w_1, \ldots, w_n)$, the client chooses the estimated least frequent

keyword [2] in $\bar{w}$, say w_1, and checks for each $\mathsf{ind} \in I_{w_1}$ whether $w_i \in L_{\mathsf{ind}}$, $i = 2, \ldots, n$. If this holds for all $2 \leq i \leq n$ then ind is added to the result set. As a performance optimization, instead of maintaining a list L_{ind} for each ind, one can fix a hash function f and keep a data structure representing the set $X = \{f(w, \mathsf{ind}) : w \in \mathsf{W}, \mathsf{ind} \in \mathsf{DB}(w)\}$. Thus, the check $w \in L_{\mathsf{ind}}$ can be replaced with the check $f(w, \mathsf{ind}) \in X$. Protocol OXT adapts this algorithm to the encrypted setting as follows (we start with the description of a simplified version, corresponding to protocol BXT in [9], and then move to the more specific details of OXT).

For each keyword $w \in \mathsf{W}$ an inverted index $\mathsf{TSet}(w)$ (corresponding to I_w above) is built pointing to all the ind values of documents in $\mathsf{DB}(w)$. Each $\mathsf{TSet}(w)$ is identified by a string $\mathsf{stag}(w)$, and ind values in $\mathsf{TSet}(w)$ are encrypted under a secret key K_e. Both $\mathsf{stag}(w)$ and K_e are computed as a PRF applied to w with secret keys known to $\mathcal{C}$ only. In addition, a data structure called XSet is built as an "encrypted equivalent" of the above set X as follows. First, for each $w \in \mathsf{W}$, a value $\mathsf{xtrap}(w) = F(K_X, w)$ is computed where K_X is a secret PRF key then for each $\mathsf{ind} \in \mathsf{DB}(w)$ a value $\mathsf{xtag} = f(\mathsf{xtrap}(w), \mathsf{ind})$ is added to XSet where f is an unpredictable function of its inputs (e.g., f can be a PRF used with $\mathsf{xtrap}(w)$ as the key and ind as input). The output EDB from the $\mathsf{EDBSetup}$ phase includes $\mathsf{TSet} = \{\mathsf{TSet}(w)\}_{w \in \mathsf{W}}$ and the set XSet. In the Search protocol for a conjunction $(w_1, \ldots, w_n)$, the client $\mathcal{C}$ chooses the conjunction's s-term (i.e., the estimated least frequent keyword in the conjunction, which we assume to be w_1), computes $\mathsf{stag}(w_1)$ and K_e using $\mathcal{C}$'s secret keys and computes $\mathsf{xtrap}_i = F(K_X, w_i)$ for each $i = 2, \ldots, n$. It then sends $(K_e, \mathsf{stag}, \mathsf{xtrap}_2, \ldots, \mathsf{xtrap}_n)$ to the server $\mathcal{E}$. $\mathcal{E}$ uses stag to retrieve $\mathsf{TSet}(w_1)$, uses K_e to decrypt the ind values in $\mathsf{TSet}(w_1)$, and sends back to $\mathcal{C}$ those ind for which $f(\mathsf{xtrap}_i, \mathsf{ind}) \in \mathsf{XSet}$ for all $i = 2, \ldots, n$.

Next, we note that using the protocol described above leads to significant leakage in that the xtrap value allows $\mathcal{E}$ to check whether $\mathsf{xtag} = f(\mathsf{xtrap}, \mathsf{ind}) \in \mathsf{XSet}$ for each value ind ever seen by $\mathcal{E}$, revealing correlation statistics between each s-term and each x-term ever queried by $\mathcal{C}$. This motivates the main mechanism in OXT, i.e. the instantiation of the function f via a two-party computation in which $\mathcal{E}$ inputs an encrypted value ind, $\mathcal{C}$ inputs xtrap and the ind-decryption key, and $\mathcal{E}$ gets the value of $\mathsf{xtag} = f(\mathsf{xtrap}, \mathsf{ind})$ without learning either the xtrap or ind values themselves. For this OXT uses a blinded DH computation over a group G (with generator g of prime order p). However, to avoid the need for interaction between $\mathcal{E}$ and $\mathcal{C}$ in the Search phase, the *blinding factors are pre-computed and stored as part of the tuples in the* TSet *lists.* Specifically, indexes ind are replaced in the computation of f with dedicated per-record values $\mathsf{xind} \in Z_p^*$ (computed as $F_p(K_I, \mathsf{ind})$ where F_p is a PRF with range Z_p^*), $\mathsf{xtrap}(w)$'s are implemented as $g^{F_p(K_X, w)}$ (K_I and K_X are secret keys kept by $\mathcal{C}$), and xtag is re-defined as $(\mathsf{xtrap}(w))^{\mathsf{xind}}$. The blinding factor in the underlying two-party computation is pre-computed during $\mathsf{EDBSetup}$ and stored in the TSet. Namely, each tuple corresponding to a keyword w and document index ind contains a blinded value $y_c = \mathsf{xind} \cdot z_c^{-1}$ for $\mathsf{xind} = F_p(F_I, \mathsf{ind})$ where z_c is an element in Z_p^* derived (via a PRF) from w and a tuple counter c (this counter, incremented for each tuple in the tuple list associated

[2] The estimated least frequent keyword is called the conjunction's *s-term*; the other terms in a conjunction are called *x-terms*.

with w, ensures independence of each blinding value z_c).

During search, the server $\mathcal{E}$ needs to compute the xtag values $g^{F_p(K_X,w_i)\cdot \mathsf{xind}}$ for each xind in $\mathsf{TSet}(w_1)$ and then test these for membership in XSet. To enable this the client sends, for the c-th tuple in $\mathbf{t}$, an n-long array $\mathsf{xtoken}[c]$ defined by $\mathsf{xtoken}[c,i] := g^{F_p(K_X,w_i)\cdot z_c}$, $i = 1,\ldots,n$, where z_c is the precomputed blinding derived by $\mathcal{C}$ from w (via a PRF) and the tuple counter c. $\mathcal{E}$ then performs the T-set search to get the results for w_1, and for each c it filters the c-th result by testing if $\mathsf{xtoken}[c,i]^{y_c} \in \mathsf{XSet}$ for all $i = 2,\ldots,n$. This protocol is correct because $\mathsf{xtoken}[c,i]^{y_c} = g^{F_p(K_X,w_i)\cdot z_c\cdot \mathsf{xind}\cdot z_c^{-1}} = g^{F_p(K_X,w_i)\cdot \mathsf{xind}}$, meaning that the server correctly recomputes the pseudorandom values in the XSet. Putting these ideas together results in the OXT protocol of Figure 1. Note that $\mathcal{C}$ sends the xtoken arrays (each holding several values of the form $g^{F_p(K_X,w_i)\cdot z_c}$) until instructed to stop by $\mathcal{E}$. There is no other communication from server to client (alternatively, server can send the number of elements in $\mathsf{TSet}(w)$ to the client who will respond with such number of xtoken arrays).

Note. The OXT protocol in Figure 1 derives keys K_e, K_z slightly differently than in the OXT description of [9]. The modified key derivation is closer to what we need for $\mathsf{MC\text{-}OXT}$ and $\mathsf{OSPIR\text{-}OXT}$ protocols presented in the following sections, without affecting the functionality or security of OXT.

3 Multi-Client SSE

We present an extension of the OXT protocol for the *multi-client SSE (MC-SSE)* setting described in the introduction and in more detail below. The extension preserves the functionality of OXT, supporting any boolean query, and superb performance, while securely serving multiple clients, all of which can behave maliciously.

Multi-Client SSE Setting. In MC-SSE there is the owner $\mathcal{D}$ of the plaintext database DB, an external server $\mathcal{E}$ that holds the encrypted database EDB, and clients that receive tokens from $\mathcal{D}$ in order to perform search queries at $\mathcal{E}$. In other words, $\mathcal{D}$ is outsourcing her search service to a third party but requires clients to first obtain search tokens from her. Her goal is to ensure service to clients via $\mathcal{E}$ while leaking as little as possible information to $\mathcal{E}$ about the plaintext data and queries, and preventing clients from running any other DB queries than those for which $\mathcal{D}$ issued them a token. This is a natural outsourcing setting of increasing value in cloud-based platforms, and it was described by Chase and Kamara [11] as an *SSE with controlled disclosure*.

Formally, the MC-SSE setting changes the syntax of an SSE scheme by including an additional algorithm $\mathsf{GenToken}$ which on input the secret key K, generated by the data owner $\mathcal{D}$ in the $\mathsf{EDBSetup}$ procedure, and a boolean query $\psi(\bar{w})$, submitted by client $\mathcal{C}$, generates a search-enabling value token. Then, procedure Search is executed by server $\mathcal{E}$ on input EDB, but instead of the client $\mathcal{C}$ running on input K and query $\psi(\bar{w})$ (as in SSE), $\mathcal{C}$ runs on input consisting only of the search token token. Correctness is defined similarly to

the SSE case, namely, assuring (except for negligible error probability) that $\mathcal{C}$'s output sets $\mathsf{DB}(\psi(\bar{w}))$ and $\mathsf{RDK}[\mathsf{DB}(\psi(\bar{w}))]$. Security is treated in Section 5.

3.1 The MC-OXT Protocol

We describe the changes to OXT from Figure 1 needed to support boolean queries in the MC-SSE setting. As before, the protocol is described for conjunctions with the adaptation to boolean queries described in Section 4.2.

$\mathsf{EDBSetup}(\mathsf{DB}, \mathsf{RDK})$. This pre-processing phase is identical to the one in OXT except for the addition of a key K_M shared between $\mathcal{D}$ and $\mathcal{E}$. The output from this phase is $K = (K_S, K_X, K_T, K_M)$ kept by $\mathcal{D}$ and $\mathsf{EDB} = (K_M, \mathsf{TSet}, \mathsf{XSet})$ stored at $\mathcal{E}$.

$\mathsf{GenToken}(K, \bar{w})$. This is the new MC-specific phase in which $\mathcal{D}$, using key K, authorizes $\mathcal{C}$ for a conjunction $\bar{w} = w_1, \ldots, w_n$ and provides $\mathcal{C}$ with the necessary tokens to enable the search at $\mathcal{E}$. We assume w_1 to be the s-term (and chosen by $\mathcal{D}$ who has knowledge of term frequencies). Specifically, $\mathcal{D}$ performs the following operations. She sets $\mathsf{stag} \leftarrow \mathsf{TSetGetTag}(K_T, w_1)$ and $\mathsf{strap} \leftarrow F_\tau(K_S, w_1)$, same as $\mathcal{C}$ does in OXT. Then, for $i = 2, ..., n$, she picks $\rho_i \leftarrow Z_p^*$ and sets $\mathsf{bxtrap}_i \leftarrow g^{F_p(K_X, w_i)\cdot\rho_i}$. Finally, she sets $\mathsf{env} \leftarrow \mathsf{AuthEnc}(K_M, (\mathsf{stag}, \rho_2, ..., \rho_n))$ and outputs $\mathsf{token} = (\mathsf{env}, \mathsf{strap}, \mathsf{bxtrap}_2, \ldots, \mathsf{bxtrap}_n)$ (which $\mathcal{C}$ uses in the search phase).

To see how this enables search as in OXT, first note that with $\mathsf{xtrap}_i = g^{F_p(K_X, w_i)}$, $i = 2, \ldots, n$, the client can produce the values $g^{F_p(K_X, w_i)\cdot z_c}$ needed in the Search phase of OXT. Hence, $\mathcal{D}$ could just provide the xtrap_i values to $\mathcal{C}$. However, $\mathcal{D}$ needs to be able to sign (or MAC) these values so that $\mathcal{E}$ can check that $\mathcal{C}$ is authorized to this query and, e.g., did not truncate a conjunction or mix parts from different queries. Signing the plain xtrap values does not work since these values must not be seen by $\mathcal{E}$ (it would allow $\mathcal{E}$ to learn information from unauthorized searches). The solution is to provide $\mathcal{C}$ with a *homomorphic signature* that $\mathcal{C}$ can convert into individual signatures for all tokens sent to $\mathcal{E}$. This is accomplished as follows: $\mathcal{D}$ picks one-time blinding factors $\rho_2, \ldots, \rho_n$ and provides $\mathcal{C}$ with *blinded* (or MAC'ed) xtrap values $\mathsf{bxtrap}_i = \mathsf{xtrap}_i^{\rho_i}$ for $j = 2, \ldots, n$, while providing the blinding factors $(\rho_2, \ldots, \rho_n)$ to $\mathcal{E}$ in an encrypted and authenticated envelope env. To produce $g^{F_p(K_X, w_i)\cdot z_c}$ during Search, $\mathcal{C}$ will compute $\mathsf{bxtrap}_i^{z_c}$ and $\mathcal{E}$ will raise this value to $1/\rho_i$. Security relies on the fact that if $\mathcal{C}$ provides $\mathcal{E}$ with values other than those given by $\mathcal{D}$, then when raised to $1/\rho_i$ by $\mathcal{E}$ (where ρ_i is random and unknown to $\mathcal{C}$) the resulting values will not correspond to elements of XSet except with negligible probability.

Search *Protocol.* The Search protocol reflects the above changes: On $\mathsf{token} = (\mathsf{env}, \mathsf{strap}, \mathsf{bxtrap}_2, ..., \mathsf{bxtrap}_n)$, $\mathcal{C}$ computes (K_z, K_e) from strap as in OXT. Then, it sends to $\mathcal{E}$ the value env as well as the sequence $\mathsf{bxtoken}[1], \mathsf{bxtoken}[2], \ldots$ which contains the same values as $\mathsf{xtoken}[1], \mathsf{xtoken}[2], \ldots$ in OXT up to the blinding exponents ρ_i. Specifically, the values $\mathsf{bxtoken}[c, i]$, for $i = 2, \ldots, n$, are computed by $\mathcal{C}$ as $(\mathsf{bxtrap}_i)^{z_c}$. On the receiving side, $\mathcal{E}$ verifies the authenticity of env and decrypts it to find stag, which it uses to retrieve $\mathsf{TSet}(w_1)$,

and $\rho_2, \ldots, \rho_n$. The only change from OXT is that the operation $\mathsf{xtoken}[c,i]^y$ is replaced with $\mathsf{bxtoken}[c,i]^{y/\rho_i}$.

Note *(masking the size of* $\mathsf{TSet}(w_1)$*).* $\mathsf{MC\text{-}OXT}$ leaks to $\mathcal{C}$ the size of $\mathsf{TSet}(w_1) = \mathsf{DB}(w_1)$ given by the number of $\mathsf{bxtoken}$ vectors requested by $\mathcal{E}$. Note that the exact size of this set can easily be masked by $\mathcal{E}$ requesting more $\mathsf{bxtoken}$ values from $\mathcal{C}$ than needed. We observe that some form of leakage on the frequency of the least frequent term in the conjunction appears to be inherent even for plaintext search algorithms. Indeed, it is likely (though there seem to be no proven lower bounds) that running time will be noticeably different for conjunctions with all terms being infrequent from the case that all terms are very frequent, except if short searches are artificially padded to full database size (or if conjunctions are pre-computed). Two considerations in masking the size of $\mathsf{TSet}(w_1)$ are that (i) the masked size should be chosen as a step function of $|\mathsf{TSet}(w_1)|$ (and not, say, a fixed linear function); (ii) During search, $\mathcal{E}$ should return to $\mathcal{C}$ all results matching a query only after $\mathcal{E}$ received all the $\mathsf{bxtoken}$ vectors from $\mathcal{C}$; indeed, sending the results as soon as they are found to match the query would leak information on $|\mathsf{TSet}(w)|$ to $\mathcal{C}$ (since no ind's will be returned after $\mathcal{C}$ sends $|\mathsf{TSet}(w)|$ tokens). Alternatively, to avoid delaying the sending of results from $\mathcal{E}$ to $\mathcal{C}$, $\mathcal{D}$ can randomly spread the counters c in each $\mathsf{TSet}(w)$ during the $\mathsf{EDBSetup}$ procedure, so that counters c range between 1 and $|\mathsf{Mask}(\mathsf{TSet}(w))|$ rather than between 1 and $|\mathsf{TSet}(w)|$ (note that the number of tuples in $\mathsf{TSet}(w)$ is not increased by this approach, only the counters in these tuples are more sparsely allocated).

4 Outsourced Symmetric PIR

The OSPIR (for Outsourced Symmetric PIR) setting augments the MC-SSE setting with an additional requirement: The database owner $\mathcal{D}$ should learn as little as possible about queries performed by clients while still being able to verify the compliance of these queries to her policy. Here we extend $\mathsf{MC\text{-}OXT}$ to this setting by augmenting the token generation component $\mathsf{GenToken}$ to support "blinding" by the client of query requests sent to $\mathcal{D}$, and adding a mechanism for $\mathcal{D}$ to enforce policy-compliance of the query when only having access to their blinded versions. OSPIR-OXT supports *attribute-based policies* where $\mathcal{D}$ learns information about query attributes but nothing about values (e.g., $\mathcal{D}$ may learn that a query includes a last-name, a zipcode and two words from a text field but nothing about the actual queried keywords). The security model adds $\mathcal{D}$ as a new adversarial entity trying to learn $\mathcal{C}$'s hidden query values. Refer to the introduction for more discussion about the OSPIR setting and attribute-based policies.

OSPIR SSE Syntax. 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 $\bar{w}$, with a two-party protocol between $\mathcal{C}$ and $\mathcal{D}$ that allows $\mathcal{C}$ to compute a search-enabling token without $\mathcal{D}$ learning $\bar{w}$. In addition, $\mathcal{D}$ should be able to enforce an attribute-based query-authorization

policy on these queries. For this, we assume that keyword set W is partitioned into m *attributes*, and let $I(w)$ denote the attribute of keyword w.[3] An attribute-based policy is represented by a set of attribute-sequences P s.t. a conjunctive query $\bar{w} = (w_1, ..., w_n)$ is *allowed by policy* P if and only if the sequence of attributes $\mathsf{av}(\bar{w}) = (I(w_1), ..., I(w_n))$ corresponding to this query is an element in set P. Using this notation, the goal of the $\mathsf{GenToken}$ protocol is to let $\mathcal{C}$ compute token corresponding to its query $\bar{w}$ only if $\mathsf{av}(\bar{w}) \in \mathsf{P}$. Reflecting these goals, an OSPIR-SSE scheme is a tuple $\Sigma = (\mathsf{EDBSetup}, \mathsf{GenToken}, \mathsf{Search})$ where $\mathsf{EDBSetup}$ is an algorithm run by $\mathcal{D}$ on inputs $(\mathsf{DB}, \mathsf{RDK})$ with outputs (EDB, K), $\mathsf{GenToken}$ is a protocol run by $\mathcal{C}$ on input $\bar{w}$ and by $\mathcal{D}$ on input (P, K), with $\mathcal{C}$ outputting token or $\perp$ and $\mathcal{D}$ outputting av, and Search is a protocol run by $\mathcal{C}$ on input token and by $\mathcal{E}$ on input EDB, with $\mathcal{C}$ outputting a set of ind's matching his query and the corresponding set of rdk's.

4.1 The OSPIR-OXT Protocol

$\mathsf{GenToken}$ protocol

- Client $\mathcal{C}$, on input $\bar{w} = (w_1, ..., w_n)$ where w_1 is chosen as s-term:
  - Compute $(a_s, r_s) \leftarrow \mathsf{OPRF}.\mathcal{C}_1(w_1)$, and $(a_i, r_i) \leftarrow \mathsf{S\text{-}OPRF}.\mathcal{C}_1(w_i)$ for each $i = 1, ..., n$.
  - Send $(a_s, a_1, ..., a_n)$ and $\mathsf{av} = (I(w_1), ..., I(w_n))$ to $\mathcal{D}$.
- Data owner $\mathcal{D}$, on input policy P and master key $K = (K_S, K_X, K_T, K_I, K_P, K_M)$:
  - Abort if av is not in policy set P. Otherwise set av as $\mathcal{D}$'s local output.
  - Compute $b_s \leftarrow \mathsf{OPRF}.\mathcal{D}(K_S, a_s)$.
  - Compute $(b_1, \rho_1) \leftarrow \mathsf{S\text{-}OPRF}.\mathcal{D}(K_T, I_1, a_1)$, and $(b_i, \rho_i) \leftarrow \mathsf{S\text{-}OPRF}.\mathcal{D}(K_X, I_i, a_i)$ for $i = 2, .., n$.
  - Set $\mathsf{env} \leftarrow \mathsf{AuthEnc}(K_M, (\rho_1, \rho_2, ..., \rho_n))$ and send $(\mathsf{env}, b_s, b_1, ..., b_n)$ to $\mathcal{C}$.
- $\mathcal{C}$ outputs $\mathsf{token} = (\mathsf{env}, \mathsf{strap}, \mathsf{bstag}, \mathsf{bxtrap}_2, ..., \mathsf{bxtrap}_n)$ where $\mathsf{strap} \leftarrow \mathsf{OPRF}.\mathcal{C}_2(b_s, r_s)$, $\mathsf{bstag} \leftarrow \mathsf{S\text{-}OPRF}.\mathcal{C}_2(b_1, r_1)$, and $\mathsf{bxtrap}_i \leftarrow \mathsf{S\text{-}OPRF}.\mathcal{C}_2(b_i, r_i)$ for $i = 2, ..., n$.

Figure 2: Token Generation in OSPIR-OXT

The $\mathsf{OSPIR\text{-}OXT}$ protocol addresses the above OSPIR setting by enhancing $\mathsf{MC\text{-}OXT}$ with query-hiding techniques that allow $\mathcal{D}$ to authorize queries without learning the queried values. Most changes with respect to $\mathsf{MC\text{-}OXT}$ are in the $\mathsf{GenToken}$ protocol. $\mathsf{EDBSetup}$ remains mostly unchanged except for the implementation of the PRFs, and Search is essentially unmodified.

We introduce the two main tools used in the design of $\mathsf{GenToken}$. First, instead of the

[3] We assume that the string representing w has its attribute encoded into it, i.e. $w = (i, val)$ for $i = I(w)$, so that Ryan as a first name is distinguished from Ryan as a last name.

use of regular PRFs for stag and xtag computations in OXT and MC-OXT, the OSPIR-OXT protocol uses an an "oblivious PRF" (OPRF) computation between $\mathcal{C}$ and $\mathcal{D}$. A PRF $F(K, w)$ is called oblivious [25] if there is a two-party protocol in which $\mathcal{C}$ inputs w, $\mathcal{D}$ inputs K, $\mathcal{C}$ learns the value of $F(K, w)$ and $\mathcal{D}$ learns nothing. A simple example is the Hashed DH OPRF which we use in our implementation of the OSPIR-OXT protocol, defined as $F(K, x) = H(x)^K$ where H is a hash function onto $G \setminus \{1\}$ where G is a group of prime order p, and K is chosen at random in Z_p^*. In this case, the OPRF protocol consists of $\mathcal{C}$ sending $a = H(x)^r$ for random r in Z_p^*, $\mathcal{D}$ sending back $b = a^K$ and $\mathcal{C}$ computing $H(x)^K$ as $b^{1/r}$. The second tool used in GenToken is needed to enforce an *attribute-based policy* and guarantee that only queries on authorized attributes can generate valid tokens for search at $\mathcal{E}$. To enforce such policies we have $\mathcal{D}$ use a different key for each possible attribute; for example, when a stag (or xtag) is requested for attribute 'zipcode' the key that $\mathcal{D}$ inputs into the OPRF computation is different than the key used for attribute 'name' or attribute 'text'. The point is that if $\mathcal{C}$ claims to be querying zipcode but actually enters the keyword "Michael" into the OPRF computation, the output for $\mathcal{C}$ will be the tag $F(K_{zip},$ "Michael"), where K_{zip} is a zipcode-specific key, which will match no tag stored at $\mathcal{E}$.

To obtain OSPIR-OXT we combine the above two tools with an authorization mechanism similar to the one used in MC-OXT via a homomorphic signature (using the ρ_i exponents) for binding together the n tokens corresponding to an n-term conjunctive query in a way that $\mathcal{E}$ can verify. We describe the changes to MC-OXT (defined via Figure 1 and the modifications in Section 3) required by OSPIR-OXT. We first replace the PRF F_p used in computing xtrap and xtag values with a PRF F_G which maps w directly onto the group G generated by g, i.e. we set xtrap as $F_G(K_X, w)$ instead of $g^{F_p(K_X,w)}$, hence $\mathsf{xtag} = (\mathsf{xtrap})^{\mathsf{xind}}$ will now be computed as $F_G(K_X, w)^{\mathsf{xind}}$ instead of $g^{F_p(K_X,w)\cdot \mathsf{xind}}$. We similarly replace the PRF F_τ used in computing the strap value with the PRF F_G, i.e. we set strap as $F_G(K_S, w)$ instead of $F_\tau(K_S, w)$. (Since we use strap as a key to F_τ in deriving (K_z, K_e), we assume that a PRF F_τ key can be extracted from a random group element.)

We also make a specific assumption on the implementation of the function TSetGetTag used to derive $\mathsf{stag}(w)$ value, i.e., the handle pointing to the set $\mathsf{TSet}(w)$ which is computed as $\mathsf{TSetGetTag}(K_T, w)$. First, we assume that TSetGetTag is implemented using PRF F_G. Second, to enable enforcement of attribute-based policies we assume that the key K_T in TSetGetTag is formed by an array of F_G keys $K_T = (K_T[1], ..., K_T[m])$, where $K_T[i]$ is the key to be used only for keywords with attribute $I(w) = i$. For notational convenience we define a PRF F_G^m s.t. $F_G^m(K_T, w) = F_G(K_T[I(w)], w)$, and we set $\mathsf{stag}(w) = \mathsf{TSetGetTag}(K_T, w)$ to $F_G^m(K_T, w)$. Since we explicitly handle the keys used in the TSetGetTag implementation we also need to modify the TSet API: We will initialize TSet as $\mathsf{TSet} \leftarrow \mathsf{TSetSetup}'(\mathbf{T})$, where $\mathbf{T}$ indexes the tuple lists $\mathbf{t}(w)$ not by the keywords w but by the corresponding $\mathsf{stag}(w)$ values. (This API change does not affect existing TSet implementations [9] because they internally use $\mathsf{stag}(w) = \mathsf{TSetGetTag}(K_T, w)$ to store the $\mathbf{t}(w)$ list.) The PRF we use in the computation of xtag's will be similarly attribute-partitioned. Namely, K_X is also an array of m independent F_G keys $K_X = (K_X[1], \ldots, K_X[m])$, the xtrap value

for keyword w is defined as $F_G^m(K_X, w)$, and the xtag corresponding to keyword w and index xind is set to $(F_G^m(K_X, w))^{\mathsf{xind}}$.

In OSPIR-OXT, there are two two-party protocols involved in the computation of F_G. In the first case, the protocol implements an OPRF computation in which $\mathcal{C}$ enters an input w, $\mathcal{D}$ enters a key K_S, and the output is $F_G(K_S, w)$ for $\mathcal{C}$ and $\perp$ for $\mathcal{D}$. In the second case, F_G^m is computed via a protocol, that we call a *shared OPRF (*S-OPRF*)*, in which $\mathcal{C}$ inputs w and $i = I(w)$, and $\mathcal{D}$ enters a key K and additional input $\rho \in Z_p^*$; the output learned by $\mathcal{D}$ is i, and the output learned by $\mathcal{C}$ is $(F_G^m(K, x))^\rho = (F_G(K[i], x))^\rho$. Note that the pair of outputs $((F_G^m(K, x))^\rho, \rho)$ can be seen as a secret sharing of $F_G^m(K, x)$, hence the name shared-OPRF. OSPIR-OXT uses the OPRF protocol to let $\mathcal{C}$ learn the strap value corresponding to the w_1 s-term, i.e. strap $= F_G(K_S, w_1)$, without $\mathcal{D}$ learning w_1. The S-OPRF protocol is used to let $\mathcal{C}$ compute a *blinded stag* bstag $= [F_G^m(K_T, w_1)]^{\rho_1}$ and the *blinded xtraps* $\mathsf{bxtrap}_i = [F_G^m($ $K_I, w_i)]^{\rho_i}$, for $i = 2, ..., n$. The functionality of the blinding ρ_i is the same as in the case of MC-OXT, namely, as a form of homomorphic signature binding and authorizing stag and xtrap's that $\mathcal{E}$ can verify. As in MC-OXT, $\mathcal{E}$ will receive the corresponding (de)blinding factors $\rho_1, \ldots, \rho_n$ in the authenticated envelope env.

To simplify the description of OSPIR-OXT, we assume that both OPRF and S-OPRF protocols take a single round of interaction between $\mathcal{C}$ and $\mathcal{D}$, as is indeed the case for several efficient OPRF's of interest [14, 20], including the Hashed Diffie-Hellman OPRF [21] used in our implementation below. We denote $\mathcal{C}$'s initial computation in the OPRF protocol as $(a, r) \leftarrow \mathsf{OPRF}.\mathcal{C}_1(x)$ (a is the value sent to $\mathcal{D}$ and r is randomness used by $\mathcal{C}$), $\mathcal{D}$'s response computation as $b \leftarrow \mathsf{OPRF}.\mathcal{D}(K, a)$, and $\mathcal{C}$'s local computation of the final output as $\mathsf{OPRF}.\mathcal{C}_2(b, r)$. We use the corresponding notation in the case of S-OPRF, except that S-OPRF.$\mathcal{D}$ takes as an input a triple (K, i, a) where i is an attribute and outputs a pair (b, ρ). See below for a simple implementation of these procedures for the case of the Hashed Diffie-Hellman OPRF.

OSPIR-OXT **Specification.** With the above ingredients and notation we specify OSPIR-OXT on the basis of MC-OXT via the following changes.

Keys. Select key K_S for F_G; K_T and K_X for F_G^m; K_I for F_p; and K_M for the authenticated encryption scheme.

EDBSetup. Follow the EDBSetup procedure of MC-OXT except for computing strap $\leftarrow$ $F_G(K_S, w)$ and xtag $\leftarrow (F_G^m(K_X, w))^{\mathsf{xind}}$, and for implementing the TSetGetTag procedure as $\mathsf{TSetGetTag}(K_T, w) = F_G^m(K_T, w)$, which means that we compute $\mathsf{stag}(w) \leftarrow F_G^m(K_T, w)$; index $\mathbf{t}(w)$ in $\mathbf{T}$ by $\mathsf{stag}(w)$ instead of by w itself; and generate TSet using the modified API procedure $\mathsf{TSetSetup}'(\mathbf{T})$.

GenToken protocol. This is the main change with respect to OSPIR-OXT; it follows the above mechanisms and is presented in Figure 2.

Search protocol. Same as MC-OXT except that stag is not included under env but rather it is provided to $\mathcal{E}$ by $\mathcal{C}$ as bstag from which $\mathcal{E}$ computes $\mathsf{stag} \leftarrow (\mathsf{bstag})^{1/\rho_1}$.

Instantiation via Hashed Diffie-Hellman OPRF. Our implementation and analysis of OSPIR-OXT assumes the use of Hashed DH OPRF mentioned above, namely, $F_G(K, x) = (H(x))^K$. The instantiations of OPRF and S-OPRF protocols in this case are as follows. OPRF.$\mathcal{C}_1(x)$ and S-OPRF.$\mathcal{C}_1(x)$ both pick random r in Z_p^*, set $a \leftarrow (H(x))^r$, and output (a, r). Procedure OPRF.$\mathcal{D}(K, a)$, where $K \in Z_p^*$ is a key for PRF F_G, outputs $b \leftarrow a^K$. Procedure S-OPRF.$\mathcal{D}(K, i, a)$, where $i \in \{1, ..., m\}$ and $K = (K[1], \ldots, K[m]) \in (Z_p^*)^m$ is a key for PRF F_G^m, picks random ρ in Z_p^*, computes $b \leftarrow a^{K[i]\cdot\rho}$, and outputs (b, ρ). Procedures OPRF.$\mathcal{C}_1(b, r)$ and S-OPRF.$\mathcal{C}_1(b, r)$ both output $b^{1/r}$. Note that if parties follow the protocol, $\mathcal{C}$'s final output is equal to $(H(x))^K = F_G(K, x)$ in the OPRF protocol, while in the S-OPRF protocol it is equal to $(H(x))^{K[i]\cdot\rho}$, which is equal to $(F_G^m(K, x))^\rho$ if $i = I(x)$. These OPRF and S-OPRF protocols emulate their corresponding ideal functionalities in ROM under so-called One-More Gap Diffie-Hellman assumption [21], see Section 5.

Figure 4 in Appendix B shows the OSPIR-OXT scheme instantiated with the above Hashed DH OPRF. It helps visualize the entire protocol and it reflects our actual implementation. In it we denote keys K_T and K_X of PRF F_G^m by vectors of exponents in Z_p^*, respectively $(k_1, \ldots, k_m)$ and $(e_1, \ldots, e_m)$, where m is the number of attributes. Also, because of the specific OPRF instantiation we equate a_s to a_1 in $\mathcal{C}$'s message of the GenToken protocol, instead of computing these two blinded versions of keyword w_1 separately, as in Figure 2.

4.2 Supporting Boolean Queries

For simplicity we presented our protocols for the case of conjunctions. The protocols can be readily adapted to search boolean queries in "searchable normal form (SNF)", i.e., of the form "$w_1 \wedge \phi(w_2, \ldots, w_m)$" (intended to return any document that matches keyword w_1 and in addition satisfies the formula ϕ on the remaining keywords). In this case, OXT and its derivatives change only in the way $\mathcal{E}$ determines which tuples match a query (i.e., which values e it sends back to $\mathcal{C}$). Specifically, in OXT the c-th tuple matches if and only if $\mathsf{xtoken}[c, i]^{y/\rho_i} \in \mathsf{XSet}$ for all $2 \le i \le n$. Instead, for boolean queries as above, $\mathcal{E}$ will have a set of boolean variables $v_2, \ldots, v_n$ and will set v_i to the truth value of the predicate $\mathsf{xtoken}[c, i]^{y/\rho_i} \in \mathsf{XSet}$. A tuple is matching if and only if the result of evaluating ϕ on these values returns true. The complexity of boolean search is same as for conjunctions, i.e., proportional to $|\mathsf{DB}(w_1)|$, and leakage to $\mathcal{E}$ is the same as for a conjunctive query on the same set of keywords except that $\mathcal{E}$ also learns the expression ϕ being evaluated. See [9] for details and support of other forms of boolean queries.

In the OSPIR-SSE setting, supporting boolean queries requires policies that are defined in terms of such queries. Specifically, a policy will determine a set of allowed pairs (ψ, I) where ψ is a symbolic boolean expression and I a sequence of attributes, one per each variable in ψ. Thus, leakage to $\mathcal{D}$ will include I (as in the case of conjunctions) plus the symbolic expression being evaluated.

4.3 Computational Cost

Here we provide an operations count for OSPIR-OXT when instantiated with the DH-based OPRF noting its (mild) overhead over the original OXT protocol from [9]. The computational cost of OSPIR-OXT is easy to quantify by inspecting Figure 4 in Appendix B.

The cost of pre-processing (EDBSetup) is dominated by operations related to the group G, mainly exponentiations: For every $w \in \mathsf{W}$:

1. One hashing operation, $H(w)$, of keyword w into an element of the group G.
2. Two exponentiations: $\mathsf{strap}(w) = (H(w))^{K_S}$ and $\mathsf{stag}(w) = (H(w))^{k_i}$.
3. For every ind in $\mathsf{DB}(w)$: One exponentiation $H(w)^{e_i \cdot \mathsf{xind}}$ for computing an XSet element.

The first two items are specific to OSPIR-OXT while the third is from the original OXT protocol, except that here the base for exponentiation is changed from the generator g in OXT to the value $H(w)$. Hence the overhead introduced by OSPIR-OXT is given by the first two items and the variable base. Importantly, the overhead for the first two items is only linear in $|\mathsf{W}|$, typically much smaller than the number of exponentiations in OXT (item 3), namely one per pair (w, ind) for $\mathsf{ind} \in \mathsf{DB}(w)$. As for the latter exponentiations, while the bases are variable, each $H(w)$ is typically used with a very large number of exponentiations (as the number of documents containing w) hence allowing for significant same-base optimizations. The hashing of w into the group G (item 1) is modeled as a random oracle, hence it rules out algebraic implementations such as $g^{h(w)}$. For the elliptic curves groups we use, H is realized by sampling a field element e and a bit b from a PRNG seeded with w until (e, b) is the compressed representation of a valid group element. Depending on the particular nature of a curve, the generic square root algorithms required in solving the Weierstrass equation, though, can be extremely inefficient: Our original implementation for the chosen NIST 224p curve using OpenSSL's standard algorithms was more than an order of magnitude slower than a normal exponentiation and considerably more when common-base optimization is used. By implementing our own algorithm inspired by [4] and optimizing it for this particular field, we could reduce the cost down to the order of an exponentiation. Once all these optimizations are in place, the performance of OSPIR-OXT is remarkable as shown in Section 4.4 and, in much more details, in a companion paper [8].

The dominating cost of GenToken is just $2n + 1$ exponentiations for the client and $n + 1$ for $\mathcal{D}$.

Finally, the cost of query processing between $\mathcal{C}$ and $\mathcal{E}$ (Search) on a n-term SNF expression is as follows:

1. $\mathcal{C}$ computes $n - 1$ exponentiations for each tuple in $\mathsf{TSet}(w_1)$
2. $\mathcal{E}$ performs *up to* $n - 1$ exponentiations for each element in $\mathsf{TSet}(w_1)$.

Note that $\mathcal{C}$ can apply same-base optimization to the exponentiations since each term in the SNF expression has its own fixed base. On the other hand, $\mathcal{E}$ cannot use same-base optimizations. However, note that as soon as one of the values $\mathsf{xtoken}[c,i]^{y/\rho_i}$ for a conjunction is found not to be in XSet, the other terms for this conjunction do not have to be evaluated (hence avoiding the need for these exponentiations). Similarly, for general Boolean expressions, early termination can be exploited to reduce costly computation. This highlights an important optimization for query processing (especially for queries with large $\mathsf{TSet}(w_1)$ sets): Besides choosing the s-term as a term with high-entropy to keep $\mathsf{TSet}(w_1)$ small, also choose the evaluation order of x-terms of an SNF expression such that it maximizes the probability of early termination and, hence, reduces the number of (expected) exponentiations executed by $\mathcal{E}$, e.g., for conjunctions order the x-terms in descending order of entropy (equivalently, ascending order of frequency).

Note (frequency ordering). In the OSPIR setting the client may not know the frequency of terms in the database and $\mathcal{D}$ will not know the query values to choose such terms optimally. Thus, the exact mechanism for determining the above ordering will depend on the specific setting. In our implementation we decide on ordering based on typical entropy of attributes; e.g., assuming last names have more entropy than names, and names more entropy than addresses, etc. (note that an attribute-based ordering is more privacy-preserving for the client than a value-based one).

4.4 Implementation and performance

The practicality of the proposed schemes was validated by experiments on DBs which included e.g. English-language Wikipedia (13,284,801 records / 2,732,311,945 indexed tuples), and a synthetic US census database (100 million records / 22,525,274,592 index tuples, resulting in EDB with 1.7 TB TSet and 0.4 TB XSet). To illustrate search efficiency, in the census DB case we executed complex queries like

SELECT ID WHERE FNAME='CHARLIE' AND SEX='FEMALE' AND NOT (STATE='NY' OR STATE='MA' OR STATE='PA' OR STATE='NJ)

in about 4 seconds on an IBM Blade dual Intel 4-core Xeon processor and storage provided by a (low-end) 6.2TB RAID-5 storage system. Preprocessing of such large DBs (TSet and XSet creation) has been feasible by, among other things, optimization of common-base exponentiations, achieving aggregated (parallel) rate of about 500,000 exp.'s/sec. for the NIST 224p elliptic curve.

See [8] for details on implementation and performance as well as for the extension of OXT and $\mathsf{OSPIR\text{-}OXT}$ to support dynamic databases (where documents can be added, deleted and modified).

5 Security

We analyze security of the OSPIR-SSE scheme. We focus on the OSPIR case as it is the more comprehensive setting and it contains MC-SSE as a special case. The SSE-OXT protocol is analyzed in the SSE setting in [9].

5.1 Extended Notion of TSet Correctness.

As we mentioned in Section 2, in the context of MC-SSE and OSPIR-SSE setting the correctness property of the TSet datastructure must be extended to include the (lack of) false positives in addition to the false negatives. Intuitively, whereas in all SSE settings we must assure that the metadata $\mathbf{T}[w]$ associated with keyword w can be recovered in its entirety given the $\mathsf{stag}(w)$ search handle, in the MC-SSE and OSPIR-SSE settings we must in addition assure that no data is returned without an explicitely provided search handle. To simplify stating and using the latter property we restrict it to the case where TSetGetTag is a deterministic function, i.e. where for each K_T, w there exists a unique $\mathsf{stag} = \mathsf{TSetGetTag}(K_T, w)$.

Formally, we call a T-set scheme (computationally) *correct* if for every $\mathbf{T}$ and every efficient algorithm A, the following two facts hold, reflecting negligible probability of false negatives and false positives, respectively. First, there must be at most negligible probability that $\mathbf{t} \neq \mathbf{T}[w]$ in the experiment where $(\mathsf{TSet}, K_T) \leftarrow \mathsf{TSetSetup}(\mathbf{T})$, $w \leftarrow A(\mathbf{T}, \mathsf{TSet}, K_T)$, $\mathsf{stag} \leftarrow \mathsf{TSetGetTag}(K_T, w)$, and $\mathbf{t} \leftarrow \mathsf{TSetRetrieve}(\mathsf{TSet}, \mathsf{stag})$. Secondly, there must be at most negligible probability that $b = 1$ in an experiment where $(\mathsf{TSet}, K_T) \leftarrow \mathsf{TSetSetup}(\mathbf{T})$, $\mathsf{stag}^* \leftarrow \mathbf{Adv}(\mathbf{T}, \mathsf{TSet}, K_T)$, $\mathbf{t} \leftarrow \mathsf{TSetRetrieve}(\mathsf{TSet}, \mathsf{stag}^*)$, and we assign $b \leftarrow 1$ if it holds that $\mathbf{t}$ is non-empty *and* $\mathsf{stag}^* \neq \mathsf{TSetGetTag}(K_T, w)$ for all $w \in \mathsf{W}$; otherwise $b \leftarrow 0$.

It is easy to see that this extended correctness notion is satisfied by TSet implementations of both [9] and [8].

5.2 OSPIR-SSE Security and Correctness Definitions

SSE security definitions where the only adversarial entity is server $\mathcal{E}$ are provided in prior work. Here we follow the definitions from [9] - which in turn follow [11, 13] - and extend them to the MC setting by considering multiple malicious clients and to the OSPIR setting by adding also the data owner $\mathcal{D}$ as an adversarial entity. All security definitions follow the ideal/real model framework of secure computation and are parametrized by a *leakage function* $\mathcal{L}$ bounding the information leaked to an adversarial party in addition to the intended output for that party. Specifically, we ask that whatever an adversary can do by running the real protocol on data and queries *chosen by the adversary*, a simulator can do solely on the basis of the leakage function.

Correctness. We say that an OSPIR-SSE scheme $\Sigma = (\mathsf{EDBSetup}, \mathsf{GenToken}, \mathsf{Search})$ is *computationally correct* if for every efficient algorithm A, there is a negligible probability that the following experiment outputs 0. On inputs $(\mathsf{DB}, \mathsf{RDK})$ and $\bar{w}^{(1)}, \ldots, \bar{w}^{(m)}$ provided by A, execute $(K, \mathsf{EDB}) \leftarrow \mathsf{EDBSetup}(\mathsf{DB}, \mathsf{RDK})$; and for $i = 1, 2, \ldots$, execute protocol $\mathsf{GenToken}$ on $\mathcal{C}$'s input $\bar{w}^{(i)}$ and $\mathcal{D}$'s inputs (P, K), denote $\mathcal{C}$'s output as $\mathsf{token}^{(i)}$, execute protocol Search between $\mathcal{C}$ on input $\mathsf{token}^{(i)}$ and $\mathcal{E}$ on input EDB and denote $\mathcal{C}$'s outputs as a pair $(\mathsf{indSet}^{(\mathsf{i})}, \mathsf{rdkSet}^{(\mathsf{i})})$. Output 1 if for each i we have that $\mathsf{indSet}^{(\mathsf{i})} = \mathsf{DB}(\bar{\mathsf{w}}^{(\mathsf{i})})$ and $\mathsf{rdkSet}^{(\mathsf{i})} = \mathsf{RDK}[\mathsf{DB}(\bar{\mathsf{w}}^{(\mathsf{i})})]$. Otherwise output 0.

Security against adversarial server $\mathcal{E}$. Security against adversarial (honest-but-curious) $\mathcal{E}$ has been the focus of prior SSE work. Adapting the definition of $\mathcal{L}$-semantic security against adaptive attacks (by the server $\mathcal{E}$) from [9] to our setting is straightforward and is omitted here.

Security against adversarial clients. The definition captures the information leaked to a malicious client in addition to the intended output $\mathsf{DB}(\bar{w})$ and the corresponding record-decrypting keys $\mathsf{RDK}[\mathsf{DB}(\bar{w})]$. The definition compares the real execution to an emulation of an interaction with algorithm $\mathsf{I\text{-}SSE}_{\mathcal{L}}$, which models an ideal functionality of the OSPIR-SSE scheme instantiated with the leakage function $\mathcal{L}$. The interactive algorithm $\mathsf{I\text{-}SSE}_{\mathcal{L}}$, running on local input $(\mathsf{DB}, \mathsf{RDK}, \mathsf{P})$, answers queries $\bar{w} \in \mathsf{W}^*$ by checking if $\mathsf{av}(\bar{w}) \in \mathsf{P}$. If the check verifies, then it replies to this $\bar{w}$ with a triple $(\mathsf{DB}(\bar{w}), \mathsf{RDK}[\mathsf{DB}(\bar{w})], \mathcal{L}(\mathsf{DB}, \bar{w}))$, and if $\mathsf{av}(\bar{w}) \notin \mathsf{P}$ then it sends back a rejection symbol $\perp$.

Definition 1 *Let $\Pi = (\mathsf{EDBSetup}, \mathsf{GenToken}, \mathsf{Search})$ be an OSPIR-SSE scheme. Given algorithms $\mathcal{L}$, A, and $S = (S_0, S_1, S_2)$ we define experiments (algorithms) $\mathbf{Real}^{\Pi}_{A}(\lambda)$ and $\mathbf{Ideal}^{\Pi}_{A,S}(\lambda)$ as follows:*

$\mathbf{Real}^{\Pi}_{A}(\lambda)$: *$A(1^\lambda)$ chooses $(\mathsf{DB}, \mathsf{RDK}, \mathsf{P})$, and the experiment runs $(K, \mathsf{EDB}) \leftarrow \mathsf{EDBSetup}(\mathsf{DB}, \mathsf{RDK})$. Adversary A can then adaptively invoke instances of the protocol $\mathsf{GenToken}$ and Search, interacting with party $\mathcal{D}$ running on input K and P in the first case and with party $\mathcal{E}$ running on input EDB in the second case. Note that A can behave arbitrarily in all these protocol instances. Let q be the number of $\mathsf{GenToken}$ instances and let av_i be $\mathcal{D}$'s local output in the i-th instance. If at any point A halts and outputs a bit b, the game outputs $(b, \mathsf{av}_1, \ldots, \mathsf{av}_q)$.*

$\mathbf{Ideal}^{\Pi}_{A,S}(\lambda)$: *$A(1^\lambda)$ chooses $(\mathsf{DB}, \mathsf{RDK}, \mathsf{P})$ as above, while the experiment initializes $S = (S_0, S_1, S_2)$ by running $\mathsf{st} \leftarrow S_0(1^\lambda)$. Subsequently, each time A invokes an instance of protocol $\mathsf{GenToken}$, it interacts with the experiment running $S_1(\mathsf{st}, \mathsf{P})$, whereas if A invokes an instance of protocol Search, it interacts with the experiment running $S_2(\mathsf{st})$. Both S_1 and S_2 algorithms are allowed to update the global simulator's state st while interacting with A. Both can issue queries $\bar{w}$ to $\mathsf{I\text{-}SSE}_{\mathcal{L}}(\mathsf{DB}, \mathsf{RDK}, \mathsf{P})$. Let q be the number of these queries and let $\mathsf{av}_i = I(\bar{w}_i)$, where $\bar{w}_i$ is the i-th query. As above, if at any point A halts and output a bit b, the game outputs $(b, \mathsf{av}_1, \ldots, \mathsf{av}_q)$.*

We call Π $\mathcal{L}$-semantically-secure against malicious clients *if for any efficient algorithm* A *there is an efficient algorithm* S *s.t. the statistical difference between tuples* $(b, \mathsf{av}_1, \ldots, \mathsf{av}_q)$ *output by experiments* $\mathbf{Real}^{\Pi}_{A}$ *and* $\mathbf{Ideal}^{\Pi}_{A,S}$ *is a negligible function of the security parameter* λ.

Security against adversarial data owner. Security against a data-owner $\mathcal{D}$ models privacy of the client's queries $\bar{w}$ against malicious $\mathcal{D}$, given an adaptive choice of the client's queries. Similarly to the case of security against either the client $\mathcal{C}$ or the EDB-storing server $\mathcal{E}$, this security definition also allows for leakage of some information $\mathcal{L}(\bar{w})$ regarding the query $\bar{w}$ to $\mathcal{D}$.

Definition 2 *Let* $\Pi = (\mathsf{EDBSetup}, \mathsf{GenToken}, \mathsf{Search})$ *be an OSPIR-SSE scheme. Given algorithms* $\mathcal{L}$, A, *and* S *we define experiments (algorithms)* $\mathbf{Real}^{\Pi}_{A}(\lambda)$ *and* $\mathbf{Ideal}^{\Pi}_{A,S}(\lambda)$ *as follows:*

$\mathbf{Real}^{\Pi}_{A}(\lambda)$**:** *Adversary* $A(1^{\lambda})$ *can adaptively invoke any number of* $\mathsf{GenToken}$ *instances by specifying a query* $\bar{w}$ *and interacting with party* $\mathcal{C}$ *running the* $\mathsf{GenToken}$ *protocol on input* $\bar{w}$. *At any point* A *can halt and output a bit, which the game uses as its own output.*

$\mathbf{Ideal}^{\Pi}_{A,S}(\lambda)$**:** *Adversary* $A(1^{\lambda})$ *can adaptively invoke any number of* $\mathsf{GenToken}$ *instances as above, but for any query* $\bar{w}$ *which* A *specifies, it interacts with* S *running on input* $\mathcal{L}(\bar{w})$. *As above if* A *halts and output a bit, the game uses this bit as its own output.*

We call Π $\mathcal{L}$-semantically-secure against malicious data owner *if for any efficient alg.* A *there is an efficient alg.* S *s.t.* $\Pr[\mathbf{Real}^{\Pi}_{A}(\lambda)=1] - \Pr[\mathbf{Ideal}^{\Pi}_{A,S}(\lambda)=1] \leq \mathrm{neg}(\lambda)$.

Note (non-collusion between $\mathcal{D}$ *and* $\mathcal{E}$*).* We stress that even though the data owner $\mathcal{D}$ can be arbitrarily malicious, we assume that $\mathcal{D}$ and $\mathcal{E}$ do not collude. Indeed, client's security in the $\mathsf{OSPIR\text{-}OXT}$ scheme we propose is not maintained against such collusion. Moreover, providing query-privacy from $\mathcal{D}$ under collusion with $\mathcal{E}$ would have the (impractical) cost of a single-server symmetric PIR protocol. The hospital example mentioned in the introduction, is a case where such non-collusion requirement makes sense. Indeed, it is the hospital interest not to learn the queries: It helps avoiding liability and complying with regulations as well as withstanding potential insider attacks. Similarly, for a service providing private access to a database (e.g. to a patent repository) preserving client privacy is part of its very business model. See also [18].

5.3 Security of OSPIR-OXT

Correctness of OSPIR-OXT. We first argue that protocol $\mathsf{OSPIR\text{-}OXT}$ is correct. Assuming a computationally correct TSet implementation (see Section 2.1), any correctness

errors can only come from collisions in PRF functions, including function F_{K_X,K_I} effectively used in computing xtag values, defined as $F_{K_X,K_I}(w,\mathsf{ind}) = (F_G^m(K_X,w))^{F_p(K_I,\mathsf{ind})}$. But assuming the PRF property of F_p, and the PRF property of F_G^m, which holds under DDH in ROM, function F_{K_X,K_I} is a PRF too, and so collision probability is negligible, resulting in negligible error probability over the execution of the OSPIR-OXT correctness experiment. Thus, we have:

Theorem 3 *The OSPIR-SSE scheme* OSPIR-OXT *instantiated with the Hashed Diffie-Hellman OPRF is computationally correct assuming that the DDH assumption holds, that the T-set implementation is computationally correct, that F_p is a secure PRF, and assuming the Random Oracle Model for hash function H.*

Security of OSPIR-OXT. Using the security notions explained above we describe the security properties of the OSPIR-SSE scheme OSPIR-OXT instantiated with the Hashed Diffie-Hellman OPRF, as shown in Figure 4 in Appendix B. We first state the OM-GDH security assumption required for the security of the OPRF and S-OPRF sub-protocols of this OSPIR-OXT instantiation.

One-More Gap Diffie-Hellman (OM-GDH). Let $G = G_\lambda$ be a prime order cyclic group of order $p = p(\lambda)$ generated by g. We say that the *One-More Gap Diffie-Hellman (OM-GDH) assumption* holds in G if $\mathbf{Adv}^{\mathrm{ddh}}_{G,A}(\lambda)$ is negligible for all efficient adversaries A, where $\mathbf{Adv}^{\mathrm{ddh}}_{G,A}(\lambda)$ is defined as the probability that A wins the following game: (1) The game chooses random t in Z_p^* and two random elements h_1, h_2 in G; (2) A, on input h_1, h_2, specifies a single query a to the Diffie-Hellman oracle, which on input a returns $b \leftarrow a^t$; (3) A can make any number of queries to a *Decisional Diffie-Hellman* oracle $\mathsf{DDH}_\mathsf{t}(\cdot,\cdot)$, which on input (h, v) returns 1 if $v = h^t$ and 0 otherwise; (4) Finally A outputs two values v_1, v_2, and we say that A *wins* the game if $v_1 = (h_1)^t$ and $v_2 = (h_2)^t$.

Security against adversarial server $\mathcal{E}$. The OSPIR-SSE scheme OSPIR-OXT is $\mathcal{L}_{\mathrm{oxt}}$-semantically-secure against adaptive server $\mathcal{E}$ under the same assumptions and for the same leakage function $\mathcal{L}_{\mathrm{oxt}}$ as the underlying SSE scheme OXT of [9]. This is because the specific PRF's used by OSPIR-OXT in EDB construction are instantiations of general PRF's considered in OXT, and because $\mathcal{E}$'s view of the Search protocol in the OSPIR-OXT scheme can be generated from $\mathcal{E}$'s view of Search in the OXT scheme. Specifically, each ρ_i in env is random in Z_p^*, and bstag and each $\mathsf{bxtoken}[c,i]$ value in Figure 4 (Appendix B) can be computed by exponentiating values stag and $\mathsf{xtoken}[c,i]$ in Figure 1 to, respectively, ρ_1 and ρ_i.

Security against adversarial client $\mathcal{C}$. Let $\mathsf{Mask}(|\mathsf{DB}(w_1)|)$ denote an upper bound on $|\mathsf{DB}(w_1)|$ used by $\mathcal{E}$ to mask the size of $\mathsf{TSet}(w_1)$ when responding to $\mathcal{C}$'s queries as described at the end of Section 3.

Theorem 4 *Let $\mathcal{L}$ be a defined as $\mathcal{L}(\mathsf{DB}, \bar{w}) = \mathsf{Mask}(|\mathsf{DB}(w_1)|)$ for $\bar{w} = (w_1, ..., w_n)$. OSPIR-SSE scheme* OSPIR-OXT *instantiated with the Hashed Diffie-Hellman OPRF is $\mathcal{L}$-semantically-secure against malicious clients assuming that the One-More Gap Diffie-Hellman assumption holds in G, that F_p is a secure PRF, that the T-set implementation is (computationally) correct, that* $(\mathsf{AuthEnc}, \mathsf{AuthDec})$ *is an IND-CPA and Strongly-UF-CMA authenticated encryption scheme, and assuming the Random Oracle Model for hash function H.*

Proof: The proof is lengthy and is presented in Appendix A. □

Security against adversarial data owner $\mathcal{D}$. In our OSPIR-SSE scheme a malicious $\mathcal{D}$ learns nothing about clients' query $\bar{w} = (w_1, \ldots, w_n)$ except for the vector of attributes $\mathsf{av}(\bar{w}) = (I(w_1), \ldots, I(w_n))$.

Theorem 5 *Let $\mathcal{L}$ be a leakage function defined as $\mathcal{L}(\bar{w}) = \mathsf{av}(\bar{w})$. OSPIR-SSE scheme* OSPIR-OXT *instantiated with the Hashed Diffie-Hellman OPRF is $\mathcal{L}$-semantically-secure against malicious data owner.*

Proof: The view of $\mathcal{D}$ in the GenToken protocol of OSPIR-OXT (Figure 2) consists of the attribute vector $\mathsf{av}(\bar{w}) = (I_1, ..., I_n)$ corresponding to the query $\bar{w}$ and the values a_s, a_1, $..., a_n$ where a_s is output by $\mathsf{OPRF}.\mathcal{C}_1(w_1)$ and each a_i is output by $\mathsf{S\text{-}OPRF}.Charlie_1(w_i)$. In the Hashed DH OPRF instantiation of this scheme in Figure 4 (Appendix B), these values are formed as $a_j \leftarrow H(w_j)^{r_j}$ for random r_j's in Z_p^* (additionally, a_s is set to a_1). Since G is of prime order, every element in $G \setminus \{1\}$ is a generator, and thus each a_j is uniform in G. Thus, it is straightforward to simulate $\mathcal{D}$'s view of the GenToken protocol from $\mathcal{L}(\bar{w}) = \mathsf{av}(\bar{w})$. □

5.4 Extensions: Reducing Leakage to $\mathcal{D}$

In the full version we show how to adapt OSPIR-OXT to a setting where a third party, called a policy manager, authorizes queries while $\mathcal{D}$ can enforce them without learning the policy, the boolean expression or the queried attributes; only the number of such attributes is learned by $\mathcal{D}$. This setting is precisely what is needed to implement searches authorized by a warrant while keeping the searched information hidden from all parties except the authorized searcher.

In addition, OSPIR-OXT can be extended (even without introducing a policy manager) so that the leakage about queried attributes to $\mathcal{D}$ is further limited to the minimum needed to make policy decisions (e.g., $\mathcal{D}$ may not need to know the exact attributes in a query but only the attribute classes they belong to).

Acknowledgment

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] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, and H. Shi. Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions. In V. Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 205–222. Springer, Aug. 2005.

[2] L. Ballard, S. Kamara, and F. Monrose. Achieving efficient conjunctive keyword searches over encrypted data. In S. Qing, W. Mao, J. López, and G. Wang, editors, *ICICS 05*, volume 3783 of *LNCS*, pages 414–426. Springer, Dec. 2005.

[3] M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In A. Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 535–552. Springer, Aug. 2007.

[4] D. J. Bernstein. Faster square roots in annoying finite fields. http://cr.yp.to/papers/sqroot.pdf, 2001.

[5] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In C. Cachin and J. Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 506–522. Springer, May 2004.

[6] D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In S. P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 535–554. Springer, Feb. 2007.

[7] J. W. Byun, D. H. Lee, and J. Lim. Efficient conjunctive keyword search on encrypted data storage system. In *EuroPKI*, pages 184–196, 2006.

[8] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner. Dynamic Searchable Encryption in Very Large Databases: Data Structures and Implementation. 21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, 2014., 2014.

[9] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. Crypto'2013. Cryptology ePrint Archive, Report 2013/169, Mar. 2013. http://eprint.iacr.org/2013/169.

[10] Y.-C. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In J. Ioannidis, A. Keromytis, and M. Yung, editors, *ACNS 05*, volume 3531 of *LNCS*, pages 442–455. Springer, June 2005.

[11] M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *ASIACRYPT 2010*, LNCS, pages 577–594. Springer, Dec. 2010.

[12] E. D. Cristofaro, Y. Lu, and G. Tsudik. Efficient techniques for privacy-preserving sharing of sensitive information. In J. M. McCune, B. Balacheff, A. Perrig, A.-R. Sadeghi, A. Sasse, and Y. Beres, editors, *TRUST*, volume 6740 of *Lecture Notes in Computer Science*, pages 239–253. Springer, 2011.

[13] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In A. Juels, R. N. Wright, and S. Vimercati, editors, *ACM CCS 06*, pages 79–88. ACM Press, Oct. / Nov. 2006.

[14] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In J. Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 303–324. Springer, Feb. 2005.

[15] E.-J. Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2003. http://eprint.iacr.org/.

[16] P. Golle, J. Staddon, and B. R. Waters. Secure conjunctive keyword search over encrypted data. In M. Jakobsson, M. Yung, and J. Zhou, editors, *ACNS 04*, volume 3089 of *LNCS*, pages 31–45. Springer, June 2004.

[17] Y. Huang and I. Goldberg. Outsourced private information retrieval with pricing and access control. Technical Report 2013-11, Centre for Applied Cryptographic Research (CACR), University of Waterloo, Feb. 2013.

[18] IARPA. Security and Privacy Assurance Research (SPAR) Program - BAA, 2011. http://www.iarpa.gov/solicitations_spar.html/.

[19] M. Islam, M. Kuzu, and M. Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS 2012)*, San Diego, CA, Feb. 2012. Internet Society.

[20] S. Jarecki and X. Liu. Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In O. Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 577–594. Springer, Mar. 2009.

[21] S. Jarecki and X. Liu. Fast secure computation of set intersection. In *SCN 10*, LNCS, pages 418–435. Springer, 2010.

[22] S. Kamara and K. Lauter. Cryptographic cloud storage. In *Financial Cryptography Workshops*, pages 136–149, 2010.

[23] S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *Proc. of CCS'2012*, 2012.

[24] K. Kurosawa and Y. Ohtaki. UC-secure searchable symmetric encryption. In *Financial Cryptography*, page 285, 2012.

[25] M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th FOCS*, pages 458–467. IEEE Computer Society Press, Oct. 1997.

[26] V. Pappas, B. Vo, F. Krell, S. G. Choi, V. Kolesnikov, A. Keromytis, and T. Malkin. Blind Seer: A Scalable Private DBMS. Manuscript, 2013.

[27] E. Shi, J. Bethencourt, H. T.-H. Chan, D. X. Song, and A. Perrig. Multi-dimensional range query over encrypted data. In *2007 IEEE Symposium on Security and Privacy*, pages 350–364. IEEE Computer Society Press, May 2007.

[28] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy*, pages 44–55. IEEE Computer Society Press, May 2000.

[29] P. van Liesdonk, S. Sedhi, J. Doumen, P. H. Hartel, and W. Jonker. Computationally efficient searchable symmetric encryption. In *Proc. Workshop on Secure Data Management (SDM)*, pages 87–100, 2010.

[30] B. R. Waters, D. Balfanz, G. Durfee, and D. K. Smetters. Building an encrypted and searchable audit log. In *NDSS 2004*. The Internet Society, Feb. 2004.

[31] WSJ. U.S. Terrorism Agency to Tap a Vast Database of Citizens. Wall Street Journal 12/13/12. http://alturl.com/ot72x.

A Proof of Theorem 4: Security Against Malicious Clients

Here we prove Theorem 4 formulated in Page 22.

The simulator algorithm $S = (S_0, S_1, S_2)$ for this security proof is shown in Figure 3. Let Π denote the scheme OSPIR-OXT instantiated with the Hashed Diffie-Hellman PRF, let A be an adversary's algorithm and let G_0 denote the experiment $\mathbf{Real}_A^{\Pi}(\lambda)$, which models A's interaction with the real scheme OSPIR-OXT. We will denote by G_i a sequence of modifications of G_0, s.t. the last game, G_{12} is identical to the experiment $\mathbf{Ideal}_{A,S}^{\Pi}(\lambda)$. Let

$\underline{S_0(1^\tau)}$

- Select key K_S for F_G, K_X and K_T for F_G^m, K_P for F_p, and K_M for authenticated encryption.
- Initialize an empty table QList, which will be indexed by ciphertexts env, and a table TList, indexed by keywords w in W, which initially holds an empty set for each w.
- Set $\mathsf{st} \leftarrow (K_S, K_X, K_T, K_P, K_M, \mathsf{QList}, \mathsf{TList})$.

$\underline{S_1(\mathsf{st}, \mathsf{P})}$

- On input $(a_s, a_1, ..., a_n), (I_1, ..., I_n)$ from A, abort if $(I_1, ..., I_n) \notin \mathsf{P}$.
- Pick $\rho'_i, \rho_i \xleftarrow{\$} Z_p^*$ for each $i = 1, ..., n$.
- Set $b_s \leftarrow (a_s)^{K_S}$, $b_1 \leftarrow (a_1)^{K_T[I_1] \cdot \rho_1}$, and $b_i \leftarrow (a_i)^{K_X[I_i] \cdot \rho_i}$ for $i = 2, ..., n$.
- Update QList in st by setting $\mathsf{QList}(\mathsf{env}) \leftarrow (I_1, ..., I_n, \rho_1, ..., \rho_n)$.
- Set $\mathsf{env} \leftarrow \mathsf{AuthEnc}(K_M, (\rho'_1, ..., \rho'_n))$ and output $(\mathsf{env}, b_s, b_1, ..., b_n)$.

$\underline{S_2(\mathsf{st}), \text{ on message } (\mathsf{env}, \mathsf{bstag}, \mathsf{xtoken}[1], \mathsf{xtoken}[2], ...) \text{ from } A:}$

- Retrieve $(I_1, ..., I_n, \rho_1, ..., \rho_n) \leftarrow \mathsf{QList}(\mathsf{env})$. Abort if $\mathsf{QList}(\mathsf{env}) = \perp$.
- Set $\mathsf{stag} \leftarrow (\mathsf{bstag})^{1/\rho_1}$. If there exists $w_1 \in \mathsf{W}$ s.t. $\mathsf{stag} = (H(w_1))^{K_T[I_1]}$ and $I(w_1) = I_1$ then set $\mathsf{strap} \leftarrow F_G(K_S, w_1)$ and $(K_z, K_e) \leftarrow (F_\tau(\mathsf{strap}, 1), F_\tau(\mathsf{strap}, 2))$. Abort if no such w_1 found.
- Set $c \leftarrow 0$ and $\mathsf{found} \leftarrow \mathsf{F}$, and perform the following loop while $\mathsf{found} = \mathsf{F}$.
  - Set $c \leftarrow c + 1$ and $z_c \leftarrow F_p(K_z, c)$. For $\mathsf{xtoken}[c] = (\mathsf{xtoken}[c, 2], ..., \mathsf{xtoken}[c, n])$, if $\exists (w_2, ..., w_n) \in \mathsf{W}^{n-1}$ s.t. $\mathsf{xtoken}[c, i] = (H(w_i))^{K_X[I_i] \cdot z_c \cdot \rho_i}$ and $I(w_i) = I_i$, for $i = 2, ..., n$, then set $\mathsf{found} \leftarrow \mathsf{T}$.
  - Abort if $\mathsf{found} = \mathsf{F}$ and $\mathsf{xtoken}[c]$ is the last element in A's message.
- Send $\bar{w} = (w_1, w_2, ..., w_n)$ to $\mathsf{I\text{-}SSE}_{\mathcal{L}}(\mathsf{DB}, \mathsf{RDK}, \mathsf{P})$ where $w_1, w_2, ..., w_n$ are the keywords found above. Since $\mathsf{av}(\bar{w}) = (I_1, ..., I_n)$ is guaranteed to be included in P, S_2 receives back $(\mathsf{DB}(\bar{w}), \mathsf{RDK}(\mathsf{DB}(\bar{w})), \mathsf{TSetL})$.
- Set $S' \leftarrow \emptyset$ and $D \leftarrow \mathsf{DB}(\bar{w})$. $\forall \mathsf{ind} \in D$ s.t. $(c, \mathsf{ind}, \mathsf{e}) \in \mathsf{TList}(w_1)$, add c to S' and delete ind from D.
- Pick S as random $|D|$-element subset in $\{1, ..., \mathsf{TSetL}\} \setminus S'$, and while S is non-empty do:
  - Remove a random element c from S and a random element ind from D.
  - Set $\mathsf{rind} \leftarrow P_\tau(K_P, \mathsf{ind})$, $\mathsf{rdk} \leftarrow \mathsf{RDK}(\mathsf{ind})$, $\mathsf{e} \leftarrow \mathsf{Enc}(K_e, (\mathsf{rind}|\mathsf{rdk}))$.
  - Update TList in st by adding $(c, \mathsf{ind}, \mathsf{e})$ to $\mathsf{TList}(w_1)$.
- Starting from the last counter c encountered above, perform the following loop while $c \leq \mathsf{TSetL}$:
  - Set $z_c \leftarrow F_p(K_z, c)$. If $\mathsf{xtoken}[c, i] = (H(w_i))^{K_X[I_i] \cdot z_c \cdot \rho_i}$ for each $i = 2, ..., n$, and if there exists $(c, \mathsf{ind}, \mathsf{e})$ in $\mathsf{TList}(w_1)$ s.t. $\mathsf{ind} \in \mathsf{DB}(\bar{w})$, then send e to A, and set $c \leftarrow c + 1$.
- Send stop to A and halt.

Figure 3: $S = (S_0, S_1, S_2)$: Simulators for the security proof of the OSPIR-OXT protocol

p_i be the probability that game G_i outputs 1. We argue that the difference between p_0 and p_{12} is a negligible function of the security parameter λ.

Games G_1 and G_2. In G_1 we modify game G_0 by adding an abort if any authenticated ciphertext env accepted by the server in the Search protocol has not been generated and signed by the data owner in the $\mathsf{GenToken}$ procedure. $p_1 \approx p_0$ by Strong-UF-CMA unforgeability of the authenticated encryption scheme. In game G_2 we add an abort if ever two $\mathsf{GenToken}$ instances generate the same env ciphertext. $p_2 \approx p_1$ e.g. because ρ_i's are generated at random from Z_p^*, and collision in the ciphertext implies collision in the plaintext.

Game G_3. We add an abort if for any two keywords $w, w' \in \mathsf{W}$ s.t. $w' \neq w$ and *either* $F_G^m(K_T, w') = F_G^m(K_T, w)$ *or* $F_G^m(K_X, w') = F_G^m(K_X, w)$. Such collisions occur with negligible probability because K_G^m is a PRF, and therefore $p_3 \approx p_2$.

Game G_4. In the $\mathsf{EDBSetup}$ procedure we skip the step when TSet is created from the array $\mathbf{T}$, and we change the way the game responds in the Search protocol. As in G_0, the game computes $\mathsf{stag} \leftarrow \mathsf{bstag}^{1/\rho_1}$, but then it searches through the keyword space W for w_1 s.t. $\mathsf{stag} = F_G^m(K_T, w_1)$. If such w_1 is found it assigns $\mathbf{t} \leftarrow \mathbf{T}[w_1]$, and otherwise it aborts. Since in game G_3 we have eliminated collisions in function F_G^m, if stag computed above is equal to $\mathsf{stag}(w_1)$ for some $w_1 \in \mathsf{W}$ then by the no-false-negatives part of the TSet correctness property the $\mathbf{t}$ retrieved in game G_3 via $\mathsf{TSetRetrieve}(\mathsf{TSet}, \mathsf{stag})$ is the same as $\mathbf{T}[w_1]$ retrieved in game G_4, except for a negligible probability of error. On the other hand, if $\mathsf{stag} \neq \mathsf{stag}(w)$ for all $w \in \mathsf{W}$, then by the no-false-positives part of the TSet correctness property the $\mathbf{t}$ retrieved in game G_3 is an empty list just like in game G_4, again except for a negligible error probability. It follows that $p_4 \approx p_3$.

Game G_5. Instead of encrypting $(\rho_1, \rho_2, ..., \rho_n)$ in env, G_5 encrypts a string of independent random values $(\rho'_1, \rho'_2, ..., \rho'_n)$. In addition, game G_5 keeps an array QList indexed by ciphertexts env, and when servicing $\mathsf{GenToken}$ requests G_5 stores in $\mathsf{QList}(\mathsf{env})$ the list of attribute indexes $(I_1, ..., I_n)$ and the true blinding values $(\rho_1, ..., \rho_n)$ used in the $\mathsf{GenToken}$ instance which generated ciphertext env. Therefore in the Search procedure, instead of decrypting ρ_i's from env, which now encrypts independent random values, G_5 retrieves these ρ_i's from $\mathsf{QList}(\mathsf{env})$. $p_5 \approx p_4$ by IND-CCA of the authenticated encryption, and because already in game G_2 the only env's which are accepted in Search are uniquely identifying some $\mathsf{GenToken}$ instance.

Game G_6. We modify the test for identifying w_1 given bstag and ρ_1 recovered from $\mathsf{QList}(\mathsf{env})$, to w_1 s.t. $(\mathsf{bstag})^{1/\rho_1} = (H(w_1))^{K_T[I_1]}$ *and* $I(w_1) = I_1$. In other words, compared to G_5, G_6 ignores w_1's s.t. $\mathsf{stag} = (\mathsf{bstag})^{1/\rho_1} = (H(w_1))^{K_T[I(w_1)]}$ but $I(w_1) \neq I_1$. We argue that the probability of finding such w_1 is negligible even against all-powerful adversary. Denote the exponent $K_T[I_1] \cdot \rho_1$ used in computing b_1 from a_1 in j-th $\mathsf{GenToken}$ instance as t_j. Note that each t_j is uniformly random in Z_p^*. Using this notation, the way game G_6 tests each keyword w_1 given bstag in the Search procedure can be rewritten as testing whether $\mathsf{bstag} = (H(w_1))^{(K_T[I(w_1)]/K_T[I_1]) \cdot t_j}$ where t_j is used in $\mathsf{GenToken}$ instance that generated

ciphertext env used in this Search instance. In particular, the discrete logarithm between $(H(w_1))^{t_j}$ and bstag must be equal to $K_T[I(w_1)]/K_T[I_1]$. Note that G_6 does not use the K_T key array in any other way except in this test. Therefore, since K_T component keys are all random in Z_p^*, and the game makes polynomially-many such tests, the probability that such test ever succeeds for $I(w_1) \neq I_1$ is negligible, and hence $p_6 \approx p_5$.

Game G_7. We add an abort if A ever invokes two instances of Search protocol which involve the same env ciphertext but two different values bstag and bstag′ s.t. there exists two keywords w_1, w_1' s.t. $(\mathsf{bstag})^{1/\rho_1} = (H(w_1))^{K_T[I_1]}$ and $(\mathsf{bstag}')^{1/\rho_1} = (H(w_1'))^{K_T[I_1]}$. Use the notation t_j for $K_T[I_1] \cdot \rho_1$ used in the j-th GenToken instance as in game G_6. Note that the above condition implies that $\mathsf{bstag} = (H(w_1))^{t_j}$ and $\mathsf{bstag}' = (H(w_1'))^{t_j}$ if ciphertext env was generated by the j-th instance of GenToken. It is easy to see that the probability of encountering such two pairs of values (w_1, bstag) and (w_1', bstag') must be negligible under the OM-GDH assumption if H is modeled as a random oracle, hence $p_7 \approx p_6$.

Let ϵ be the probability of the above event. The reduction on the input a OM-GDH challenge h_1, h_2, emulates game G_3 except that on each A's query w to H, it picks random r_1, r_2 in Z_p^* and sets $H(w) \leftarrow (h_1)^{r_1}(h_2)^{r_2}$. Furthermore, the reduction picks a random index j between 1 and the maximum number of GenToken instances A can invoke, and when servicing the j-th instance of GenToken, the reduction sends the a_1 value in this instance to the OM-GDH challenger, who replies with $b_1 \leftarrow (a_1)^t$ for t chosen by the OM-GDH challenge game. The reduction passes this b_1 in its response to A. Finally, for each Search protocol instance on which A sends ciphertext env, the reduction takes bstag sent by A along with env, and for each query w made by A to H it consults the DDH oracle if $(a_1, b_1, H(w), \mathsf{bstag})$ is a DDH tuple. If the reduction finds two instances of Search on which the above check verifies, one for (w, bstag) and the other for $(w', \mathsf{bstag}') \neq (w, \mathsf{bstag})$, it computes h_1^t and h_2^t as, respectively, $(\mathsf{bstag}^{r_2}(\mathsf{bstag}')^{-r_2})^{1/(r_1 r_2' - r_1' r_2)}$ and $(\mathsf{bstag}^{r_1}(\mathsf{bstag}')^{-r_1})^{1/(r_1' r_2 - r_1 r_2')}$, where $H(w) = (h_1)^{r_1}(h_2)^{r_2}$ and $H(w') = (h_1)^{r_1'}(h_2)^{r_2'}$. The probability of reduction's success is $(1/m) \cdot \epsilon$ where m is the upper-bound on the number of GenToken instances A invokes, minus the negligible probability that $r_1 r_2' = r_1' r_2$.

Game G_8. We replace the PRF $F_p(K_I, \cdot)$ with a random function $F_I(\cdot)$ onto Z_p^*. $p_8 \approx p_7$ by the PRF property of F_p.

Game G_9. We modify the way G_8 processes the Search procedure as follows: Once G_9 identifies w_1 given bstag and the (I_1, ρ_1) in QList(env) as described above, it computes $\mathsf{strap} \leftarrow F_G(K_s, w_1)$ and $(K_z, K_e) \leftarrow (F_\tau(\mathsf{strap}, 1), F_\tau(\mathsf{strap}, 2))$. Then, for each c, G_9 computes $z_c \leftarrow F_p(K_z, c)$, and given $\mathsf{xtoken}[c] = (\mathsf{xtoken}[c, 2], ..., \mathsf{xtoken}[c, n])$, for each $i = 2, ..., n$, G_9 searches for w_i in W s.t. $\mathsf{xtoken}[c, i] = (F_G(K_X, w_i))^{z_c \cdot \rho_i}$ and $\mathsf{ind}_c \in \mathsf{DB}(w_i)$, where ind_c corresponds to the c-th tuple (e, y) in $\mathbf{t} = \mathbf{T}[w_1]$. If G_9 finds such w_i for all i then it sends e to A.

Let us use the notation $W(\mathsf{ind})$ for the set of keywords in record ind, i.e. the set of w's in W s.t. $\mathsf{ind} \in \mathsf{DB}(w)$. Note that if the above check succeeds for any c, i in game G_9 then it must also succeed in game G_8, because if $\mathsf{xtoken}[c, i] = (F_G(K_X, w_i))^{z_c \cdot \rho_i}$ then

$(\mathsf{xtoken}[c,i])^{y_c/\rho_i}$ is equal to $(F_G(K_X, w_i))^{\mathsf{xind}_c}$, and if $\mathsf{ind}_c \in \mathsf{DB}(w_i)$ then this xtrap value is included in set XSet. Hence the only difference between G_9 and G_8 can occur if for some c, i we have that $(\mathsf{xtoken}[c,i])^{y_c/\rho_i} \in \mathsf{XSet}$ but $\mathsf{xtoken}[c,i] \neq (F_G(K_X, w))^{z_c \cdot \rho_i}$ for all $w \in W(\mathsf{ind}_c)$.

Let us denote exponent $K_X[I_i] \cdot \rho_i$ used in exponentiating a_i in the j-th $\mathsf{GenToken}$ instance as $t_{i,j}$. Clearly, G_8 can pick $t_{i,j}$'s at random in Z_p^* and define the corresponding ρ_i value as $t_{i,j}/K_X[I_i]$. Using this notation, the event that $(\mathsf{xtoken}[c,i])^{y_c/\rho_i}$ matches some value in XSet is equivalent to existence of some ind^* and $w^* \in W(\mathsf{ind}^*)$ s.t.

$$(\mathsf{xtoken}[c,i])^{1/z_c} = (H(w^*))^{t_{i,j} \cdot \frac{K_x[I(w^*)]}{K_x[I_i]} \cdot \frac{F_I(\mathsf{ind}^*)}{F_I(\mathsf{ind}_c)}} \tag{1}$$

At the same time, the constraint that $\mathsf{xtoken}[c,i] \neq (F_G(K_x, w))^{z_c \cdot \rho_i}$ for all $w \in W(\mathsf{ind}_c)$ implies that for all $w \in W(\mathsf{ind}_c)$ we have

$$(\mathsf{xtoken}[c,i])^{1/z_c} \neq (H(w))^{t_{i,j} \cdot \frac{K_x[I(w)]}{K_x[I_i]}} \tag{2}$$

We will argue that there is at most negligible probability that equation (1) holds while equation (2) also holds for every $w \in W(\mathsf{ind}_c)$. It will follow that $p_9 \approx p_8$.

Observe that if equation (1) holds for $\mathsf{ind}^* = \mathsf{ind}_c$ and $I(w^*) = I_i$, then $(\mathsf{xtoken}[c,i])^{1/z_c} = (H(w^*))^{t_{i,j}}$ for $w^* \in W(\mathsf{ind}_c)$, but this contradicts equation (2), which in the case $I(w^*) = I_i$ implies that $(\mathsf{xtoken}[c,i])^{1/z_c} \neq (H(w^*))^{t_{i,j}}$. Below we argue that equation (1) can hold only with negligible probability for either if $\mathsf{ind}^* \neq \mathsf{ind}_c$ or if $I(w^*) \neq I_i$, which together implies that the two equations can hold together only with negligible probability.

(Case a:) We argue that equation (1) can hold for $\mathsf{ind}^* \neq \mathsf{ind}_c$ with at most negligible probability. Let us modify game G_8 into G' s.t. G' does not append pair (e, y) into the $\mathbf{T}[w]$ tuple list, but instead appends triples $(\mathsf{e}, \mathsf{ind}, z)$, i.e. in particular G' does not query F_I in creating the $\mathbf{T}$ lists. Secondly, game G' also does not create the XSet data structure during $\mathsf{EDBSetup}$. Instead, it modifies the test procedure performed by G_8 for each c and i in the Search protocol: Instead of checking whether $(\mathsf{xtoken}[c,i])^{(y_c/\rho_i)}$ is in the XSet, G' searches for $w^* \in \mathsf{W}$ and $\mathsf{ind}^* \in \mathsf{DB}(w)$ s.t. equation (1) holds, using the ind_c, z_c values kept in the c-th tuple in $\mathbf{T}[w_1]$. As we argued above, these are equivalent conditions, and therefore G' provides an identical view as G_8. Note furthermore that the test in equation (1) can be implemented by testing whether $a = b^{F_I(\mathsf{ind}^*)/F_I(\mathsf{ind}_c)}$ where $a = (\mathsf{xtoken}[c,i])^{1/z_c}$ and $b = (H(w^*))^{t_{i,j} \cdot (K_X[I(w^*)]/K_X[I_i])}$. In other words, game G' can operate with an access to F_I restricted as follows: G' specifies a tuple (a, b, x, x^*) and the oracle returns 1 if $a^{F_I(x)} = b^{F_I(x^*)}$, and 0 otherwise. Since H is a random function with range G, we have that except for negligible probability all (a, b, x, x^*) queries of G' involve $b \neq 1$. Since F_I is a random function onto Z_p^* and game G' makes polynomially-many such queries, there is only a negligible probability that the oracle returns 1 for any query s.t. $x \neq x^*$, which implies that there is at most negligible probability that equation (1) holds for $\mathsf{ind}_c \neq \mathsf{ind}$.

(Case b:) Using a similar reasoning, we argue that equation (1) can hold for $I(w^*) \neq I_i$ also with at most negligible probability. Note that the only time game G' accesses key array

K_X is also in testing equation (1), which is equivalent to specifying a query (a, b, I, I^*) for $a = (\mathsf{xtoken}[c, i])^{(1/z_c)\cdot F_I(\mathsf{ind}_c)}$, $b = (H(w^*))^{t_{i,j}\cdot F_I(\mathsf{ind}^*)}$, $I = I_i$, and $I^* = I(w^*)$, and receiving back 1 if $a^{K_X[I]} = b^{K_X[I^*]}$. Note that probability that $b = 1$ is negligible because H is a random function onto G. Since the individual keys in array K_X are independently chosen in Z_p^* and bG' makes polynomially many such queries, it follows that there is a negligible probability that any such query succeeds for $I \neq I^*$, i.e. for $I_i \neq I(w^*)$.

Game G_{10}. We modify the test game G_9 uses to identify w_i given $\mathsf{xtoken}[c, i]$ and z_c, ρ_i by amending verification that $\mathsf{xtoken}[c, i] = (F_G(K_X, w_i))^{z_c\cdot\rho_i}$ and $\mathsf{ind}_c \in \mathsf{DB}(w_i)$ with the additional constraint that $I(w_i) = I_i$ for I_i is retrieved from $\mathsf{QList}(\mathsf{env})$. The event that differentiates the two games is that $\mathsf{xtoken}[c, i] = (H(w_i))^{K_X[I(w_i)]\cdot z_c\rho_i}$ for $\mathsf{ind}_C \in \mathsf{DB}(w_i)$ but $I(w_i) \neq I_i$. We argue that this event occurs with negligible probability, and hence $p_{10} \approx p_9$, even against an all-powerful adversary. Using the notation $t_{i,j}$ for $K_X[I_i] \cdot \rho_i$ used on the j-th $\mathsf{GenToken}$ session, the above equation can be rewritten as $\mathsf{xtoken}[c, i] = (H(w_i))^{z_c\cdot t_{i,j}\cdot\frac{K_X[I(w_i)]}{K_X[I_i]}}$. In particular, the discrete logarithm between $\mathsf{xtoken}[c, i]$ and $(H(w_i))^{z_c\cdot t_{i,i}}$ must be equal to $K_X[I_i]/K_X[I(w_i)]$. Note the G_{10} does not use the key array K_X in any other way except in this test. Therefore, since K_X component keys are random in Z_p^*, and the game makes polynomially-many such tests, the probability that such test ever succeeds for $I(w_i) \neq I_i$ is negligible, and hence $p_{10} \approx p_9$.

Game G_{11}. We add an abort if the game ever encounters a ciphertext env, index i, a pair of counters c, c' and two keywords $w_{/=} w_i'$ s.t. $\mathsf{xtoken}[c, i] = (H(w_i))^{K_X[I_i]\cdot z_c\cdot\rho_i}$ and $\mathsf{xtoken}[c', i] = (H(w_i'))^{K_X[I_i]\cdot z_{c'}\cdot\rho_i}$. We intend this abort to include any Search session that some ciphertext env is used, including the case that this "collision" occurs on two different Search sessions using the same env for the same counter $c' = c$. Using the $t_{i,j}$ notation for $K_X[I_1] \cdot \rho_i$ used in the j-th session of $\mathsf{GenToken}$, we rewrite this as $\mathsf{xtoken}[c, i]^{1/z_c} = (H(w_i))^{t_{i,j}}$ and $\mathsf{xtoken}[c', i]^{1/z_{c'}} = (H(w_i'))^{t_{i,j}}$. It is easy to see that the probability of encountering such two tuples $(w_i, \mathsf{xtoken}[c, i], z_c)$ and $(w_i', \mathsf{xtoken}[c', i], z_{c'})$ for $w_i \neq w_i'$ must be negligible under the OM-GDH assumption in the ROM model for H, and hence $p_{11} \approx p_{10}$.

The reduction is essentially the same as the one given in the argument for game G_7. Let ϵ be the probability of the above event. The reduction on the input a OM-GDH challenge h_1, h_2, emulates game G_{11} except that on each A's query w to H, it picks random r_1, r_2 in Z_p^* and sets $H(w) \leftarrow (h_1)^{r_1}(h_2)^{r_2}$. The reduction also picks a random index j between 1 and the maximum number of $\mathsf{GenToken}$ instances A can invoke, and when servicing the j-th instance of $\mathsf{GenToken}$, the reduction picks i at random between 2 and n, sends the a_i value in this $\mathsf{GenToken}$ instance to the OM-GDH challenger, who replies with $b_i \leftarrow (a_i)^t$ for the exponent t chosen in the OM-GDH challenge game. The reduction passes this b_i in its response to A. Finally, for each Search protocol instance on which A sends ciphertext env, for each c, the reduction takes $\mathsf{xtoken}[c, i]$ sent by A along with env, and for each query w made by A to H it consults the DDH oracle if $(a_1, b_1, H(w), (\mathsf{xtoken}[c, i])^{1/z_c})$ is a DDH tuple. If the reduction finds two such instances on which the above check verifies, one for $(w, \mathsf{xtoken}[c, i], z_c)$ and the other for $(w', \mathsf{xtoken}[c', i], z_{c'})$ s.t. $w \neq w'$, it

computes h_1^t and h_2^t as, respectively, $((\mathsf{xtoken}[c,i])^{r_2/z_c}(\mathsf{xtoken}[c',i])^{-r_2/z_{c'}})^{1/(r_1r_2'-r_1'r_2)}$ and $((\mathsf{xtoken}[c,i])^{r_1/z_c}(\mathsf{xtoken}[c',i])^{-r_1/z_{c'}})^{1/(r_1'r_2-r_1r_2')}$, where $H(w)=(h_1)^{r_1}(h_2)^{r_2}$ and $H(w')=(h_1)^{r_1'}(h_2)^{r_2'}$. The probability of reduction's success is $(1/(mn))\cdot\epsilon$ where m is the upper-bound on the number of $\mathsf{GenToken}$ instances A invokes and n an upper-bound on the length of a conjunctive query, minus the negligible probability that $r_1r_2'=r_1'r_2$.

Game G_{12}. Note that in game G_{11} there is at most one vector $\bar{w}=(w_1,...,w_n)$ of keywords the game finds in all Search protocol instances for a given ciphertext env which the game produced in some $\mathsf{GenToken}$ protocol instance. This is because from game G_7 on, the game finds at most one w_1 corresponding to any env, and from game G_{11} on, the game also finds at most one w_i corresponding to any env and i between 2 and the index n used in the $\mathsf{GenToken}$ session which produced env. This unique query $\bar{w}$ corresponding to env should be thought of as the effective query A made in the $\mathsf{GenToken}$ instance which produced the env. Note, moreover, that $\mathsf{av}(\bar{w})$ is equal to the vector of attributes $(I_1,...,I_n)$ which A specified in that $\mathsf{GenToken}$ instance, and therefore the only way this instance produced a ciphertext env is if $(I_1,...,I_n)\in\mathsf{P}$. Note finally, that the only ciphertext e the game G_{11} supplies to A correspond to indexes ind_c in $\mathsf{DB}(w_1)$ s.t. $\mathsf{ind}_c\in\mathsf{DB}(w_i)$ for $i=2,...,n$, i.e. for ind_c's in $\mathsf{DB}(\bar{w})$.

These observations lead us to the new game G_{12}, a modification of G_{11}, which operates as follows. Game G_{12} does not create either TSet or XSet data structure in $\mathsf{EDBSetup}$, only picks keys K_S,F_X,K_T,K_P,K_M (it skips choosing key K_I, which it will not need). Game G_{12} services the $\mathsf{GenToken}$ instances as G_{11}, but on calls to Search, when it finds counter c s.t. $\mathsf{xtoken}[c,i]=F_G(K_X,w_i))^{z_c\cdot\rho_i}$, $\mathsf{ind}_c\in\mathsf{DB}(w_i)$, and $I(w_i)=I_i$, for all $i=2,...,n$, it assembles query $\bar{w}$ from the identified $w_1,...,w_n$ terms, and sends it to $\mathsf{I\text{-}SSE}_{\mathcal{L}}(\mathsf{DB},\mathsf{RDK},\mathsf{P})$. Since $\mathsf{av}(\bar{w})=(I_1,...,I_n)\in\mathsf{P}$, it receives back $(\mathsf{DB}(\bar{w}),\mathsf{RDK}(\mathsf{DB}(\bar{w})),\mathsf{Mask}(|\mathsf{DB}(w_1)|))$. Game G_{12} then assigns the indexes in $\mathsf{DB}(\bar{w})$ to random indexes between 1 and $\mathsf{Mask}(|\mathsf{DB}(w_1)|)$, unless some of these were previously assigned. To keep track of previously seen indexes, G_{12} keeps a list TList, indexed by w_1's, s.t. $\mathsf{TList}(w_1)$ stores tuples $(c,\mathsf{ind},\mathsf{e})$ s.t. counter c in $\mathbf{T}[w_1]$ was assigned during some Search instance to index ind, and the corresponding ciphertext was created as e. Initially $\mathsf{TList}(w_1)$ is empty for all $w_1\in\mathsf{W}$, but when G_{12} identifies the query $\bar{w}=(w_1,...,w_n)$ in the Search protocol, sends it to $\mathsf{I\text{-}SSE}_{\mathcal{L}}$, and receives the tuple $(\mathsf{DB}(\bar{w}),\mathsf{RDK}(\mathsf{DB}(\bar{w})),\mathsf{Mask}(|\mathsf{DB}(w_1)|))$, G_8 checks if for any $\mathsf{ind}\in\mathsf{DB}(\bar{w})$ the ind value was already in the $\mathsf{TList}(w_1)$. For all ind's in $\mathsf{DB}(\bar{w})$ which are *not* in $\mathsf{TList}(w_1)$, G_8 assigns them to a random subset of remaining counters c between 1 and TSetL, and for each (c,ind) pair it computes a corresponding ciphertext e using the K_e key computed using K_S from w_1 as in game G_9 above, setting $\mathsf{rind}\leftarrow P_\tau(K_P,\mathsf{ind}_c)$, $\mathsf{rdk}\leftarrow\mathsf{RDK}(\mathsf{ind})$, and $\mathsf{e}\leftarrow\mathsf{Enc}(K_e,(\mathsf{rind}|\mathsf{rdk}))$.

In this way G_{12} creates a view which the adversary gets of some T-set $\mathsf{TSet}(w_1)$ throughout its interaction with the $\mathsf{MC\text{-}OXT}$ scheme, and this view matches that created by G_{11}, where the assignment between indexes $\mathsf{ind}\in\mathsf{DB}(w_1)$ and the counters in $\{1,...,\mathsf{Mask}(|\mathsf{DB}(w_1)|)\}$ is done during the $\mathsf{EDBSetup}$ procedure, because in both cases this assignment is random. The full description of game G_{12} is shown in Figure 3 as a code of a simulator algorithm S, broken down into three subprocesses S_0,S_1,S_2, servicing respectively $\mathsf{EDBSetup}$, $\mathsf{GenToken}$,

and Search protocols. It follows that $p_{12} = p_{11}$.

This completes the proof of security because it concludes the argument that there is at most negligible difference between p_0, which is the probability that $\mathbf{Real}^{\Pi}_{A}(\lambda)$ outputs 1, and p_{12}, which is the probability that $\mathbf{Ideal}^{\Pi}_{A,S}(\lambda)$ outputs 1. Moreover, the simulator S maintains the constraint demanded by the security definition that the number of unique queries $\bar{w}$ it makes to $\mathsf{I\text{-}SSE}_{\mathcal{L}}$ does not exceed the number of GenToken instances which adversary A makes.

B OSPIR-OXT Instantiated with Hashed-DH OPRF

In Figure 4 we present the instantiation of protocol OSPIR-OXT with the Hashed-DH OPRF described at the end of Section 4.1. It serves as an illustration of a concrete realization of the protocol and it is the basis for our analysis and implementation. We note some notational differences relative to the abstract OSPIR-OXT protocol that arise from the specific Hashed-DH instantiation: We denote keys K_T and K_X of PRF F^m_G by vectors of exponents in Z^*_p, respectively $(k_1, \ldots, k_m)$ and $(e_1, \ldots, e_m)$, where m is the number of attributes. Also, because of the specific OPRF instantiation we equate a_s to a_1 in $\mathcal{C}$'s message of the GenToken protocol, instead of computing these two blinded versions of keyword w_1 separately, as in Figure 2.

Group operations. G is a cyclic group of prime order p generated by an element g. H is a hash function with range in $G \setminus \{1\}$.

EDBSetup(DB, RDK)

Key Generation. $\mathcal{D}$ picks key K_S and two vectors of elements $K_T = (k_1, \ldots, k_m)$ and $K_X = (e_1, \ldots, e_m)$ at random in Z_p^* (m = number of attributes); key K_I for PRF F_p; and key K_M for a symmetric authenticated encryption $\mathsf{AuthEnc}$. F_p and F_τ are PRF's which outputs strings in respectively Z_p^* and $\{0,1\}^\tau$, and τ is a security parameter.

- Initialize XSet to an empty set, and initialize $\mathbf{T}$ to an empty array indexed by group elements in G.
- For each $w = (i, val) \in \mathsf{W}$, build the tuple list $\mathbf{t}$ and add elements to set XSet as follows:
  - Initialize $\mathbf{t}$ to an empty list.
  - Set $\mathsf{strap} \leftarrow (H(w))^{K_S}$, $\mathsf{stag} \leftarrow (H(w))^{k_i}$ $[= F_G^m(K_T, w)]$, $(K_z, K_e) \leftarrow (F_\tau(\mathsf{strap}, 1), F_\tau(\mathsf{strap}, 2))$.
  - Initialize $c \leftarrow 0$; then for all ind in $\mathsf{DB}(w)$ in random order:
    - 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 $c \leftarrow c + 1$, $z_c \leftarrow F_p(K_z, c)$, $y \leftarrow \mathsf{xind} \cdot z_c^{-1}$. Append (e, y) to $\mathbf{t}$.
    - Set $\mathsf{xtag} \leftarrow H(w)^{e_i \cdot \mathsf{xind}}$ $[= (F_G^m(K_X, w))^{\mathsf{xind}}]$ and add xtag to XSet.
  - $\mathbf{T}[\mathsf{stag}] \leftarrow \mathbf{t}$.
- Create $\mathsf{TSet} \leftarrow \mathsf{TSetSetup}'(\mathbf{T})$, and output key $K = (K_S, K_X, K_T, K_I, K_M)$ and $\mathsf{EDB} = (\mathsf{TSet}, \mathsf{XSet}, K_M)$.

GenToken protocol

Client $\mathcal{C}$ on input a conjunctive query $\bar{w} = (w_1, ..., w_n)$, where w_1 is chosen as s-term, proceeds as follows:

- Pick $r_1, ..., r_n \xleftarrow{\$} Z_p^*$ and set $a_j \leftarrow (H(w_j))^{r_j}$ for $j = 1, \ldots, n$.
- Send to $\mathcal{D}$ the blinded queries $a_1, \ldots, a_n$ and the attribute sequence $\mathsf{av} = (I(w_1), \ldots, I(w_n))$.

Data owner $\mathcal{D}$ on input policy P and key K proceeds as follows:

- Abort if $\mathsf{av} \notin P$. Otherwise set av as a local output. Pick $\rho_1, \ldots, \rho_n \xleftarrow{\$} Z_p^*$.
- Set $\mathsf{strap}' \leftarrow (a_1)^s$, $\mathsf{bstag}' \leftarrow (a_1)^{k_{i_1} \cdot \rho_1}$ $[= (a_1)^{K_T[i_1] \cdot \rho_1}]$, and $\mathsf{bxtrap}'_j \leftarrow (a_j)^{e_{i_j} \cdot \rho_j}$ $[= (a_j)^{K_X[i_j] \cdot \rho_j}]$ for $j = 2, \ldots, n$.
- Reply to $\mathcal{C}$ with $(\mathsf{strap}', \mathsf{bstag}', \mathsf{bxtrap}'_2, \ldots, \mathsf{bxtrap}'_n)$ and $\mathsf{env} = \mathsf{AuthEnc}_{K_M}(\rho_1, \ldots, \rho_n)$.

$\mathcal{C}$ sets:

- $\mathsf{strap} \leftarrow (\mathsf{strap}')^{r_1^{-1}}$; $\mathsf{bstag} \leftarrow (\mathsf{bstag}')^{r_1^{-1}}$; $\mathsf{bxtrap}_j \leftarrow (\mathsf{bxtrap}'_j)^{r_j^{-1}} (\forall_{j=2,\ldots,n})$;
- $\mathsf{token} \leftarrow (\mathsf{env}, \mathsf{strap}, \mathsf{bstag}, \mathsf{bxtrap}_2, \ldots, \mathsf{bxtrap}_n)$.

Search protocol

Client $\mathcal{C}$ on input token proceeds as follows:

- Set $(K_z, K_e) \leftarrow F(_\tau(\mathsf{strap}, 1), F(\mathsf{strap}, 2))$; send to $\mathcal{E}$ the message $(\mathsf{env}, \mathsf{bstag}, \mathsf{bxtoken}[1], \mathsf{bxtoken}[2], \ldots)$ defined as:
  - For $c = 1, 2, \ldots$, until $\mathcal{E}$ sends stop:
    - Set $z_c \leftarrow F_p(K_z, c)$ and set $\mathsf{bxtoken}[c, i] \leftarrow (\mathsf{bxtrap}_i)^{z_c}$ for $i = 2, \ldots, n$.
    - Set $\mathsf{bxtoken}[c] \leftarrow (\mathsf{bxtoken}[c, 2], \ldots, \mathsf{bxtoken}[c, n])$.

Server $\mathcal{E}$ on input $\mathsf{EDB} = (\mathsf{TSet}, \mathsf{XSet}, K_M)$ responds as follows:

- Upon receiving $\mathsf{env}, \mathsf{bstag}$ from $\mathcal{C}$, decrypt/verify env; if verification fails return "no results" and stop.
- Set $\mathsf{stag} \leftarrow (\mathsf{bstag})^{1/\rho_1}$ and retrieve $\mathbf{t} \leftarrow \mathsf{TSetRetrieve}(\mathsf{TSet}, \mathsf{stag})$ from TSet.
- For $c = 1, \ldots, |\mathbf{t}|$ do:
  - Receive $\mathsf{bxtoken}[c]$ from $\mathcal{C}$ and retrieve value (e, y) from the c-th tuple in $\mathbf{t}$.
  - Check if $\mathsf{bxtoken}[c, i]^{y/\rho_i} \in \mathsf{XSet}$ for all $i = 2, \ldots, n$. If so, send e to $\mathcal{C}$ (else nothing is returned for this tuple).
  - When last tuple in $\mathbf{t}$ is reached, send stop to $\mathcal{C}$ and halt.

For each received e client $\mathcal{C}$ computes $(\mathsf{ind}|\mathsf{rdk}) \leftarrow \mathsf{Dec}(K_e, \mathsf{e})$ and outputs $(\mathsf{ind}, \mathsf{rdk})$.

Figure 4: OSPIR-OXT Instantiated for Conjuntions with the Hashed Diffie-Hellman OPRF