Conjunctive Searchable Symmetric Encryption from Hard Lattices

Debadrita Talapatra
IIT Kharagpur, India

Sikhar Patranabis
IBM Research, India

Debdeep Mukhopadhyay
IIT Kharagpur, India

June 14, 2024

Abstract

Searchable Symmetric Encryption (SSE) supports efficient keyword searches over encrypted outsourced document collections while minimizing information leakage. All practically efficient SSE schemes supporting conjunctive queries rely crucially on *quantum-broken* cryptographic assumptions (such as discrete-log hard groups) to achieve compact storage and fast query processing. On the other hand, quantum-safe SSE schemes based on purely symmetric-key cryptoprimitives either do not support conjunctive searches, or are practically inefficient. In particular, there exists no quantum-safe yet practically efficient conjunctive SSE scheme from *lattice-based hardness assumptions*.

We solve this open question by proposing Oblivious Post-Quantum Secure Cross Tags (OQXT) – the first lattice-based practically efficient and highly scalable conjunctive SSE scheme. The technical centerpiece of OQXT is a novel oblivious cross-tag generation protocol with provable security guarantees derived from lattice-based hardness assumptions. We prove the post-quantum simulation security of OQXT with respect to a rigorously defined and thoroughly analyzed leakage profile. We then present a prototype implementation of OQXT and experimentally validate its practical efficiency and scalability over extremely large real-world databases. Our experiments show that OQXT has competitive end-to-end search latency when compared with the best (quantum-broken) conjunctive SSE schemes.

Contents

1 Introduction

Searchable Symmetric Encryption (SSE). Searchable Symmetric Encryption (SSE) [SWP00, CGKO06, CPPJ18, CK10, CJJ+13, CJJ+14] allows a (potentially untrusted) server to execute keyword search queries directly on a collection of a client's encrypted documents in an efficient manner, while ensuring client privacy by minimizing the amount of information "leakage" to the server in the process. The most general notion of SSE with optimal security guarantees (wherein little or no information is leaked to the server) can be achieved using techniques like fully homomorphic encryption (FHE) [Gen09] and Oblivious RAM (ORAM) [GO96]. However, such an ideal notion of privacy comes at the cost of significant computational and/or communication overhead today. Hence, existing SSE schemes [CK10, CJJ+13, CJJ+14, Bos16, BMO17, CPPJ18] prefer to trade-off security for practical efficiency by leaking "some" information during query execution.

Conjunctive SSE. Consider a client that offloads an encrypted database of (potentially sensitive) emails to an untrusted server and later issues a *conjunctive* query of the form "*retrieve all emails received from xyz@foobar.org with the keyword "research" in the subject field*" (which is a conjunction of the queries "*retrieve all emails received from xyz@foobar.org* or "*retrieve all emails with the keyword "research" in the subject field*"). For any SSE scheme to be truly practical, it should at least support such conjunctive keyword queries, i.e., given a set of keywords $(\mathsf{w}_1, \ldots, \mathsf{w}_n)$, it should be able to find and return the set of documents that contain *all* of these keywords. There exist today efficient SSE schemes that support conjunctive (and more general Boolean) queries [CJJ+13, CJJ+14, LPS+18, PM21].

Public-Key Techniques in SSE. An often overlooked aspect of the above mentioned SSE constructions is the following: while SSE is an ostensibly symmetric-key primitive, almost all of these SSE constructions supporting conjunctive (and general Boolean) queries resort to public-key cryptographic techniques to achieve compact storage of the encrypted database and/or fast encrypted query processing. For example, the Oblivious Cross-Tags (OXT) protocol by Cash et al. [CJJ+13] achieves two highly desirable features from the point of view of practical efficiency: (i) a storage overhead for the encrypted database that grows linearly with the size of the plaintext database, and (ii) a conjunctive query complexity that grows with the frequency of the *least frequent keyword*.[1] However, OXT *crucially* relies on discrete-log hard groups (typically implemented in practice using elliptic-curve based cryptography). Similar dependencies on discrete-log hard groups are inherent to essentially all advanced SSE schemes supporting rich Boolean queries [CJJ+14, LPS+18, PM21], many of which build upon OXT. Consequently, all of these constructions are *quantum-broken*.

Quantum-Safety vs Practical Efficiency. Unfortunately, all quantum-safe SSE schemes today rely on purely symmetric-key primitives, and are impractical for real-world databases because they all suffer from one of the following drawbacks: (i) either they are severely limited in terms of the expressiveness of queries they support, or (ii) they are highly inefficient. The first category includes SSE schemes supporting only single-keyword search [CGKO06, CK10, Bos16, BMO17, CPPJ18].[2] The second category includes SSE schemes that sup-

[1] Concretely, given a conjunctive query of the form $(\mathsf{w}_1 \wedge \ldots \wedge \mathsf{w}_n)$ where w_1 is the least frequent keyword, OXT incurs a computational and communication complexity of $O(\mathsf{n} \cdot f(\mathsf{w}_1))$, where $f(\mathsf{w}_1)$ denotes the frequency of w_1.

[2] In real-life applications, such as querying large remotely stored email databases, a single keyword query

port highly expressive Boolean queries, but are practically inefficient [KM17, JP22, PPSY21] since they incur a storage complexity that grows *quadratically* with the size of the plaintext database in the worst case. Ideally, we want practically efficient conjunctive SSE schemes with *linear storage overhead*.

Our Motivation. In this paper, we investigate the possibility of constructing practically efficient conjunctive SSE scheme based on plausibly quantum-safe assumptions. To the best of our knowledge, prior to our work, there existed no SSE scheme from quantum-safe assumptions, such as lattice-based assumptions. Motivated by this, we ask the following question:

Can we design a quantum-safe yet efficient conjunctive SSE scheme from lattice-based hardness assumptions?

1.1 Our Contributions

We design, analyze and prototype implement the *first lattice-based SSE scheme* that efficiently supports conjunctive queries over large encrypted databases. Our proposed scheme is called *Oblivious Post-quantum Secure Cross-tags Protocol* (OQXT). OQXT achieves the similar storage and conjunctive query complexities as OXT [CJJ$^+$13], i.e., it achieves: (i) a storage overhead for the encrypted database that grows linearly with the size of the plaintext database (unlike IEX [KM17] and ConjFilter [PPSY21]), and (ii) a conjunctive query complexity that grows with the frequency of the *least frequent keyword*.

The main difference between OXT and our proposed OQXT scheme is that, while OXT derives its efficiency and (classical) security guarantees from computational assumptions over discrete-log hard groups, our proposed OQXT scheme derives its *post-quantum security* guarantees from computational assumptions over hard lattices, such as the *Learning With Rounding* (LWR) assumption [BPR12]. We rigorously prove the post-quantum security of OQXT with respect to a well defined leakage profile in the simulation-based real world/ideal world paradigm against a semi-honest server. Finally, we describe a prototype implementation of OQXT and present experimental validation of its storage and conjunctive query complexities over large real-world databases.

1.2 Lattice-based Oblivious Cross Tags

The technical centerpiece of OQXT is a *lattice-based* non-interactive search protocol executed between the client and the server, where server takes as input the encrypted database, while the client takes as input a conjunction of keywords and some secret state information. The outcome of this protocol is a filtered, significantly smaller set of encrypted records, which the client can then locally decrypt to compute the identifiers for documents containing all of the queried keywords. We provide an informal overview of our core techniques below.

would potentially return a large number of matching records/documents that the client would need to download and filter locally, which is clearly impractical.

Classical Oblivious Cross Tags in OXT**.** We begin by noting that OXT achieved a classically secure realization of the above protocol using discrete-log hard groups. In its simplest embodiment, OXT pre-computes an encrypted version of the plaintext database (using a secret symmetric key) and stores it at a server that is presumed to be *honest-but-curious*. A client with access to this symmetric key, breaks a (two-) conjunctive query $q = \mathsf{w}_1 \wedge \mathsf{w}_2$ into two search tokens for the server. The first search token (stag) yields all entries for the first conjunct w_1 and the second search token (**xtoken**) is used to search for exactly the conjunct w_2 using a "*cross-tag* (**xtag**) *helper token*" stored as part of the entries for w_1. The **xtag** helper token is independent of the second attribute and hence only one **xtag** helper token per (keyword-document) pair is stored (this at most doubles the total space requirement).

We explain the technique at a high level using the example of the two-conjunctive query from the discussion above: "*retrieve all emails received from xyz@foobar.org with the keyword "research" in the subject field*". In the OXT protocol, the client computes two search tokens (using its secret key): one for (sender; XYZ@FOOBAR.ORG), say $\mathsf{p1}$, and another for (keyword; RESEARCH), say $\mathsf{p2}$. It sends to the server a *symmetric-encryption* key $\mathsf{k1}$ derived from $\mathsf{p1}$, and an **xtoken** of the form $\mathsf{h}^{\mathsf{p2}/\mathsf{p1}}$ (in a discrete-log hard group $\mathbb{G}$ with generator h). The server uses $\mathsf{k1}$ to look-up and retrieve an encrypted set (stored in a lookup-table called the XSet that is pre-computed and stored with the server as part of the encrypted database) corresponding to (sender; XYZ@FOOBAR.ORG), and uses $\mathsf{k1}$ to decrypt it. Next, for each record, in this decrypted set D, the server is allowed to look-up an "**xtag** *helper token*" $\mathsf{z} = \mathsf{p1} * \mathsf{rind}$ (where, rind stands for randomized-document-identifier). Observe that the **xtoken** $\mathsf{h}^{\mathsf{p2}/\mathsf{p1}}$ raised to the power z yields an **xtag**, $\mathsf{h}^{\mathsf{p2}*\mathsf{rind}}$, which is then checked in a lookup-table called XSet (also pre-computed and stored with the server as part of the encrypted database). This lookup-table XSet stored with the server has every valid member of the form $\mathsf{h}^{\mathsf{p2}*\mathsf{rind}}$, and hence this check allows the server to confirm whether or not a record in the initial decrypted set D satisfies the second conjunct. Note that the size of this set XSet is exactly the number of keyword-document pairs in the database, which is nothing but the size of the database (hence the linear storage overhead of OXT). That this lookup-table reveals no information, a priori, is proved in [CJJ+13] under the classically secure decisional Diffie-Hellman (DDH) assumption over the group $\mathbb{G}$.

OXT **is Quantum-Broken.** We note here that OXT is devastatingly broken in the presence of a quantum (honest-but-curious) adversary that is capable of solving the discrete log problem (and hence break the DDH assumption) in the group $\mathbb{G}$. As mentioned earlier, the OXT protocol relies crucially on the DDH assumption in the group $\mathbb{G}$ to argue that the XSet data structure reveals no information, a priori, about the plaintext database and the plaintext inverted index. A quantum adversary can, however, compute the discrete-logs corresponding to all **xtag** elements in the XSet data-structure, thereby reconstructing completely the frequency-distribution of keywords across documents in the plaintext inverted index. This constitutes a serious loss of data privacy for the client. Additionally, equipped with this leakage about the database, the adversary can launch leakage-abuse attacks to recover all of the client's queries with high accuracy, as discussed in many prior works (notably in [IKK12, CGPR15, BKM20, GPP23]), thus also violating the query privacy guarantees of OXT. This motivates designing a post-quantum secure version of OXT, which was an open question prior to our work.

Switching to Lattice-based Cross Tags. The main technical novelty of OQXT is that, when generating the **xtag** entries in the XSet as above, we switch from the usage of a DDH-hard group (which is only classically secure) to a hard lattice over which the Learning with Rounding (LWR) assumption holds. Along the way, we encounter several technical challenges, which we solve by introducing novel techniques, the most notable being: (a) a new construction of **xtag**s in the XSet look-up table with security guarantees derived from the LWR assumption, and (b) a novel usage of *trapdoors for hard lattices* to design the **xtag** helper tokens and **xtoken**s such that all computationally intensive trapdoor-based operations are performed at setup, thus allowing for fast online search performance. We expand more on these challenges and solution techniques in the next subsection.

1.3 Technical Challenges and Solution Overview

For simplicity of exposition, we first explain the technical challenges and our core solution ideas using the more popularly studied Learning with Errors (LWE) assumption (the LWR assumption can be viewed as a *deterministic* variant of the LWE assumption, and is, in fact, implied by the LWE assumption [Reg09]). We subsequently elucidate the need to switch from using the (randomized) LWE assumption to the (deterministic) LWR assumption. We begin by (informally) recalling the LWE assumption. Let $n, m, q \in \mathbb{N}$ be positive integers and let χ be a noise distribution over $\mathbb{Z}_q$. In the $\mathsf{LWE}(n, m, q, \chi)$ problem, the adversary's goal is to distinguish between the two distributions:

$$(\mathbf{A}, \mathbf{As} + \mathbf{e}) \quad \text{and} \quad (\mathbf{A}, \mathbf{u})$$

where $\mathbf{A} \leftarrow \mathbb{Z}_q^{m \times n}$, $\mathbf{s} \leftarrow \mathbb{Z}_q^n$, $\mathbf{e} \leftarrow \chi^m$, and $\mathbf{u} \leftarrow \mathbb{Z}_q^m$. The corresponding $\mathsf{LWE}(n, m, q, \chi)$ assumption (informally) states that the $\mathsf{LWE}(n, m, q, \chi)$ problem is computationally intractable.

Challenge-1: LWE-based Cross-Tags. The first challenge in our realization of LWE-based oblivious **xtag**s is to devise a technique that maps each entry in the look-up table XSet to an LWE sample (looking ahead, such a mapping would allow us to invoke the LWE assumption to argue that the XSet entries reveal no information, a priori, about the underlying plaintext database).

Concretely, as in the example above, let the search token (**xtoken**) corresponding to the second conjunct (keyword; RESEARCH) be $\mathsf{p2}$, and let the randomized-document-identifier be rind. Instead of generating the **xtag** as $\mathsf{h}^{\mathsf{p2}*\mathsf{rind}}$ as in the original OXT protocol, we now generate the **xtag** as

$$\mathbf{u} = \mathbf{A}(\mathsf{p2})\mathbf{s}(\mathsf{rind}) + \mathbf{e}$$

where $\mathbf{A}(\cdot)$ and $\mathbf{s}(\cdot)$ are functions mapping $\mathsf{p2}$ and rind to a matrix and a vector of the appropriate dimension respectively, and $\mathbf{e}$ is an appropriately sampled error vector. Note that each **xtag** is essentially an LWE sample, and we can rely on the LWE assumption to prove that the lookup-table XSet reveals no information, a priori, about the underlying keyword-document identifier pairs in the plaintext database. This is conceptually similar to the usage of DDH in OXT, but presents additional challenges for query processing, as outlined next.

Challenge-2: LWE-based Helper Tokens. The next challenge is to allow the server to obliviously compute the above **xtag**, which in turn requires the client to pre-compute a (quantum-safe) **xtag** helper token, that the server can subsequently look up during the query execution. Concretely, in the example above, let the search token corresponding to the first conjunct (sender; XYZ@FOOBAR.ORG) be $\mathsf{p1}$. A straightforward adaptation of the approach in OXT would be to set the **xtag** helper token (pre-computed and stored as part of TSet look-up table) as another LWE sample of the form

$$\mathbf{z} = \mathbf{A}(\mathsf{p1})\mathbf{s}(\mathsf{rind}) + \mathbf{e}'$$

and the corresponding **xtoken** to be something akin to $\mathbf{W} = \mathbf{A}(\mathsf{p2})\left(\mathbf{A}(\mathsf{p1})\right)^{-1}$, such that the server can compute and search for the **xtag** $\mathbf{u}$ as $\mathbf{Wz} = \mathbf{u}$. Unfortunately, such a straightforward adaptation of the approach in OXT presents several technical challenges:

- $\mathbf{A}(\mathsf{p1})$ is not a square matrix (this follows from the definition of the LWE assumption as used in our construction), and is therefore unlikely to have an efficiently computable "inverse" matrix.
- Crucially, *even* if we designed $\mathbf{A}(\mathsf{p1})$ to be a square matrix with an efficiently computable inverse $\left(\mathbf{A}(\mathsf{p1})\right)^{-1}$, the product matrix $\left(\mathbf{A}(\mathsf{p1})\right)^{-1}\mathbf{A}(\mathsf{p2})$ is not guaranteed to have *short entries*, and hence, with high probability, $\mathbf{Wz} \neq \mathbf{u}$ due to a blow-up in the error of the resulting product in the left-hand side. This is a major technical challenge that we need to overcome in order to ensure correctness of search *without compromising on practical search efficiency.*

Our Solution: Trapdoors for Hard Lattices. We address these technical challenges by a novel usage of *trapdoors for hard lattices* [Ajt96, MP12] to re-design the **xtag** helper token as follows. We pre-compute and store as part of the TSet look-up index a "short" vector (i.e., a vector with short entries) $\mathbf{y}$ such that

$$\left(\mathbf{A}(\mathsf{p1})\right)^{\mathbf{t}}\mathbf{y} = \mathbf{s}(\mathsf{rind})$$

where $\left(\mathbf{A}(\mathsf{p1})\right)^{\mathbf{t}}$ denotes the transpose of the matrix $\mathbf{A}(\mathsf{p1})$. We generate this vector $\mathbf{y}$ using well-known techniques to generate a short basis (equivalently, a trapdoor) for the lattice generated by the matrix $\mathbf{A}(\mathsf{p1})$, and then using this short basis to solve a short integers solution (SIS) instance with respect to $\mathbf{A}(\mathsf{p1})$ and the target vector $\mathbf{s}(\mathsf{rind})$ to generate the corresponding SIS solution vector $\mathbf{y}$. We defer the details of the trapdoor generation procedure to Section 3. We note here that the **xtag** helper token is plausibly quantum-safe assuming that the SIS problem is quantum-hard (which is indeed the case, since SIS is implied by LWE [Reg09]).

Challenge-3: Retaining Search Efficiency. An astute reader might observe that, since the process of generating lattice trapdoors is generally computationally expensive, our approach might hinder practically efficient searches. However, we already account for this concern in our generation and usage of LWE-based **xtag**s and helper tokens using lattice trapdoors. In particular, we *pre-compute and securely store at the server* all trapdoors at setup; this represents a one-time offline computational effort for the client that does

not impact online search latency. In the online search phase, the client-server protocol allows efficiently retrieving these pre-computed trapdoors to efficiently compute and look up the corresponding LWE-based **xtag**s, thus retaining practical search efficiency. This claim is practically validated via experiments where we showcase that our prototype implementation of OQXT achieves fast conjunctive searches over large real-world databases (see Section 5 for details).

Putting Everything Together. We now exemplify the end-to-end query processing steps in OQXT. As in OXT, the client computes two search tokens (using its secret key): one for (sender; XYZ@FOOBAR.ORG), say $\mathsf{p1}$, and another for (keyword; RESEARCH), say $\mathsf{p2}$. It sends to the server a *symmetric-encryption* key $\mathsf{k1}$ derived from $\mathsf{p1}$, and an **xtoken** of the form

$$\mathbf{W} = \mathbf{A}(\mathsf{p2})\left(\mathbf{A}(\mathsf{p1})\right)^{\mathbf{t}} + \mathbf{E}$$

where $\mathbf{E}$ is an appropriately sampled error matrix. The server uses $\mathsf{k1}$ to look-up and retrieve an encrypted set (pre-computed and stored in the look-up table XSet as part of the encrypted database) corresponding to (sender; XYZ@FOOBAR.ORG), and uses $\mathsf{k1}$ to decrypt it. Next, for each record, in this decrypted set D, the server is allowed to look-up the new quantum-safe **xtag** helper token $\mathbf{y}$ as described above. Finally, the server uses the **xtoken** $\mathbf{W}$ and the **xtag** helper token $\mathbf{y}$ to compute the **xtag**

$$\begin{aligned}\mathbf{u}' &= \mathbf{W}\mathbf{y} = \left(\mathbf{A}(\mathsf{p2})\left(\mathbf{A}(\mathsf{p1})\right)^{\mathbf{t}} + \mathbf{E}\right)\mathbf{y} \\ &= \mathbf{A}(\mathsf{p2})\left(\mathbf{A}(\mathsf{p1})\right)^{\mathbf{t}}\mathbf{y} + \mathbf{E}\mathbf{y} \\ &\approx \mathbf{A}(\mathsf{p2})\mathbf{s}(\mathsf{rind}) + \mathbf{e} = \mathbf{u}\end{aligned}$$

where the approximate equality follows from the fact that the vector $\mathbf{y}$ and the error matrix $\mathbf{E}$ consist of short entries. The approximately reconstructed **xtag** is then looked up in the XSet (again, pre-computed and stored with the server as part of the encrypted database).

Challenge-4: Approximate Look-ups and Potential Attacks. While the above solution seems to provide search correctness and quantum-safety from the LWE assumption, there remain two issues. The first (and minor) issue is how to implement the "approximate" **xtag** lookup in the XSet data structure in an efficient way. The original OXT protocol has an exact **xtag** lookup, and hence implements the XSet using some probabilistic membership-check data structure (such as a Bloom filter). This is not immediately achievable in the case of OQXT.

The second (and major) issue is that the computation of an approximate **xtag** implies that, each time a query results in an approximate version of this **xtag**, the server essentially gains access to two LWE samples of the form $\mathbf{u}$ and $\mathbf{u}'$, where

$$\mathbf{u} = \mathbf{A}(\mathsf{p2})\mathbf{s}(\mathsf{rind}) + \mathbf{e}, \quad \mathbf{u}' = \mathbf{A}(\mathsf{p2})\mathbf{s}(\mathsf{rind}) + \mathbf{e}'$$

and, upon receipt of polynomially many such samples, the server could potentially use noise-averaging techniques to gain access to the noise-free sample $\mathbf{A}(\mathsf{p2})\mathbf{s}(\mathsf{rind})$. This is a violation of LWE security (in other words, we cannot simulate the corresponding query transcripts observed by the adversary in our security proof based on LWE).

Our Solution: LWR-based Cross-Tags. To avoid both the above issues, we switch from relying on the LWE assumption to its deterministic counterpart – the LWR assumption.

More concretely, we now generate the **xtag** and the **xtoken** deterministically as

$$\mathbf{u} = \lfloor \mathbf{A}(\mathsf{p2})\mathbf{s}(\mathsf{rind}) \rceil_p, \quad \mathbf{W} = \lfloor (\mathbf{A}(\mathsf{p1}))^{\mathbf{t}}\, \mathbf{A}(\mathsf{p2}) \rceil_p$$

where $p < q$ is an appropriately chosen rounding modulus. The **xtag** helper re-construction procedure remains unchanged. Note that we still have correctness, since

$$\mathbf{u}' = \mathbf{y}\mathbf{W} = \mathbf{y} \cdot \left(\lfloor (\mathbf{A}(\mathsf{p1}))^{\mathbf{t}}\, \mathbf{A}(\mathsf{p2}) \rceil_p \right) = \mathbf{u}$$

where the equality follows from the fact that the vector $\mathbf{y}$ consists of short entries.

Note that instead of adding a random error term, we now introduce the error via rounding in a deterministic manner, and the rounding modulus p is chosen relative to q such that the re-constructed **xtag** matches the original **xtag** exactly with overwhelmingly large probability. This solves both of the above-mentioned issues, since we can now implement the XSet to support exact look-ups using a Bloom filter, while also avoiding the security vulnerabilities associated with approximate **xtag** reconstruction as discussed above. See Section 3 for a more complete technical description.

1.4 Performance and Security of OQXT

In this section, we present a brief summary of the practical performance and security guarantees of OQXT.

Storage and Search Overheads. In OQXT, the server-side storage requirement grows linearly with the number of keyword-document pairs in the database, which is asymptotically optimal. This is similar to OXT and asymptotically better than both IEX [KM17] and ConjFilter [PPSY21], where the server-side storage grows quadratically in the worst-case.

The search protocol of OQXT entails a single round of communication between the client and the server. For a conjunctive search query, the search complexity is independent of the total number of documents in the database and scales with the frequency of the least frequent conjunct. Thus OQXT retains the sub-linear search complexity of OXT, and supports fast conjunctive keyword searches in practice.

Leakage Profile and Security Proof. We present a detailed enumeration and analysis of the leakage profile for OQXT, and then provide a formal proof that this is indeed the leakage incurred by OQXT against a semi-honest server with quantum-computation capabilities. Our proof is in the standard simulation-based real world/ideal world paradigm against a semi-honest server capable of adaptively issuing search queries of its choice, and only assumes the (plausibly post-quantum) hardness of the LWR problem. The proof consists of establishing formally that a probabilistic polynomial-time simulation algorithm can simulate the view of the adversarial server (in a computationally indistinguishable manner) given access to only the leakage profile for our scheme. The detailed leakage enumeration and security proof are presented in Section 4.

Experimental Evaluation. We present a C++ implementation of OQXT along with

performance figures in Section 5. We experimented over the Enron email corpus[3] for compatibility with previous SSE literature. The data set contains around 6K keywords, 10K documents and 80 thousand unique keyword-document pairs. Our experiments validate that OQXT supports extremely fast conjunctive queries, and asymptotically scales with OXT. Storage requirements for the encrypted database of OQXT is higher than OXT due to the use of quantum-safe lattice-based instantiations. This however is considered as a trade-off for the post-quantum security provided by OQXT while supporting efficient conjunctive search. The search execution time of OQXT maintains the sublinear search time of OXT and is hence very efficient. [4]

1.5 Related Work

SSE for Single-Keyword Search. The seminal works of Song et al. [SWP00] and Goh et al. [Goh03] introduced SSE schemes for single-keyword search with linear search complexity. Subsequent works by Curtmola et al. [CGKO06] and Chase et al. [CK10] introduced practically efficient SSE schemes for single-keyword search with sub-linear search complexity. Unfortunately, these schemes are restricted to static databases and do not support secure updates. An alternative line of works [CM05, KPR12, KP13] initially investigated SSE for dynamic databases, and were subsequently improved upon (both in terms of security and practical efficiency) in a more recent line of works [SPS14, Bos16, GMP16, KKL$^+$17, BMO17, EKPE18, SDY$^+$18, SYL$^+$18, CPPJ18, DCPP20, SSL$^+$21, CPKD22]. An alternative line of works has investigated SSE schemes for single-keyword search incurring minimal leakage [KM19a, PPYY19a, GPPW20, GKM21] with the goal of countering *leakage-abuse attacks* [IKK12, CGPR15, BKM20, KKM$^+$22, GPP23]. All of above schemes rely on purely symmetric-key primitives and are inherently quantum-safe; however, they are extremely limited in terms of query expressiveness. In this paper, our focus is on SSE for conjunctive and richer Boolean queries.

SSE for Boolean Queries. The first SSE scheme to support conjunctive (and a restricted class of general Boolean) queries over static encrypted document collections was OXT by Cash et al. [CJJ$^+$13]. As already mentioned, OXT crucially relies on discrete log hard groups and is quantum-broken. A sequence of works have introduced several refinements and extensions of OXT, including highly optimized practical implementations [CJJ$^+$14], extensions to range and substring queries [FJK$^+$15], leakage suppression [LPS$^+$18], and extension to dynamic databases [PM21]. As mentioned earlier, all of these schemes also rely on discrete log hard groups, and are hence also quantum-broken.

There exist inherently quantum-safe SSE schemes from purely symmetric-key primitives that support highly expressive Boolean queries, but are practically inefficient [KM17, JP22, PPSY21, APP$^+$23a] since they incur a storage complexity that grows *quadratically* with the size of the plaintext database in the worst case, and hence do not scale to real-world databases, where the number of keywords typically range from tens of thousands to even millions. As already mentioned, we want practically efficient conjunctive SSE schemes with

[3] https://www.cs.cmu.edu/~enron/,
https://www.kaggle.com/wcukierski/enron-email-dataset

[4] We plan to make our prototype implementation open-source when the paper is accepted.

linear storage overhead. In this paper, we investigate the possibility of designing such a scheme from quantum-safe lattice-based hardness assumptions.

Encrypted Search for Additional Query Classes. An alternative line of works investigates SSE schemes supporting point, range and substring queries [FJK$^+$15, DPP$^+$, DPP$^+$18, DPPS20], as well as SSE schemes supporting join and group-by queries [KM18, DPPS20, JP22]. We note here that SSE schemes supporting range queries have been cryptanalyzed extensively, notably in [KKNO16, LMP18, GLMP18, GLMP19, GJW19]. Similarly, there exists a large body of work on order-preserving and property-preserving encryption [AKSX04, PRZB11, BCO11, Ker15, LW16, TK20] supporting a rich class of SQL queries over encrypted relational databases, many of which have also been broken by leakage-abuse and inference attacks [NKW15]. Our goal in this paper is to design SSE schemes for Boolean queries (specifically, conjunctive queries) from lattice-based assumptions, and we do not consider range (or other classes of) queries in the present version.

Comparison with FHE. Since OQXT relies on lattice-based assumptions, it is natural to compare it with FHE [Gen09, BV11, BGV14, CGGI16, CGGI20], which also relies on lattice-based assumptions. At a high level, OQXT avoids several inherent inefficiencies incurred by using full-fledged FHE for searching over encrypted document collections: (a) the high storage requirements typically incurred by encrypting the entire document collection using FHE (in OQXT, only the inverted index is encrypted using lattice techniques, while the actual documents are simply encrypted using AES-256; as mentioned earlier, in our implementation and experiments, the plaintext size is only 0.05% of the size of the overall plaintext database), and (b) the computational costs of FHE-bootstrapping, which is a major bottleneck when scaling FHE to extremely large databases (in particular, keyword-search operations, when modeled as Boolean/arithmetic circuits, could have high depth and thus high bootstrapping costs, leading to high query latency). As a result, we expect OQXT to offer significantly faster searches and higher scalability in practice as compared to traditional FHE-based solutions for encrypted keyword search.

2 Preliminaries and Background

In this section, we present some preliminary background material used in our construction. We begin by describing the notations used throughout the paper followed by a generic SSE syntax, a brief discussion on the TSet data structure, some basic crypto-primitves and a brief overview of hard lattice problems that guarantees the security of OQXT.

2.1 Notations

We write $x \xleftarrow{\$} \chi$ to represent that an element x is sampled uniformly at random from a set/distribution $\mathcal{X}$. The output x of a deterministic algorithm $\mathcal{A}$ is denoted by $x = \mathcal{A}$ and the output x' of a randomized algorithm $\mathcal{A}'$ is denoted by $x' \leftarrow \mathcal{A}'$. We refer to $\lambda \in \mathbb{N}$ as the security parameter, and denote by $\mathsf{poly}(\lambda)$ and $\mathsf{negl}(\lambda)$ any generic (unspecified)

polynomial function and negligible function in λ, respectively[5]. We denote integers by $\mathbb{Z}$ and multiplicative group modulo some prime (q) over integers as $\mathbb{Z}_q$. Vectors are denoted by lower-case bold letters (e.g., $\mathbf{x}$) and are always in column form ($\mathbf{x^t}$ is a row vector). Matrices are denoted by upper-case bold letters $\mathbf{X}$. F and F_q are pseudorandom functions with output range in $\{0,1\}^\lambda$ and $\mathbb{Z}_q$ respectively.

Databases. Let $\mathcal{W} = \{\mathsf{w}_1, \ldots, \mathsf{w}_N\}$ be a dictionary of keywords where N is the total number of keywords in the database. The total number of documents in the database is denoted by d, each document is associated with a unique identifier denoted as id and contains keywords from $\mathcal{W}$. We denote by $\mathbf{DB}$ a database of identifier-keyword pairs, such that $(\mathsf{id}, \mathsf{w}) \in \mathbf{DB}$ if and only if the document with identifier id contains the keyword w. We denote by $\mathbf{DB}(\mathsf{w})$ the set of all identifiers corresponding to documents containing w. We denote by $|\mathcal{W}|$ the number of distinct keywords in $\mathbf{DB}$, by $|\mathbf{DB}|$ the number of distinct id-w pairs in $\mathbf{DB}$, and by $|\mathbf{DB}(\mathsf{w})|$ the number of documents containing w. Maximum number of keywords in a query in denoted by n and the result set returned by the server to the client is denoted by R_q.

Lattice-based Instantiations. For lattice based instantiations we use rounding parameters and statistical errors as mentioned in the respective implementations that we incorporate in this paper [BPR12, MP12]. Concretely, we take $r \simeq ln(2/\epsilon)/\pi$ where ϵ is a desired bound on the statistical error introduced by each randomized-rounding operation for $\mathbb{Z}$.

The discrete Gaussian probability distribution over $\mathbb{Z}$ with parameter $r > 0$, denoted $\mathbb{D}_{\mathbb{Z},r}$, assigns probability proportional to $\exp(\pi x^2/r^2)$ to each $x \in \mathbb{Z}$. It is possible to efficiently sample from the discrete Gaussian distribution over a desired coset $\bigwedge^{\perp}(\mathbf{A})$ using a trapdoor for $\mathbf{A}$ via an efficient $\mathsf{SampleD}$ function [MP12].

We define a 'rounding' function as done in [AKPW13] $\lfloor\cdot\rceil_p$: $\mathbb{Z}_q \to \mathbb{Z}_p$, where $q \geq p \geq 2$ and,

$$\lfloor x \rceil_p = \lfloor (p/q) \cdot \bar{x} \rceil \mod p$$

where $\bar{x} \in \mathbb{Z}$ is any integer congruent to $x \mod q$. We naturally identify elements of $\mathbb{Z}_k$ with integers in the interval $\{0, \ldots, k-1\}$. Intuitively, $\lfloor\cdot\rceil_p$ partitions $\mathbb{Z}_q$ into intervals of length $\simeq \frac{q}{p}$ which it maps to the same image. We naturally extend the rounding function to vectors over $\mathbb{Z}_q$ by applying it component-wise.

Conjunctive Queries. We represent a conjunctive query over n distinct keywords $(\mathsf{w}_1, \ldots, \mathsf{w}_\mathsf{n})$ as $q = (\mathsf{w}_1 \wedge \ldots \wedge \mathsf{w}_\mathsf{n})$. We consider w_1 as the least frequent keyword in the query without loss of generality. Throughout the paper we denote w_1 as sterm and rest of the keywords in the query $\mathsf{w}_2, \ldots, \mathsf{w}_\mathsf{n}$ as the xterm.

2.2 SSE: Syntax and Security Model

In this section, we formally define Searchable Symmetric Encryption (SSE) for static databases.

[5]Note that a function $f : \mathbb{N} \to \mathbb{N}$ is said to be negligible in λ if for every positive polynomial p, $f(\lambda) \leq 1/p(\lambda)$ when λ is sufficiently large

Static SSE. A *Searchable Symmetric Encryption (SSE) scheme* Π for static databases consists of an algorithm $\mathsf{EDBSetup}$ and a protocol Search between the client and server, mentioned as follows:

- $\mathsf{EDBSetup}$ takes as input a database $\mathbf{DB}$, and outputs a secret key K along with an encrypted database $\mathbf{EDB}$.
- The Search protocol is between a *client* and *server*, where the client takes as input the secret key K and a query q and the server takes as input $\mathbf{EDB}$. At the end, the client outputs a set of identifiers, and the server has no output.

Correctness of SSE. An SSE scheme is *correct* if for all inputs $\mathbf{DB}$ and queries q, if $(K, \mathbf{EDB}) \xleftarrow{\$} \mathsf{EDBSetup}(\mathbf{DB})$, after running Search with client input (K, q) and server input $\mathbf{EDB}$, the client outputs the set of indices $\mathbf{DB}(q)$.

Adaptive Security of SSE. We recall the semantic security definitions of SSE from [CK10, CGKO06]. The definition is parameterized by a *leakage function* $\mathcal{L}$, which describes what an adversary (the server) is allowed to learn about the database and queries. Formally, security says that the server's view during an adaptive attack (where the server selects the database and queries) can be simulated given only the output of $\mathcal{L}$.

Definition 1. *Let* $\Pi = (\mathit{EDBSetup}, \mathit{Search})$ *be an SSE scheme and let* $\mathcal{L}$ *be a stateful algorithm. For algorithms* $\mathcal{A}$ *(denoting the adversary) and* $\mathcal{S}$ *(denoting a simulator), we define the experiments (algorithms)* $\mathbf{Real}^{\Pi}_{\mathcal{A}}(\lambda)$ *and* $\mathbf{Ideal}^{\Pi}_{\mathcal{A},\mathcal{S}}(\lambda)$ *as follows:*

$\mathbf{Real}^{\Pi}_{\mathcal{A}}(\lambda)$ *:* $\mathcal{A}(1^{\lambda})$ *chooses* $\mathbf{DB}$*. The experiment then runs* $(K, \mathbf{EDB}) \leftarrow \mathit{EDBSetup}(\mathbf{DB})$*, and gives* $\mathbf{EDB}$ *to* $\mathcal{A}$*. Then* $\mathcal{A}$ *repeatedly chooses a query* q*. To respond, the game runs the* Search *protocol with client input* (K, q) *and server input* $\mathbf{EDB}$ *and gives the transcript and client output to* $\mathcal{A}$*. Eventually* $\mathcal{A}$ *returns a bit that the game uses as its own output.*

$\mathbf{Ideal}^{\Pi}_{\mathcal{A},\mathcal{S}}(\lambda)$ *: The game initializes a counter* $\mathsf{cnt} = 0$ *and an empty list* $\mathbf{q}$*.* $\mathcal{A}(1^{\lambda})$ *chooses* $\mathbf{DB}$*. The experiment runs* $\mathbf{EDB} \leftarrow \mathcal{S}(\mathcal{L}(\mathbf{DB}))$ *and gives* $\mathbf{EDB}$ *to* $\mathcal{A}$*. Then* $\mathcal{A}$ *repeatedly chooses a query* q*. To respond, the game records this as* $\mathbf{q}[i]$*, increments* i*, and gives to* $\mathcal{A}$ *the output of* $\mathcal{S}(\mathcal{L}(\mathbf{DB}, \mathbf{q}))$*. (Note that here,* $\mathbf{q}$ *consists of all previous queries in addition to the latest query issued by* $\mathcal{A}$*.) Eventually* $\mathcal{A}$ *returns a bit that the game uses as its own output.*

We say that Π *is* $\mathcal{L}$*-semantically-secure against adaptive attacks if for all adversaries* $\mathcal{A}$ *there exists an algorithm* $\mathcal{S}$ *such that*

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

Selective Security of SSE. We also consider a weaker version of *selective* security for SSE that is identical to the adaptive security definition except that: (a) in the real world experiment,

the adversary $\mathcal{A}$ does not get to choose its queries adaptively, but is required to specify all such queries non-adaptively at the beginning of the protocol along with the plaintext database $\mathbf{DB}$, and receives $\mathbf{EDB}$ and the transcript and client output corresponding to each of its queries together at the end of the experiment. Also, (b) in the ideal world experiment, the adversary $\mathcal{A}$ directly receives as output the final response of a non-adaptive simulator $\mathcal{S}$, computed as $\mathcal{S}(\mathcal{L}(\mathbf{DB}, \{\mathbf{q}[i]\}_{i\in[Q]}))$, where Q is the total number of queries issued by the adversary $\mathcal{A}$ non-adaptively.

Remark. In this paper, we choose to define SSE for encrypted document collections, with queries structured as membership-finding of (one or more) keywords in the documents. This model is naturally applicable when the underlying database is a collection of free-text documents, and is most commonly used in the vast majority of the SSE literature [CGKO06, CJJ+13, CJJ+14, KM17, Bos16, BMO17, LPS+18, PPSY21, SSL+21]. We note here that certain prior works (e.g. [CJJ+13, KM18, JP22]) have used an alternative formulation of SSE for encrypted relational databases, with queries structured as (one or more) attribute-value pairs. We note that the document collection-oriented formulation of SSE can, in fact, also be used to model the relational-database oriented formulation of SSE as follows: each (attribute, value) pair is modeled as a (unique) keyword, and each record containing this attribute-value pair is modeled as a document. For this reason, and also because it is more widely adopted in the SSE literature [CGKO06, CJJ+13], we opt for the document collection-oriented formulation of SSE in this paper.

2.3 TSets

We briefly explain the syntax of the special data structure introduced in [CJJ+13, CJJ+14], namely, the tuple set or TSet. Intuitively, a TSet associates a list of fixed-sized data tuples (list for each keyword is made of every document identifier that contains the particular keyword) with each keyword in the database. The original OXT scheme uses it as an "expanded inverted index". For conjunctive keyword search, the TSet is used to store the encrypted indices of document along with some additional information, that is later used by the server to obliviously compute tokens for **xtag** generation. Formally TSet instantiation consists of three algorithms $\Sigma = (\mathsf{TSet.SetUp}, \mathsf{TSet.GetTag}, \mathsf{TSet.Retrieve})$, each of these algorithm is briefly explained below.

TSet Syntax. Formally, a TSet implementation $\Sigma = (\mathsf{TSet.SetUp}, \mathsf{TSet.GetTag}, \mathsf{TSet.Retrieve})$ will consist of three algorithms with the following syntax:

- $\mathsf{TSet.SetUp}$ takes as input $T = (T_1, \ldots, T_N)$, where each T_i for $i \in [N]$ is an array of lists of equal-length bit strings indexed by the elements of $\mathcal{W}_i$, and outputs (TSet, K_T).
- $\mathsf{TSet.GetTag}$ takes as input the key K_T and a tuple (i, w) and outputs stag_i.
- $\mathsf{TSet.Retrieve}$ takes as input TSet and stag_i, and returns a list of strings.

TSet Correctness. We say that Σ is *correct* if for all $\{\mathcal{W}_i\}_{i\in[N]}$, all $T = (T_1, \ldots, T_N)$, and any $\mathsf{w} \in \mathcal{W}_i$, we have

$$\mathsf{TSet.Retrieve}(\mathsf{TSet}, \mathsf{stag}) = T_i[\mathsf{w}]$$

when we have $(\mathsf{TSet}, K_T) \leftarrow \mathsf{TSet.SetUp}(T)$ and $\mathsf{stag} \leftarrow \mathsf{TSet.GetTag}(K_T, (i, \mathsf{w}))$. Intuitively, T holds lists of tuples associated with keywords and correctness guarantees that the $\mathsf{TSet.Retrieve}$ algorithm returns the data associated with the given keyword.

TSet Security. The security goal of a TSet implementation is to hide as much as possible about the tuples in $T = (T_1, \ldots, T_N)$ and the attribute-value pairs these tuples are associated to, except for the vectors $T_i[\mathsf{w}_1], T_i[\mathsf{w}_2], \ldots$ of tuples revealed by the client's queried attribute-value pairs $\mathsf{w}_1, \mathsf{w}_2, \ldots$. (For the purpose of TSet implementation we equate client's query with a single attribute-value pair.)

The formal definition of security for TSet is similar to that of keyword-search based SSE for single-keyword queries. We refer the reader to prior works [CJJ+13, CJJ+14, JP22] for the formal definition and concrete instantiations of TSet; in our paper, we adopt the same definition of security and the same concrete instantiation as used in these works.

2.4 Basic Cryptographic Primitives

This section defines basic cryptographic primitives used throughout the paper.

Pseudorandom Function (PRF). A pseudorandom function (PRF) is a polynomial-time computable function

$$F : \{0,1\}^\lambda \times \{0,1\}^\ell \longrightarrow \{0,1\}^{\ell'}$$

such that for any security parameter λ and any PPT algorithm $\mathcal{A}$, we have

$$\left| \Pr\left[\mathcal{A}^{F(K,\cdot)} = 1\right] - \Pr\left[\mathcal{A}^{G(\cdot)} = 1\right] \right| \leq \mathsf{negl}(\lambda)$$

where $K \leftarrow \{0,1\}^\lambda$ and G is uniformly sampled from the set of all functions that map $\{0,1\}^\ell$ to $\{0,1\}^{\ell'}$.

Symmetric-Key Encryption (SKE). A symmetric-key encryption scheme SKE consists of the following polynomial-time algorithms:

- $\mathsf{Gen}(\lambda)$: Takes the security parameter λ as input and outputs a secret-key sk.
- $\mathsf{Enc}(\mathsf{sk},x)$: Takes as input a key sk and a plaintext x. Outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{sk},\mathsf{ct})$: Takes as input a key sk and a ciphertext ct. Outputs the decrypted plaintext x.

Correctness. A symmetric-key encryption scheme is said to be correct if for any security parameter λ and any plaintext x, we have $\mathsf{Dec}(\mathsf{sk}, Enc(\mathsf{sk}, x)) = x$, where $\mathsf{sk} \leftarrow \mathsf{Gen}(\lambda)$.

IND-CPA Security. A symmetric-key encryption scheme is said to be IND-CPA secure if for any security parameter λ, for any two *arbitrary* plaintext messages x_0 and x_1, for any $\mathsf{sk} \leftarrow \mathsf{Gen}(\lambda)$, letting $\gamma_b = \Pr\left[\mathcal{A}^{\mathsf{Enc}(\mathsf{sk},\cdot)}(\mathsf{Enc}(\mathsf{sk}, x_b)) = 1\right]$ for each $b \in \{0,1\}$, we have $|\gamma_0 - \gamma_1| \leq \mathsf{negl}(\lambda)$.

2.5 Lattice Preliminaries

Short Integer Solution Problem. For $\beta > 0$, the *short integer solution problem* $\mathsf{SIS}_{n,m,q,\beta}$ [Ajt96] is an average-case version of the approximate shortest vector problem on lattice $\bigwedge^{\perp}(\mathbf{A})$. The problem states that: given uniformly random matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ for any $m = \mathsf{poly}(n)$, find a short non-zero $\mathbf{z} \in \bigwedge^{\perp}(\mathbf{A})$ or a non-zero $\mathbf{z} \in \mathbb{Z}^m$, such that $\mathbf{Az} = 0$ mod q and $\|z\| \leq \beta$. When $q \geq \beta n \cdot \omega(logn)$, solving this problem (with any non-negligible probability over the random choice of $\mathbf{A}$) is at least as hard as (probabilistically) approximating the Shortest Independent Vectors Problem (SIVP), a classic hard problem in the computational study of lattices [MG02].

Learning With Rounding Problem. Let $n \geq 1$ be the main security parameter and moduli $q \geq p \geq 2$ be integers. For a vector $s \in \mathbb{Z}_q^n$, the LWR distribution L is defined as the distribution over $\mathbb{Z}_q^n \times \mathbb{Z}_p$ obtained by choosing a vector $a \leftarrow \mathbb{Z}_q^n$ uniformly at random, and outputting $(a, b = \lfloor \langle a, s \rangle \rceil_p)$ [BPR12]. For a given distribution over $s \in \mathbb{Z}_q^n$ the *decision*-$\mathsf{LWR}_{n,q,p}$ problem is to distinguish (with advantage non-negligible in n) between any desired number of independent samples $(a_i, b_i) \leftarrow L$, and the same number of samples drawn uniformly and independently from $\mathbb{Z}_q^n \times \mathbb{Z}_p$.

Lattice Trapdoors. Advanced lattice cryptographic primitives require generating a random matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ along with some *trapdoor* $\mathbf{T} \in \mathbb{Z}_q^{m \times m}$ such that $\mathbf{AT} = 0$ *mod* q. It is evident from the above relation that the trapdoor is a matrix of *short* integer vectors and is often interpreted as the basis of the lattice defined by using the parity check matrix $\mathbf{A}$. The efficiency of a cryptographic algorithm incorporating such lattice trapdoors is determined by the "quality" of the trapdoor. Quality (s) of a trapdoor $\mathbf{T}$ is equivalent to the Euclidean length (norm) of its vectors. We consider here n as the main security parameter that determines the robustness of the functions and m is the dimension of a lattice associated with $\mathbf{A}$, generated by the basis $\mathbf{T}$. Typically it is considered that $m = \theta$(n log q), which is optimal up to constant factors. The trapdoor generation algorithm devised by Micciancio and Peikert [MP12] shows to generate a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ within negligible statistical distance from uniform, the dimension of lattice should be $m \simeq 2$n log q. They propose an efficient algorithm for trapdoor generation with the quality of the trapdoor $s \simeq 1.6$ n log q. Also using this trapdoor they devised an optimised SIS preimage sampling algorithm with a bound on the length of the preimage as $\beta \simeq s\sqrt{m}$.

Discrete Gaussian Distribution The discrete Gaussian distribution $\mathbb{D}_{(\bigwedge + c), \sqrt{\Sigma}}$, is a Gaussian distribution $\mathbb{D}_{\sqrt{\Sigma}}$ that is restricted to support the coset $(\bigwedge + c)$, where $\bigwedge \in \mathbb{R}^n$ is a lattice, $c \in \mathbb{R}^n$ and Σ is a positive semi-definite matrix such that $(\bigwedge + c) \cap$ span (Σ) is non-empty. That is, for all $a \in (\bigwedge + c)$,

$$\mathbb{D}_{\bigwedge + c, \sqrt{\Sigma}}(a) = \frac{\rho_{\sqrt{\Sigma}}(a)}{\rho_{\sqrt{\Sigma}}(\bigwedge + c)} \propto \rho_{\sqrt{\Sigma}}(a)$$

where ρ is a Gaussian function $\rho : \mathbb{R}^n \to (0, 1]$ defined as -

$$\rho(x) \triangleq \exp(-\pi \cdot \|x\|^2) = \exp(-\pi \cdot \langle x, x \rangle)$$

3 Oblivious Post-Quantum Secure Cross-tags (OQXT)

In this section we elucidate the techniques devised in OQXT more formally. We address the limitation of OXT in a quantum setting and provide a plausibly quantum-safe solution. OQXT is broadly bifurcated into two phases - *Setup* and *Search* phase. We formally describe them in algorithmic form in Algorithm 1 and Algorithm 2 respectively.

In order to explicate the motivation behind developing a post-quantum secure construction of OXT, we first try to answer the question - *Why is* OXT *not quantum-safe?*

Oblivious Cross-Tag Protocol (OXT**).** To answer the above mentioned question we first provide a brief understanding of the the technical details of the OXT protocol. OXT [CJJ$^+$13] relies on specially structured pseudo-random functions that can be incorporated using discrete-log hard groups. The idea is that the client encrypts the data (using symmetric-key encryption) and offloads it to the server (*honest-but-curious*). The client queries the server with a conjunction of keywords to which it returns a set of encrypted pointers that point to documents containing all the client's queried keywords. The client decrypts these pointers locally to obtain the required documents matching the conjunctive query. The server does not have the ability to perform decryption nor can it learn any information about the queried keywords. The entire protocol constitutes one-round of interaction/communication between the client and the server. This is delegated by an *oblivious shared computation of cross-tags* between client and server. In order to achieve minimum communication complexity while ensuring privacy of the client's queried keywords and corresponding documents, OXT incorporates an *oblivious cross-tag* generation process. The cross-tag is computed using blinded exponentiation in prime order cyclic groups, analogous to *Diffie-Hellman based oblivious PRF*. It pre-computes the blinding part of the oblivious computation and stores them in encrypted form at the server. During search, the client sends tokens to the server using which it unlocks these pre-computed values and computes the cross-tag *obliviously* using which it matches the documents corresponding to the queried keywords.

The core technical idea of OXT is the process of oblivious cross-tag generation, that relies on public-key computations in a discrete-log hard group. The hardness assumption is based on the fact that *DDH assumption* is conjectured to hold in a prime order subgroup of Z_p^* (p is a large prime). The inherent reliance of OXT on discrete-log hard groups renders it vulnerable to quantum attacks and can be effectively broken by a scalable quantum computer. Therefore, although OXT provides security against efficient adaptive adversaries in classical setting, it is not secure in the post-quantum setting. The motivation of our work is hence, to develop a post-quantum secure construction of OXT that preserves its asymptotic search and communication complexity while ensuring its scalability with arbitrary large datasets. In order to understand the core technical novelty devised in OQXT let us analyze the most crucial component of OXT– the *cross-tags* (**xtag**).

The Cross-Tag. The most fundamental component of OXT which is incorporated to check for the presence of a keyword in a particular document without leaking any extra information to the server, is the *cross-tag* (denoted as **xtag** in the paper). The fundamental building block that makes OXT one of the most efficient, highly scalable and secure conjunctive SSE scheme is an *oblivious computation of the* **xtag** *at the server*, which is incorporated

using a DH-based oblivious PRF type computation. The idea is, for every keyword and every document in which it is present, the client pre-computes a **xtag** (an element in the prime-order subgroup of Z_p^*) and stores it in a data-structure called XSet which is offloaded to the server.

$$\mathbf{xtag} \leftarrow g^{F_p(K_X,\mathsf{w})\cdot F_p(K_I,\mathsf{id})}$$

where, g is the generator of the subgroup of Z_p^* and F_p is a PRF that takes as input a keyword or a document identifier and outputs an element in Z_p^*.

Along with the queried keywords, the server receives some tokens (**xtoken**) from the client (also an element in the prime-order subgroup of Z_p^*) during any conjunctive search. The beauty of OXT lies in the fact that the whole process of **xtag** computation by the server takes place obliviously without revealing the keyword-document pair for which the **xtag** is being computed. This is done by raising the **xtoken** to a blinded value y (an element in Z_p^*) which is pre-computed by the client and stored at the server.

3.1 Description of OQXT

Core Idea. We propose a potential solution to the above limitation by replacing computations in discrete-log hard groups with post-quantum secure lattice instantiations. More specifically, we transform the cross-tags and in order to preserve the oblivious computations as done in OXT the **xtoken** and *blinding factor* computation is also transformed in accordance with post-quantum secure instantiations over lattices. The cross-tags generated in OXT as ($g^{F_p(K_X,\mathsf{w})\cdot\mathbf{xid}}$) i.e. elements in a discrete-log hard group (where g is the generator of the prime order subgroup of Z_p^*) is modified and generated as LWR samples in $\mathbb{Z}_q^n \times \mathbb{Z}_p$. By hardness of LWR assumption **xtag** is indistinguishable from any random element in $\mathbb{Z}_q^n \times \mathbb{Z}_p$ even to adversaries with unbounded computational powers. The main crux of OXT constitutes the *oblivious* computation of the cross-tags by the server, using some form of blinded exponentiation, like in DH-based oblivious PRFs. We preserve the innate oblivious computation of cross tags in a post-quantum secure framework by incorporating special *lattice trapdoor* techniques and hard SIS instances.

Modified Cross-Tags in XSet. The first step towards a post-quantum secure transformation of OXT is to modify the *cross-tags* - elements in discrete-log hard groups which are indistinguishable from random elements in Z_p^* by the hardness of DDH assumption. The technical centerpiece of OQXT is a novel oblivious cross-tag generation protocol with provable security guarantees derived from post-quantum secure hardness assumptions of *Learning With Rounding* problem. The cross-tags (**xtag**) are LWR samples generated from a public matrix $\mathbf{xw} \in \mathbb{Z}_q^{n\times n}$ and a secret $\mathbf{xid} \in \mathbb{Z}_q^{n\times n}$, which are outputs of PRF F_q. In particular, for some $p < q$, we divide up the elements of $\mathbb{Z}_q$ into p contiguous intervals of roughly q/p elements each and define the rounding function $\lfloor\cdot\rfloor_p : \mathbb{Z}_q \to \mathbb{Z}_p$ that maps each element in $\mathbb{Z}_q$ into the index of the interval that it belongs to. For example here, we consider both q and p as powers of 2, therefore, this results **xtag** to be the $log(p)$ most significant bits of $[\mathbf{xid}\cdot\mathbf{xw}]$. According to [BPR12] as long as $q/p \geq \sqrt{n}$ is an integer, the LWR problem appears to be exponentially hard (in n) for any $p = poly(n)$, and super-polynomially hard for $p \leq 2^{n^\epsilon}$ for any $\epsilon < 1$.

$$\mathbf{xtag} = \lfloor \mathbf{xid}\cdot\mathbf{xw} \rceil_p$$

The **xtag** constitutes of two components - the public matrix $\mathbf{xw}$, an element in $\mathbb{Z}_q^{n\times n}$ which is calculated for every keyword w. The LWR secret **xid**, which corresponds to all the documents in which w occurs. Hence, for each w there can be multiple LWR secrets, which makes **xid** an element in $\in \mathbb{Z}_q^{n\times n}$.

Lattice Trapdoor Generation. In order to delegate the oblivious computation of cross-tags to the server during any conjunctive search in a post-quantum secure framework the client performs some pre-computations during *Setup* phase. We incorporate the trapdoor generation algorithm devised in [MP12] with a slight tweak to generate a random matrix in $\mathbb{Z}_q^{n\times m}$ along with a trapdoor $\mathbf{T}$. The idea is to generate for every keyword w, a matrix $\mathbf{z_w} \in \mathbb{Z}_q^{n\times m}$ and a trapdoor $\mathbf{T}$ for $\mathbf{z_w}$ using special lattice trapdoor function ($\mathsf{Gen\_Trapdoor}$), that takes as input some randomness r calculated as $r \leftarrow F'_q(K_Z, \mathsf{w}||c)$ where c is a counter that keeps count of the frequency of w. The significance of generating this trapdoor is to use it for efficient pre-image sampling from desired coset (**xid**) of the lattice $\bigwedge^{\perp}(\mathbf{z_w})$.

Pre-image Sampling from Discrete Gaussian. In [MP12] the authors propose an efficient and simple methodology to sample SIS pre-images from a specific coset of a discrete Gaussian distribution. We use the $\mathsf{SampleD}$ function from [MP12] to sample a (*short*) pre-image from a given coset of a discrete Gaussian distribution (we sample from a similar distribution as discussed in Section 2) with a relatively small parameter $s \simeq s_1(T)\cdot s_1(\sqrt{\sum_{\mathbf{G}}})$ (as in [MP12]). With the help of the trapdoor, it is possible to to generate a short $\mathbf{y}_{\mathsf{id}}$ appropriately from a (nearly spherical) *discrete Gaussian* distribution $\mathbb{D}_{\bigwedge^{\perp}_{\mathbf{xid}}(\mathbf{z_w}),s}$ so as to ensure that the output *syndrome* (**xid**) is uniformly random i.e., for any random $\mathbf{y}_{\mathsf{id}}$, the product $\mathbf{y}_{\mathsf{id}}^{\mathbf{t}} \cdot \mathbf{z}_{\mathbf{w}}^{\mathbf{t}}$ should be uniformly random. The hardness assumption here is given $\mathbf{z_w}$ and the coset $\mathbf{xid} \in \mathbb{Z}_q^{n\times n}$, it is hard to sample a *short* $\mathbf{y}_{\mathsf{id}} \in \mathbb{Z}_q^{m\times n}$ from $\bigwedge^{\perp}_{\mathbf{xid}}(\mathbf{z_w})$ by the hardness of *Short Integer Solutions* (SIS) problem.

$$\mathbf{y}_{\mathsf{id}}^{\mathbf{t}} \cdot \mathbf{z}_{\mathbf{w}}^{\mathbf{t}} = \mathbf{xid}$$

Thus if a trapdoor $\mathbf{T}$ is not known and a random **xid** is provided, it is hard to find a *short* pre-image $\mathbf{y}_{\mathsf{id}}$, this is equivalent to solving the SIS problem, which is conjectured to be a hard lattice problem.

OQXT **Setup Phase.** The Setup phase is entirely executed by the client, where it generates the encrypted database **EDB** and offloads it to the server. In OXT construction **EDB** comprises of two structures, the XSet and the TSet. We use similar structure of **EDB** for our scheme and formally elaborate the construction of XSet and TSet in Algorithm 1. The client generates *cross-tags* (LWR samples) and stores in the XSet. Also it samples a random parity-check matrix $\mathbf{z}_{\mathsf{w}}$ from $\mathbb{Z}_q^{m\times n}$ with a trapdoor $\mathbf{T}$ for every keyword $\mathsf{w} \in \mathcal{W}$. The novelty of OQXT as a post-quantum secure construction of OXT lies in the subtle observation that the trapdoor matrix generated from every keyword can be used to sample a short vector from a specific coset (defined by **xid**) of a *discrete* Gaussian distribution. This ensures that it is difficult to sample a *short* $\mathbf{y}_{\mathsf{id}}$ for every keyword-id pair without the knowledge of the trapdoor $\mathbf{T}$, thereby making $\mathbf{y}_{\mathsf{id}}$ quantum safe. The client encrypts the document identifiers (e_c) and for every document containing a particular keyword w (i.e. every $\mathsf{id} \in \mathbf{DB}(\mathsf{w})$) it generates $\mathbf{y}_{\mathsf{id}}$. Thereafter it stores the $(e_c, \mathbf{y}_{\mathsf{id}}^{\mathbf{t}})$ pairs for every keyword-id pair in TSet using $\mathsf{TSet.SetUp}$ routine (Section 2.3).

Oblivious Conjunctive Search. During a conjunctive search client generates **xtoken**

Algorithm 1 OQXT.Setup

Input: $1^\lambda, \mathbf{DB}$
Output: $\mathsf{mk}, \mathbf{EDB}$

function OQXT.Setup($1^\lambda, \mathbf{DB}$)
  Initialise $T \leftarrow \phi$ indexed by keywords $\mathcal{W}$
  Select key K_S for PRF F
  Select keys K_I, K_X for PRF F_q
  Select keys K_Z for PRF F'_q
  Initialise $\mathbf{EDB} \leftarrow \{\}$
  Initialise $\mathsf{XSet} \leftarrow \{\}$
  for $\mathsf{w} \in \mathcal{W}$ **do**
    Initialise $\mathbf{tset} \leftarrow \{\}$
    Compute $k_e \leftarrow F(K_S, \mathsf{w})$
    Compute $\mathbf{xw}\ (\in \mathbb{Z}_q^{n\times n}) \leftarrow F_q(K_X, \mathsf{w})$
    Set counter $c \leftarrow 1$
    Sample a random matrix $\bar{\mathbf{A}} \in \mathbb{Z}_q^{n\times \bar{m}}$ and a tag $\mathbf{H} = \mathbf{I} \in \mathbb{Z}_q^{n\times n}$
    for $\mathsf{id}\ \in \mathbf{DB}(\mathsf{w})$ **do**
      Compute $\mathbf{xid}\ (\in \mathbb{Z}_q^{n\times n}) \leftarrow F_q(K_I, \mathsf{id})$
      Compute $r \leftarrow F'_q(K_Z, \mathsf{w}||c)$
      $\mathbf{z_w} (\in \mathbb{Z}_q^{n\times m}), \mathbf{T} \leftarrow \mathsf{Gen\_Trapdoor}(\bar{\mathbf{A}}, \mathbf{H}, r)$
      $\mathbf{y}^{\mathbf{t}}_{\mathsf{id}} \leftarrow \mathsf{SampleD}(\mathbf{T}, \mathbf{z}^{\mathbf{t}}_{\mathbf{w}}, \mathbf{xid}^{\mathbf{t}})$
      Compute $e_c \leftarrow Sym.Enc(k_e, \mathsf{id})$
      Set $\mathbf{xtag} \leftarrow \lfloor \mathbf{xid} \cdot \mathbf{xw} \rceil_p$
      Add $\mathbf{xtag}$ to XSet
      Append $(\mathbf{y}^{\mathbf{t}}_{\mathsf{id}}, e_c)$ to $\mathbf{tset}$ and set $c \leftarrow c + 1$
    Set $T[\mathsf{w}] \leftarrow \mathbf{tset}$
  Compute $\{\mathsf{TSet}, K_T\} \leftarrow \mathsf{TSet.SetUp}(T)$
  return $\mathsf{mk} = \{K_S, K_I, K_Z, K_X, K_T\}$, $\mathbf{EDB} = (\mathsf{TSet}, \mathsf{XSet})$

as LWR samples for every queried keyword other than the stag (computed by calling the TSet.GetTag routine (Section 2.3) with the least frequent keyword as input) and sends it to the server, for all ids corresponding to the documents containing the *least frequent* keyword in the query.

$$\mathbf{xtoken}[\mathsf{id}, \mathsf{w}] = \lfloor \mathbf{z_w} \cdot \mathbf{xw} \rceil_p$$

The computation of **xtoken** interestingly combines the matrix $\mathbf{z_w} (\in \mathbb{Z}_q^{n\times m})$ and $\mathbf{xw} (\in \mathbb{Z}_q^{n\times n})$ and rounding the value by the same factor p by which the **xtag** is rounded at Setup. This helps in oblivious computation of the cross-tags at the server. The $\mathbf{y}_{\mathsf{id}}$ values stored in the TSet are retrieved by the server and multiplied with **xtoken**. Due to this **xtoken** is designed as a combination of $\mathbf{z_w}$ and **xw**. A unique $\mathbf{z_w}$ is sampled for every keyword in the query. It is important to note that this sampling process is deterministic because while generating the **xtoken** during *search* phase we need to ensure that the same $\mathbf{z_w}$ with a trapdoor is generated as during the *setup* phase for the keyword w. This is necessary to fetch the correct record $\mathbf{y}_{\mathsf{id}}$ from the TSet, which is calculated using the trapdoor for the particular keyword w and the corresponding ids in $\mathbf{DB}(\mathsf{w})$.

Oblivious Cross-Tag Computation. The server retrieves from TSet all $(e, \mathbf{y}^{\mathbf{t}}_{\mathsf{id}})$ pairs corresponding to stag (by invoking TSet.Retrieve routine (Section 2.3) and computes the matrix $\mathbf{z}$ with a trapdoor $\mathbf{T}$ for the stag using the Gen_Trapdoor function. The oblivious

Algorithm 2 OQXT.Search

Input: $\mathsf{mk}, q = (\mathsf{w}_1 \wedge \ldots \wedge \mathsf{wn}), \mathbf{EDB}$
Output: Result R_q

function OQXT.SEARCH($\mathsf{mk}, q, \mathbf{EDB}$)
  Client's inputs are (mk, q) and server's inputs are $(\mathbf{EDB})$
  Client
  $R_q \leftarrow \{\}$
  $\mathsf{stag} \leftarrow \mathsf{TSet.GetTag}(K_T, \mathsf{w}_1)$
  Sends stag to the server
  Sample a random matrix $\bar{\mathbf{A}} \in \mathbb{Z}_q^{n \times \bar{m}}$ and a tag $\mathbf{H} = \mathbf{I} \in \mathbb{Z}_q^{n \times n}$
  Computes **xtoken** as follows:
  for $c = 1$: until server sends *stop* **do**
    $r \leftarrow F'_q(K_Z, \mathsf{w}_1 || c)$
    $\mathbf{z}_{\mathsf{w}_1} (\in \mathbb{Z}_q^{n \times m}), \mathbf{T_1} \leftarrow \mathsf{Gen\_Trapdoor}(\bar{\mathbf{A}}, \mathbf{H}, r)$
    for $l = 2 : \mathsf{n}$ **do**
      $\mathbf{xw}_l \leftarrow F_q(K_X, l)$
      $\mathbf{xtoken}[c, l] \leftarrow \lfloor \mathbf{z}^{\mathbf{t}}_{\mathbf{w_1}} \cdot \mathbf{xw}_l \rceil_p$
    $\mathbf{xtoken}[c] \leftarrow (\mathbf{xtoken}[c, 2], \ldots, \mathbf{xtoken}[c, n])$
    Client send $\mathbf{xtoken}[c]$ to server
  Server
  Recovers $\mathbf{EDB} = \mathsf{TSet}$
  Computes $\mathbf{tset} \leftarrow \mathsf{TSet.Retrieve}(\mathsf{TSet}, \mathsf{stag})$ for TSet routines)
  for $c = 1 : |\mathbf{tset}|$ **do**
    Server recovers $(\mathbf{y}^{\mathbf{t}}_{\mathsf{id}}, e_c)$ from c-th component of **tset**.
    for $l = 2 : \mathsf{n}$ **do**
      Server computes $\lfloor \mathbf{xtag} \rceil_p = \mathbf{y}^{\mathbf{t}}_{\mathsf{id}} \cdot \lfloor \mathbf{xtoken} \rceil_p$
    If $\forall l \in [2, \mathsf{n}]$, $\mathbf{xtag} \in \mathsf{XSet}$, then send e_c to the client.
    When last tuple in t is reached, send *stop* to client and halt.
  Client
  $K_e \leftarrow F(K_S, \mathsf{w}_1)$.
  $\mathsf{id}_c \leftarrow Sym.Dec(K_e, e_c)$, and adds id_c to R_q for all e_c received.
  return R_q

computation of **xtag** is crafted carefully by multiplying $\mathbf{y}^{\mathbf{t}}_{\mathsf{id}}$ with rounded **xtoken** values for each (w-id) pair to obtain rounded **xtag** value without leaking any extra information to the server as follows.

$$\lfloor \mathbf{xtag} \rceil_p = \mathbf{y}^{\mathbf{t}}_{\mathsf{id}} \cdot \lfloor \mathbf{xtoken} \rceil_p$$

3.2 Proof Of Correctness of OQXT

The proof of correctness of OQXT follows from the correctness of Gen_Trapdoor and SampleD function from [MP12] along with the correctness of TSet instantiations from OXT [CJJ+13] which ensures OQXT to be a correct conjunctive SSE scheme. Correctness of a conjunctive SSE scheme implies that given a conjunctive query $q = \mathsf{w}_1 \wedge \cdots \wedge \mathsf{w}_\mathsf{n}$ over an encrypted database it should satisfy the following relations.

$$\mathbf{EDB} = \mathsf{Setup}(\mathbf{DB})$$

$$\mathbf{DB}(\mathsf{w}_1) \cap \cdots \cap \mathbf{DB}(\mathsf{w}_\mathsf{n}) = \mathsf{Search}(q, \mathbf{EDB})$$

SSE Scheme	Post-Quantum Secure	Hardness Assumption	Search Complexity	Storage Overhead
OXT [CJJ+13]	No	DDH	$O(n\|\mathbf{DB}(\mathsf{w}_1)\|)$	$O(\|\mathbf{DB}\|)$
ConjFilter [PPSY21]	Yes	Random Oracle	$O(n\|\mathbf{DB}(\mathsf{w}_1 \wedge \mathsf{w}_2)\|)$	$O(\sum_{\mathsf{w}_1,\mathsf{w}_2 \in \mathcal{W}} \|\mathbf{DB}(\mathsf{w}_1 \wedge \mathsf{w}_2)\| + \|\mathbf{DB}(\delta)\|)$
IEX-2Lev [KM17]	Yes	Symmetric-key Encryption	$O(n^2\|\mathbf{DB}(\mathsf{w}_1)\|)$	$O(\|\mathcal{W}\|\|\mathbf{DB}\|)$
OQXT (This Work)	Yes	Learning with Rounding	$O(n\|\mathbf{DB}(\mathsf{w}_1)\|)$	$O(\|\mathbf{DB}\|)$

Table 1: Comparison of OQXT with OXT [CJJ+13], IEX-2Lev [KM17] and ConjFilter [PPSY21]. The storage overhead of OQXT scales linearly with the number of keyword-id pairs in the database ($|\mathbf{DB}|$) as in OXT. It is important to note that OQXT significantly reduces the quadratic storage overhead incurred by both IEX-2Lev and ConjFilter. The search complexity of OQXT scales linearly with the least frequent keyword in the conjunctive query and is asymptotically comparable to all the other schemes. (Note: The above comparison of search complexity is done for conjunctive queries of the form $q = \mathsf{w}_1 \wedge \mathsf{w}_2 \wedge \ldots \wedge \mathsf{w}_\mathsf{n}$, where w_1 is the least frequent keyword.)

We state the proof of correctness for OQXT below.

Proof. For the proof of correctness of OQXT we consider the functions $\mathsf{Gen\_Trapdoor}$ and $\mathsf{SampleD}$ gives correct output. Considering $\mathsf{Gen\_Trapdoor}$ produces a random matrix $\mathbf{z}_\mathsf{w} \in \mathbb{Z}_q^{n \times m}$ with a trapdoor $\mathbf{T}$ correctly according to [MP12], such that we can invert $\mathbf{z}_\mathsf{w}$ using this trapdoor. Also, we sample a pre-image using $\mathbf{T}$ from a desired coset of a discrete Gaussian using the $\mathsf{SampleD}$ function, and we assume the correctness of the function follows from [MP12]. Under these assumptions we verify the correctness of OQXT. Consider a conjunctive query q as stated below.

$$q = \mathsf{w}_1 \wedge \cdots \wedge \mathsf{w}_\mathsf{n}$$

According to $\mathsf{OQXT.Setup}$ the client generates for each $(\mathsf{w} - \mathsf{id})$ pair a random matrix $\mathbf{z}_\mathsf{w} \in \mathbb{Z}_q^{n \times m}$ with a trapdoor $\mathbf{T}$ and a pre-image $(\mathbf{y}_\mathsf{id})$ from the coset $\bigwedge(\mathbf{xid})$. It stores $\mathbf{y}_\mathsf{id}$ in the TSet and offloads it to the server. It also generates $\mathbf{xtag}$ as LWR samples for every $(\mathsf{w} - \mathsf{id})$ pair, $\mathbf{xtag} = \lfloor \mathbf{xid} \cdot \mathbf{xw} \rceil_p$ and stores these values in XSet. The $\mathbf{xtag}$ values are rounded upto least significant p bits. When a query of the form q is received at the server, the server first finds the $\mathbf{DB}(\mathsf{w}_1)$ corresponding to the least frequent keyword in the query. This follows from the correctness of the TSet. For id in $\mathbf{DB}(\mathsf{w}_1)$ and the rest of the keywords in q, the client generates $\mathbf{xtoken}$ as $\mathbf{xtoken}[\mathsf{id}, \mathsf{w}] = \lfloor \mathbf{z}_{\mathsf{w}1} \cdot \mathbf{xw} \rceil_p$, where $\mathbf{z}_{\mathsf{w}1}$ is generated by $\mathsf{Gen\_Trapdoor}$ for w_1 along with a trapdoor $\mathbf{T}_1$. $\mathbf{xtoken}$s are again rounded by the same factor as the $\mathbf{xtag}$s. The server on receiving the rounded $\mathbf{xtoken}$ values calculates a rounded $\mathbf{xtag}$ value by multiplying it with $\mathbf{y}_\mathsf{id}$ which it retrieves from TSet for w_1 i.e., $\lfloor \mathbf{xtag} \rceil_p = \mathbf{y}_\mathsf{id}^\mathbf{t} \cdot \lfloor \mathbf{xtoken} \rceil_p$. The correctness of this is ensured by the correctness of the $\mathsf{SampleD}$ function, which inturn ensures that a *short* pre-image $\mathbf{y}_\mathsf{id}$ is sampled from the specific coset of the discrete Gaussian distribution. Since $\mathbf{y}_\mathsf{id}$ is short, we claim that multiplying noisy rounded $\mathbf{xtoken}$ with short $\mathbf{y}_\mathsf{id}$ will give rounded $\mathbf{xtag}$ with overwhelming probability. The noise will not propagate if $\mathbf{xtoken}$ is correct towards the most significant bits (since we drop the least significant bits during rounding).

3.3 Complexity Analysis

Search Overhead. Originally OXT was designed as a *sublinear* conjunctive SSE scheme which implies the search complexity scales linearly to the frequency of the *least frequent* keyword in the query (sterm). In OQXT we preserve the innate sublinear property of OXT and ensure during a conjunctive search the complexity at the server scales with the least frequent keyword in the query (sterm). The novel cross-tag generation of OQXT using secure lattice instantiations add up to some delay during the pre-computation phase, but since it is a one-time process, we consider it as a trade-off for post-quantum security that OQXT provides. The conjunctive search process is similar to that of OXT and our lattice implementations scale asymptotically with the sublinear search complexity of OXT.

Storage Overhead. The server stores the dictionaries TSet and XSet that is computed by the client during Setup. The client locally stores only the metadata using which it keeps a track of the frequency of the keywords in the database. The storage overhead at server follows a similar trend as OXT i.e.TSet and XSet are offloaded to the server with encrypted entries for each keyword-document pair in the database. Incorporation of the lattice-based instantiations, does not incur excessive storage blowup and the asymptotic storage overhead of OXT is retained.

Comparison with Other Schemes. Table 1 presents a comparison of the asymptotic performance overheads of OQXT with that of OXT [CJJ+13], IEX-2Lev [KM17] and ConjFilter [PPSY21] (the latter two are chosen for comparison since they represent the only other state-of-the-art SSE schemes capable of supporting conjunctive queries while also providing post-quantum security). We highlight that OQXT does not incur quadratic storage overhead as in IEX-2Lev and ConjFilter while achieving competitive search overheads for conjunctive queries. We refer the reader to Section 5 for a concrete comparison of the performance overheads of these schemes over real-world databases.

4 Security Analysis of OQXT

The security of OQXT fundamentally relies on the hardness of LWR and SIS problems. The short preimages $\mathbf{y}_{\mathsf{id}}$ sampled from specific cosets of a discrete Gaussian distribution are stored in TSet and used by the server to calculate **xtag** values and check for their membership in the XSet. Using lattice-based trapdoors [MP12] we sample *short* pre-images ($\mathbf{y}_{\mathsf{id}}$) from specific cosets defined by **xid** of a discrete Gaussian distribution. This ensures $\mathbf{y}_{\mathsf{id}}{}^{\mathbf{t}} \cdot \mathbf{z}_{\mathsf{w}}^{\mathbf{t}} = \mathbf{xid}$, where $\mathbf{z}_{\mathsf{w}}$ is a matrix (with elements sampled from $\mathbb{Z}_q^{n\times m}$) with a trapdoor $\mathbf{T}$ generated by the $\mathsf{Gen\_Trapdoor}$ function. This in turn forms a SIS instance given $\mathbf{y}_{\mathsf{id}}^{\mathbf{t}}$ is *short*. While generating matrix $\mathbf{z}_{\mathsf{w}}$ with trapdoor $\mathbf{T}$ we add some fixed randomness r, so as to ensure for every w, $\mathbf{z}$ is unique, hence $\mathbf{y}_{\mathsf{id}}$ is statistically random. The **xtag** values stored in XSet are LWR samples ($\in \mathbb{Z}_q^{n\times n}$). By the hardness of LWR these values are indistinguishable from random values in $\mathbb{Z}_q^{n\times n}$, thus the server learns no information about the underlying keyword or the corresponding document in which it belongs from the **xtag**. Similarly the **xtoken** values sent by the client to the server are LWR samples in $\mathbb{Z}_q^{m\times n}$ and are hence indistinguishable from random values in $\mathbb{Z}_q^{m\times n}$, thus the server learns no information about

the keywords present in the query.

In the rest of this section, we formally enumerate the leakage profile of OQXT and analyze its post-quantum security.

4.1 Leakage Profile Analysis

The leakage profile of OQXT is similar to original OXT scheme. Our scheme ensures no extra information leakage occurs in addition to the information leaked in OXT while ensuring security in the presence of an efficient and scalable quantum computer. We formalise the leakage profile and provide a detailed simulation-based proof of security in Section 4.2.

We describe the function $\mathcal{L}_{\mathrm{OQXT}}$ that captures the leakage profile of OQXT under the scenario where all queries are non-adaptive. We eventually show that our claim for non-adaptive queries also holds for adaptive queries. In addition to the leakage from TSet implementation $\mathcal{L}_T$ (described in [CJJ+13]), $\mathcal{L}_{\mathrm{OQXT}}$ captures all of the information leaked by our quantum-safe SSE construction. We proceed in a similar manner as in OXT by representing a sequence of n non-adaptive queries of two-conjunction as $q = (s, x)$ where each query $q[i]$ is a conjunction of two keywords, we represent as, $q[i] = s[i] \wedge x[i]$. The leakage function $\mathcal{L}_{\mathrm{OQXT}}(\mathbf{DB}, q)$ gets as input the database $\mathbf{DB} = (\mathsf{id}_i, \mathcal{W}_i)_{i=1}^{d}$ and a set of queries $q = (s, x)$ and outputs $N, \bar{s}, \mathsf{SP}, \mathsf{RP}, \mathsf{IP}$ as defined below.

Defining and apprehending the components of $\mathcal{L}_{\mathrm{OQXT}}$. The significance of each component of the leakage function in OQXT is equivalent to that in OXT. We define each leakage component as follows -

- $N = \sum_{i=1}^{d} |\mathcal{W}_i|$ - the total number of appearances of keywords in documents. The parameter N signifies an upper bound which is equivalent to the total size of **EDB**. Leaking such a bound is unavoidable and is considered as a trivial leakage in literature of SSE.
- $\bar{s} \in [m]^{\mathsf{n}}$ - equality pattern of $s \in \mathcal{W}^{\mathsf{n}}$ that indicates which queries have the equal sterms. Repetition of sterms of different queries is leaked by $\bar{s}$. This occurs due to the optimization technique devised in OXT in order to ensure sublinear search complexity by filtering out the least frequent term during search.
- SP - size pattern of the queries i.e., the number of documents matching the sterm in each query. Formally, $\mathsf{SP} \in [d]^{\mathsf{n}}$ and $\mathsf{SP}[i] = |\mathbf{DB}(s[i])|$. It leaks the number of documents satisfying the sterm in a query.
- RP - result pattern of the queries or the indices of documents matching the entire conjunction. Formally, RP is vector of size n with $\mathsf{RP}[i] = \mathbf{DB}(s[i]) \cap \mathbf{DB}(x[i])$ for each i. It is the final output of the search query and is not considered as a real leakage in the context of SSE.

- IP - conditional intersection pattern, a $n \times n$ table defined as -

$$\mathsf{IP}[i,j] = \begin{cases} \mathbf{DB}(s[i]) \cap \mathbf{DB}(s[j]), \\ \quad if\ i \neq j\ and\ x[i] = x[j] \\ \phi, \quad otherwise \end{cases}$$

  IP captures a subtle leakage which occurs due to the specialised computation of **xtag**s and storing a unique **xtag** value in the XSet for each (w-id) pair. Fundamentally IP captures the leakage which occurs when two distinct queries have a common xterm but different sterms and there exists a document that satisfies both the sterms. In such a scenario the set of document indices matching both sterms is leaked (if no document matching both sterms exist then nothing is leaked).

Theorem 1. *Let $\mathcal{L}_{\mathrm{OQXT}}$ be as defined above, and we assume that the TSet implementation $\sum$ from [CJJ+13] is secure . Then SSE scheme* OQXT *is $\mathcal{L}_{\mathrm{OQXT}}$-post-quantum-secure against adaptive attacks, assuming that the $\mathsf{LWR}_{n,p,q}$ problem is hard in $\mathbb{Z}_q^n \times \mathbb{Z}_p$, that solving $\mathsf{SIS}_{n,m,q}$ is hard i.e. sampling a short pre-image from a specific coset of a discrete Gaussian distribution without using a trapdoor T is hard, that F and F_q are secure PRFs, and (Enc, Dec) is an IND-CPA secure symmetric encryption scheme.*

Proof Overview. We begin by proving that OQXT is post-quantum secure against non-adaptive adversaries. For simplicity we begin by describing our proof for non-adaptive security, while our ultimate goal is to prove security against adaptive adversaries. We assume that all conjunctive queries are given non-adaptively at a time to the simulator. Eventually we show this can be extended to similar conjunctive queries in an adaptive setting.

Non-adaptive Simulator Description. The security proof of OQXT (proof of theorem 1) can be elucidated in a similar tone as done is OXT. The first step towards this would be to justify that the leakage captured by $\mathcal{L}_{\mathrm{OQXT}}$ is atleast necessary for correct simulation. We briefly describe why each of the individual leakage components are necessary for a simulator to produce correct results. In order to simulate OQXT correctly each of the leakage components are critically analysed and their significance is justified. N or the total number of appearances of keywords in the database gives the size of the XSet, i.e. number of **xtag** entries for each keyword-id pair in the database. The equality pattern $\bar{s}$ is important as it indicates the queries with same stags. This is required since the stags are deterministic the server can observe repetition in stags of different queries. In order to know $\mathbf{DB}(\mathsf{w}_1)$ or the number of documents matching the sterm, the *size pattern* leakage component is important. This is equal to the number of tuples returned by the TSet. To enable the simulator to produce the final result of the query consistent with the conditional intersection pattern, the *result pattern leakage* is important. From the conditional intersection pattern the server can observe the queries in which the xterms repeat and that have a document identifier common in their stag. We give a simulator description in Algorithm 3. The proof of Theorem 1 is shown via hybrids in Section 4.2. The simulator S takes as input the leakage components as defined by $\mathcal{L}_{\mathrm{OQXT}}$ and $\mathcal{L}_T$ and produces the result pattern which is similar to the document identifiers returned by the original scheme on input of a non-adaptive conjunctive query. The routines $\mathsf{Initialize}$, $\mathsf{XSet.SetUp}$, $\mathsf{GenTrans}$ are elaborately explained in the detailed proof in Section 4.2.

Algorithm 3 Simulator for the Selective Security Proof of OQXT

function $\underline{\text{INITIALIZE}}(N, \bar{s}, \mathsf{SP}, \mathsf{RP}, \mathsf{IP}, \mathcal{L}_T(\mathbf{DB}, s), T[s])$
  for $\mathsf{w} \in \hat{x}$ and $\mathsf{id} \in \bigcup_{i=1} \mathsf{RP}[i]$ **do**
    $H[\mathsf{id}, \mathsf{w}] \xleftarrow{\$} \mathbb{Z}_q^{n \times n}$
  for $\mathsf{w} \in \bar{s}$ **do**
    $\mathsf{WPerms}[\mathsf{w}] \xleftarrow{\$} \mathsf{Perm}([\mathsf{SP}[i]])$
  for $\mathsf{w} \in \hat{x}$ **do**
    Sample a random matrix $\bar{\mathbf{A}} \in \mathbb{Z}_q^{n \times \bar{m}}$ and a tag $\mathbf{H} = \mathbf{I} \in \mathbb{Z}_q^{n \times n}$
    $K_e \xleftarrow{\$} \{0,1\}^\lambda$
    $\mathbf{tset} \leftarrow \perp$
    for For $c = 1, \ldots, T_w$ **do**
      $e_c \leftarrow Sym.Enc(K_e, 0^\lambda)$
      $\mathbf{xid} \leftarrow \mathbb{Z}_q^{n \times n}$
      $r \leftarrow f_Z(\mathsf{w}||c)$
      $\mathbf{z_w}, \mathbf{T} \leftarrow \mathsf{Gen\_Trapdoor}(\bar{\mathbf{A}}, \mathbf{H}, r)$
      $\mathbf{y_c^t} \leftarrow \mathsf{SampleD}(\mathbf{T}, \mathbf{z_w^t}, \mathbf{xid^t})$
      $\mathbf{tset}[c] \leftarrow (\mathbf{y_c^t}, e)$
    $(\mathsf{TSet}, \mathsf{STags}) \leftarrow \mathcal{S}(\mathcal{L}_\mathcal{T}(\mathbf{DB}, s), T[s])$
    $\mathsf{XSet} \leftarrow \mathsf{XSet.SetUp}(N, \mathsf{RP}, \mathcal{L}_\mathcal{T}(\mathbf{DB}, s), H)$
    $\mathbf{EDB} \leftarrow (\mathsf{TSet}, \mathsf{XSet})$
    for $i = 1, \ldots, Q$ **do**
      $\mathbf{tset}[i] \leftarrow \mathsf{GenTrans}(\mathsf{RP}, \mathsf{IP}, \mathsf{SP}, \bar{s}[i], H, \mathcal{L}_\mathcal{T}(\mathbf{DB}, s),$
      $\mathsf{STags}[i])$
  return Return $(\mathbf{EDB}, \mathbf{tset})$

function $\underline{\text{XSET.SETUP}}(N, \mathsf{RP}, \mathcal{L}_\mathcal{T}(\mathbf{DB}, s), H)$
  $\mathsf{XSet} \leftarrow \phi$; $j \leftarrow 0$
  for $\mathsf{w} \in \hat{x}$ and $\mathsf{id} \in \bigcup_{i:\hat{x}[i]=\mathsf{w}} \mathsf{RP}[i]$ **do**
    $\mathsf{XSet} \leftarrow \mathsf{XSet} \cup \{H[\mathsf{id}, \hat{x}[i]]\}$
    $j \leftarrow j + 1$
  for $i = j + 1, \ldots, N$ **do**
    $h \xleftarrow{\$} \mathbb{Z}_q^{n \times n}$
    $\mathsf{XSet} \leftarrow \mathsf{XSet} \cup \{h\}$
  return XSet

function $\underline{\text{GENTRANS}}(\mathsf{RP}, \mathsf{IP}, \mathsf{SP}, \bar{s}[i], H, \mathcal{L}_\mathcal{T}(\mathbf{DB}, s), \mathsf{stag})$
  $R = \mathsf{RP}[i] \cup \bigcup_{j=1}^{n} \mathsf{IP}[i, j]$
  $(\mathsf{id}_1, \ldots, \mathsf{id}_{T'}) \leftarrow R$
  for $k = T' + 1, \ldots, \mathsf{SP}[i]$ **do**
    $\perp \leftarrow \mathsf{id}_k$
    $R \leftarrow \mathsf{id}_k$
  $\mathbf{tset} \leftarrow \mathsf{TSetRetrieve}(\mathsf{TSet}, \mathsf{stag}[i])$
  $\sigma \leftarrow \mathsf{WPerms}[\bar{\mathsf{s}}[i]]$
  Sample a random matrix $\bar{\mathbf{A}} \in \mathbb{Z}_q^{n \times \bar{m}}$ and a tag
  $\mathbf{H} = \mathbf{I} \in \mathbb{Z}_q^{n \times n}$
  for For $c = 1, \ldots, \mathsf{SP}[i]$ **do**
    $r_x \leftarrow f_Z(\bar{\mathsf{s}}[i]||c)$
    $(\mathbf{y}, e) \leftarrow \mathbf{tset}[c]$
    if $\bar{\mathsf{id}}_{\sigma(c)} \neq \perp$ **then**
      $\mathbf{xtoken}, \mathbf{T_x} \leftarrow \mathsf{Gen\_Trapdoor}(\bar{\mathbf{A}}, \mathbf{H}, r_x)$ such that, $\mathbf{y_c^t} \cdot \mathbf{xtoken} = H[\bar{\mathsf{id}}_{\sigma(c)}, \hat{x}[i]]$
    else
      $\mathbf{xtoken}, \mathbf{T_x} \leftarrow \mathsf{Gen\_Trapdoor}(\bar{\mathbf{A}}, \mathbf{H}, r_x)$ such that, $\mathbf{y_c^t} \cdot \mathbf{xtoken} = H[\bar{\mathsf{id}}_{\sigma(c)}, \hat{x}[i]]$
  for $c = \mathsf{SP}[i] + 1, \ldots, T$ **do**
    $\mathbf{xtoken}[c] \xleftarrow{\$} \mathbb{Z}_q^{m \times n}$
  $\mathsf{Res} \leftarrow \mathsf{ServerSearch}(\mathbf{EDB}, (\mathsf{stag}, \mathbf{xtoken}))$
  return $((\mathsf{stag}, \mathbf{xtoken}), \mathsf{Res}, \mathsf{RP}[i])$

Extension to Adaptive Security. Our adaptive security proof for OQXT is also in the standard model, and is conceptually very similar to the selective security proof, hence we only provide a brief overview here. For the adaptive proof of security, we assume an adaptively secure instantiation of TSet in the standard model. While the original construction of TSet [CJJ+13] requires random oracles for adaptive security, the authors of [CJJ+13] also discuss an alternative instantiation of adaptively secure TSets in the standard model without incurring additional rounds of communication. In the context of our OQXT protocol, using the standard model instantiation of TSet increases the communication overhead for the TSet component, but not the (asymptotic) communication overhead for the overall search protocol.

The main crux of our adaptive security proof is that the simulator for OQXT initializes the XSet to consist entirely of uniformly random elements initially (while relying on the LWR assumption for indistinguishability of the real and simulated XSet entries). Additionally, the simulator for OQXT can directly invoke the simulator for the adaptively secure TSet to simulate the TSet entries at setup and the corresponding TSet tokens during searches. The simulator also uses the (adaptive) result pattern leakage and the (adaptive) conditional intersection pattern leakage to program the **xtoken** entries to be consistent with the adversarially issued queries.

4.2 Proof Of Theorem 1

For formal security proof, we rely on the following definition for non-adaptive security -

Definition 2. *Let* $\Pi = (\mathsf{Setup}, \mathsf{Search})$ *be an SSE scheme and let* $\mathcal{L}$ *be an algorithm. For efficient algorithms* $\mathcal{A}$ *and* $\mathcal{S}$*, we define experiments (algorithms)* $\mathbf{Real}^{\Pi}_{\mathcal{A}}(\lambda)$ *and* $\mathbf{Ideal}^{\Pi}_{\mathcal{A},\mathcal{S}}(\lambda)$ *as follows:*

- $\mathbf{Real}^{\Pi}_{\mathcal{A}}(\lambda)$*:* $\mathcal{A}(1^{\lambda})$ *chooses* **DB** *and a list of queries* q*. The experiment then runs* $(\mathsf{mk}, \mathbf{EDB}) \leftarrow \mathsf{Setup}(\mathbf{DB})$*. For each* $i \in |q|$*, it runs the* Search *protocol with client input* $(\mathsf{mk}, q[i])$ *and server input* **EDB** *and stores the transcript and the client's output in* $\mathbf{tset}[i]$*. Finally the game gives* **EDB** *and* **tset** *to* $\mathcal{A}$*, which returns a bit that the game uses as its own output.*
- $\mathbf{Ideal}^{\Pi}_{\mathcal{A},\mathcal{S}}(\lambda)$ *:* $\mathcal{A}(1^{\lambda})$ *chooses* **DB** *and a list of queries* q*. The experiment then runs* $\mathcal{S}(\mathcal{L}(\mathbf{DB}, q))$ *and gives its output to* $\mathcal{A}$*, which returns a bit that the game uses as its own output.*

Informally, an SSE scheme Π *is secure with respect to a leakage function* $\mathcal{L}$ *if the adversarial server provably learns no more information about* **DB** *other than that encapsulated by* $\mathcal{L}$*. Formally* Π *is* $\mathcal{L}$*-semantically-secure against non-adaptive attacks if for all stateful PPT adversaries* $\mathcal{A}$ *there exists a stateful probabilistic polynomial time simulator (algorithm)* $\mathcal{S} = (\mathcal{S}.\mathsf{Setup}, \mathcal{S}.\mathsf{Search})$ *such that the following holds -*

$$\Pr[\mathbf{Real}^{\Pi}_{\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{Ideal}^{\Pi}_{\mathcal{A},\mathcal{S}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$$

In OQXT the leakage function is defined by the $\mathcal{L}_{\text{OQXT}}$ along with the TSet leakage function $\mathcal{L}_T$ (as described in [CJJ+13]). The vector T is computed as follows given the input $\mathbf{DB}$ and $(\mathbf{s}, \mathbf{x})$ -

For $\mathsf{w} \in \Delta$ do
  $K \xleftarrow{\$} 0,1^\lambda; \mathbf{tset} \leftarrow \perp$
  For $c = 1, \ldots, T_{\mathsf{w}}$ do
    $\mathbf{y_c} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}; e_c \leftarrow Sym.Enc(K, 0^\lambda);$
    $\mathbf{tset}[c] \leftarrow (\mathbf{y}_{\mathsf{id}}, e_c)$
  $T[\mathsf{w}] \leftarrow \mathbf{tset}$

It outputs $(\mathcal{L}_{\text{OQXT}}(\mathbf{DB}, \mathbf{s}, \mathbf{x}), \mathcal{L}_T(T, s), T[s])$, where $T[s] = (T[s[1]], \ldots, T[s[Q]])$.

Theorem 2. *Let $\mathcal{L}$ be the leakage function as defined above. The SSE scheme* OQXT *is $\mathcal{L}$-post-quantum secure against non-adaptive attacks where all queries are 2-conjunctions, assuming that the $\mathsf{LWR}_{n,p,q}$ problem is hard in $\mathbb{Z}_q^n \times \mathbb{Z}_p$, that solving $\mathsf{SIS}_{n,m,q}$ instance in $\mathbb{Z}_q^{n \times m}$ is hard i.e. sampling a short pre-image from a specific coset of a discrete Gaussian distribution without using a lattice trapdoor $\mathbf{T}$ is hard, that F and F_q are secure PRFs, that (Enc, Dec) is an IND-CPA secure symmetric encryption scheme and that Σ is a (non-adaptively) $\mathcal{L}_T$-secure and computationally correct TSet instantiation.*

Proof. We construct the proof using various games $G_0, G_1, ...$ In each game, $\mathcal{A}$ starts by supplying $(\mathbf{DB}, \mathbf{q})$, which is then given to an $\mathsf{Initialize}$ routine that produces an output, which is given to $\mathcal{A}$ who then outputs a bit that becomes the game output. Game G_0 is designed to generate exactly the same distribution as $\mathbf{Real}_{\mathcal{A}}^{\mathbf{\Pi}}(\lambda)$ (assuming no false positives occur) and the final game is structured so that it is easy to simulate exactly given the leakage profile instead of the actual $(\mathbf{DB}, \mathbf{q})$ input. By relating the games we can argue that the final simulator satisfies Definition 2 with OQXT, completing the proof.

Game G_0. The first game G_0 is an implementation of the real game. The game starts by running the $\mathsf{Initialize}$ routine, which passes $(\mathbf{DB}, \mathbf{s}, \mathbf{x})$ from $\mathcal{A}$ and simulates OQXT.Setup, with some minor changes. While building T vector, it records the permutations σ in a vector WPerms indexed by keywords. The game computes an array Stags of all of the stag values used. For each $i = 1, \ldots, Q$ it lets $\mathsf{Stags}[i] \leftarrow \mathsf{TSetGetTag}(K_T, s[i])$. To compute the transcript array $\mathbf{tset}$, for $i = 1, \ldots, Q$ it sets $\mathbf{tset}[i]$ to the output of $\mathsf{GenTrans}(\mathbf{EDB}, K_X, K_Z, s[i], x[i], \mathsf{STags}[i])$, which is defined in game G_0. As employed in OXT we use a subroutine $\mathsf{ServerSearch}$ which we take to be the server's computation as defined in OQXT in response to the first client message. The $\mathsf{GenTrans}$ routine generates the transcripts as in the real game but it computes the final result set $\mathsf{ResInds}$, in a different way i.e., instead of decrypting the ciphertext it computes the $\mathbf{DB}(s[i])$ and finds the common ids between $\mathbf{DB}(s[i])$ and $\mathbf{DB}(x[i])$. Assuming F_p is a secure PRF it can be stated -

$$\Pr[G_0 = 1] \leq \Pr[\mathbf{Real}_{\mathcal{A}}^{\Pi}(\lambda) = 1] + \mathsf{negl}(\lambda)$$

Game G_1. This game is exactly same as G_0 the only difference is we replace every evaluation of PRF $F(K_S, \cdot), F(K_X, \cdot), F(K_I, \cdot), F(K_Z, \cdot)$ with independent random functions with appropriate domain and range. The keys are chosen uniformly at random. By the security of PRF and using some standard hybrid argument it can be shown that there exists

$\underline{\mathsf{Initialize}(\mathbf{DB}, s, x)}$ // G_0

$K_S, K_I, K_X, K_Z \xleftarrow{\$} \{0,1\}^\lambda$
$(\mathsf{id}_i, \mathcal{W}_i)_{i=1}^d \leftarrow \mathbf{DB}$
For $\mathsf{w} \in \mathcal{W}$
  $(\mathsf{id}_1, \ldots, \mathsf{id}_{T_\mathsf{w}}) \leftarrow \mathbf{DB}(\mathsf{w})$
  $\sigma \leftarrow \mathsf{Perm}([T_\mathsf{w}]); \mathsf{WPerms}[\mathsf{w}] \leftarrow \sigma$
  Sample a random matrix $\bar{\mathbf{A}} \in \mathbb{Z}_q^{n \times \bar{m}}$ and a tag
  $\mathbf{H} = \mathbf{I} \in \mathbb{Z}_q^{n \times n}$
  $K_e \leftarrow F(K_S, \mathsf{w}); \mathbf{tset} \leftarrow \perp$
  For $c = 1, \ldots, T_\mathsf{w}$ do
    $e_c \leftarrow Sym.Enc(K_e, \mathsf{id}_{\sigma(c)}); \mathbf{xid} \leftarrow F_q(K_I, \mathsf{id}_{\sigma(c)})$
    $r \leftarrow F'_q(K_Z, \mathsf{w}||c)$; $\mathbf{z_w}, \mathbf{T} \leftarrow \mathsf{Gen\_Trapdoor}(\bar{\mathbf{A}}, \mathbf{H}, r)$
    $\mathbf{y_c^t} \leftarrow \mathsf{SampleD}(\mathbf{T}, \mathbf{z_w^t}, \mathbf{xid^t})$
    $\mathbf{tset}[c] \leftarrow (\mathbf{y_c^t}, e_c)$
  End
  $T[w] \leftarrow \mathbf{tset}$
End
$(\mathsf{TSet}, K_T) \leftarrow \mathsf{TSet.SetUp}(T)$
For $i = 1, \ldots, Q$ do
  $\mathsf{STags}[i] \leftarrow \mathsf{TSet.GetTag}(K_T, s[i])$
$\mathsf{XSet} \leftarrow \mathsf{XSet.SetUp}(K_X, K_I, \mathbf{DB})$
$\mathbf{EDB} \leftarrow (\mathsf{TSet}, \mathsf{XSet})$
For $i = 1, \ldots, Q$ do
  $t[i] \leftarrow \mathsf{GenTrans}(K_X, K_Z, s[i], x[i], \mathsf{STags}[i])$
Return $(\mathbf{EDB}, \mathbf{tset})$

$\underline{\mathsf{XSet.SetUp}(K_X, K_I, \mathbf{DB})}$ // G_0

$(\mathsf{id}_i, \mathcal{W}_i)_{i=1}^d \leftarrow \mathbf{DB}$
For $\mathsf{w} \in \mathcal{W}$ and $\mathsf{id} \in \mathbf{DB}(\mathsf{w})$ do
  $\mathbf{xw} \leftarrow F_q(K_X, \mathsf{w})$
  $\mathbf{xid} \leftarrow F_q(K_I, \mathsf{id})$
  $\mathbf{xtag} \leftarrow \lfloor \mathbf{xid} \cdot \mathbf{xw} \rceil_p$
  $\mathsf{XSet} \leftarrow \mathsf{XSet} \cup \{\mathbf{xtag}\}$
End
Return XSet

$\underline{\mathsf{GenTrans}(\mathbf{EDB}, K_X, K_Z, s, x, \mathsf{stag})}$ // G_0

$\mathbf{xw} \leftarrow F_q(K_X, \mathsf{w})$
Sample a random matrix $\bar{\mathbf{A}} \in \mathbb{Z}_q^{n \times \bar{m}}$ and a tag
$\mathbf{H} = \mathbf{I} \in \mathbb{Z}_q^{n \times n}$
For $c = 1, \ldots, T$ do
  $r \leftarrow F'_q(K_Z, s||c)$
  $\mathbf{z_s}, \mathbf{T} \leftarrow \mathsf{Gen\_Trapdoor}(\bar{\mathbf{A}}, \mathbf{H}, r)$
  $\mathbf{xtoken}[c] \leftarrow \lfloor \mathbf{z_s} \cdot \mathbf{xw} \rceil_p$
$\mathsf{Res} \leftarrow \mathsf{Server.Search}(\mathbf{EDB}, (\mathsf{stag}, \mathbf{xtoken}))$
$\mathsf{ResInds} \leftarrow \mathbf{DB}(s) \cap \mathbf{DB}(x)$
Return $((\mathsf{stag}, \mathbf{xtoken}), \mathsf{Res}.\ \mathsf{ResInds})$

Table 2: Game G_0

unbounded efficient adversaries $B_{1,1}, B_{1,2}, B_{1,3}$ such that -

$$\Pr[G_1 = 1] - \Pr[G_0 = 1] \leq \mathbf{Adv}^{prf}_{F,B_{1,1}}(\lambda) + 3 \cdot \mathbf{Adv}^{prf}_{F_p,B_{1,2}}(\lambda) + \mathbf{Adv}^{prp}_{P,B_{1,3}}(\lambda)$$

Game G_2. In this game we modify G_1 to include the boxed code. The modification is done in the Initialize routine, where every ciphertext e_c is always generated as an encryption of 0^λ (with the same key). Similar to OXT we can claim here that there exists an unbounded efficient adversary B_2 such that -

$$\Pr[G_2 = 1] - \Pr[G_1 = 1] \leq m \cdot \mathbf{Adv}^{IND-CPA}_{\Sigma,B_2}(\lambda)$$

Game G_3.In G_3 the XSet values are pre-computed along with the **xtoken** values. In Initialize subroutine, two arrays H, Y are computed and used in XSet.SetUp and GenTrans. H is indexed by an identifier and a keyword from $\mathcal{W}$ and contains elements from $\mathbb{Z}_q^{n \times n}$. The elements of H are computed as $\lfloor \mathbf{xid} \cdot \mathbf{xw} \rceil_p$, where **xid** and **xw** are output of a random function with range in $\mathbb{Z}_q^{n \times n}$. Basically elements from H is used as **xtag** in XSet.SetUp. Y is indexed by a keyword and each i in $|\mathbf{DB}(\mathsf{w})|$. It is filled with elements from $\mathbb{Z}_q^{n \times m}$.

$\underline{\mathsf{Initialize}(\mathbf{DB}, s, x)}$ // G_1, $\boxed{G_2}$

$f_I, f_X \leftarrow \mathsf{Fun}(\{0,1\}^\lambda, \mathbb{Z}_q^{n\times n})$
$f_Z \leftarrow \mathsf{Fun}(\{0,1\}^\lambda, \mathbb{Z}_q^{n})$
$(\mathsf{id}_i, \mathcal{W}_i)_{i=1}^d \leftarrow \mathbf{DB}$
For $\mathsf{w} \in \mathcal{W}$
  $(\mathsf{id}_1, \ldots, \mathsf{id}_{T_\mathsf{w}}) \leftarrow \mathbf{DB}(\mathsf{w}))$
  $\sigma \leftarrow \mathsf{Perm}([T_\mathsf{w}]); \mathsf{WPerms}[\mathsf{w}] \leftarrow \sigma$
  Sample a random matrix $\bar{\mathbf{A}} \in \mathbb{Z}_q^{n\times \bar{m}}$ and a tag
  $\mathbf{H} = \mathbf{I} \in \mathbb{Z}_q^{n\times n}$
  $K_e \leftarrow F(K_S, \mathsf{w}); \mathbf{tset} \leftarrow \perp$
  For $c = 1, \ldots, T_\mathsf{w}$ do
    $e_c \leftarrow Sym.Enc(K_e, \mathsf{id}_{\sigma(c)})$
    $\boxed{e_c \leftarrow Sym.Enc(K_e, 0^\lambda)}$
    $\mathbf{xid} \leftarrow f_I(\mathsf{id}_{\sigma(c)})$
    $r \leftarrow f_Z(\mathsf{w}||c)$; $\mathbf{z_w}, \mathbf{T} \leftarrow \mathsf{Gen\_Trapdoor}(\bar{\mathbf{A}}, \mathbf{H}, r)$
    $\mathbf{y_c^t} \leftarrow \mathsf{SampleD}(\mathbf{T}, \mathbf{z_w^t}, \mathbf{xid^t})$
    $\mathbf{tset}[c] \leftarrow (\mathbf{y_c^t}, e_c)$
  End
  $T[w] \leftarrow \mathbf{tset}$
End
$(\mathsf{TSet}, K_T) \leftarrow \mathsf{TSet.SetUp}(T)$
For $i = 1, \ldots, Q$ do
  $\mathsf{STags}[i] \leftarrow \mathsf{TSet.GetTag}(K_T, s[i])$
$\mathsf{XSet} \leftarrow \mathsf{XSet.SetUp}(f_X, f_I, \mathbf{DB})$
$\mathbf{EDB} \leftarrow (\mathsf{TSet}, \mathsf{XSet})$
For $i = 1, \ldots, Q$ do
  $\mathbf{tset}[i] \leftarrow \mathsf{GenTrans}(f_X, f_Z, s[i], x[i], \mathsf{STags}[i])$
Return $(\mathbf{EDB}, \mathbf{tset})$

$\underline{\mathsf{XSet.SetUp}(f_X, f_I, \mathbf{DB})}$ // G_1, G_2

$(\mathsf{id}_i, \mathcal{W}_i)_{i=1}^d \leftarrow \mathbf{DB}$
$\mathsf{XSet} \leftarrow \phi$
For $\mathsf{w} \in \mathcal{W}$ and $\mathsf{id} \in \mathbf{DB}(\mathsf{w})$ do
  $\mathbf{xw} \leftarrow f_X(\mathsf{w})$
  $\mathbf{xid} \leftarrow f_I(\mathsf{id})$
  $\mathbf{xtag} \leftarrow \lfloor \mathbf{xid} \cdot \mathbf{xw} \rceil_p$
  $\mathsf{XSet} \leftarrow \mathsf{XSet} \cup \{\mathbf{xtag}\}$
End
Return XSet

$\underline{\mathsf{GenTrans}(\mathbf{EDB}, f_X, f_Z, s, x, \mathsf{stag})}$ // G_1, G_2

$\mathbf{xw} \leftarrow f_X(\mathsf{w})$
  Sample a random matrix $\bar{\mathbf{A}} \in \mathbb{Z}_q^{n\times \bar{m}}$ and a tag
  $\mathbf{H} = \mathbf{I} \in \mathbb{Z}_q^{n\times n}$
For $c = 1, \ldots, T$ do
  $r \leftarrow f_Z(s||c)$
  $\mathbf{z_s}, \mathbf{T} \leftarrow \mathsf{Gen\_Trapdoor}(\bar{\mathbf{A}}, \mathbf{H}, r)$
  $\mathbf{xtoken}[c] \leftarrow \lfloor \mathbf{z_s} \cdot \mathbf{xw} \rceil_p$
$\mathsf{Res} \leftarrow \mathsf{Server.Search}(\mathbf{EDB}, (\mathsf{stag}, \mathbf{xtoken}))$
$\mathsf{ResInds} \leftarrow \mathbf{DB}(s) \cap \mathbf{DB}(x)$
Return $((\mathsf{stag}, \mathbf{xtoken}), \mathsf{Res}, \mathsf{ResInds})$

Table 3: Games G_1 and G_2

$\underline{\mathsf{Initialize}(\mathbf{DB}, s, x)}$ // G_3, $\boxed{G_4}$

$f_I, f_X \leftarrow \mathsf{Fun}(\{0,1\}^\lambda, \mathbb{Z}_q^{n\times n})$
$f_Z \leftarrow \mathsf{Fun}(\{0,1\}^\lambda, \mathbb{Z}_q^{n})$
$(\mathsf{id}_i, \mathcal{W}_i)_{i=1}^d \leftarrow \mathbf{DB}$
For each w and each id_i do
  $\mathbf{xw} \leftarrow f_X(\mathsf{w})$
  $\mathbf{xid} \leftarrow f_I(\mathsf{id})$
  $H[\mathsf{id}_i, \mathsf{w}] \leftarrow \lfloor \mathbf{xid} \cdot \mathbf{xw} \rceil_p$
  $\boxed{H[\mathsf{id}_i, \mathsf{w}] \xleftarrow{\$} \mathbb{Z}_q^{n\times n}}$
End
For $\mathsf{w} \in \mathcal{W}$
  $(\mathsf{id}_1, \ldots, \mathsf{id}_{T_\mathsf{w}}) \leftarrow \mathbf{DB}(\mathsf{w}))$
  $\sigma \leftarrow \mathsf{Perm}([T_\mathsf{w}])$; $\mathsf{WPerms}[\mathsf{w}] \leftarrow \sigma$
  Sample a random matrix $\bar{\mathbf{A}} \in \mathbb{Z}_q^{n\times \bar{m}}$ and a tag
  $\mathbf{H} = \mathbf{I} \in \mathbb{Z}_q^{n\times n}$
  $K_e \leftarrow F(K_S, \mathsf{w})$; $\mathbf{tset} \leftarrow \perp$
  For $c = 1, \ldots, T_\mathsf{w}$ do
    $e_c \leftarrow Sym.Enc(K_e, 0^\lambda)$
    $\mathbf{xid} \leftarrow f_I(\mathsf{id}_{\sigma(c)})$
    $r \leftarrow f_Z(\mathsf{w}||c)$; $\mathbf{z_w}, \mathbf{T} \leftarrow \mathsf{Gen\_Trapdoor}(\bar{\mathbf{A}}, \mathbf{H}, r)$
    $\mathbf{y_c^t} \leftarrow \mathsf{SampleD}(\mathbf{T}, \mathbf{z_w^t}, \mathbf{xid^t})$
    $\mathbf{tset}[c] \leftarrow (\mathbf{y_c^t}, e_c)$
  End
  $T[w] \leftarrow \mathbf{tset}$
  For $u \in \mathcal{W} \setminus \mathsf{w}$ do
    $\mathsf{xu} \leftarrow f_X(u)$
    $r \leftarrow f_Z(\mathsf{w}||c)$;
    $\mathbf{z_w}, \mathbf{T} \leftarrow \mathsf{Gen\_Trapdoor}(\bar{\mathbf{A}}, \mathbf{H}, r)$
    $Y[w, u, c] \leftarrow \lfloor \mathbf{z_w} \cdot \mathsf{xu} \rceil_p$
    $\boxed{Y[w, u, c] \xleftarrow{\$} \mathbb{Z}_q^{m\times n}}$
  End
  For $c = T_w + 1, \ldots, T$ do
End
$(\mathsf{TSet}, K_T) \leftarrow \mathsf{TSet.SetUp}(T)$
For $i = 1, \ldots, Q$ do
  $\mathsf{STags}[i] \leftarrow \mathsf{TSet.GetTag}(K_T, s[i])$
$\mathsf{XSet} \leftarrow \mathsf{XSet.SetUp}(\mathbf{DB}, H)$
$\mathbf{EDB} \leftarrow (\mathsf{TSet}, \mathsf{XSet})$
For $i = 1, \ldots, Q$ do
  $\mathbf{tset}[i] \leftarrow \mathsf{GenTrans}(\mathbf{DB}, \mathbf{EDB}, H, s[i], x[i], \mathsf{STags}[i])$
Return $(\mathbf{EDB}, \mathbf{tset})$

$\underline{\mathsf{XSet.SetUp}(\mathbf{DB}, H)}$ // G_3, G_4

$(\mathsf{id}_i, \mathcal{W}_i)_{i=1}^d \leftarrow \mathbf{DB}$
$\mathsf{XSet} \leftarrow \phi$
For $\mathsf{w} \in \mathcal{W}$ and $\mathsf{id} \in \mathbf{DB}(\mathsf{w})$ do
  $\mathsf{XSet} \leftarrow \mathsf{XSet} \cup \{H[\mathsf{id}, \mathsf{w}]\}$
Return XSet

$\underline{\mathsf{GenTrans}(\mathbf{EDB}, f_X, f_Z, s, x, \mathsf{stag})}$ // G_3, G_4

$t \leftarrow \mathsf{TSet.Retrieve}(\mathsf{TSet}, \mathsf{stag})$
$(\mathsf{id}_1, \ldots, \mathsf{id}_{T_s}) \leftarrow \mathbf{DB}(s)$; $\sigma \leftarrow \mathsf{WPerms}[s]$
Sample a random matrix $\bar{\mathbf{A}} \in \mathbb{Z}_q^{n\times \bar{m}}$ and a tag
$\mathbf{H} = \mathbf{I} \in \mathbb{Z}_q^{n\times n}$
For $c = 1, \ldots, T_s$ do
  $(\mathbf{y_c^t}, e) \leftarrow \mathbf{tset}[c]$;
  $r_x \leftarrow f_Z(s||c)$;
  $\mathbf{xtoken}, \mathbf{T_x} \leftarrow \mathsf{Gen\_Trapdoor}(\bar{\mathbf{A}}, \mathbf{H}, r_x)$ such that
  $\mathbf{y_c^t} \cdot \mathbf{xtoken} = H[\mathsf{id}_{\sigma(c)}, x]$
For $c = T_s + 1, \ldots, T$ do
  $\mathbf{xtoken}[c] \leftarrow Y[s, x, c]$
$\mathsf{Res} \leftarrow \mathsf{Server.Search}(\mathbf{EDB}, (\mathsf{stag}, \mathbf{xtoken}))$
$\mathsf{ResInds} \leftarrow \mathbf{DB}(s) \cap \mathbf{DB}(x)$
Return $((\mathsf{stag}, \mathbf{xtoken}), \mathsf{Res}, \mathsf{ResInds})$

Table 4: Games G_3 and G_4

Both H and Y are used in $\mathsf{GenTrans}$. For each $c \in |T|$ in G_2 the $\mathsf{GenTrans}$ routine generates **xtoken** values as $\lfloor \mathbf{z_w} \cdot \mathbf{xw} \rceil_p$, where $\mathbf{z_w}$ is a matrix with trapdoor $\mathbf{T}$ generated by $\mathsf{Gen\_Trapdoor}$. In G_3 the $\mathsf{GenTrans}$ function, uses $\sigma \leftarrow \mathsf{WPerms}[s], \mathbf{DB}(s) = (\bar{\mathsf{id}}_1, \ldots, \bar{\mathsf{id}}_{T_s})$ and **tset**. In $\mathsf{GenTrans}$ for $c = 1, \ldots, T_s$,

$$H[\mathsf{id}_{\sigma(c)}, x] = \mathbf{y_c^t} \cdot \mathbf{xtoken}$$

and for $c = T_{s+1}, \ldots, T$ **xtoken** is set as -

$$\mathbf{xtoken}[c] \leftarrow Y[s, x, c]$$

this is exactly similar to the **xtoken** value as in G_2. Therefore we have,

$$\Pr[G_3 = 1] = \Pr[G_2 = 1]$$

Game G_4. In game G_4 all steps are similar to game G_3 except the boxed code is included. We claim that

$$Pr[G_4 = 1] = Pr[G_3 = 1]$$

The above claim holds as the random, function f_Z for sampling a random value r is chosen only once in $\mathsf{Initialize}$ routine and not used later in any of the other routines. Moreover, for any $\mathsf{w} \in \mathcal{W}$ and $c = 1, \ldots, T_{\mathsf{w}}$, the value of $r = f_Z(\mathsf{w})||c$ is uniform and independent of the rest of the randomness in the game. Thus the value $\lfloor \mathbf{z_w} \cdot \mathsf{xu} \rceil_p$ is also uniform and independent of the rest of the game, which makes our above claim valid. In this game the values of H and Y arrays are independently chosen from $\mathbb{Z}_q^{n \times n}$ and $\mathbb{Z}_q^{m \times n}$ respectively. We claim that by the hardness of $\mathsf{LWR}_{n,p,q}$ Game G_3 and G_4 are indistinguishable.

Game G_5. The $\mathsf{Initialize}$ code for game G_5 is described in Table 4 and the other routines remain same as in game G_4. In G_6 soome irrelevant codes like selecting random functions are removed and TSet is generated using a simulator, which by the security guarantees of TSet is expected to exist. We replace all the call to TSet.SetUp and loop generating STags with a call to the simulator $\mathcal{S}$ on input $(\mathcal{L}_{\mathcal{T}}(\mathbf{DB}, s), \mathbf{T}[s])$. By the assumed $\mathcal{L}_{\mathcal{T}}$-security of TSet instantiations we can say that -

$$\Pr[G_6 = 1] - \Pr[G_5 = 1] \leq \mathsf{negl}(\lambda)$$

Game G_6. In this game the way in which array H is accessed in altered so as to aid the final simulator to work with the given leakages. Whenever the game access the H array at index (id, x) it checks intuitively whether the game will ever access this location again in future. If yes, then the value in the corresponding index is used and if not then the value returned by accessing the corresponding index of H array is replaced by some random value. After H array is generated it is used in $\mathsf{XSet.SetUp}$ and $\mathsf{GenTrans}$. The routine $\mathsf{XSet.SetUp}$ will never repeat an access to H therefore for a particular index (id, x) it needs to check whether $\mathsf{GenTrans}$ will read the same location or not. But if we carefully observe we see $\mathsf{GenTrans}$ will read positions such that $\mathsf{id} \in \mathbf{DB}(s[i])$ and $x = x[i]$ for some i. This is exactly same as done in $\mathsf{XSet.SetUp}$ routine of game G_6. From this observation we can say -

$$\Pr[G_6 = 1] = \Pr[G_5 = 1]$$

$\underline{\mathsf{Initialize}(\mathbf{DB}, s, x)}$ // G_5, G_6, G_7

$(\mathsf{id}_i, \mathcal{W}_i)_{i=1}^d \leftarrow \mathbf{DB}$
For $\mathsf{w} \in \mathcal{W}$ and $i \in [d]$ do $H[\mathsf{id}_i, \mathsf{w}] \xleftarrow{\$} \mathbb{Z}_q^{n \times n}$
For $\mathsf{w} \in s$ do $\mathsf{WPerms}[\mathsf{w}] \xleftarrow{\$} \mathsf{Perm}([T_s])$
For $\mathsf{w} \in \mathcal{W}$ do
  Sample a random matrix $\bar{\mathbf{A}} \in \mathbb{Z}_q^{n \times \bar{m}}$ and a tag $\mathbf{H} = \mathbf{I} \in \mathbb{Z}_q^{n \times n}$
  $K_e \xleftarrow{\$} \{0,1\}^\lambda$; $\mathbf{tset} \leftarrow \perp$
  For $c = 1, \ldots, T_w$ do
    $e_c \leftarrow Sym.Enc(K_e, 0^\lambda)$
    $\mathbf{xid} \leftarrow \mathbb{Z}_q^{n \times n}$
    $r \leftarrow f_Z(\mathsf{w}||c)$; $\mathbf{z_w}, \mathbf{T} \leftarrow \mathsf{Gen\_Trapdoor}(\bar{\mathbf{A}}, \mathbf{H}, r)$
    $\mathbf{y_c^t} \leftarrow \mathsf{SampleD}(\mathbf{T}, \mathbf{z_w^t}, \mathbf{xid^t})$
    $\mathbf{tset}[c] \leftarrow (\mathbf{y_c^t}, e_c)$
  End
  $T[\mathsf{w}] \leftarrow \mathbf{tset}$
End
$(\mathsf{TSet}, \mathsf{STags}) \leftarrow \mathcal{S}(\mathcal{L}_{\mathcal{T}}(\mathbf{DB}, s), T[s])$
$\mathsf{XSet} \leftarrow \mathsf{XSet.SetUp}(\mathbf{DB}, \mathsf{H})$
$\mathbf{EDB} \leftarrow (\mathsf{TSet}, \mathsf{XSet})$
For $i = 1, \ldots, Q$ do
  $\mathbf{tset}[i] \leftarrow \mathsf{GenTrans}(\mathbf{DB}, \mathbf{EDB}, H, s[i], x[i], \mathsf{STags}[i])$
End
Return $(\mathbf{EDB}, \mathbf{tset})$

$\underline{\mathsf{XSet.SetUp}(\mathbf{DB}, H)}$ // G_6, G_7
$\mathsf{XSet} \leftarrow \phi$
For $\mathsf{w} \in \mathcal{W}$ and $\mathsf{id} \in \mathbf{DB}(\mathsf{w})$ do
  If $\exists i : \mathsf{id} \in \mathbf{DB}(s[i]) \wedge x[i] = w$ then
    $\mathsf{XSet} \leftarrow \mathsf{XSet} \cup \{H[\mathsf{id}, \mathsf{w}]\}$
  Else $h \xleftarrow{\$} \mathbb{Z}_q^{n \times n}$; $\mathsf{XSet} \leftarrow \mathsf{XSet} \cup \{h\}$
End
Return XSet

$\underline{\mathsf{GenTrans}(\mathbf{DB}, \mathbf{EDB}, H, s, x, \mathsf{stag}, i)}$ // G_7

$\mathbf{tset} \leftarrow \mathsf{TSet.Retrieve}(\mathsf{TSet}, \mathsf{stag})$
$(\mathsf{id}_1, \ldots, \mathsf{id}_{T_s}) \leftarrow \mathbf{DB}(s)$; $\sigma \leftarrow \mathsf{WPerms[s]}$
Sample a random matrix $\bar{\mathbf{A}} \in \mathbb{Z}_q^{n \times \bar{m}}$ and a tag $\mathbf{H} = \mathbf{I} \in \mathbb{Z}_q^{n \times n}$
For $c = 1, \ldots, T_s$ do
  $(\mathbf{y_c^t}, e) \leftarrow \mathbf{tset}[c]$;
  $r_x \leftarrow f_Z(s||c)$;
  If $\mathsf{id}_{\sigma(c)} \in \mathbf{DB}(s) \cap \mathbf{DB}(x)$ then
    $\mathbf{xtoken}, \mathbf{T_x} \leftarrow \mathsf{Gen\_Trapdoor}(\bar{\mathbf{A}}, \mathbf{H}, r_x)$ such that,
    $\mathbf{y_c^t} \cdot \mathbf{xtoken} = H[\mathsf{id}_{\sigma(c)}, x]$
  ElseIf $\exists j \neq i : \mathsf{id}_{\sigma(c)} \in \mathbf{DB}(s[j]) \wedge x[j] = x$ then
    $\mathbf{xtoken}, \mathbf{T_x} \leftarrow \mathsf{Gen\_Trapdoor}(\bar{\mathbf{A}}, \mathbf{H}, r_x)$ such that,
    $\mathbf{y_c^t} \cdot \mathbf{xtoken} = H[\mathsf{id}_{\sigma(c)}, x]$
  Else
    $\mathbf{xtoken}, \mathbf{T_x} \leftarrow \mathsf{Gen\_Trapdoor}(\bar{\mathbf{A}}, \mathbf{H}, r_x)$ such that,
    $\mathbf{y_c^t} \cdot \mathbf{xtoken} = H[\mathsf{id}_{\sigma(c)}, x]$
End
For $c = T_s + 1, \ldots, T$ do
  $\mathbf{xtoken}[c] \xleftarrow{\$} \mathbb{Z}_q^{m \times n}$
$\mathsf{Res} \leftarrow \mathsf{Server.Search}(\mathbf{EDB}, (\mathsf{stag}, \mathbf{xtoken}))$
$\mathsf{ResInds} \leftarrow \mathbf{DB}(s) \cap \mathbf{DB}(x)$
Return $((\mathsf{stag}, \mathbf{xtoken}), \mathsf{Res}, \mathsf{ResInds})$

Table 5: Games G_5, G_6 and G_7

Game G_7. In this game we change the way GenTrans accesses array H. To check for a possible repeated access it should check if either XSet.SetUp reads the particular index or if GenTrans reads the same index again. The first test is done by checking the XSet.SetUp access only those index positions $H[\mathsf{id}, \mathsf{w}]$ that satisfy the condition $\mathsf{id} \in \mathbf{DB}(s[i]) \cap \mathbf{DB}(x[i])$. Next it checks if there exists an index of H that is accessed by GenTrans twice. For an index (id, x) to be accessed twice , either of the following condition must be satisfied $\mathsf{id} \in \mathbf{DB}(s[i]) \cap \mathbf{DB}(s[j])$ for some $i \neq j$ or $x[i] = x[j]$ for some $i \neq j$. An argument similar to the one for the previous game transition gives -

$$\Pr[G_7 = 1] = \Pr[G_6 = 1]$$

Simulator. The simulator $\mathcal{S}$ uses its leakage to compute the same distribution as G_7. $\mathcal{S}$ takes as input $\mathcal{L}(\mathbf{DB}$, s,x), which consists of all the leakages encapsulated by $\mathcal{L}_{\mathrm{OQXT}}$. Therefore, input to the simulator include $N, \bar{s}$, SP, RP, IP, $\mathcal{L}_T(\mathbf{DB}, s), \mathbf{T}[s]$. The output of the simulator is a simulated $\mathbf{EDB}=$ (TSet, XSet) and the transcript array **tset**. The simulator will first compute the *restricted equality pattern* of x, denoted as $\hat{x}$. Basically $\hat{x}$ denotes which xterms are known to be "equal" by the server.

The simulator works as follows - On input $N, \bar{s}$, SP, RP, IP, $\mathcal{L}_T(\mathbf{DB}, s), \mathbf{T}[s]$, it computes $\hat{x}$ in a very similar way as shown in OXT. Next it follows the following steps to produce $\mathbf{EDB}=$ (TSet, XSet) -

1. For $\mathsf{w} \in \hat{x}$ and $\mathsf{id} \in \bigcup_{i=1} \mathsf{RP}[i]$ do $H[\mathsf{id}, \mathsf{w}] \xleftarrow{\$} \mathbb{Z}_q^{n \times n}$
2. For $\mathsf{w} \in \bar{s}$ do $\mathsf{WPerms}[\mathsf{w}] \xleftarrow{\$} \mathsf{Perm}([\mathsf{SP}[\mathrm{i}]])$.
3. $(\mathsf{TSet}, \mathsf{STags}) \leftarrow \mathcal{S}(\mathcal{L}_T(\mathbf{DB}, s), \mathbf{T}[\mathbf{s}])$
4. $\mathsf{XSet} \leftarrow \phi$; $j \leftarrow 0$
5. For $\mathsf{w} \in \hat{x}$ and $\mathsf{id} \in \bigcup_{i:\hat{x}=\mathsf{w}} \mathsf{RP}[i]$ do $\mathsf{XSet} \leftarrow \mathsf{XSet} \cup \{H[\mathsf{id}, \hat{x}[i]]\}$; $j \leftarrow j + 1$
6. For $i = j + 1, \ldots, N$ do $h \xleftarrow{\$} \mathbb{Z}_q^{n \times n}$; $\mathsf{XSet} \leftarrow \mathsf{XSet} \cup \{h\}$

The transcript array **tset** is computed by $\mathcal{S}$ by computing the revealed document-identifiers for the corresponding query as $R = \mathsf{RP}[i] \cup \bigcup_{j=1}^{Q} \mathsf{IP}[i, j]$. The document identifiers are placed in R in canonical order as $(\bar{\mathsf{id}}_1, \ldots, \bar{\mathsf{id}}_{T'})$. The simulator pads $|R|$ upto $|\mathsf{SP}[i]|$ by setting $\bar{\mathsf{id}}_k = \perp$ for $k = T' + 1, \ldots, \mathsf{SP}[i]$. It computes the transcript array as $\mathbf{tset} \leftarrow \mathsf{TSet.Retrieve}(\mathsf{TSet}, \mathsf{stag}[i])$; $\sigma \leftarrow \mathsf{WPerms}[\bar{s}[i]]$. Using this it computes $((\mathsf{stag}, \mathbf{xtoken}), \mathsf{Res}, \mathsf{RP}[i])$ as shown in Algorithm 3. On the basis of the similar argument as shown in OXT we can claim that the output of $\mathcal{S}$ has the same distribution as the output of the Initialize routine of G_7.

4.3 Discussion on the Leakage Profile of OQXT

In this section, we present additional discussion on the leakage of OQXT. We note here that OQXT is an *index-only* SSE scheme (following the vast majority of the SSE literature

today [CGKO06, CJJ+13, CJJ+14]), and we leave it as an interesting direction of future research to extend it into an end-to-end SSE system with system-wide leakage suppression [GPPW20, GPP23].

Output Leakage. The output leakage or result-pattern leakage (RP) is incurred by a significant number of existing SSE schemes supporting both single keyword or conjunctive keyword queries [CGKO06, CJJ+13, LPS+18]. This is considered as an acceptable leakage in literature. RP essentially reveals the indices of documents matching a particular query. This evidently reveals the number of documents returned as a result of a query. Since OQXT inherits the exact leakage profile of OXT [CJJ+13], it also incurs result pattern leakage and hence reveals the number of documents returned as the result of a conjunctive query. The few known data/query recovery attacks that manage to exploit this leakage [CGPR15, BKM20, KM19b, ZKP16] assume extremely strong adversarial models where the adversary has partial knowledge of the plaintext database/queries.

s-term Leakages. We focus next on the leakages related to the sterm, namely the size and equality pattern leakages. We begin by noting that these leakages are somewhat inherent to the design paradigm of OXT, which we inherit while designing OQXT. Even in the simpler setting of single keyword search, most existing scheme [Bos16, BMO17, CJJ+14, CPPJ18, CK10, CGKO06] also incur size and equality pattern leakages; the only constructions not to incur such leakages seem to rely on the use of ORAM-style data structures [BMO17, CPPJ18]. Fortifying OQXT with such data structures in an attempt to prevent this leakage is an interesting open challenge, although this would probably have to trade-off with some degradation in search performance (mostly in terms of communication complexity and number of rounds of communication during searches). It is also possible (and perhaps conceptually simpler) to mask this leakage by using volume-hiding techniques such as padding and encrypted multi-maps (EMMs) [APP+23b, PPYY19b, KM19b]. This would incur a degradation in search performance, and it is up to the designer to decide on a suitable trade-off between performance and leakage. However, we would like to point out that there are no known data/query recovery attacks on SSE schemes that specially exploit leakages related to the sterm. So we believe that even without the aforementioned fortifications, OQXT is not vulnerable to any known attacks due to the leakages related to the sterm, in realistic/practical adversarial settings.

x-term Leakages. Next, we focus on the xterm leakages. These leakages are also inherent to the design paradigm of OXT, which we inherit while designing OQXT. The only known attack on conjunctive SSE schemes that exploits a form of xterm leakages is the file injection attack proposed by Zhang et al. in [ZKP16]. More concretely, the adversarial server must be able to compute $|\mathbf{DB}(\mathsf{w}_1) \cap \mathbf{DB}(\mathsf{w}_i)|$ (where w_i is the xterm) when processing the search query. However for file injection attacks to work efficiently, the adversarial server must recover, for every xterm, the result size corresponding to each sub-query of the form $\mathsf{w}_1 \cap \mathsf{w}_i$. The xterm leakage profile of OQXT is not sufficient to compute this term, since the set of **xtoken** values sent to the server is randomly permuted precisely to mask such inference-style attacks. Finally, fortifying implementations of OQXT by using either ORAM-style data structures or adopting volume-hiding techniques such as padding or EMMs may be useful in masking this leakage even further; however, even without such additional fortifications, it appears that OQXT is not vulnerable to file injection attacks, or any other known attacks for that matter, due to the leakages related to the xterm, in realistic/practical adversarial

settings.

5 Experimental Results

We describe a prototype implementation of OQXT and evaluate its performance over real-world databases. We present experimental results comparing the storage requirements and search performance of OQXT with that of OXT [CJJ+13], IEX-2Lev [KM17] and ConjFilter [PPSY21].

5.1 Experimental Setup

In this section, we describe the experimental setup used to evaluate the performance of OQXT, as well as the parameter choices and other low-level details of our prototype implementation(s) of OQXT.

Data Set and Platform. We used the Enron email dataset[6] for our experiments. The Enron email data set used in our experiments contains around 10K documents (emails) and 80 thousand keyword-document pairs, with a total size of 1.3 GB.[7]

The complete OQXT implementation was done using C++ (with GCC 9 compiler) and we used Redis as the database backend. We ran the experiments on a *single* node with Intel Xeon E5-2690 v4 2.6 GHz CPU with 128 GB RAM and 512 GB SSD storage.

Lattice Parameters. We now discuss the lattice parameters used in our experiments. Concretely, for the LWR samples and associated lattice algorithms, we used two different OQXT implementations based on two different sets of LWR parameters, as described below.

- The first (smaller) set of LWR parameters are as follows: $(n, q, p) = \left(71, 2^8, 2^4\right)$. In the rest of this section, we denote by OQXT-I the concrete OQXT realization using these parameters. We note that these are not realistically secure LWR parameters (and we **do not recommend** using these parameters for practical deployments of OQXT), and are mainly used to depict a toy implementation of OQXT, which we call OQXT-I. The toy implementation is used because it better depicts the query complexity and storage overheads of OQXT (and the corresponding comparison with IEX-2Lev and ConjFilter) for the kinds of database sizes that could be supported given the (relatively) resource constrained environment that we used for our experiments.
- The second (larger) set of LWR parameters are as follows: $(n, q, p) = \left(512, 2^{20}, 2^8\right)$. The concrete OQXT realization using these parameters is denoted by OQXT-II in the rest of this section. These parameters resist state-of-the-art lattice cryptanalytic

[6] https://www.cs.cmu.edu/~enron/, https://www.kaggle.com/wcukierski/enron-email-dataset

[7] The raw Enron dataset consists of folders of emails of individual employees. Since the emails are in plain English text, we performed a dictionary-based pre-processing to filter the keywords (we removed the stop-words and punctuation in the process), and created a list of (lexicographically ordered) keywords.

attacks, and we **recommend** using these parameters for practical deployments of OQXT. We note, however, that the performance overheads incurred by this set of parameters is comparatively higher, especially on the (relatively) resource constrained environment used for the performance evaluation experiments. We envision, however, that even for these parameters, OQXT would prove to be more scalable as compared to IEX-2Lev and ConjFilter (especially for extremely large databases and, in particular, in terms of storage costs) when deployed on powerful servers with larger resources (which we believe would be used by enterprises and organizations for real deployment of SSE schemes over extremely large databases). We leave it as an interesting open question to implement and deploy OQXT on such servers, and compare the corresponding storage and conjunctive search overheads with the other SSE schemes.

The remaining lattice parameters for OQXT (namely m, $\bar{m}$ and w, which are basically related to lattice trapdoors) are chosen based on the recommendations in [MP12] as follows:

$$\bar{m} = 2n, \quad w = n \log q, \quad m = (\bar{m} + w).$$

The concrete figures used for these parameters would depend on the LWR parameters for OQXT-I or OQXT-II. We present an extended discussion on the performance of OQXT for these two choices of parameter sets at the end of this section.

Low-level Implementation Details. As already mentioned, we parse the raw Enron dataset to create a list of keywords, and we associate with each keyword the list of documents (we map each email to a document associated with a unique document identifier). This yields our plaintext inverted index **DB** (this plaintext index has size approximately 650KB, which is 0.05% of the actual plaintext Enron database), which we then encrypt using the setup algorithm of our OQXT protocol to create the encrypted outsourced database **EDB**, comprising of the TSet and XSet data structures. Note that, since we model the Enron dataset as a collection of documents and keywords, our formulation of OQXT as an SSE scheme for encrypted document collections proves to be the natural choice for prototyping and experimentation.

Concretely, we build the TSet and XSet data structures for the Enron dataset using Setup routine [1]. The OQXT.Setup routine incorporates specialized lattice based operations to generate the elements that are stored in the TSet and XSet. The $\mathsf{Gen\_Trapdoor}$ function generates a matrix with trapdoor for every keyword-id pair. The $\mathsf{SampleD}$ function samples a *short* vector $\mathbf{y}_{\mathsf{id}}$ from a specific coset of a discrete Gaussian distribution. These two algorithms are directly incorporated from [MP12]. The heavy computational overhead due to lattice instantiations are incurred at the setup phase only once and the encrypted data-structures are then offloaded to the server.

5.2 Experimental Evaluation

In this section, we present the results of our experimental evaluation of OQXT-I and OQXT-II, and compare the overheads incurred by these implementations with those incurred by OXT, IEX-2Lev, and ConjFilter.

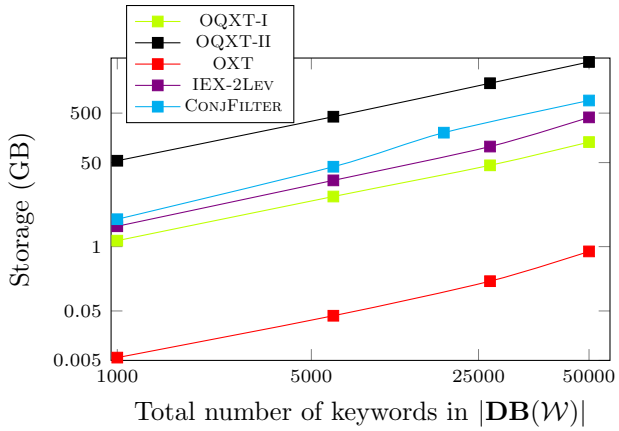

Figure 1: Server Storage (both axes are in log scale)

Evaluation of Storage Overhead. The difference of OQXT from OXT lies in the incorporation of lattice-based instantiations replacing the elliptic curve operations. In order to achieve post-quantum security lattice-based constructions rely upon matrices of large dimensions due to which the storage overhead of our plausibly quantum-safe construction is slightly higher than classically secured OXT. The trade-off in storage with security is however conveniently accepted because storage is considerably cheaper and unconstrained resource. Figure 1 compares the storage overhead of OQXT and OXT on Enron database. We vary the number of keywords in the database and observe the change in the storage pattern. It is observed that although the storage overhead of OQXT surpasses that of OXT, it increases linearly with the increase in number of keywords in **DB**.

In Figure 1 we compare the storage overhead of OQXT-I and OQXT-II with IEX-2Lev and ConjFilter scheme. Both IEX-2Lev and ConjFilter have a quadratic storage overhead and exploit the low size of mutual intersections for all pairs of keywords in **DB**. The storage overhead scales with the size of the intersection. In a sparse data set, the size of these intersections for most of the pairs of keywords is very low. However, if the database is not sparse, this results in large intersections for pairs of keywords and the overhead becomes truly quadratic. The storage overhead of OQXT increases linearly with the increase in the number of keywords in the database. OQXT incurs higher storage overhead due to incorporation of lattice-based constructions.

Evaluation of End-to-End Search Latency. Figure 2 and Figure 3 compares the end-to-end search latency of OQXT-I and OQXT-II with that of OXT, IEX-2Lev and ConjFilter for conjunctive queries. Search performance of OQXT-I and OQXT-II scales asymptotically with the search performance of OXT and ConjFilter and shows $10\times$ faster performance than IEX-2Lev. To validate this, we consider a three-keyword query of the form $q = (\mathsf{a}_1 \wedge \mathsf{a}_2) \wedge \mathsf{v}$, where $\mathsf{a}_1, \mathsf{a}_2$ and v are three keywords from **DB**. Without loss of generality, we consider the first term of q (or a_1 here) to be the least frequent keyword. We vary the frequency of v (referred to as the variable term) with different queries where as the frequency of a is kept constant (constant term). Figure 2 shows a constant time overhead for conjunctive queries of this form with OQXT-I and OQXT-II, which is identical with

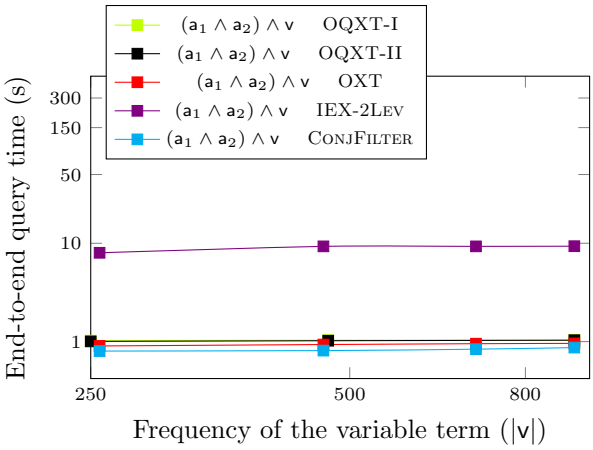

Figure 2: End-to-End Search latency with constant frequency of the least frequent term (both axes are in log scale)

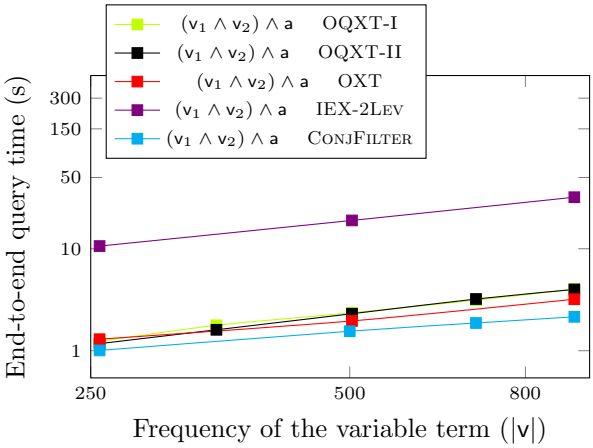

Figure 3: End-to-End Search latency with variable frequency of the least frequent term (both axes are in log scale)

OXT and ConjFilter but considerably faster than IEX-2Lev. This happens because the conjunctive search time of OXT and ConjFilter depends upon the least frequent keyword/conjunct. If the least frequent keyword is kept constant the time for searching a conjunctive query remains constant. In our quantum-safe construction OQXT we preserve the sub-linear search complexity of the original OXT scheme.

Another set of experiments is carried out on queries of the form $q = (v_1 \wedge v_2) \wedge a$ (Figure 3). This time we vary the frequency of least frequent keyword a_1 with different queries and the frequency of v is kept constant (since we *vary* the least frequent keyword, we denote this variable term as $(v_1 \wedge v_2)$ and the constant term as a in Figure 3). It is observed from Figure 3 that the search time of OXT and ConjFilter increases linearly with the increase in the frequency of the least frequent keyword. OQXT-I and OQXT-II preserves similar

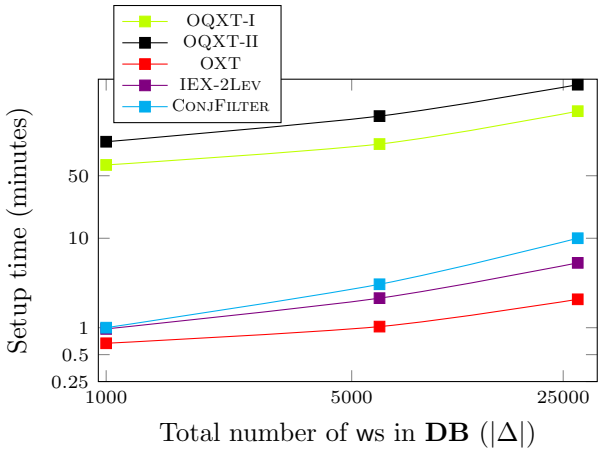

Figure 4: Setup time (minutes) (both axes are in log scale)

linear dependency of search time with an increase in the frequency of the least frequent keyword hence is efficiently comparable to both OXT and ConjFilter. The search time complexity of IEX-2Lev scales following a similar trend but is significantly (10×) more as compared to OQXT-I and OQXT-II.

The lattice-based instantiations used in OQXT uses matrices of larger dimensions which is generally compute intensive and increases the execution time. But it is important to note here that the two lattice-based functions, Gen_Trapdoor and SampleD in OQXT are primarily used during the setup (pre-comptutation) phase. Gen_Trapdoor function mainly incorporates matrix multiplication and only this lattice-based function is the incorporated during the search phase. The majority of compute intensive operations are carried out during the setup phase to construct the TSet and XSet elements. Since this is a one-time process we can safely not include this time in the overall search-query execution time. We report the data points on the set-up time of OQXT-I and OQXT-II and provide a comparison graph in Figure 4. The search phase of OQXT incorporates Gen_Trapdoor function as the only lattice based instantiation other than the generation of LWR samples. All the operations fundamentally reduces to matrix multiplications which can be highly optimized using various libraries or optimization methods, thereby ensuring efficient query processing.

Pre-Processing Overheads for Setup. Figure 4 compares the setup time required by OQXT-I and OQXT-II with that of IEX-2Lev, ConjFilter and OXT. It is observed that OQXT takes considerably more time than the other schemes. This is due to compute intensive lattice-based operations in- cluding trapdoor generation for every $\mathsf{w}-\mathsf{id}$ pair which is carried out during the setup phase in OQXT to construct the TSet and XSet elements. The time for setup increases linearly with the increase in the number of keywords in the database. However the setup time for IEX-2Lev and ConjFilter increases exponentially with the increase in number of keywords. We note here that, for static SSE, setup is essentially a one-time pre-processing step done at the client and does not impact the online query processing time. As compared to OXT, we view the increased pre- processing time for OQXT-I and OQXT-II as a necessary tradeoff for achieving post-quantum security while maintaining reasonable storage overheads and competitive online query processing over-

heads. As compared to IEX-2Lev and ConjFilter, we view the increased pre-processing time for OQXT as a tradeoff for achieving linear storage overheads (as compared to the worst-case quadratic overheads for IEX-2Lev and ConjFilter) while maintaining fast practically efficient search overheads.

5.3 Discussion on Our Experimental Results

We conclude the experimental evaluation section with some discussion on the performance of OQXT.

Storage Overheads. We note that, at first glance, OQXT (especially OQXT-II) incurs significant storage overheads and setup time as compared to IEX-2Lev and ConjFilter, both of which are also quantum-safe SSE schemes for conjunctive queries. However, this is mainly due to the somewhat constrained nature of the experimental setup used for our experiments. Asymptotically, the storage overheads of OQXT scale linearly with the size of the database (this is particularly demonstrated by the storage overheads incurred by the toy implementation OQXT-I in our experiments). On the other hand, the storage overheads for IEX-2Lev and ConjFilter are quadratic in the number of keywords. For precisely this reason, we envision that OQXT-II would prove to be more scalable in terms of storage costs as compared to IEX-2Lev and ConjFilter (especially for extremely large databases with millions of keywords) when deployed on powerful servers with larger resources (which would typically be used by enterprises and organizations for real deployment of SSE schemes over extremely large databases). We leave it as an interesting open question to implement and deploy OQXT on such servers, and compare the corresponding storage and conjunctive search overheads with the other SSE schemes.

Conjunctive Query Performance. We note that both OQXT-I and OQXT-II are competitive with OXT and ConjFilter in terms of conjunctive query performance, and significantly outperforming IEX-2Lev. We leave it as an interesting open question to further improve the concrete search time of OQXT (perhaps via more optimized implementation of the underlying mathematical operations related to lattice trapdoor sampling and the corresponding matrix operations). Regardless, as demonstrated by our experiments, OQXT achieves reasonably fast and practically efficient conjunctive search overheads while providing strong lattice-based post-quantum security guarantees (including for the larger set of parameters that resists state-of-the-art lattice cryptanalysis techniques).

Setup Time. Finally, we note that setup time essentially represents a a one-time preprocessing-style cost at the client and, for many applications, might not be a significant bottleneck (especially for static databases). Hence, we do not view the high storage costs of OQXT as a major issue, and treat it as a tradeoff for the better (asymptotic storage) complexity, practical query performance and lattice-based quantum-safe security guarantees of OQXT.

6 Conclusion and Future Directions

In this paper, we proposed Oblivious Post-Quantum Secure Cross Tags (OQXT) – the first lattice-based SSE scheme that supports highly scalable and practically efficient conjunctive keyword searches over large encrypted document collections. Along the way, we introduced a novel oblivious cross-tag generation protocol with provable security guarantees derived from the Learning with Rounding (LWR) assumption. We formally defined the leakage profile of OQXT and proved its simulation-based post-quantum security with respect to this leakage profile against a semi-honest adversarial server. We finally presented a prototype implementation of OQXT and experimentally validated its performance over real-world databases. Our work gives rise to many directions of future research. We summarize some of these below.

Extension to Disjunctive and Richer Boolean Queries. We conjecture that OQXT can be immediately plugged into extensions of OXT to support disjunctive and richer Boolean queries, thereby yielding quantum-safe variants of duch extensions. As a concrete example, a recent work [BTR+22] proposes a generic framework called TWINSSE that upgrades any conjunctive SSE scheme into an SSE scheme that supports disjunctive and richer Boolean queries. Instantiating this framework using OQXT would yield the first plausibly quantum safe SSE scheme capable of supporting conjunctive, disjunctive and richer Boolean queries in an efficient and scalable manner. We leave formalizing such an instantiation as an interesting future direction of research.

Extension to Join Queries. Another recent work [JP22] shows how to extend OXT to also support join queries over relational databases using a purely symmetric-key framework called JXT. We believe that OQXT is fully compatible with JXT, and that combining OQXT with JXT would yield the first plausibly quantum safe SSE scheme capable of supporting Boolean queries as well as join queries over encrypted relational databases in an efficient and scalable manner. We again leave formalizing such a combination of OQXT with JXT as an interesting future direction of research.

References

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 99–108, 1996.

[AKPW13] Joël Alwen, Stephan Krenn, Krzysztof Pietrzak, and Daniel Wichs. Learning with rounding, revisited - new reduction, properties and applications. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, 2013.

[AKSX04] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 563–574, 2004.

[APP+23a] Ghous Amjad, Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. Dynamic volume-hiding encrypted multi-maps with applications to searchable encryption. *Proc. Priv. Enhancing Technol.*, 2023(1):417–436, 2023.

[APP+23b] Ghous Amjad, Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. Dynamic volume-hiding encrypted multi-maps with applications to searchable encryption. *Proc. Priv. Enhancing Technol.*, 2023.

[BCO11] Alexandra Boldyreva, Nathan Chenette, and Adam O'Neill. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In *Annual Cryptology Conference*, pages 578–595. Springer, 2011.

[BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *TOCT*, 6(3):13:1–13:36, 2014.

[BKM20] Laura Blackstone, Seny Kamara, and Tarik Moataz. Revisiting leakage abuse attacks. In *NDSS 2020*, 2020.

[BMO17] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. In *ACM CCS 2017*, pages 1465–1482, 2017.

[Bos16] Raphael Bost. $\sum o\varphi o\varsigma$: Forward secure searchable encryption. In *ACM CCS 2016*, pages 1143–1154, 2016.

[BPR12] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 719–737. Springer, 2012.

[BTR+22] Arnab Bag, Debadrita Talapatra, Ayushi Rastogi, Sikhar Patranabis, and Debdeep Mukhopadhyay. Two-in-one-sse: Fast, scalable and storage-efficient searchable symmetric encryption for conjunctive and disjunctive boolean queries. *IACR Cryptol. ePrint Arch.*, page 1096, 2022.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *CRYPTO 2011*, pages 505–524, 2011.

[CGGI16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *ASIACRYPT 2016*, volume 10031, pages 3–33, 2016.

[CGGI20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: fast fully homomorphic encryption over the torus. *J. Cryptol.*, 33(1):34–91, 2020.

[CGKO06] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *ACM CCS 2006*, pages 79–88, 2006.

[CGPR15] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In *ACM CCS 2015*, pages 668–679, 2015.

[CJJ+13] David Cash, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *CRYPTO 2013*, pages 353–373, 2013.

[CJJ+14] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *NDSS 2014*, 2014.

[CK10] Melissa Chase and Seny Kamara. Structured encryption and controlled disclosure. In *ASIACRYPT 2010*, pages 577–594, 2010.

[CM05] Yan-Cheng Chang and Michael Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *ACNS 2005*, pages 442–455, 2005.

[CPKD22] Javad Ghareh Chamani, Dimitrios Papadopoulos, Mohammadamin Karbasforushan, and Ioannis Demertzis. Dynamic searchable encryption with optimal search in the presence of deletions. In *USENIX Security 2022*, pages 2425–2442, 2022.

[CPPJ18] Javad Ghareh Chamani, Dimitrios Papadopoulos, Charalampos Papamanthou, and Rasool Jalili. New constructions for forward and backward private symmetric searchable encryption. In *ACM CCS 2018*, pages 1038–1055, 2018.

[DCPP20] Ioannis Demertzis, Javad Ghareh Chamani, Dimitrios Papadopoulos, and Charalampos Papamanthou. Dynamic searchable encryption with small client storage. In *NDSS 2020*, 2020.

[DPP+] Ioannis Demertzis, Stavros Papadopoulos, Odysseas Papapetrou, Antonios Deligiannakis, and Minos N. Garofalakis. Practical private range search revisited. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*.

[DPP+18] Ioannis Demertzis, Stavros Papadopoulos, Odysseas Papapetrou, Antonios Deligiannakis, Minos N. Garofalakis, and Charalampos Papamanthou. Practical private range search in depth. *ACM Trans. Database Syst.*, 2018.

[DPPS20] Ioannis Demertzis, Dimitrios Papadopoulos, Charalampos Papamanthou, and Saurabh Shintre. SEAL: attack mitigation for encrypted databases via adjustable leakage. In *USENIX Security 2020*, 2020.

[EKPE18] Mohammad Etemad, Alptekin Küpçü, Charalampos Papamanthou, and David Evans. Efficient dynamic searchable encryption with forward privacy. *PoPETs*, 2018(1):5–20, 2018.

[FJK+15] Sky Faber, Stanislaw Jarecki, Hugo Krawczyk, Quan Nguyen, Marcel-Catalin Rosu, and Michael Steiner. Rich queries on encrypted data: Beyond exact matches. In *ESORICS 2015*, pages 123–145, 2015.

[Gen09] C. Gentry. Fully homomorphic encryption using ideal lattices. In *ACM STOC'09*, pages 169–178, 2009.

[GJW19] Zichen Gui, Oliver Johnson, and Bogdan Warinschi. Encrypted databases: New volume attacks against range queries. In *ACM CCS 2019*, pages 361–378, 2019.

[GKM21] Marilyn George, Seny Kamara, and Tarik Moataz. Structured encryption and dynamic leakage suppression. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021*, volume 12698, pages 370–396, 2021.

[GLMP18] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. Pump up the volume: Practical database reconstruction from volume leakage on range queries. In *ACM CCS 2018*, pages 315–331. ACM, 2018.

[GLMP19] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. Learning to reconstruct: Statistical learning theory and encrypted database attacks. In *IEEE SP 2019*, pages 1067–1083, 2019.

[GMP16] Sanjam Garg, Payman Mohassel, and Charalampos Papamanthou. TWORAM: efficient oblivious RAM in two rounds with applications to searchable encryption. In *CRYPTO 2016*, pages 563–592, 2016.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, 1996.

[Goh03] Eu-Jin Goh. Secure indexes. *IACR Cryptology ePrint Archive*, 2003:216, 2003.

[GPP23] Zichen Gui, Kenneth G. Paterson, and Sikhar Patranabis. Rethinking searchable symmetric encryption. In *IEEE Symposium on Security and Privacy, SP 2023 (to appear)*, 2023. Available from `https://eprint.iacr.org/2021/879`.

[GPPW20] Zichen Gui, Kenneth G. Paterson, Sikhar Patranabis, and Bogdan Warinschi. Swissse: System-wide security for searchable symmetric encryption. *IACR Cryptol. ePrint Arch.*, page 1328, 2020.

[IKK12] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *NDSS 2012*, 2012.

[JP22] Charanjit S. Jutla and Sikhar Patranabis. Efficient searchable symmetric encryption for join queries. In *ASIACRYPT 2022*, volume 13793, pages 304–333, 2022.

[Ker15] Florian Kerschbaum. Frequency-hiding order-preserving encryption. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 656–667, 2015.

[KKL+17] Kee Sung Kim, Minkyu Kim, Dongsoo Lee, Je Hong Park, and Woo-Hwan Kim. Forward secure dynamic searchable symmetric encryption with efficient updates. In *ACM CCS 2017*, pages 1449–1463, 2017.

[KKM+22] Seny Kamara, Abdelkarim Kati, Tarik Moataz, Thomas Schneider, Amos Treiber, and Michael Yonli. Sok: Cryptanalysis of encrypted search with LEAKER - A framework for leakage attack evaluation on real-world data. In *EuroS&P 2022*, pages 90–108, 2022.

[KKNO16] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. Generic attacks on secure outsourced databases. In *ACM CCS 2016*, pages 1329–1340, 2016.

[KM17] Seny Kamara and Tarik Moataz. Boolean searchable symmetric encryption with worst-case sub-linear complexity. In *EUROCRYPT 2017*, pages 94–124, 2017.

[KM18] Seny Kamara and Tarik Moataz. SQL on structurally-encrypted databases. In *Advances in Cryptology - ASIACRYPT 2018 - 24th International*, 2018.

[KM19a] Seny Kamara and Tarik Moataz. Computationally volume-hiding structured encryption. In *EUROCRYPT 2019*, pages 183–213, 2019.

[KM19b] Seny Kamara and Tarik Moataz. Computationally volume-hiding structured encryption. In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II*, 2019.

[KP13] Seny Kamara and Charalampos Papamanthou. Parallel and dynamic searchable symmetric encryption. In *FC 2013*, pages 258–274, 2013.

[KPR12] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In *ACM CCS 2012*, pages 965–976, 2012.

[LMP18] Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. Improved reconstruction attacks on encrypted data using range query leakage. In *IEEE SP 2018*, pages 297–314. IEEE Computer Society, 2018.

[LPS+18] Shangqi Lai, Sikhar Patranabis, Amin Sakzad, Joseph K. Liu, Debdeep Mukhopadhyay, Ron Steinfeld, Shifeng Sun, Dongxi Liu, and Cong Zuo. Result pattern hiding searchable encryption for conjunctive queries. In *ACM CCS 2018*, pages 745–762, 2018.

[LW16] Kevin Lewi and David J. Wu. Order-revealing encryption: New constructions, applications, and lower bounds. In *ACM CCS 2016*, pages 1167–1178, 2016.

[MG02] Daniele Micciancio and Shafi Goldwasser. *Complexity of lattice problems: a cryptographic perspective*, volume 671. Springer Science & Business Media, 2002.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 700–718. Springer, 2012.

[NKW15] Muhammad Naveed, Seny Kamara, and Charles V. Wright. Inference attacks on property-preserving encrypted databases. In *ACM CCS 2015*, pages 644–655, 2015.

[PM21] Sikhar Patranabis and Debdeep Mukhopadhyay. Forward and backward private conjunctive searchable symmetric encryption. In *NDSS 2021*, 2021.

[PPSY21] Sarvar Patel, Giuseppe Persiano, Joon Young Seo, and Kevin Yeo. Efficient boolean search over encrypted data with reduced leakage. In *ASIACRYPT 2021*, volume 13092, pages 577–607, 2021.

[PPYY19a] Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing. In *ACM CCS 2019*, pages 79–93, 2019.

[PPYY19b] Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, 2019.

[PRZB11] Raluca A. Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *SOSP 2011*, pages 85–100, 2011.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, pages 34:1–34:40, 2009.

[SDY+18] Xiangfu Song, Changyu Dong, Dandan Yuan, Qiuliang Xu, and Minghao Zhao. Forward private searchable symmetric encryption with optimized I/O efficiency. *IACR Cryptology ePrint Archive*, 2018:497, 2018.

[SPS14] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. Practical dynamic searchable encryption with small leakage. In *NDSS 2014*, 2014.

[SSL+21] Shifeng Sun, Ron Steinfeld, Shangqi Lai, Xingliang Yuan, Amin Sakzad, Joseph K. Liu, Surya Nepal, and Dawu Gu. Practical non-interactive searchable encryption with forward and backward privacy. In *NDSS 2021*, 2021.

[SWP00] Dawn Xiaodong Song, David A. Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *IEEE S&P 2000*, pages 44–55, 2000.

[SYL+18] Shifeng Sun, Xingliang Yuan, Joseph K. Liu, Ron Steinfeld, Amin Sakzad, Viet Vo, and Surya Nepal. Practical backward-secure searchable encryption from symmetric puncturable encryption. In *ACM CCS 2018*, pages 763–780, 2018.

[TK20] Anselme Tueno and Florian Kerschbaum. Efficient secure computation of order-preserving encryption. In Hung-Min Sun, Shiuh-Pyng Shieh, Guofei Gu, and Giuseppe Ateniese, editors, *ACM ASIA CCS 2020*, pages 193–207, 2020.

[ZKP16] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In *USENIX Security Symposium 2016*, pages 707–720, 2016.