Highly Scalable Searchable Symmetric Encryption for Boolean Queries from NTRU Lattice Trapdoors

Debadrita Talapatra
IIT Kharagpur, India

Sikhar Patranabis
IBM Research India

Debdeep Mukhopadhyay
IIT Kharagpur, India

Abstract

Searchable symmetric encryption (SSE) enables query execution directly over symmetrically encrypted databases. To support realistic query executions over encrypted document collections, one needs SSE schemes capable of supporting *both* conjunctive *and* disjunctive keyword queries. Unfortunately, existing solutions are either practically inefficient (incur large storage overheads and/or high query processing latency) or are quantum-unsafe.

In this paper, we present the first practically efficient SSE scheme with fast conjunctive *and* disjunctive keyword searches, compact storage, and security based on the (plausible) quantum-hardness of well-studied *lattice-based* assumptions. We present NTRU-OQXT – a highly compact NTRU lattice-based conjunctive SSE scheme that outperforms *all* existing conjunctive SSE schemes in terms of search latency. We then present an extension of NTRU-OQXT that additionally supports disjunctive queries, we call it NTRU-TWINSSE. Technically, both schemes rely on a novel oblivious search protocol based on highly optimized Fast-Fourier trapdoor sampling algorithms over NTRU lattices. While such techniques have been used to design other cryptographic primitives (such as digital signatures), they have not been applied before in the context of SSE.

We present prototype implementations of both schemes, and experimentally validate their practical performance over a large real-world dataset. Our experiments demonstrate that NTRU-OQXT achieves 2× faster conjunctive keyword searches as compared to *all other* conjunctive SSE schemes (including the best quantum-unsafe conjunctive SSE schemes), and substantially outperforms many of these schemes in terms of storage requirements. These efficiency benefits also translate to NTRU-TWINSSE, which is practically competitive with the best quantum-unsafe SSE schemes capable of supporting both conjunctive and disjunctive queries.

Contents

1 Introduction **3**

1.1 Our Contributions 4

1.2 Extensions and Future Work 6

2 Overview of Our Contributions **7**

3 Preliminaries and Background **11**

3.1 Notations 11

3.2 Lattice Preliminaries 11

3.3 The GPV Signature Scheme 12

3.4 The FALCON Signature Scheme 13

3.5 Basic Cryptographic Primitives 14

3.6 SSE: Syntax and Security Model 15

4 NTRU-OQXT **16**

4.1 Revisiting OQXT 17

4.2 Core Idea 18

4.3 Technical Details 19

4.4 Proof Of Correctness of NTRU-OQXT 23

5 Security analysis of NTRU**-**OQXT **26**

5.1 Leakage Profile Analysis 26

5.2 Proof Of Theorem 2 28

6 NTRU-TWINSSE **35**

6.1 Correctness and Security Analysis of NTRU-TWINSSE 36

7 Experimental Results **38**

7.1 Experimental Evaluation of NTRU-OQXT 39

7.2 Evaluation of NTRU-TWINSSE 41

8 Concluding Remarks **42**

1 Introduction

Searchable Symmetric Encryption (SSE). Searchable Symmetric Encryption (SSE) [SWP00, Goh03, CK10, CJJ+13, CJJ+14] is a cryptographic system that allows a (potentially resource-constrained) client to execute keyword search queries *directly* over symmetrically encrypted document collections stored at an *untrusted* third-party server. SSE schemes achieve efficient query processing over encrypted data while allowing the server to learn some controlled amount of benign information (called "leakage") during query execution.

A long line of works [CJJ+13, CJJ+14, KM17, SYL+18, PPSY21, PM21] has studied the expressiveness of queries supported by SSE schemes. In real-life applications, such as querying a large medical database that is remotely stored, a single keyword query would potentially return a large number of matching records/documents that the client would need to download and filter locally. As such, for an SSE scheme to be practically deployable, it should crucially support richer query processing capabilities such as conjunctions, disjunctions, and more general Boolean formulae over collections of keywords.

SSE for General Boolean Queries. Starting with the proposal of the Oblivious Cross-Tags (OXT) scheme in [CJJ+13], several works have proposed SSE schemes supporting conjunctive keyword queries with a variety of leakage vs efficiency trade-offs in different settings [CJJ+13, CJJ+14, LPS+18, PM21]. Unfortunately, these schemes do not support disjunctive queries efficiently. Only a handful of existing SSE schemes additionally support disjunctive and more general Boolean queries [KM17, PPSY21]. However, the scheme proposed in [KM17] exhibits significantly inferior performance for conjunctive queries over real-world databases as compared to OXT. Moreover, the schemes from [KM17, PPSY21] incur (worst-case) storage overheads that grow *quadratically* with the number of keywords in the database. This hinders practical deployment over large real-world databases.

TWINSSE. A very recent work from PoPETS '23 [BTR+23] proposed TWINSSE– an efficient and highly scalable SSE for conjunctive, disjunctive, and more general Boolean queries. TWINSSE builds upon OXT [CJJ+13] (which already supported conjunctive queries, but not disjunctive queries) in a black-box manner to also support disjunctive queries. The technical core of TWINSSE is the idea of "meta-keywords". Informally speaking, each meta-keyword is a disjunctive combination of specific keywords in the database. Given such a (pre-processed) collection of meta-keywords, the authors of [BTR+23] proposed a framework for simulating any disjunctive query over the original keywords in the database using *a conjunctive query over the meta-keywords*, which can then be evaluated using OXT. The main advantage of TWINSSE over other state-of-the-art schemes such as [KM17, PPSY21] is that it incurs a storage overhead that scales *linearly* (and *not quadratically*) with the size of the plaintext database, while maintaining the *sub-linear* search complexity of the original conjunctive OXT scheme.

TWINSSE is Quantum-Unsafe. While SSE is, by definition, an inherently symmetric-key cryptosystem, almost all existing SSE constructions supporting conjunctive (and more general Boolean) queries rely on public-key cryptographic techniques to achieve compact storage of the encrypted database and/or fast encrypted query processing [CJJ+13, LPS+18, PM21, BTR+23]. For example, TWINSSE builds upon OXT, which in turn relies crucially

on discrete-log hard groups (typically implemented in practice using elliptic-curve-based cryptography) for efficient and secure query processing. As a result, OXT, and by extension, TWINSSE are quantum-unsafe, and will be devastatingly broken once scalable quantum computers capable of solving the discrete log problem become feasible.

OQXT. Another recent work [TPM23] introduced OQXT – a plausibly quantum-safe counterpart to OXT that relies on the presumed (quantum) hardness of solving certain lattice problems, such as the *Short Integer Solutions* (SIS) [Ajt96] problem and the *Learning with Rounding* (LWR) [Reg09, BPR12] problems. Unfortunately, OQXT has two major drawbacks. First of all, like OXT, it only supports conjunctive keyword queries, and not disjunctive or more general Boolean queries. More crucially, OQXT works over integer lattices, and incurs significantly higher storage overheads as compared to OXT, which is not desirable from a practical point of view.

Our Motivation. It is evident from the above discussions that the existing SSE schemes supporting *both* conjunctive *and* disjunctive keyword queries either incur extremely large encrypted storage overheads [KM17, PPSY21], or are not quantum-safe [BTR$^+$23]. Moreover, existing lattice-based SSE schemes have limited query expressiveness [TPM23]. To the best of our knowledge, there does not exist today a practically efficient SSE scheme that supports conjunctive and disjunctive queries while relying on the plausible quantum-hardness of lattice problems. Motivated by this, we ask the following question -

Can we design a lattice-based practically efficient SSE scheme supporting ***both*** *conjunctive and disjunctive keyword queries?*

1.1 Our Contributions

In this paper, we answer the above question in the affirmative. We propose the *first* practically efficient lattice-based SSE scheme capable of supporting *both* conjunctive *and* disjunctive queries over symmetrically encrypted static document collections. Our contributions are as follows.

Quantum-Safe Conjunctive SSE from NTRU Lattices. As a core technical contribution, we present a novel and highly optimized alternative to OQXT [TPM23] that operates *over structured NTRU lattices* (as opposed to unstructured integer lattices in OQXT). We call this scheme NTRU-OQXT. The design of NTRU-OQXT incorporates state-of-the-art practically efficient techniques for lattice trapdoor sampling based on *Fast-Fourier Sampling*. The design of NTRU-OQXT is partly inspired by the digital signature scheme Falcon [PFH$^+$17] (currently a NIST standard for quantum-safe digital signatures), and derives its data and query privacy guarantees from the same set of (plausibly quantum-safe) hardness assumptions over NTRU lattices as Falcon[1].

Extension to Disjunctive Queries. We then show how to incorporate NTRU-OQXT

[1]Note, the translation from unstructured lattices to NTRU is not straightforward and requires a thorough technical understanding of the underlying mathematical structures and devising new algorithmic constructs using the mathematical structures.

into the TWINSSE framework of [BTR+23]. The resulting scheme, which we call NTRU-TWINSSE , supports *both* conjunctive *and* disjunctive queries over static datasets. This yields, to the best of our knowledge, the *first* lattice-based static SSE scheme that supports both conjunctive and disjunctive keyword queries, while attaining linear storage overheads and sub-linear query complexity.

Implementation and Experimentation. We present a prototype implementation of NTRU-OQXT and experimentally demonstrate that it outperforms OQXT substantially (with $3-4\times$ faster conjunctive keyword searches and $10^4\times$ smaller storage requirements), and scales smoothly to large real-world document collections. In fact, NTRU-OQXT achieves at least $2\times$ faster conjunctive keyword searches as compared to *all other* conjunctive SSE schemes [CJJ+13,KM17,PPSY21], and substantially outperforms [KM17,PPSY21] in terms of storage requirements, while achieving only slightly higher storage as compared to OXT [CJJ+13]. To the best of our knowledge, NTRU-OQXT is the first conjunctive SSE scheme that is competitive with the overall practical efficiency of the OXT scheme, while relying on the plausible quantum-safe hardness assumptions over NTRU lattices (unlike OXT, which relies on discrete log-hard groups, and is quantum-unsafe).

As noted previously, prior conjunctive SSE schemes such as OXT leverages public-key primitives for optimally compact encrypted storage, which the purely symmetric-key schemes do not achieve (IEX-2Lev and ConjFilter incur *quadratic* storage overheads and do not scale to large databases). We show how to use plausibly quantum-safe public-key techniques to achieve NTRU-OQXT – a conjunctive SSE with optimally compact encrypted storage (which IEX-2Lev, ConjFilter, and OQXT do not achieve), resistance to quantum attacks (which OXT does not achieve), and fast conjunctive searches (which IEX-ZMF and OQXT do not achieve).

Finally, we also present a prototype implementation of NTRU-TWINSSE , and experimentally demonstrate its performance over large real-world datasets. Our experiments establish that NTRU-TWINSSE substantially outperforms [KM17, PPSY21] in terms of storage requirements, and is closely competitive with [BTR+23] both in terms of search time (for conjunctive *and* disjunctive queries) and storage overheads, while additionally achieving security based on plausibly quantum-safe hardness assumptions over NTRU lattices (unlike $\text{TWINSSE}_{\text{OXT}}$ [BTR+23], which inherits reliance on discrete log-hard groups from OXT, and is quantum-unsafe).

Comparison with FHE. Traditional fully homomorphic encryption (FHE) [Gen09, BV11, BGV14, CGGI16, CGGI20] is capable of a much more general class of encrypted computations as compared to encrypted keyword searches. However, for the specific privacy-preserving computation scenarios that we focus on in this paper, NTRU-OQXT and NTRU-TWINSSE serve as more practically efficient lattice-based alternatives. With traditional FHE, one would encrypt the entire document collection. On the other hand, NTRU-OQXT and NTRU-TWINSSEonly encrypt the inverted index using lattice techniques, while the documents themselves would be encrypted using simple and efficient symmetric-key encryption techniques (e.g., AES-256). In addition, the proposed schemes avoid the computational overheads associated with FHE-bootstrapping [CGGI16, CGGI20], which sometimes hinder FHE computations from scaling to large databases.

1.2 Extensions and Future Work

Extensions to General Boolean Queries. We note here that Boolean queries, which include conjunctions, disjunctions, and negations, are typically expressed in the conjunctive normal form (CNF) or disjunctive normal form (DNF). The SSE literature has traditionally focused on conjunctive and disjunctive queries, which is also the focus of our work. The original TWINSSE paper demonstrated how their core scheme could be extended to support more general CNF and DNF queries, though this came with a significant storage blowup, and was tightly coupled with their underlying conjunctive scheme, OXT. While similar extensions could potentially be developed for NTRU-OQXT, we leave a detailed exploration of this direction to future work.

Extension to Dynamic Databases. We acknowledge the importance of handling dynamic databases. However, we believe that our contribution wr.t. static databases is meaningful in its own right. First of all, static SSE is widely studied [CGKO06, CK10, CJJ+13], and is particularly relevant to real-world applications where the client wishes to outsource a snapshot of a large database for query processing. Consider a situation where an organization outsources a snapshot of its current sales database for auditing by a third-party organization, but wishes to do this without revealing the entire database in plaintext for privacy reasons. This precisely requires static SSE, since the auditing organization only wishes to query the database without updating it. We note here that all of the schemes that we compared against [CJJ+13, KM17, PPSY21, BTR+23, TPM23] are also static SSE schemes and do not support dynamic databases (in fact, as the authors of [BTR+23] point out, extending the TWINSSE framework to the dynamic setting is, by itself, a challenging research direction).

Additionally, a two-phase approach of first focusing on static databases before extending it to the dynamic setting allows for an easier exposition of our core technical ideas. We point to several relevant examples of this approach in the SSE literature. OXT was only studied in the static setting in [CJJ+13] before being extended to the dynamic setting in [CJJ+14] and [PM21]. In leakage-suppressing SSE, volume-hiding encrypted multi-maps were first analyzed extensively in the static SSE setting [KM19, PPYY19] before being extended to dynamic databases [GKM21].

While we believe that NTRU-OQXT can be extended to the dynamic setting (e.g., using techniques similar to those in [PM21] for extending OXT), we believe that a detailed treatment deserves dedicated future work, with the following possible modifications to the static scheme: (i) mapping the static cross-tags to dynamic cross-tags, (ii) signing the secret message and storing the signatures dynamically, (iii) ensure forward/backward privacy guarantees in the post-quantum setting, (iv) handle additions and deletions indistinguishably, and so on. We leave it as an interesting open question to extend both NTRU-OQXT and NTRU-TWINSSE to dynamic databases.

Supplementary Leakage Analysis. An interesting follow-up to this work would be to further analyze the leakage profile of NTRU-OQXT and devise a post-quantum secure conjunctive SSE scheme with a more concise leakage profile. For example, the keyword-pair result pattern leakage in OXT translates to NTRU-OQXT which is a non-trivial leakage, and has been exploited in literature by several leakage abuse attacks. In [LPS+18] the

authors propose a scheme (HXT) that improves upon this leakage in OXT. Since HXT builds upon similar OXT like construction, it would be interesting to incorporate lattice-based improvisations into HXT and develop an even more robust quantum-safe conjunctive SSE scheme. Another plausible direction would be to investigate the leakage suppression technique for mitigating 2-conjunction leakages as demonstrated in [PPSY21].

2 Overview of Our Contributions

OQXT. We begin with a very brief background on the OQXT scheme from [TPM23], which is a lattice-based static conjunctive SSE scheme. The OQXT scheme is conceptually similar to OXT, and essentially adopts the techniques of OXT from discrete-log hard groups to the setting of unstructured integer lattices. In particular, OQXT uses *lattice trapdoors* for short integers solution (SIS) [MP12] to enable an *oblivious search operation* at the server. Unfortunately, over integer lattices, using SIS trapdoors necessitates matrices of very large dimensions, which leads to a huge blowup in both the setup time as well as the storage overheads for OQXT. See Section 4.1 for a more detailed overview of OQXT.

OQXT and GPV. The starting point of our core technical contribution is the observation that OQXT can actually be explained in terms of the well-known GPV signatures [GPV08], which also rely on SIS trapdoors over integer lattices. In fact, at a very high level, OQXT enables oblivious searches at the server by using the following template:

- At setup[2], for each keyword-document pair in the underlying plaintext database, the client: (i) creates a signing key-verification key pair (derived uniquely from the keyword) and a message (derived uniquely from the document identifier), (ii) computes a unique GPV signature on the message using the signing key, and (iii) stores this signature along with an encryption of the document identifier at the server (as part of a specialized data structure representing the encrypted database).
- During search, depending on its query, the client sends two tokens to the server with the following (informal) semantics: (i) the first token allows the server to derive a GPV public key, and (ii) the second token allows the server to retrieve some GPV signature (and an associated encryption of the corresponding document identifier) from the data structure representing the encrypted database.
- The server verifies the retrieved signature under the derived public key. If the verification succeeds, the server infers that corresponding identifier represents a document matching the query. In this case, it returns the encrypted document identifier to the client, and the client decrypts it locally.

The last two steps are actually repeated for several candidate documents (depending on the client's query), and the server filters which (encrypted) document identifier to send to the client depending on whether the signature verifies. The actual scheme is significantly more involved and exploits some additional algebraic properties of the GPV signature scheme,

[2]Refer to Section 3.6 for SSE-specific syntax.

but we deliberately present a very simplified exposition here to illustrate our key ideas. Note that this connection between GPV and OQXT was *not elucidated explicitly* in [TPM23], and is the key insight that we build upon to achieve a significantly more efficient alternative to OQXT.

Switching from GPV to Falcon. Given the above connection between GPV signatures and OQXT, one can immediately interpret the practical inefficiency of OQXT in terms of the well-known practical inefficiency of the original GPV signature scheme over integer lattices. Fortunately, this also presents us with the idea for a solution. While the GPV signature scheme is inefficient in its original form, one can achieve a significantly more efficient version of it by shifting from unstructured integer lattices to more structured NTRU lattices. This efficient version is nothing but the digital signature scheme Falcon [PFH+17] (currently a NIST standard for quantum-safe digital signatures). Conceptually, Falcon also uses SIS trapdoors, but over NTRU lattices, and achieves significantly more compact parameters in practice as compared to GPV. This leads us to ask if, by building upon Falcon in the same way as OQXT seems to build upon GPV, can we achieve a significantly more efficient alternative to OQXT over NTRU lattices[3]?

Note that this is not just a question of simply substituting GPV with Falcon in the design of OQXT. As mentioned earlier, the actual OQXT scheme exploits additional algebraic features of the GPV signature scheme. In particular, it exploits the following (informal) property of GPV: a signature σ that verifies on a message m with hash h under a public key pk, also verifies on a message m' with hash h' under a public key pk' if there exists an affine function f such that $\mathsf{pk}' = f(\mathsf{pk})$ and $\mathsf{h}' = f(\mathsf{pk})$. Also, OQXT has certain additional requirements on top of GPV to maintain data and query privacy; concretely, it needs both the public key pk and the message m to be *pseudorandom* so as to hide the underlying keyword and document identifier hidden from the server. For this, it uses some additional techniques while relying on the learning with rounding (LWR) assumption, while maintaining compatibility with GPV signature generation and verification. While this is somewhat natural over integer lattices, the techniques do not translate immediately to NTRU lattices. Hence, substituting GPV with Falcon in the design of OQXT would require carefully re-designing all of these associated techniques such that they now work over NTRU lattices, while achieving the same functional properties and security guarantees as OQXT. See Section 4.2 for more detailed discussion.

NTRU-OQXT. Our core technical contribution lies in addressing the aforementioned challenges. In particular, by switching from GPV to NTRU, and by carefully designing NTRU-based alternatives to all of the additional lattice techniques used in OQXT, we arrive at a significantly more efficient alternative to OQXT over NTRU lattices, which we call NTRU-OQXT. In particular, we replace the original SIS trapdoor sampler over integer lattices used by OQXT with the *Fast-Fourier trapdoor sampling* algorithm used in Falcon to achieve a significantly faster Setup algorithm for NTRU-OQXT. In addition, the compactness of Falcon public keys and signatures translates into significantly lower storage requirements for NTRU-OQXT. From a security standpoint, NTRU-OQXT relies on Falcon and a

[3]NTRU-OQXT adopts the original OQXT scheme, which was based on GPV signatures over integer lattices, to use Falcon over NTRU lattices for compact storage and faster query processing. One could certainly adopt OQXT to other variants/instantiations of GPV (e.g., HAWK), thereby inheriting their desirable features (e.g., the avoidance of floating-point operations in HAWK).

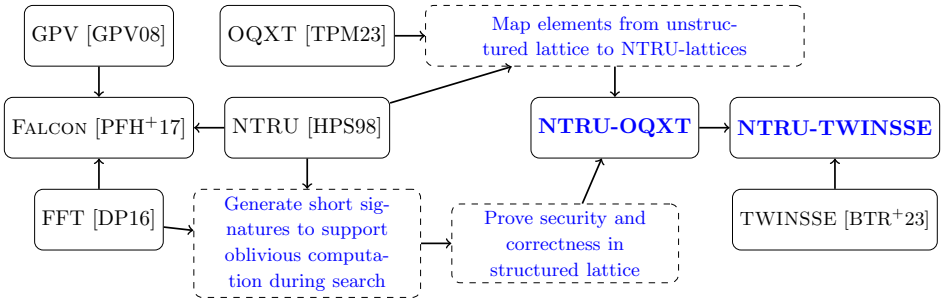


Figure 1: Lineage of NTRU-OQXT and NTRU-TWINSSE. Text in blue represents the contribution of this work.

(well-studied) variant of the assumption of LWR on polynomial rings. See Section 4.3 for the complete formal description and Section 4.4 for the formal proof of correctness.

We emphasize that the construction of NTRU-OQXT cannot not use Falcon as a black box. This is because NTRU-OQXT requires deterministic encrypted table look-up to retrieve correct search results, hence the verification routine of Falcon which is not deterministic (it accepts signatures within a given bound) cannot be incorporated into the design of NTRU-OQXT. Therefore, transforming OQXT from unstructured lattice to structured NTRU lattice and replacing its trapdoor sampler with FFT trapdoor sampler, devising analogous constructs and proving the correctness and security of NTRU-OQXT, is not straightforward and requires a thorough technical understanding of the underlying mathematical structures. Technically, to devise its structural constructs NTRU-OQXT deploys some efficient primitives used in the design of Falcon (see Figure 1); thereby leverages their security guarantees. NTRU-OQXT is built on structured NTRU lattice primarily to reduce the size of cross-tags (stored at the server) and search tokens (sent during a conjunctive search), thereby reducing the overall storage and communication overhead of OQXT significantly. For this, we draw a mapping from Falcon's public/private keys to corresponding elements in OQXT (public matrix $\mathbf{z}_{\mathsf{w}}$ and secret trapdoor $\mathbf{T}$) and devise the public/private matrix in NTRU-OQXT. The trapdoor sampler deployed in OQXT is not directly compatible with NTRU lattices, hence, the efficient fast Fourier trapdoor sampler (used to generate short signatures in Falcon) is deployed to generate short vectors used for oblivious cross-tag generation at the server during a conjunctive search. This trapdoor sampler works significantly faster in NTRU lattices using efficient NTT transformations, thereby allowing NTRU-OQXT to exhibit extremely efficient polynomial multiplications with significantly reduced parameter sizes as compared to OQXT.

Security Analysis of NTRU-OQXT. We present a detailed analysis of the leakage profile for NTRU-OQXT, and then provide formal proof that this is indeed the leakage incurred by NTRU-OQXT against a semi-honest server (while the server is classical, the adversary corrupting the server is allowed to have quantum computing 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 ring-LWR problem over NTRU lattices (this in turn implies the security of the Falcon signature scheme). 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 analysis and security proof are presented in Section 5.

Extension to Disjunctive Queries. While NTRU-OQXT achieves significant efficiency gains over OQXT, it still only supports conjunctive keyword queries. Our goal, however, is to achieve a lattice-based SSE scheme that supports both conjunctive and disjunctive keyword queries in a practically efficient manner. Towards this end, we propose a novel integration of our NTRU-OQXT scheme into the TWINSSE framework of [BTR+23]. The resulting scheme, which we call NTRU-TWINSSE , is (to the best of our knowledge) the *first* SSE scheme that supports both conjunctive and disjunctive keyword queries, while attaining linear storage overheads and sub-linear query complexity, and while relying on the hardness of plausibly quantum-safe hardness assumptions over NTRU lattices. In particular, NTRU-TWINSSE offers a quantum-safe alternative to the original $\text{TWINSSE}_{\text{OXT}}$ scheme, which is quantum-unsafe due to its inherent reliance on discrete log-hard groups.

The detailed construction and analysis of NTRU-TWINSSE appears in Section 6. Note that the correctness and security guarantees of NTRU-TWINSSE follow directly from the correctness and security of NTRU-OQXT, with no additional assumptions necessitated by the integration into the TWINSSE framework.

Performance Overhead and Experimental Evaluation. We practically validate the efficiency improvements of NTRU-OQXT over OQXT by presenting a prototype implementation of NTRU-OQXT and evaluating its search performance over the Enron email corpus[4]. We also present a prototype implementation of NTRU-TWINSSE and compare its search performance and storage overheads (with respect to the same Enron email corpus) with those of [CJJ+13, KM17, PPSY21, TPM23]. The implementation and experiments are described in Section 7.

Our experiments validate that, in NTRU-OQXT, the server-side storage requirement grows linearly with the number of keyword-document pairs in the database, which is asymptotically optimal, and better than the quadratic overheads incurred by [KM17, PPSY21]. Practically, NTRU-OQXT reduces the storage overhead incurred by OQXT by several orders of magnitude (for example, the RAM requirement is reduced from gigabytes to megabytes). These improvements directly translate to the case of NTRU-TWINSSE , which substantially outperforms [KM17, PPSY21] in terms of storage requirements, and approaches the practical search efficiency of the $\text{TWINSSE}_{\text{OXT}}$ scheme from [BTR+23], while additionally achieving security based on plausibly quantum-safe hardness assumptions over NTRU lattices (unlike $\text{TWINSSE}_{\text{OXT}}$, which inherits reliance on discrete log-hard groups from OXT, and is quantum-unsafe).

Ethics Discussion. We believe that it is justifiable to use the Enron dataset for our experiments given: (i) the lack of alternative realistic datasets for conducting experiments, (ii) the extensive usage of this dataset for experiments in prior SSE literature [IKK12, BKM20, OK21], (iii) the fact that the data has long already been public, and (iv) there has been an effort by the researchers curating the version of the dataset used in this paper to

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

remove users upon request.[5]

3 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 brief discussion on some lattice preliminaries. The GPV signature scheme (Section 3.3), FALCON signature scheme (Section 3.4) and additional background material on basic crypto primitives (Section 3.5) and SSE (Section 3.6) are presented to provide an overview of the background.

3.1 Notations

Table 1 summarizes the notations and symbols used in the paper.

Conjunctive and Disjunctive Queries. We represent a conjunctive query over n distinct keywords $\mathsf{w}, \ldots, \mathsf{w_n}$ as $query = \mathsf{w} \wedge \ldots \wedge \mathsf{w_n}$ and define the set $\mathbf{DB}(query)$ as $\mathbf{DB}(query) = \cap_{i=1}^{\mathsf{n}} \mathbf{DB}(\mathsf{w}_i)$. Similarly, we represent a disjunctive query over n distinct keywords $\mathsf{w}_1, \ldots, \mathsf{w_n}$ as $query = (\mathsf{w}_1 \vee \ldots \vee \mathsf{w_n})$ and define the set $\mathbf{DB}(query)$ as $\mathbf{DB}(query) = \cup_{i=1}^{\mathsf{n}} \mathbf{DB}(\mathsf{w}_i)$.

Lattice Notations. We use rounding parameters and statistical errors as mentioned in the respective implementations that we incorporate in this paper [BPR12, PFH+17]. 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 $\mathbb{Z}$. 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.

3.2 Lattice Preliminaries

Ring Learning With Rounding Problem (ring-LWR). Let $n \geq 1$ be the main security parameter and moduli $q \geq p \geq 2$ be integers. Let R_q and R_p denote polynomial rings $\mathbb{Z}_q[x]/(\phi)$ and $\mathbb{Z}_p[x]/(\phi)$ respectively, for $\phi = x^n + 1$. For a vector $s \in R_q$, the ring-LWR distribution L is defined as the distribution over $R_q \times R_p$ obtained by choosing a vector $a \xleftarrow{\$} R_q$ uniformly at random and outputting $(a, b = \lfloor \langle a, s \rangle \rceil_p)$ [BPR12]. For a given distribution over $s \in R_q$ the *decision*-LWR$_{R,q,p}$ problem is to distinguish (with advantage

[5] See `https://www.cs.cmu.edu/~enron/` for detailed documentation.

Notations	Meaning
n	security parameter
id	document identifier
w	a keyword
N	total number of keywords in the database
$\mathcal{W}$	dictionary of keywords $\mathcal{W} = \{\mathsf{w}_1, \ldots, \mathsf{w}_N\}$
$\mathbf{DB}$	database $(\mathsf{id}_i, \mathsf{w}_i)_{i=1}^{d} \in \mathbf{DB}$
$\lvert\mathbf{DB}(\mathsf{w})\rvert$	number of documents containing w
n	max. number of keywords per conjunctive query.
$\mathcal{R}_{query}$	$= \mathbf{DB}(query)$ result set returned to the client after a *query*.
$x \xleftarrow{\$} \chi$	uniformly sampling x from χ
$x = \mathcal{A}$	x is output of a deterministic algorithm
$x \leftarrow \mathcal{A}'$	x is output of a randomized algorithm
$\mathbb{Z}_q^*$	multiplicative group modulo some prime q
$\mathbb{Z}_q$	quotient ring of integers modulo some prime q
F	Pseudo-random function with output range in $\{0,1\}^\lambda$
F'	Pseudo-random function with output range in a multiplicative group modulo some prime

Table 1: Notations

non-negligible in n) between any desired number of independent samples $(a_i, b_i) \xleftarrow{\$} L$, and the same number of samples drawn uniformly and independently from $R_q \times R_p$.

NTRU **Lattices.** The NTRU lattices [HPS98] incorporates a ring structure that reduces the public keys' size by a factor $O(n)$, while significantly accelerating the computations by a factor at least $O(n/logn)$. Let $\phi = x^n + 1$ for $n = 2^k$ a power of two, and $q \in \mathbb{N}^*$.

1. Private Keys: Consists of a set of four polynomials $f, g, F, G \in \mathbb{Z}[x]/(\phi)$. The polynomials satisfy the NTRU equation -

$$fG - gF = q \ mod \ \phi$$

2. Public Key: In the above equation, if f is invertible modulo q a polynomial h is defined as $h \leftarrow g' \cdot f^{-1} \bmod q$. The polynomial h is considered the public key.

The NTRU *assumption* corresponds to the hardness of solving the problem of finding small polynomials f', g', such that $h \leftarrow g' \cdot f'^{-1} \bmod q$.

SIS Problem On NTRU **Lattice (**NTRU-**SIS).** Given parameters n, m, q, a uniformly random polynomial $h \in \mathbb{Z}_q[x]/(\phi)$, and a real number $\beta > 0$, the NTRU-SIS problem is defined as follows: To find two non-zero small polynomials $(u, v) \in \mathbb{Z}_q^2[x]/(\phi)$ satisfying $u + v \cdot h = 0 \bmod q$ and $||u||, \ ||v|| \leq \beta$.

NTRU-*SIS Assumption.* Given a system linear equations modulo q, without any information about the trapdoor, the advantage of finding two non-zero small polynomials $(u, v) \in \mathbb{Z}_q^2[x]/(\phi)$ such that $u + v \cdot h = 0 \bmod q$ and $||u||, \ ||v|| \leq \beta$ can be neglected for any probabilistic polynomial time (PPT) algorithm.

3.3 The GPV Signature Scheme

GPV [GPV08] is a hash-and-sign lattice-based signature scheme can be described briefly as follows -

1. **Public key**: A full-rank matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ $(m > n)$ generating a q-ary lattice Λ.
2. **Private key:** A matrix $\mathbf{B} \in \mathbb{Z}_q^{m \times m}$ generating a lattice $\Lambda_q^{\perp}$ which is orthogonal to Λ modulo q. Equivalently, the rows of $\mathbf{A}$ and $\mathbf{B}$ are pairwise orthogonal i.e., $\mathbf{B} \times \mathbf{A^t} = 0$.
3. **Signature Generation:** Given a message m, a signature of m is a *short* vector $\sigma \in \mathbb{Z}_q^m$ such that, $\sigma \mathbf{A^t} = \mathtt{H}(\mathsf{m})$, where $\mathtt{H}$ is a hash function, $\mathtt{H} : \{0,1\}^* \rightarrow \mathbb{Z}_q^n$. For computing a valid signature, a pre-image $\mathbf{c_0} \in \mathbb{Z}_q^m$ is computed such that $\mathbf{c_0} \mathbf{A^t} = \mathtt{H}(\mathsf{m})$ (here, $\mathbf{c_0}$ is not required to be short). The private key $\mathbf{B}$ is then used to compute a vector $\mathbf{v} \in \Lambda_q^{\perp}$ close to $\mathbf{c_0}$. The difference $\sigma = \mathbf{c_0} - \mathbf{v}$ is a valid signature, and if $\mathbf{c_0}$ and $\mathbf{v}$ are close, then σ is *short*.
4. **Signature Verification:** Given $\mathbf{A}$ to verify whether a signature σ is a valid signature, it is required to be *short* and should satisfy $\sigma \mathbf{A^t} = \mathtt{H}(\mathsf{m})$.

3.4 The Falcon Signature Scheme

Falcon [PFH+17] is based on the GPV [GPV08] framework which is a hash-and-sign lattice-based signature scheme. To obtain a compact instantiation of the GPV framework, Falcon chooses a class of cryptographic lattices - NTRU lattices. The trapdoor sampler used in Falcon is known as the Fast Fourier Sampling. We outline briefly the steps followed in Falcon below -

1. **Public key**: It is a basis for a lattice of dimension $2n$: $\left[\begin{array}{c|c} -h & I_n \\ \hline qI_n & O_n \end{array}\right]$ where, I_n is an n-dimension identity matrix, O_n is a zero-matrix and h is a polynomial modulo ϕ that stands for an $n \times n$ sub-matrix with integer coefficients that range between 0 to $q-1$ (q is a small prime).
2. **Private key**: It is a basis for the same lattice but with smaller entries. $\left[\begin{array}{c|c} g & -f \\ \hline G & -F \end{array}\right]$ where, f, g, F and G are short integer polynomials modulo ϕ, such that - $h = g/f \ mod \ \phi \ mod \ q$; $fG - gF = q \ mod \ \phi$.
3. **Key pair generation**: The `KeyGen` algorithm takes as argument a monic polynomial $\phi \in \mathbb{Z}[x]$ and a modulus q. The final result of the algorithm is a pair of secret key sk and a public key pk ($h = gf^{-1} \bmod q$).
4. **Signature Generation**: The `Sign` algorithm takes as input a message m, a secret key sk and a bound $\lfloor \beta^2 \rfloor$. It consists of the following steps -
   (i) Hash the message m along with a random nonce r to sign into a polynomial (using the `HashToPoint` function) - $c \in \mathbb{Z}_q[x]/(\phi)$ modulo ϕ.
   (ii) The secret key (f, g, F, G) is used by the signer to produce a pair of *short* polynomials (s_1, s_2) such that, $s_1 = c - s_2 h \ mod \ \phi \ mod \ q$.
   (iii) A trapdoor sampler called fast Fourier sampling, is used to generate the short polynomials (s_1, s_2) without leaking the private key. s_2 is compressed to a bit-string s and the final signature comprise of (r, s) (r is a random salt added to the message before hashing it).

5. **Signature Verification** The `Verify` algorithm takes as input a message m, a signature $\mathsf{sig} = (\mathsf{r}, \mathsf{s})$, a public key $\mathsf{pk} = h \in \mathbb{Z}_q[x]/(\phi)$ and a bound $\lfloor \beta^2 \rfloor$. It follows the following steps -

   (i) Hash the message m along with a random nonce r to sign into a polynomial (`HashToPoint` function) - $c \in \mathbb{Z}_q[x]/(\phi)$ modulo ϕ.

   (ii) s is decompressed to a polynomial $s_2 \in \mathbb{Z}[x]/(\phi)$.

   (iii) Compute $s_1 = c - s_2 h \ mod \ q$.

   (iv) Check if $||(s_1, s_2)^2|| \leq \lfloor \beta^2 \rfloor$, ($\lfloor \beta^2 \rfloor$ is an acceptance bound) the signature is accepted, otherwise it is rejected.

Parameter Choices and Security Analysis. Standardized variants of FALCON use one of two possible ring degrees: either $n = 512$ (corresponds to NIST security level-I) or $n = 1024$ (corresponds to NIST security level-V). The modulus is fixed to a prime of the from $q = k \cdot 2n + 1$, concretely, $q = 12289$. This specific choice of modulus enables efficient number-theoretic transform (NTT) operations while resisting hybrid attacks. The trapdoor sampler samples signatures proportional to a bounded Gram-Schmidt norm of the secret basis ($\|B\|_{\mathrm{GS}} \leq 1.17\sqrt{q}$). [GJK24] provides the first formal analysis of the (strong) existential unforgeability security guarantees of FALCON in the Random Oracle Model (ROM) based on variants of the inhomogeneous SIS problem. Their analysis establishes that the standardized parameter sets for FALCON-512 and FALCON-1024 achieve $\approx$ 120-bits of security and 256-bits of security, respectively.

3.5 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 $|\gamma_0 - \gamma_1| \leq \mathsf{negl}(\lambda)$.

3.6 SSE: Syntax and Security Model

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

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 $\mathcal{L}$.

Let $\Pi = (\mathsf{EDBSetup}, \mathsf{Search})$ 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 $(K, \mathbf{EDB}) \leftarrow \mathsf{EDBSetup}(\mathbf{DB})$, 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.

4 NTRU-OQXT

In this section, we explain the technical details of our optimized post-quantum secure SSE, which we call, NTRU-OQXT. The construction is essentially an improvement of the existing construction of OQXT with a shift to structured NTRU lattices for compactness and efficiency. The translation from unstructured lattices to NTRU is not straightforward and we elucidate the core technical contribution in detail subsequently. To motivate the premise of our work we begin by discussing a brief overview of the quantum-safe SSE scheme OQXT.

4.1 Revisiting OQXT

OQXT [TPM23] is a lattice-based quantum-safe conjunctive SSE scheme, that elevates the classically secure conjunctive SSE, OXT [CJJ+13] to a post-quantum secure paradigm by relying on hard lattices over which the *Learning with Rounding* (LWR) assumption holds (replacing OXT's reliance on a discrete-log hard group, the public-key component that is quantum unsafe). The premise of OQXT can be connected to the GPV hash-and-sign lattice-based signature scheme that essentially relies on the SIS problem to derive its security guarantees.

LWR-Based Cross-Tags. OQXT entails a cross-tag generation protocol with provable security guarantees derived from post-quantum secure hardness assumptions of the LWR 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 a pseudorandom function. In particular, for some $p < q$, the elements of $\mathbb{Z}_q$ are divided into p contiguous intervals of roughly q/p elements each. Both q and p are considered as powers of 2, therefore, this results **xtag** to be the $log(p)$ most significant bits of $[\mathbf{xid} \cdot \mathbf{xw}]$.

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

The **xtag** constitutes of two components - the public matrix **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.

Lattice Trapdoors For Oblivious Computations. The correlation of OQXT with the theoretical framework of GPV [GPV08] that derives its security guarantees from the hardness of SIS problem is motivated here. Essentially, the crux of GPV signature scheme is that given the hash of a message, $\mathtt{H}(\mathsf{m})$ and a public matrix $\mathbf{A}$, the signature of m is a *short* vector σ such that $\sigma\mathbf{A^t} = \mathtt{H}(\mathsf{m})$; since σ is short, by the hardness of the SIS problem, it is difficult to find σ given the public matrix and the message without any information about the secret key (trapdoor).

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 of OQXT. It incorporates the SampleD function from [MP12] to sample a (*short*) pre-image from a given coset of a discrete Gaussian distribution. A short $\mathbf{y}_{\mathsf{id}}$ is appropriately generated from a (nearly spherical) *discrete Gaussian* distribution $\mathbb{D}_{\Lambda^{\perp}_{\mathbf{xid}}(\mathbf{z_w}),s}$ to ensure that the output *syndrome* (**xid**) is uniformly random i.e., for any random $\mathbf{y}_{\mathsf{id}}$, the product $\mathbf{y}_{\mathsf{id}} \cdot \mathbf{z_w^t}$ should be uniformly random. The hardness assumption here is given $\mathbf{z_w}$ (a public matrix) and the coset $\mathbf{xid} \in \mathbb{Z}_q^{n\times n}$, it is hard to sample a *short* $\mathbf{y}_{\mathsf{id}} \in \mathbb{Z}_q^{n\times m}$ from $\Lambda^{\perp}_{\mathbf{xid}}(\mathbf{z_w})$ (without any information about the secret trapdoor $\mathbf{T}$) by the hardness of *Short Integer Solution* (SIS) problem.

Limitations of OQXT**.** To achieve post-quantum security guarantees, lattice-based constructions rely upon matrices of large dimensions due to which the storage overhead and pre-processing time of OQXT grows significantly with the increase in plaintext database size. As a result, although the storage overhead of OQXT scales linearly with the size of the database, it incurs significant storage cost as well as pre-processing time as compared to IEX-2Lev [KM17] and ConjFilter [PPSY21] (both plausibly quantum-safe SSE schemes

for disjunctive and conjunctive queries respectively, based on purely symmetric-key primitives), whose storage overheads are quadratic in the number of keywords in the database. We observe that OQXT is built over unstructured integer lattices ($\mathbb{Z}_q^n$). This inherently requires the LWR (cross-tags) and SIS (trapdoor) matrices to be of large dimensions. Since each such matrix is generated for millions of keyword-document pairs in a database, the storage requirement blows-up significantly. This limits OQXT's scalability and practical deployability, especially for extremely large databases with millions of keywords. In this work we address this issue in OQXT which is otherwise *crucial* for a practically deployable and scalable quantum-safe SSE scheme.

4.2 Core Idea

We propose NTRU-OQXT, an optimized, extremely efficient, and scalable construction in the structured NTRU lattice, as a potential solution to the issues encountered in the construction of OQXT. We emphasize that our improved construction caters to both algorithmic as well as implementation-specific optimizations.

Switching To NTRU **Lattice.** The NTRU [HPS98] lattices incorporate a ring structure that reduces the size of the public-key by orders of magnitude and leads to more structured and compact instantiations. We transform the framework of OQXT to more compact ring structures by implementing the entire construction over NTRU lattices. The construction of LWR matrices for computing the cross-tags and lattice trapdoors incur huge storage overheads in OQXT due to their large dimensions, leading to several gigabytes of memory storage. This restricts OQXT from scaling to large real-world databases. We, therefore, elevate the unstructured integer lattice framework to more compact and structured NTRU lattices that reduce the storage requirement by several orders of magnitude (from gigabytes to megabytes of RAM storage).

The GPV **Framework over** NTRU **Lattices.**

Several theoretical instantiations of the GPV framework over NTRU lattices have been proposed in the literature [SS11, DLP14, DP16]. A highly optimized practical instantiation of the GPV framework over NTRU lattices is proposed recently in Falcon [PFH+17] (a NIST standard for quantum-safe digital signatures). In NTRU-OQXT we incorporate a highly optimized and efficient instantiation of GPV framework over NTRU, motivated by Falcon's construction. The reason for this shift is because the trapdoor sampling algorithm [MP12] used in OQXT is not straightforwardly compatible with NTRU and does not achieve the same level of compactness as NTRU. Furthermore, Falcon uses an extremely efficient trapdoor sampling function called *Fast Fourier Sampling*; it is based on a randomized variant of the fast Fourier nearest plane algorithm proposed in [DP16] which combines the quality of Klein's [Kle00] algorithm, the efficiency of Peikert's [MP12] and is compatible with NTRU lattices.

Cross-tags in NTRU **lattice.** The translation of cross-tag elements from unstructured integer lattices i.e. elements in $\mathbb{Z}_q^n \times \mathbb{Z}_p$ to NTRU lattices i.e. elements in polynomial ring $\mathbb{Z}_q[x]/(\phi) \times \mathbb{Z}_p/(\phi)$, (where $\phi = x^n + 1$ and n is the degree of the polynomial) is not straightforward. The novelty of NTRU-OQXT lies in devising cross-tags as LWR samples

in the polynomial ring. The coefficients of the polynomials are stored in the form of vectors (cross-tags are now stored as vectors of dimension n) which reduces the storage overhead considerably, unlike OQXT where cross-tags are matrices of large dimensions. This design choice not only makes NTRU-OQXT scalable to large databases but also enhances its performance efficiency significantly due to fast (FFT and NTT) operations in polynomial rings. As inferred from [TPM23] a cross-tag (**xtag**) is essentially a combination of a keyword (w) and the corresponding document (id) in which it occurs. As such in NTRU-OQXT we hash a keyword and the document in which it occurs to a point in the polynomial ring ($\mathbf{xw} = \mathtt{H}(\mathsf{w})$ and $\mathbf{xid}' = \mathtt{H}(\mathsf{id}||r)$ respectively, where r is a random salt) using the `HashToPoint` function (Section 3.4). We generate a short signature (s_1, s_2) of the hashed document $\mathbf{xid}'$ using FALCON's Sign algorithm and compute $\mathbf{xid} = \mathbf{xid}' - s_1$ and $\mathbf{y}_{\mathsf{id}} = s_2 \cdot m^{-1}$ (where $m \leftarrow \mathtt{HashToPoint}(F'(K_Z, \mathsf{w}||c))$ is computed for every keyword)[6]. **xtag** is computed by multiplying the two polynomials (**xid** and **xw**) and rounding the product to an LWR sample in $\mathbb{Z}_p[x]/(\phi)$ (where $q > p$).

4.3 Technical Details

We propose NTRU-OQXT, a client-server protocol for single-round conjunctive query processing over encrypted real-world databases. The protocol consists of two phases: the encrypted database generation phase (NTRU-$\mathsf{OQXT.Setup}$) and the search phase (NTRU-$\mathsf{OQXT.Search}$).

In the setup phase, the client encrypts data using a symmetric-key algorithm, generates an encrypted search index (TSet and XSet), and offloads it to the server. The TSet stores masked *short* signatures and encrypted document identifiers, while the XSet contains cross-tags (ring-LWR samples) for keyword-document pairs.

During the search phase, the client generates search tokens (**xtoken**s), sends them to the server, and the server computes **xtag**s via a double-rounding process. The correctness of this process depends on appropriate parameter choices ($q > p_1 > p$). The server matches **xtag**s in the XSet and returns the corresponding encrypted document identifiers to the client. We explain each algorithm explicitly subsequently.

Modified Cross-Tags in the Polynomial Ring. The cross-tag elements consist of two components, a keyword and the document in which it occurs. The document identifiers (id) are considered an analog to the secret message in FALCON. A document identifier is hashed to a point ($\mathbf{xid}' = \mathtt{H}(\mathsf{id}||r)$ where r is a random salt) in $\mathbb{Z}_{p_1}[x]/(\phi)$ (using `HashToPoint` function, $\mathtt{H}(\cdot)$ [PFH+17]). Similarly, a keyword is hashed to a point ($\mathbf{xw} = \mathtt{H}(\mathsf{w})$) in $\mathbb{Z}_q[x]/(\phi)$. A short signature (s_1, s_2) is generated using FALCON's Sign algorithm and $\mathbf{xid} = \mathbf{xid}' - s_1$ is computed. The **xtag** is computed by multiplying the two polynomials (**xw** and **xid**) and rounding the product up to $log(p)$ most significant bits (where $q > p$). By the hardness of ring-LWR assumption, **xtag**s are indistinguishable from random elements in the $\mathbb{Z}_p[x]/(\phi)$.

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

Trapdoor Sampler From FALCON Specifications. To delegate the oblivious compu-

[6] m^{-1} exists in $\mathbb{Z}_{p_1}[x]/(\phi)$.

tation 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 sampler devised in [PFH+17] which involves a *Fast Fourier Sampling* algorithm. The trapdoor sampling algorithm is leveraged while computing the signature of a hashed message in FALCON. While translating NTRU-OQXT with specifications of FALCON we inculcate the following modifications to the original OQXT framework. The idea is to generate a random matrix $\mathbf{z} \in \mathbb{Z}_{p_1}[x]/(\phi)$ and a trapdoor $\mathbf{T}$ for $\mathbf{z}$ using special lattice trapdoor function incorporated in (Sign [PFH+17]). The significance of using the Sign algorithm is to generate *short* signature (s_1, s_2) for every keyword-document pair; compute $\mathbf{y}_{\mathsf{id}} = s_2 \cdot m^{-1}$ ($m \leftarrow \texttt{HashToPoint}(F'(K_Z, \mathsf{w}||c))$ is a mask generated for every keyword-document pair where m^{-1} exists in $\mathbb{Z}_{p_1}[x]/(\phi)$). The exact mapping of elements of OQXT to FALCON specifications is explained in details below -

1. Public matrix $\mathbf{z}$: Mapped to the public key which is a basis for a lattice of dimension $2n$: $\left[\begin{array}{c|c} -h & I_n \\ \hline qI_n & O_n \end{array}\right]$ where, h is a polynomial modulo ϕ that stands for an $n \times n$ submatrix with integer coefficients that range between 0 to $p_1 - 1$ (p_1 is a small prime).
2. The trapdoor (secret) $\mathbf{T}$: Private key which is also a basis for the same lattice but with smaller entries. $\left[\begin{array}{c|c} g & -f \\ \hline G & -F \end{array}\right]$ where, f, g, F and G are short integer polynomials modulo ϕ, such that -
$$h = g/f \ mod \ \phi \ mod \ p_1$$
$$fG - gF = p_1 \ mod \ \phi$$
3. The document identifier id is analogous to the secret message m.
4. $\mathbf{y}_{\mathsf{id}}$ is computed as a masked *short* signature $(s_2) \in \mathbb{Z}_{p_1}[x]/(\phi)$ ($\mathbf{y}_{\mathsf{id}} = s_2 \cdot m^{-1}$ of the message m (here id) which is sampled using the optimized trapdoor sampler using fast Fourier Sampling.

To guarantee security against forgery of signature FALCON relies on the NTRU-SIS hardness assumption which states that given the hash of the message and the public key, it is difficult to guess the signature without the knowledge of the secret key (trapdoor).

$$s_1 + s_2 h = \mathtt{H}(\mathsf{r}||\mathsf{m})$$

On a similar note, we say that NTRU-OQXT relies on the hardness on NTRU-SIS to guarantee that given a public matrix $\mathbf{z}$, the hash of the document-identifier $\mathbf{xid}'$, it is difficult to find a *short* (s_1, s_2) without the knowledge of the trapdoor $\mathbf{T}$ i.e,

$$s_1 + s_2 \cdot \mathbf{z} = \mathbf{xid}'$$

NTRU-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 OQXT, **EDB** comprises of two structures, the XSet and the TSet. We use a similar structure of **EDB** for

Algorithm 1 NTRU-OQXT.Setup

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

Parameters: $q > p_1 > p$
function NTRU-OQXT.Setup($1^n, \phi, \mathbf{DB}$)
  Initialise $T \leftarrow \{\}$ indexed by keywords $\mathcal{W}$
  Select key K_S for PRF F
  Select key K_Z for PRF F' ▷ *To sample pseudo-random elements from* $\mathbb{Z}^*_{p_1}$
  Initialise $\mathbf{EDB} \leftarrow \{\}$
  Initialise $\mathsf{XSet} \leftarrow \{\}$
  $\mathbf{z}, \mathbf{T} \leftarrow \mathsf{KeyGen}(\phi, p_1)$ ▷ KeyGen *[PFH+17]*
  for $\mathsf{w} \in \mathcal{W}$ **do**
    Initialise $\mathbf{tset} \leftarrow \{\}$
    Compute $K_e \leftarrow F(K_S, \mathsf{w})$
    Hash w to a point in $\mathbb{Z}_q[x]/(\phi)$ using $\mathsf{HashToPoint}(\mathsf{w}, q, n)$
    $\mathbf{xw} \leftarrow \mathtt{H}(\mathsf{w}, q, n)/(\phi)$ ▷ $\mathtt{H}(\cdot) \rightarrow$ HashToPoint *[PFH+17]*
    Set counter $c \leftarrow 0$
    for $\mathsf{id} \in \mathbf{DB}(\mathsf{w})$ **do**
      $m \leftarrow \mathtt{H}(F'(K_Z, \mathsf{w}||c), p_1, n)/(\phi)$ ▷ $\mathtt{H}(\cdot) \rightarrow$ HashToPoint *[PFH+17]*
      Hash (id) to a point in $\mathbb{Z}_{p_1}[x]/(\phi)$ using $\mathsf{HashToPoint}(\mathsf{id}, p_1, n)$
      $\mathbf{xid}' \leftarrow \mathtt{H}(\mathsf{id}, p_1, n)/(\phi)$ ▷ $\mathtt{H}(\cdot) \rightarrow$ HashToPoint *[PFH+17]*
      $(s_1, s_2) \leftarrow \mathsf{Sign}(\mathbf{xid}', \mathbf{T}, \lfloor \beta^2 \rfloor)$ ▷ Sign *[PFH+17]*
      $\mathbf{xid} \leftarrow \mathbf{xid}' - s_1$
      $\mathbf{yid} \leftarrow s_2 \cdot m^{-1}$ ▷ m^{-1} *exists in* $\mathbb{Z}_{p_1}[x]/(\phi)$
      Compute $e_c \leftarrow Sym.Enc(K_e, \mathsf{id})$
      Append $(\mathbf{y}_{\mathsf{id}}, e_c)$ to $\mathbf{tset}$
      $\mathbf{xtag} \leftarrow \lfloor \mathbf{xid} \cdot \mathbf{xw} \rceil_p$
      Add $\mathbf{xtag}$ to XSet
      $c \leftarrow c + 1$
    Set $T[\mathsf{w}] \leftarrow \mathbf{tset}$
  Compute $\{\mathsf{TSet}, K_T\} \leftarrow \mathsf{TSet.SetUp}(T)$
  return $\mathsf{mk} = \{K_S, K_T\}$, $\mathbf{EDB} = (\mathsf{TSet}, \mathsf{XSet})$

NTRU-OQXT and formally elaborate the construction of XSet and TSet in Algorithm 1. The client generates *cross-tags* (ring-LWR samples in $\mathbb{Z}_q[x]/(\phi) \times \mathbb{Z}_p[x]/(\phi)$ (where $q > p$)) and stores them in XSet. The **xtag** values are rounded *once* from $\mathbb{Z}_q[x]/(\phi)$ to $\mathbb{Z}_p[x]/(\phi)$ during Setup. Also it samples a random public matrix $\mathbf{z}$ along with a secret trapdoor $\mathbf{T}$ once for every keyword, using the KeyGen algorithm of Falcon. A signature is generated for $\mathtt{H}(\mathsf{id})$. The same signature is generated for the same id present in different keywords (say $\mathsf{id} \in \mathsf{w}_1$ and $\mathsf{id} \in \mathsf{w}_2$), because we need to match the same id present in two different keywords during the construction of **xtag** in the Search phase. Generate $\mathbf{y}_{\mathsf{id}} \leftarrow s_2 \cdot m^{-1}$ ($m \leftarrow F'(K_Z, \mathsf{w}||c)$ is generated for every w-id pair where m^{-1} exists in $\mathbb{Z}_{p_1}[x]/(\phi)$) and c is a counter that records the frequency of each w. We store a masked s_2 at the server. Hence, even if the same s_2 has been generated for the same id for multiple keywords, it essentially doesn't leak anything to the server. The NTRU-SIS assumption that guarantees security of Falcon's Sign algorithm ensures that it is difficult to sample a *short* (s_1, s_2) for every keyword-id pair without the knowledge of the secret trapdoor $\mathbf{T}$. The client encrypts the documents (e_c) using an IND-CPA secure symmetric-key encryption scheme ($Sym.Enc$), like AES-GCM, and the encrypted documents are indexed by a (randomized) document

Algorithm 2 NTRU-OQXT.Search

Input: $\mathsf{mk}, \phi, query = (\mathsf{w}_1 \wedge \ldots \wedge \mathsf{w}_n), \mathbf{EDB}$
Output: Result R_{query}

Parameters: $q > p_1 > p$
function NTRU-OQXT.Search($\mathsf{mk}, \phi, query, \mathbf{EDB}$)
  Client's inputs are ($\mathsf{mk}, query$) and server's input ($\mathbf{EDB}$)
  Client
  $R_{query} \leftarrow \{\}$
  $\mathsf{stag} \leftarrow \mathsf{TSet.GetTag}(K_T, \mathsf{w}_1)$
  Sends stag to the server
  $\mathbf{z}, \mathbf{T} \leftarrow \mathsf{KeyGen}(\phi, p_1)$ ▷ KeyGen *[PFH+17]*
  Computes **xtoken** as follows:
  for $c = 1$: until server sends *stop* **do**
    $m \leftarrow \mathtt{H}(F'(K_Z, \mathsf{w}_1||c), p_1, n)/(\phi))$ ▷ $\mathtt{H}(\cdot) \rightarrow$ HashToPoint *[PFH+17]*
    for $l = 2 : \mathsf{n}$ **do**
      Hash w_l to a point in $\mathbb{Z}_q[x]/(\phi)$ using $\mathsf{HashToPoint}(\mathsf{w}_l, q, n)$
      $\mathbf{xw}_l \leftarrow \mathtt{H}(\mathsf{w}_l, q, n)/(\phi)$ ▷ $\mathtt{H}(\cdot) \rightarrow$ HashToPoint *[PFH+17]*
      $\mathbf{xtoken}'[c, l] \leftarrow \lfloor \mathbf{z} \cdot \mathbf{xw}_l \rceil_{p_1}$
      $\mathbf{xtoken}[c, l] \leftarrow \mathbf{xtoken}[c, l]' \cdot m$
    $\mathbf{xtoken}[c] \leftarrow (\mathbf{xtoken}[c, 2], \ldots, \mathbf{xtoken}[c, n])$
    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 $(\mathbf{y}_{\mathsf{id}}, e_c)$ from c-th component of $\mathbf{tset}$.
    for $l = 2 : \mathsf{n}$ **do**
      Server computes $\mathbf{xtag} = \lfloor \mathbf{y}_{\mathsf{id}} \cdot \mathbf{xtoken} \rceil_p$
    If $\forall l \in [2, \mathsf{n}]$, $\mathbf{xtag} \in \mathsf{XSet}$, then send e_c to the client.
    When last tuple in $\mathbf{tset}$ 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_{query} for all e_c received.
  return R_{query}

identifier. For every document containing a particular keyword w (i.e. every $\mathsf{id} \in \mathbf{DB}(\mathsf{w})$) it generates (s_1, s_2) by invoking the Sign routine and computes $\mathbf{y}_{\mathsf{id}} = s_2 \cdot m^{-1}$. Thereafter it stores the $(e_c, \mathbf{y}_{\mathsf{id}})$ pairs for every keyword-id pair in TSet using TSet.SetUp routine [CJJ+13].

Oblivious Conjunctive Search. During a conjunctive search client generates an stag corresponding to the least frequent queried keyword (by calling the TSet.GetTag routine [CJJ+13]). It retrieves all documents corresponding to this keyword using the stag. Next, for all the remaining queried keywords and retrieved document pairs it computes $\mathbf{xtoken}'$ as ring-LWR samples and sends them to the server.

$$\mathbf{xtoken}'[\mathsf{id}, \mathsf{w}] = \lfloor \mathbf{z} \cdot \mathbf{xw}_l \rceil_{p_1}$$

The computation of $\mathbf{xtoken}'$ interestingly combines the matrix $\mathbf{z}$ (in $\mathbb{Z}_{p_1}[x]/(\phi)$) and $\mathbf{xw}$ (in $\mathbb{Z}_q[x]/(\phi)$) and rounding the value to $\mathbb{Z}_{p_1}[x]/(\phi)$ where $q > p_1 > p$. This step is instrumental in the exact oblivious computation of the cross-tags at the server during a search.

$\mathbf{xtoken}'$ is then masked with the random mask m (same mask for w_1 that is used during the Setup phase) to generate **xtoken**. The $\mathbf{y}_{\mathsf{id}}$ values stored in the TSet are retrieved by the server (using the stag) and multiplied with **xtoken**.

Oblivious Cross-Tag Computation. The public matrix $\mathbf{z}$ is used to compute **xtoken** as mentioned above. The oblivious computation of **xtag** is crafted carefully by a *double-rounding* process. As shown, the **xtoken** values are rounded to $\mathbb{Z}_{p_1}[x]/(\phi)$ $(q > p_1)$ and masked. After that, **xtoken** is multiplied with $\mathbf{y}_{\mathsf{id}}$ for each (w-id) pair, the masks cancel out due to commutative multiplication in $\mathbb{Z}[x]/\phi(x)$ and the resultant product is rounded to $\mathbb{Z}_p[x]/(\phi)$ $(q > p_1 > p)$ to obtain correct **xtag** values without leaking any extra information to the server.

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

Remark. During the Setup phase the **xtag** is computed and rounded *once* to $\mathbb{Z}_p[x]/(\phi)$. During a Search phase the **xtag** value is obliviously computed by a *double-rounding*. Compute **xtoken** and round it to $\mathbb{Z}_{p_1}[x]/(\phi)$ first and then multiply the **xtoken** with the short signature $\mathbf{y}_{\mathsf{id}}$ and round the product to $\mathbb{Z}_p[x]/(\phi)$. This results in the same **xtag** value because of the choice of parameters $q > p_1 > p$. In particular, q and p are chosen such that their difference is large enough to accommodate the noise due to double rounding. We also validate this claim experimentally for the correct choice of parameters.

4.4 Proof Of Correctness of NTRU-OQXT

The proof of correctness of NTRU-OQXT follows from the correctness of KeyGen and Sign function from [PFH+17] along with the correctness of TSet instantiations from [CJJ+13] and for a correct choice of parameters $q > p_1 > p$. We state the following theorem for the correctness of NTRU − OQXT.

Theorem 1 (Correctness of NTRU − OQXT)**.** *Assuming that* KeyGen *generates the correct public and private keys in [PFH+17] and* Sign *generates a short signature correctly in [PFH+17], the* TSet *instantiations from [CJJ+13] works correctly, and for correctly chosen parameters $q > p_1 > p$,* NTRU − OQXT *satisfies correctness for an SSE scheme supporting conjunctive queries.*

We state the proof of correctness for OQXT below.

Proof. Correctness of a conjunctive SSE scheme implies that given a conjunctive query, $query = \mathsf{w}_1 \wedge \cdots \wedge \mathsf{w}_\mathsf{n}$ over an encrypted database, it should satisfy the following relations.

$$\begin{aligned} \mathbf{EDB} &= \mathsf{Setup}(\mathbf{DB}) \\ \mathbf{DB}(\mathsf{w}_1) \cap \cdots \cap \mathbf{DB}(\mathsf{w}_\mathsf{n}) &= \mathsf{Search}(query, \mathbf{EDB}) \end{aligned}$$

For the proof of correctness of NTRU-OQXT we consider the functions KeyGen and Sign to give the correct output. Considering KeyGen produces a random public matrix $\mathbf{z} \in \mathbb{Z}_{p_1}[x]/(\phi)$ with a trapdoor $\mathbf{T}$ (secret) correctly according to [PFH+17], such that the NTRU equation $h \leftarrow g^{-1} \cdot f$ (See Section 4.3 for the mapping of $\mathbf{z}$ and $\mathbf{T}$ to the

NTRU public-key h and secret keys (f, g, F, G) respectively, according to FALCON specifications) is satisfied. We sample a *short* signature, (s_1, s_2) using $\mathbf{T}$ from a coset defined by $\mathbf{xid}'$ of a discrete Gaussian using FALCON's Sign function, and we assume the correctness of the function follows from [PFH+17]. We then compute $\mathbf{y}_{\mathsf{id}} = s_2 \cdot m^{-1}$, where $m \leftarrow \mathtt{HashToPoint}(F'(K_Z, \mathsf{w}||c))$ is calculated for every keyword-id pair[7]. We carefully choose the parameters for rounding $q > p_1 > p$, such that the oblivious computation of an $\mathbf{xtag}$ $(\lfloor \mathbf{y}_{\mathsf{id}} \cdot \mathbf{xtoken} \rceil_p)$ during a conjunctive search gives an exact correct value with overwhelming probability. Under these assumptions and the correct parameter choice, we verify the correctness of NTRU-OQXT. Consider a conjunctive query, *query* as stated below.

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

According to NTRU-$\mathsf{OQXT}.\mathsf{Setup}$ the client generates a random public matrix $\mathbf{z} \in \mathbb{Z}_{p_1}[x]/(\phi)$ with a trapdoor $\mathbf{T}$ using the KeyGen function. It then generates *short* signature (s_1, s_2) for the secret hashed message $\mathbf{xid}' = \mathtt{H}(\mathsf{id}||r)$, computes $\mathbf{y}_{\mathsf{id}} = s_2.m^{-1}$ ($m \leftarrow \mathtt{HashToPoint}(F'(K_Z, \mathsf{w}||c))$ is calculated for every keyword), stores $\mathbf{y}_{\mathsf{id}}$ in the TSet and offloads it to the server. It also generates $\mathbf{xtag}$ as ring-LWR samples for every (w-id) pair, $\mathbf{xtag} = \lfloor \mathbf{xid} \cdot \mathbf{xw} \rceil_p$ (where, $\mathbf{xid} = \mathbf{xid}' - s_1$ and $\mathbf{xw} = \mathtt{H}(\mathsf{w})$) and stores these values in XSet. The $\mathbf{xtag}$ values are rounded (*once*) up to the most significant $log_2 p$ bits (where $q > p$), during the Setup phase. When *query* is received at the server, the server first finds $\mathbf{DB}(\mathsf{w}_1)$ using the stag, corresponding to the least frequent keyword (w_1) in the query. This follows from the correctness of the TSet. For every $\mathsf{id} \in \mathbf{DB}(\mathsf{w}_1)$ and each keyword in *query* (other than w_1), the client generates $\mathbf{xtoken}'[\mathsf{id}, \mathsf{w}] = \lfloor \mathbf{z} \cdot \mathbf{xw} \rceil_{p_1}$ where $\mathbf{z}$ is generated by KeyGen along with a trapdoor $\mathbf{T}$. An $\mathbf{xtoken}'$ is rounded to p_1, where $q > p_1 > p$. $\mathbf{xtoken}$ is then calculated as $\mathbf{xtoken} = \mathbf{xtoken}' \cdot m$, where $m \leftarrow \mathtt{HashToPoint}(F'(K_Z, \mathsf{w}||c)$ is calculated for w_1. The Search phase entails a *double-rounding* process to compute an exact $\mathbf{xtag}$ value. The server on receiving the rounded $\mathbf{xtoken}$ values (rounded to p_1) calculates $\mathbf{xtag}$ (rounded to p) by multiplying $\mathbf{xtoken}$ with $\mathbf{y}_{\mathsf{id}}$ (which it retrieves from TSet for w_1) and rounding the product as - $\mathbf{xtag} = \lfloor \mathbf{y}_{\mathsf{id}} \cdot \mathbf{xtoken} \rceil_p$. The correctness of this is ensured by the correctness of the Sign function, which in turn ensures that a *short* signature (s_1, s_2) is sampled from the coset specified by $\mathbf{xid}'$ of the discrete Gaussian distribution. Also because we are working in a commutative ring the mask m in $\mathbf{xtoken}$ and m^{-1} in $\mathbf{y}_{\mathsf{id}}$ cancels each other out. Another crucial factor for an exact $\mathbf{xtag}$ value computation is a correct choice of parameters $q > p_1 > p$. Since $\mathbf{y}_{\mathsf{id}}$ is short, we claim that multiplying noisy rounded $\mathbf{xtoken}$ (rounded to p_1) with short $\mathbf{y}_{\mathsf{id}}$ and rounding the product to p will give a correct $\mathbf{xtag}$ value (rounded to p) with overwhelming probability because the difference between p_1 and p chosen is large enough to eliminate the noise by double-rounding. The noise will not propagate if the product is correct towards the most significant $log_2 p$ bits (since we drop the least significant bits during rounding).

Asymptotic Analysis of Parameter Choice: To prove the correctness of NTRU-OQXT mathematically, the following must hold with non-negligible probability -

$$\begin{aligned} \lfloor \mathbf{y}_{\mathsf{id}} \cdot \mathbf{xtoken} \rceil_p &= \mathbf{xtag} \\ \lfloor s_2 \cdot m^{-1} \cdot \lfloor \mathbf{z} \cdot \mathbf{xw} \rceil_{p_1} \cdot m \rceil_p &= \lfloor \mathbf{xid} \cdot \mathbf{xw} \rceil_p \\ \lfloor s_2 \cdot \lfloor \mathbf{z} \cdot \mathbf{xw} \rceil_{p_1} \rceil_p &= \lfloor \mathbf{xid} \cdot \mathbf{xw} \rceil_p \end{aligned}$$

[By commutative multiplication in $\mathbb{Z}_{p_1}[x]/\phi(x)$]

[7] m^{-1} exists in $\mathbb{Z}_{p_1}[x]/(\phi)$.

The product on the LHS, $(\mathbf{y_{id}} \cdot \mathbf{xtoken})$ can be considered as repeated additions of each i-th component of $\mathbf{xtoken}$, $\mathbf{y_{id}}^i$ times. Now, since the l_2-norm, β, of $\mathbf{y_{id}}$ (essentially s_2)[8] is small (as guaranteed by FALCON [PFH+17]), we can say that, (total number of additions) $\leq \| \mathbf{y_{id}} \|_\infty \leq \frac{\beta^2}{\delta^2}$, where δ is a threshold which filters out values that have a negligible contribution to the addition[9]. Thus, each product $(\mathbf{y_{id}}^i \cdot \mathbf{xtoken}^i)$ can be viewed as at most $\frac{\beta^2}{\delta^2}$ additions, implying that the total number of effective additions to compute $\mathbf{xtag}^i$ is asymptotically bounded by $O(\frac{\beta^2}{\delta^2})$.[10] We note that, because FALCON uses efficient NTT transformation, a polynomial $f(x) \in \mathbb{Z}[x]/\phi(x)$ is transformed from its coefficient representation into a vector of evaluations at n-th roots of unity modulo ϕ. As a result, the modular reductions are handled during the NTT transformation itself. Thus, every polynomial multiplication in $\mathbb{Z}[x]/\phi(x)$ is simple coefficient-wise multiplication.

The product on the RHS is first computed and then rounded from q to p (where $q >> p$ and $q/p \geq \sqrt{n}$ for ring-LWR assumption to hold in an n degree polynomial). While on the LHS, we first round $\mathbf{xtoken}^i$ from q to p_1, multiply with $\mathbf{y}_{\mathsf{id}}^i$ (equivalent to $O(\frac{\beta^2}{\delta^2})$ additions as discussed above), and finally round the product to p (where $q > p_1 > p$ and $q/p_1 \geq \sqrt{n}$ for ring-LWR assumption to hold in an n degree polynomial). Notice that without the final rounding step on the LHS, while computing $(\mathbf{y_{id}}^i \cdot \mathbf{xtoken}^i)$ the least significant $log_2 q - log_2 p_1$ bits of $\mathbf{xtoken^i}$ do not participate in the addition (while on the RHS, the $log_2 q - log_2 p_1$ bits of $\mathbf{xw}$ or $\mathbf{xid}$ do participate). As a result, if one addition on the RHS introduces one carry bit in the least significant $log_2 q - log_2 p_1$ bits, it is propagated towards the most significant $log_2 p$ bits. In the worst case for $O(\frac{\beta^2}{\delta^2})$ additions, $O(\frac{\beta^2}{\delta^2})$ errors can be propagated from the least significant bits to the most significant bits on the RHS. This propagation will not be reflected in $(\mathbf{y_{id}}^i \cdot \mathbf{xtoken}^i)$ on the LHS, thereby resulting in a mismatch with the RHS. Therefore, in order for correctness to hold, where only the most significant $log_2 p$ bits of LHS are compared with the most significant $log_2 p$ bits of RHS, we add a final rounding step to the product $\mathbf{y_{id}}^i \cdot \mathbf{xtoken}^i$ on the LHS from $log_2 p_1$ to $log_2 p$. We also ensure that the propagation of $O(\frac{\beta^2}{\delta^2})$ errors must be subsumed by $log_2 p_1 - log_2 p$ bits, or are subsumed by the final polynomial reduction wrt. (ϕ).

Let $\Pi_{\text{NTRU}-\text{OQXT}}$ be a correct SSE scheme (i.e. the propagated $O(\frac{\beta^2}{\delta^2})$ errors are subsumed in $log_2 p_1 - log_2 p$ least significant bits, or by the final polynomial reduction wrt. $(\langle\rangle\phi)$) we show that -

$$\Pr[(\Pi_{\text{NTRU}-\text{OQXT}} = 1)] \geq \left(1 - \frac{O(\frac{\beta^2}{\delta^2})}{2^{log_2 p_1 - log_2 p}}\right)^n$$

where, $\left(\frac{O(\frac{\beta^2}{\delta^2})}{2^{(log_2 p_1 - log_2 p)}}\right)$ is negligible since $log_2 p_1 > log_2 p$; $\beta, \delta < 2^{(log_2 p_1 - log_2 p)}$; n is the

[8] In this analysis we will use $\mathbf{y_{id}}$ and s_2 interchangeably because essentially $\mathbf{y_{id}}$ is masked s_2 ($\mathbf{y_{id}} = s_2 \cdot m^{-1}$); the mask gets canceled out with $\mathbf{xtoken}' \cdot m$ because $\mathbb{Z}[x]/\phi(x)$ is a commutative ring.

[9] The above bound is obtained by Chebyshev's inequality that states, the number of components that significantly contribute to the dot product can be roughly bounded by how concentrated β is. Informally stating, a short l_2-norm implies short l_∞- norm.

[10] This gives a loose asymptotic bound on the total number of additions and assumes δ is chosen appropriately such that elements with magnitude less than δ are ignored, and β is sufficiently small.

degree of the polynomial.[11,12]

5 Security analysis of NTRU-OQXT

The security analysis of NTRU-OQXT follows from the security notions of FALCON [PFH+17] (that relies on ring-LWR and NTRU-SIS problems). We formally prove the post-quantum simulation security of NTRU-OQXT with respect to a rigorously defined and thoroughly analyzed leakage profile. The leakage profile of NTRU-OQXT is similar to OQXT. Our scheme ensures no extra information leakage occurs in addition to the information leaked in OQXT while ensuring security in the presence of an efficient and scalable quantum computer. We formalize the leakage profile and provide a detailed simulation-based proof of security of NTRU-OQXT.

5.1 Leakage Profile Analysis

The leakage profile of NTRU-OQXT is similar to original OQXT scheme. Our scheme ensures no extra information leakage occurs in addition to the information leaked in OQXT while exhibiting significant performance improvement.

We describe the function $\mathcal{L}_{\mathrm{NTRU-OQXT}}$ that captures the leakage profile of NTRU-OQXT under the scenario where all queries are non-adaptive. In addition to the leakage from TSet implementation $\mathcal{L}_T$ (described in [CJJ+13]), $\mathcal{L}_{\mathrm{NTRU-OQXT}}$ captures all of the information leaked by our quantum-safe SSE construction. We proceed similarly as in OQXT by representing a sequence of Q 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 $\mathcal{L}_{\mathrm{NTRU-OQXT}}(\mathbf{DB}, q)$ 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 $\mathcal{L}_{\mathrm{NTRU-OQXT}}$. The significance of each component of the leakage function in NTRU-OQXT is equivalent to that in OQXT. 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.
- $\bar{s} \in [m]^Q$ - equality pattern of $s \in \mathcal{W}^Q$ that indicates which queries have the equal stermss. Repetition of stermss of different queries is leaked by $\bar{s}$.

[11] Since $O(\frac{\beta^2}{\delta^2})$ gives a loose upper bound on the number of errors that can propagate towards the most significant $log_2 p$ bits of the RHS, we say that the equation above gives a loose lower bound on the probability of correctness.

[12] The $(\cdot)^n$ holds since the computation of each $\mathbf{xtag}^i$ is an independent event, so the above bound considers simultaneous correctness of *all* components of **xtag**.

- SP - size pattern of the queries i.e., the number of documents matching the sterm in each query. Formally, $\mathsf{SP} \in [d]^Q$ 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 $Q \times Q$ 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 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 2. *Let $\mathcal{L}_{\text{NTRU}-\text{OQXT}}$ be as defined above, and we assume that the TSet implementation $\sum$ from [CJJ+13] is secure. Then SSE scheme* NTRU-OQXT *is $\mathcal{L}_{\text{NTRU}-\text{OQXT}}$-post-quantum-secure against adaptive attacks, assuming that the ring-$\mathsf{LWR}_{n,p,q}$ problem and ring-$\mathsf{LWR}_{n,p_1,q}$ problem is hard in $\mathbb{Z}_q[x]/(\phi) \times \mathbb{Z}_p[x]/(\phi)$ and $\mathbb{Z}_q[x]/(\phi) \times \mathbb{Z}_{p_1}[x]/(\phi)$ respectively, that solving* $\text{NTRU} - \mathsf{SIS}_{n,m,q}$ *is hard i.e. given the* NTRU *public key sampling a short pre-image from the* NTRU *lattice without using a secret-key $\mathbf{T}$ is hard, that F and F' are secure PRFs, and (Enc, Dec) is an IND-CPA secure symmetric encryption scheme.*

Proof Overview. We begin by proving that NTRU-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 NTRU-OQXT (proof of theorem 2) can be elucidated in a similar tone as done is OQXT. The first step towards this would be to justify that the leakage captured by $\mathcal{L}_{\text{NTRU}-\text{OQXT}}$ is at least necessary for correct simulation. N or the total number of appearances of keywords in the database gives the size of the XSet, i.e. number of $\mathbf{xtag}$ entries for each (w-id) pair in the database. The equality pattern $\bar{s}$ is important as it indicates the queries with the same stags. This is required since the stags are deterministic the server can observe repetition in stags of different queries. 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 6. The proof of Theorem 2 is shown via hybrids in Section 5.2.

Extension to Adaptive Security. Our adaptive security proof for NTRU-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 NTRU-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 NTRU-OQXT initializes the XSet to consist entirely of uniformly random elements initially (while relying on the ring-LWR assumption for indistinguishability of the real and simulated XSet entries). Additionally, the simulator for NTRU-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.

5.2 Proof Of Theorem 2

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

Definition 1. *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 -*

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

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

$K_S, K_Z \xleftarrow{\$} \{0,1\}^\lambda$
$(\mathsf{id}_i, \mathcal{W}_i)_{i=1}^d \leftarrow \mathbf{DB}$
$\mathbf{z}, \mathbf{T} \leftarrow \mathsf{KeyGen}(\phi, p_1)$ ▷ KeyGen() [PFH+17]
For $\mathsf{w} \in \mathcal{W}$
  $(\mathsf{id}_1, \ldots, \mathsf{id}_{T_\mathsf{w}}) \leftarrow \mathbf{DB}(\mathsf{w})$
  $\sigma \leftarrow \mathsf{Perm}([T_\mathsf{w}]); \mathsf{WPerms}[\mathsf{w}] \leftarrow \sigma$
  $K_e \leftarrow F(K_S, \mathsf{w}); \mathbf{tset} \leftarrow \perp$
  For $c = 1, \ldots, T_w$ do
    $m \leftarrow F'(K_Z, \mathsf{w}||c)$
    $e_c \leftarrow Sym.Enc(K_e, \mathsf{id}_{\sigma(c)})$
    Hash id to a point in $\mathbb{Z}_{p_1}[x]/(\phi)$
    $\mathbf{xid}' \leftarrow \mathsf{H}(\mathsf{id}, p_1, n)$ ▷ H(·) → HashToPoint()/(ϕ) [PFH+17]
    $(s_1, s_2) \leftarrow \mathsf{Sign}(\mathbf{xid}', \mathbf{T}, \lfloor \beta^2 \rfloor)$ ▷ Sign() [PFH+17]
    $\mathbf{xid} \leftarrow \mathbf{xid}' - s_1$
    $\mathbf{y_c} \leftarrow s_2 \cdot m^{-1}$
    $\mathbf{tset}[c] \leftarrow (\mathbf{y_c}, e_c)$
  End
  $T[w] \leftarrow \mathbf{tset}$
End
$(\mathsf{TSet}, K_T) \leftarrow \mathsf{TSet.SetUp}(T)$
$\mathbf{EDB} \leftarrow (\mathsf{TSet}, \mathsf{XSet})$
For $i = 1, \ldots, Q$ do
  $\mathsf{STags}[i] \leftarrow \mathsf{TSet.GetTag}(K_T, s[i])$
$\mathsf{XSet} \leftarrow \mathsf{XSet.SetUp}(\mathbf{DB})$
For $i = 1, \ldots, Q$ do
  $\mathbf{Tr}[i] \leftarrow \mathsf{GenTrans}(K_Z, s[i], x[i], \mathsf{STags}[i])$
Return $(\mathbf{EDB}, \mathbf{Tr})$

$\underline{\mathsf{XSet.SetUp}(\mathbf{DB})}$ // G_0
$(\mathsf{id}_i, \mathcal{W}_i)_{i=1}^d \leftarrow \mathbf{DB}$
$\mathbf{z}, \mathbf{T} \leftarrow \mathsf{KeyGen}(\phi, p_1)$
For $\mathsf{w} \in \mathcal{W}$
  For $\mathsf{id} \in \mathbf{DB}(\mathsf{w})$; set $c = 0$, do
    $c \leftarrow c + 1$
    $\mathbf{xw} \leftarrow \mathsf{H}(\mathsf{w}, q, n)$
    $\mathbf{xid}' \leftarrow \mathsf{H}(\mathsf{id}, p_1, n)$
    $(s_1, s_2) \leftarrow \mathsf{Sign}(\mathbf{xid}', \mathbf{T}, \lfloor \beta^2 \rfloor)$
    $\mathbf{xid} \leftarrow \mathbf{xid}' - s_1$
    $\mathbf{xtag} \leftarrow \lfloor \mathbf{xid} \cdot \mathbf{xw} \rceil_p$
    $\mathsf{XSet} \leftarrow \mathsf{XSet} \cup \{\mathbf{xtag}\}$
End
Return XSet

$\underline{\mathsf{GenTrans}(\mathbf{EDB}, K_Z, s, x, \mathsf{stag})}$ // G_0
$\mathbf{z}, \mathbf{T} \leftarrow \mathsf{KeyGen}(\phi, p_1)$
$\mathbf{xw} \leftarrow \mathsf{H}(x, q, n)$
For $c = 1, \ldots, T$ do
  $m \leftarrow F'(K_Z, s||c)$
  $\mathbf{xtoken}'[c] \leftarrow \lfloor \mathbf{z} \cdot \mathbf{xw} \rceil_{p_1}$
  $\mathbf{xtoken}[c] \leftarrow \mathbf{xtoken}'[c] \cdot m$
$\mathsf{Res} \leftarrow \mathsf{Server.Search}(\mathbf{EDB}, (\mathsf{stag}, \mathbf{xtoken}))$
$\mathsf{ResInds} \leftarrow \mathbf{DB}(s) \cap \mathbf{DB}(x)$
Return $((\mathsf{stag}, \mathbf{xtoken}), \mathsf{Res}, \mathsf{ResInds})$

Table 2: Game G_0

0 In NTRU-OQXT the leakage function is defined by the $\mathcal{L}_{\mathrm{NTRU-OQXT}}$ 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
  $K \xleftarrow{\$} \{0,1\}^\lambda; \mathbf{tset} \leftarrow \perp$
  $m \xleftarrow{\$} \mathbb{Z}_{p_1}^*$
  For $c = 1, \ldots, T_\mathsf{w}$ do
    $(s_1, s_2) \xleftarrow{\$} \mathbb{Z}_{p_1}[x]/(\phi); \mathbf{y_c} \leftarrow s_2 \cdot m^{-1}$
    $e_c \leftarrow Sym.Enc(K, 0^\lambda)$;
    $\mathbf{tset}[c] \leftarrow (\mathbf{y_c}, e_c)$
  $T[\mathsf{w}] \leftarrow \mathbf{tset}$

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

Theorem 3. *Let $\mathcal{L}$ be the leakage function as defined above. The SSE scheme* NTRU-OQXT *is $\mathcal{L}$-post-quantum secure against non-adaptive attacks where all queries are 2-conjunctions, assuming that the ring-LWR$_{n,p,q}$ and ring-LWR$_{n,p_1,q}$ problem is hard in $\mathbb{Z}_q[x]/(\phi) \times \mathbb{Z}_p[x]/(\phi)$ and $\mathbb{Z}_q[x]/(\phi) \times \mathbb{Z}_{p_1}[x]/(\phi)$, that solving* $\mathrm{NTRU} - SIS_{n,m,q}$ *instance in $\mathbb{Z}_q[x]/(\phi)$ 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' 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.*

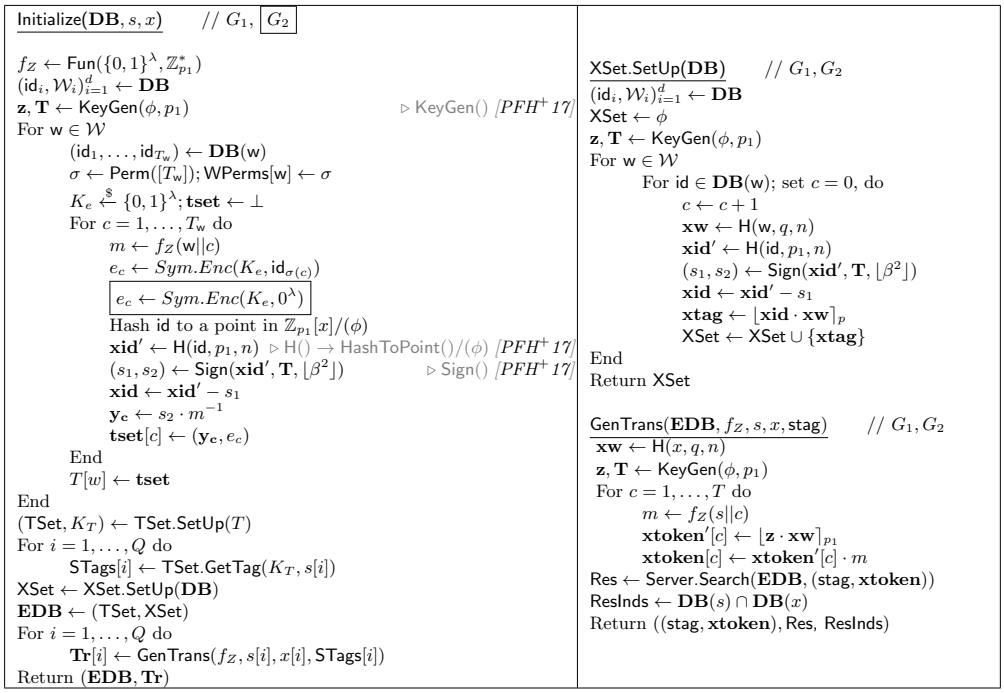


Table 3: Games G_1 and G_2

Proof. We construct the proof using various games $G_0, G_1, ...$ In each game, the adversary $\mathcal{A}$ starts by supplying $(\mathbf{DB}, \mathbf{q})$, which is then given to an 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 $\mathbf{Real}^{\mathbf{\Pi}}_{\mathcal{A}}(\lambda)$ (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 1 with NTRU-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 Initialize routine, which passes $(\mathbf{DB}, \mathbf{s}, \mathbf{x})$ from $\mathcal{A}$ and simulates NTRU-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 $\mathbf{Tr}$, for $i = 1, \ldots, Q$ it sets $\mathbf{Tr}[i]$ to the output of GenTrans$(\mathbf{EDB}, K_S, K_Z, s[i], x[i],$ STags$[i])$, which is defined in game G_0. As employed in OQXT we use a subroutine ServerSearch which we take to be the server's computation of NTRU-OQXT.Search in response to the first client message. The GenTrans routine generates the transcripts as in the real game but it computes the final result set 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 $\mathbf{DB}(x[i])$. Thus it can be stated -

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

Game G_1. This game is exactly same as G_0 the only difference is we replace every evaluation of PRF $F(K_S, \cdot), F'(K_Z, \cdot)$ 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 unbounded efficient adversaries $B_{1,1}, B_{1,2}$ such that -

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

Game G_2. In this game we modify G_1 to include the boxed code. The modification is done in the $\mathsf{Initialize}$ routine, where every ciphertext e_c is always generated as an encryption of 0^λ (with the same key). Similar to OQXT 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 $\mathsf{Initialize}$ subroutine, two arrays H, Y are computed and used in $\mathsf{XSet.SetUp}$ and $\mathsf{GenTrans}$. H is indexed by an identifier and a keyword from $\mathcal{W}$ and contains elements from $\mathbb{Z}_p[x]/(\phi)$. The elements of H are computed as $\lfloor \mathbf{xid} \cdot \mathbf{xw} \rceil_p$, where $\mathbf{xid} = \mathbf{xid}' - s_1$; $\mathbf{xid}'$ and $\mathbf{xw}$ are elements of $\mathbb{Z}_{p_1}[x]/(\phi)$ and $\mathbb{Z}_q[x]/(\phi)$ respectively and s_1 is a short signature generated by Sign algorithm [PFH+17]. Basically elements from H are used as **xtag** in $\mathsf{XSet.SetUp}$. Y is indexed by a keyword and each i in $|\mathbf{DB}(\mathsf{w})|$. It is filled with elements from $\mathbb{Z}_{p_1}[x]/(\phi)$.

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 $\lfloor \mathbf{z} \cdot \mathbf{xw} \rceil_{p_1}$, where $\mathbf{z}$ is a public matrix with a (secret) trapdoor $\mathbf{T}$ generated by KeyGen [PFH+17]. 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 $c = 1, \ldots, T_s$,

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

under the condition that p divides q. 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

$$\Pr[G_4 = 1] - \Pr[G_3 = 1] \leq \mathsf{negl}(\lambda)$$

The random function f_Z is used for sampling a random value m for every keyword-doc-id pair in the $\mathsf{Initialize}$ routine and the $\mathsf{GenTrans}$ routine. For any $\mathsf{w} \in \mathcal{W}$ and $c = 1, \ldots, T_{\mathsf{w}}$, the value of $m = f_Z(\mathsf{w}||c)$ is uniform and independent of the rest of the randomness in the game. Thus the value $\lfloor \mathbf{z} \cdot \mathsf{xu} \rceil_{p_1}$ is also uniform and independent of the rest of the game, which

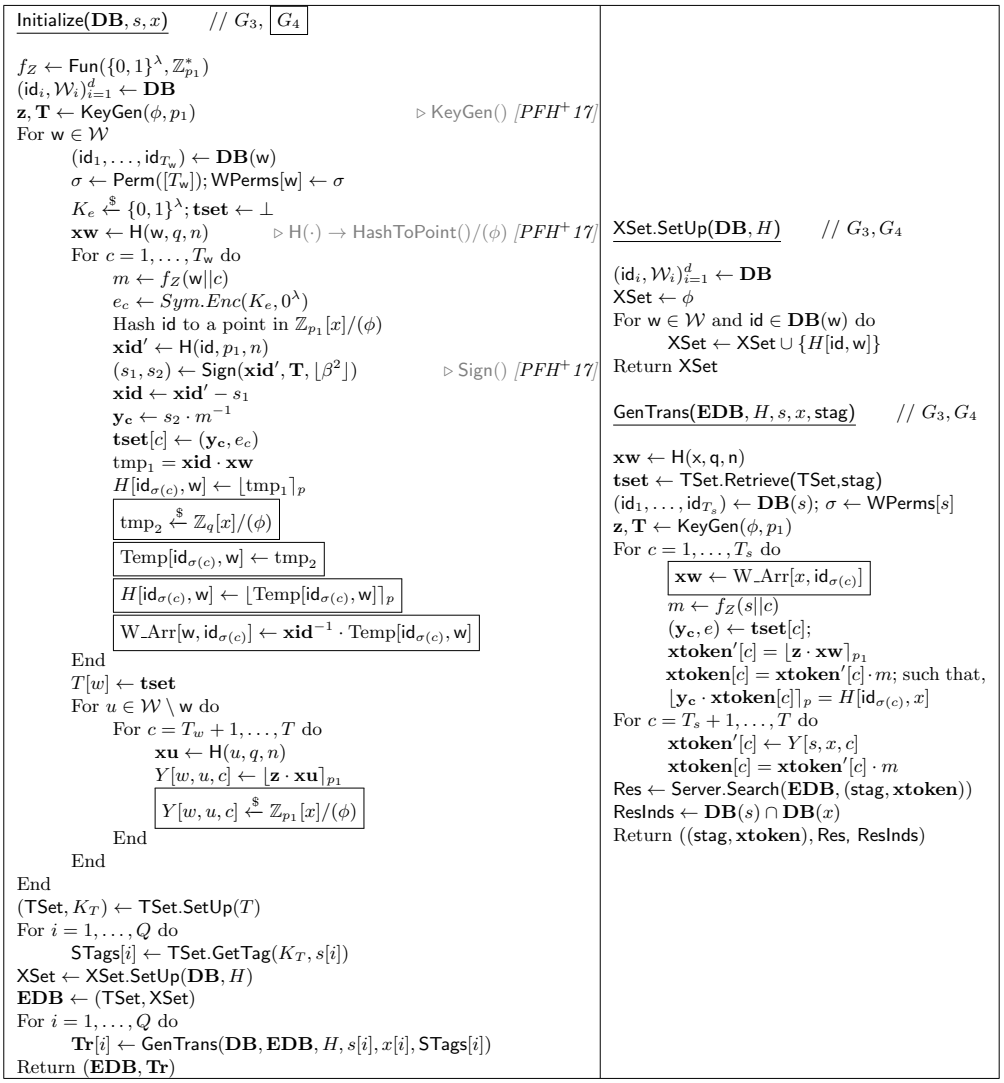


Table 4: Games G_3 and G_4

makes our above claim valid. In this game, the values of array Y are independently chosen from $\mathbb{Z}_{p_1}[x]/(\phi)$ and H is populated as $\lfloor \text{Temp}[\mathsf{id}_{\sigma(c)}, \mathsf{w}] \rceil_p$, where Temp consists of uniformly random samples from $\mathbb{Z}_q[x]/(\phi)$. We also compute another array $\text{W_Arr}[\mathsf{w}, \mathsf{id}_{\sigma(c)}] \leftarrow \mathbf{xid}^{-1} \cdot \text{Temp}[\mathsf{id}_{\sigma(c)}, \mathsf{w}]$ and claim its correctness by the existence of $\mathbf{xid}^{-1}$ in $\mathbb{Z}_{p_1}[x]/(\phi)$. Finally, by the hardness of ring-$\mathsf{LWR}_{n,p,q}$ and ring-$\mathsf{LWR}_{n,p_1,q}$, we prove that Game G_3 and G_4 are indistinguishable.

Game G_5. The Initialize code for game G_5 is described in Table 4 and the other routines remain same as in game G_4. In G_5 some 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

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

$(\mathsf{id}_i, \mathcal{W}_i)_{i=1}^{d} \leftarrow \mathbf{DB}$
$\mathbf{z}, \mathbf{T} \leftarrow \mathsf{KeyGen}(\phi, p_1)$ ▷ KeyGen() /PFH^+17/
For $\mathsf{w} \in \mathcal{W}$ and $i \in [d]$ do
  Hash id to a point in $\mathbb{Z}_{p_1}[x]/(\phi)$
  $\mathrm{Arr}[\mathsf{id}_i] \leftarrow \mathsf{H}(\mathsf{id}, p_1, n)$ ▷ $\mathsf{H}(\cdot) \rightarrow$ HashToPoint()/(ϕ) /PFH^+17/
  $\mathrm{tmp} \xleftarrow{\$} \mathbb{Z}_q[x]/(\phi)$
  $\mathrm{Temp}[\mathsf{id}_i, \mathsf{w}] \leftarrow \mathrm{tmp}$
  $H[\mathsf{id}_i, \mathsf{w}] \leftarrow \lfloor \mathrm{Temp}[\mathsf{id}_i, \mathsf{w}] \rceil_p$
For $\mathsf{w} \in s$ do $\mathsf{WPerms}[\mathsf{w}] \xleftarrow{\$} \mathsf{Perm}([T_s])$
For $\mathsf{w} \in \mathcal{W}$ do
  $K_e \xleftarrow{\$} \{0,1\}^\lambda$; $\mathbf{tset} \leftarrow \perp$
  $\mathsf{WPerms}[\mathsf{w}] \rightarrow \sigma$
  For $c = 1, \ldots, T_w$ do
    $m \leftarrow f_Z(\mathsf{w}||c)$
    $e_c \leftarrow Sym.Enc(K_e, 0^\lambda)$
    $\mathbf{xid}' \leftarrow \mathrm{Arr}[\mathsf{id}_{\sigma(c)}]$
    $(s_1, s_2) \leftarrow \mathsf{Sign}(\mathbf{xid}', \mathbf{T}, \lfloor \beta^2 \rfloor)$ ▷ Sign() /PFH^+17/
    $\mathbf{xid} \leftarrow \mathbf{xid}' - s_1$
    $\mathrm{W_Arr}[\mathsf{w}, \mathsf{id}_{\sigma(c)}] \leftarrow \mathbf{xid}^{-1} \cdot \mathrm{Temp}[\mathsf{id}_{\sigma(c)}, \mathsf{w}]$
    $\mathbf{y_c} \leftarrow s_2 \cdot m^{-1}$
    $\mathbf{tset}[c] \leftarrow (\mathbf{y_c}, e_c)$
  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
  $\mathbf{Tr}[i] \leftarrow \mathsf{GenTrans}(\mathbf{DB}, \mathbf{EDB}, H, \mathrm{W_Arr}, s[i], x[i], \mathsf{STags}[i])$
End
Return $(\mathbf{EDB}, \mathbf{Tr})$

$\underline{\mathsf{XSet.SetUp}(\mathbf{DB}, H)}$ // 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 $h \xleftarrow{\$} \mathbb{Z}_p[x]/(\phi)$; $\mathsf{XSet} \leftarrow \mathsf{XSet} \cup \{h\}$
End
Return XSet

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

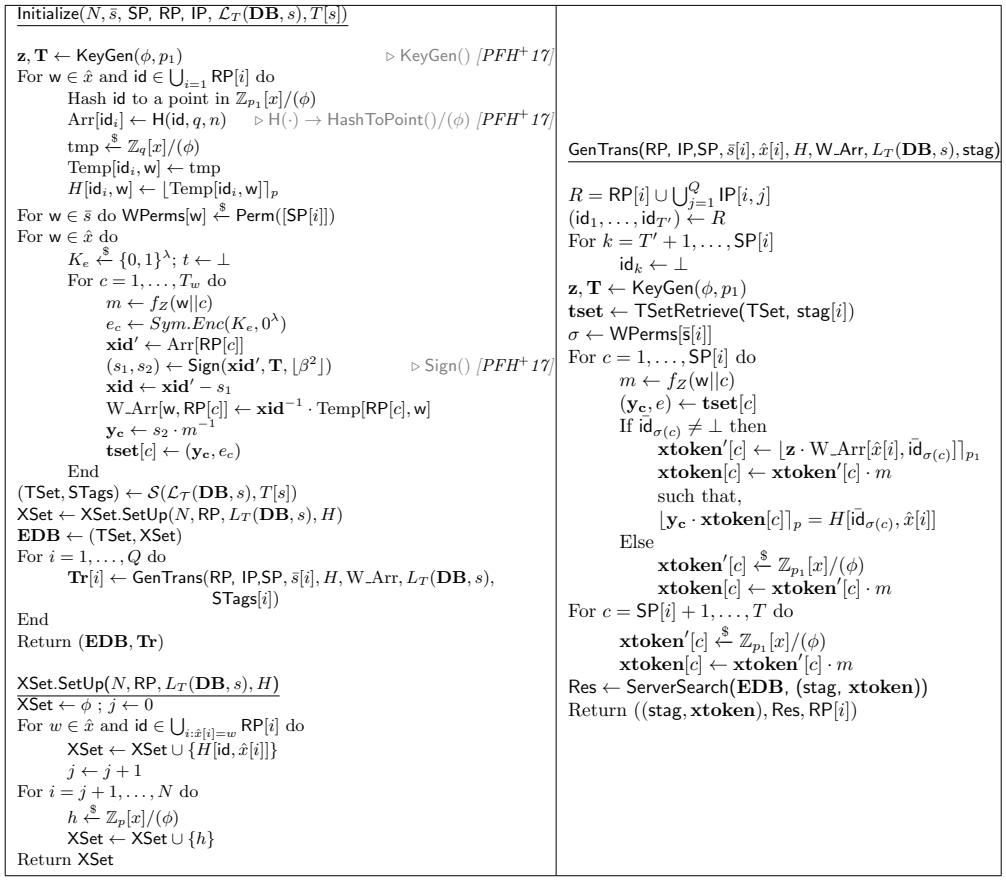
$\mathbf{tset} \leftarrow \mathsf{TSet.Retrieve}(\mathsf{TSet}, \mathsf{stag})$
$(\mathsf{id}_1, \ldots, \mathsf{id}_{T_s}) \leftarrow \mathbf{DB}(s)$; $\sigma \leftarrow \mathsf{WPerms}[\mathsf{s}]$
$\mathbf{z}, \mathbf{T} \leftarrow \mathsf{KeyGen}(\phi, p_1)$
For $c = 1, \ldots, T_s$ do
  $m \leftarrow f_Z(s||c)$
  $(\mathbf{y_c}, e) \leftarrow \mathbf{tset}[c]$;
  If $\mathsf{id}_{\sigma(c)} \in \mathbf{DB}(s) \cap \mathbf{DB}(x)$ then
    $\mathbf{xtoken}'[c] \leftarrow \lfloor \mathbf{z} \cdot \mathrm{W_Arr}[x, \mathsf{id}_{\sigma(c)}] \rceil_{p_1}$
    $\mathbf{xtoken}[c] \leftarrow \mathbf{xtoken}'[c] \cdot m$
    such that,
    $\lfloor \mathbf{y_c} \cdot \mathbf{xtoken}[c] \rceil_p = H[\mathsf{id}_{\sigma(c)}, x]$
  ElseIf $\exists j \neq i : \mathsf{id}_{\sigma(c)} \in \mathbf{DB}(s[j]) \wedge x[j] = x$
    $\mathbf{xtoken}'[c] \leftarrow \lfloor \mathbf{z} \cdot \mathrm{W_Arr}[x] \rceil_{p_1}$
    $\mathbf{xtoken}[c] \leftarrow \mathbf{xtoken}'[c] \cdot m$
    such that,
    $\lfloor \mathbf{y_c} \cdot \mathbf{xtoken}[c] \rceil_p = H[\mathsf{id}_{\sigma(c)}, x]$
  Else
    $\mathbf{xtoken}'[c] \xleftarrow{\$} \mathbb{Z}_{p_1}[x]/(\phi)$
    $\mathbf{xtoken}[c] \leftarrow \mathbf{xtoken}'[c] \cdot m$
End
For $c = T_s + 1, \ldots, T$ do
  $\mathbf{xtoken}'[c] \xleftarrow{\$} \mathbb{Z}_{p_1}[x]/(\phi)$
  $\mathbf{xtoken}[c] \leftarrow \mathbf{xtoken}'[c] \cdot m$
$\mathsf{Res} \leftarrow \mathsf{Server.Search}(\mathbf{EDB}, (\mathsf{stag}, \mathbf{xtoken}))$
$\mathsf{ResInds} \leftarrow \mathbf{DB}(s) \cap \mathbf{DB}(x)$
Return $((\mathsf{stag}, \mathbf{xtoken}), \mathsf{Res}, \mathsf{ResInds})$

Table 5: Games G_5, G_6 and G_7

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 -

$$\Pr[G_5 = 1] - \Pr[G_4 = 1] \leq \mathsf{negl}(\lambda)$$

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 accesses 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 XSet.SetUp and GenTrans. The routine XSet.SetUp will never repeat an access to H therefore for a particular index (id, x) it needs to check whether GenTrans will read the same location or not. But if we carefully observe we see GenTrans will read positions such that $\mathsf{id} \in \mathbf{DB}(s[i])$ and $x = x[i]$ for some i. This is exactly



Initialize($N, \bar{s}$, SP, RP, IP, $\mathcal{L}_T(\mathbf{DB}, s), T[s]$)

$\mathbf{z}, \mathbf{T} \leftarrow \mathsf{KeyGen}(\phi, p_1)$ ▷ KeyGen() /PFH+ 17/
For $\mathsf{w} \in \hat{x}$ and $\mathsf{id} \in \bigcup_{i=1} \mathsf{RP}[i]$ do
  Hash id to a point in $\mathbb{Z}_{p_1}[x]/(\phi)$
  $\mathrm{Arr}[\mathsf{id}_i] \leftarrow \mathsf{H}(\mathsf{id}, q, n)$ ▷ $\mathsf{H}(\cdot) \rightarrow$ HashToPoint()/(ϕ) /PFH+ 17/
  $\mathrm{tmp} \xleftarrow{\$} \mathbb{Z}_q[x]/(\phi)$
  $\mathrm{Temp}[\mathsf{id}_i, \mathsf{w}] \leftarrow \mathrm{tmp}$
  $H[\mathsf{id}_i, \mathsf{w}] \leftarrow \lfloor \mathrm{Temp}[\mathsf{id}_i, \mathsf{w}] \rceil_p$
For $\mathsf{w} \in \bar{s}$ do $\mathsf{WPerms}[\mathsf{w}] \xleftarrow{\$} \mathsf{Perm}([\mathsf{SP}[i]])$
For $\mathsf{w} \in \hat{x}$ do
  $K_e \xleftarrow{\$} \{0,1\}^\lambda$; $t \leftarrow \perp$
  For $c = 1, \ldots, T_w$ do
    $m \leftarrow f_Z(\mathsf{w}||c)$
    $e_c \leftarrow Sym.Enc(K_e, 0^\lambda)$
    $\mathbf{xid}' \leftarrow \mathrm{Arr}[\mathsf{RP}[c]]$
    $(s_1, s_2) \leftarrow \mathsf{Sign}(\mathbf{xid}', \mathbf{T}, \lfloor \beta^2 \rfloor)$ ▷ Sign() /PFH+ 17/
    $\mathbf{xid} \leftarrow \mathbf{xid}' - s_1$
    $\mathrm{W_Arr}[\mathsf{w}, \mathsf{RP}[c]] \leftarrow \mathbf{xid}^{-1} \cdot \mathrm{Temp}[\mathsf{RP}[c], \mathsf{w}]$
    $\mathbf{y_c} \leftarrow s_2 \cdot m^{-1}$
    $\mathbf{tset}[c] \leftarrow (\mathbf{y_c}, e_c)$
  End
$(\mathsf{TSet}, \mathsf{STags}) \leftarrow \mathcal{S}(\mathcal{L}_T(\mathbf{DB}, s), T[s])$
$\mathsf{XSet} \leftarrow \mathsf{XSet.SetUp}(N, \mathsf{RP}, L_T(\mathbf{DB}, s), H)$
$\mathbf{EDB} \leftarrow (\mathsf{TSet}, \mathsf{XSet})$
For $i = 1, \ldots, Q$ do
  $\mathbf{Tr}[i] \leftarrow \mathsf{GenTrans}(\mathsf{RP}, \mathsf{IP}, \mathsf{SP}, \bar{s}[i], H, \mathrm{W_Arr}, L_T(\mathbf{DB}, s), \mathsf{STags}[i])$
End
Return $(\mathbf{EDB}, \mathbf{Tr})$

XSet.SetUp($N, \mathsf{RP}, L_T(\mathbf{DB}, s), H$)
$\mathsf{XSet} \leftarrow \phi$; $j \leftarrow 0$
For $w \in \hat{x}$ and $\mathsf{id} \in \bigcup_{i:\hat{x}[i]=w} \mathsf{RP}[i]$ do
  $\mathsf{XSet} \leftarrow \mathsf{XSet} \cup \{H[\mathsf{id}, \hat{x}[i]]\}$
  $j \leftarrow j + 1$
For $i = j+1, \ldots, N$ do
  $h \xleftarrow{\$} \mathbb{Z}_p[x]/(\phi)$
  $\mathsf{XSet} \leftarrow \mathsf{XSet} \cup \{h\}$
Return XSet

GenTrans(RP, IP,SP, $\bar{s}[i], \hat{x}[i], H$, W_Arr, $L_T(\mathbf{DB}, s)$, stag)

$R = \mathsf{RP}[i] \cup \bigcup_{j=1}^{Q} \mathsf{IP}[i, j]$
$(\mathsf{id}_1, \ldots, \mathsf{id}_{T'}) \leftarrow R$
For $k = T' + 1, \ldots, \mathsf{SP}[i]$
  $\mathsf{id}_k \leftarrow \perp$
$\mathbf{z}, \mathbf{T} \leftarrow \mathsf{KeyGen}(\phi, p_1)$
$\mathbf{tset} \leftarrow \mathsf{TSetRetrieve}(\mathsf{TSet}, \mathsf{stag}[i])$
$\sigma \leftarrow \mathsf{WPerms}[\bar{s}[i]]$
For $c = 1, \ldots, \mathsf{SP}[i]$ do
  $m \leftarrow f_Z(\mathsf{w}||c)$
  $(\mathbf{y_c}, e) \leftarrow \mathbf{tset}[c]$
  If $\mathsf{id}_{\sigma(c)} \neq \perp$ then
    $\mathbf{xtoken}'[c] \leftarrow \lfloor \mathbf{z} \cdot \mathrm{W_Arr}[\hat{x}[i], \bar{\mathsf{id}}_{\sigma(c)}] \rceil_{p_1}$
    $\mathbf{xtoken}[c] \leftarrow \mathbf{xtoken}'[c] \cdot m$
    such that,
    $\lfloor \mathbf{y_c} \cdot \mathbf{xtoken}[c] \rceil_p = H[\bar{\mathsf{id}}_{\sigma(c)}, \hat{x}[i]]$
  Else
    $\mathbf{xtoken}'[c] \xleftarrow{\$} \mathbb{Z}_{p_1}[x]/(\phi)$
    $\mathbf{xtoken}[c] \leftarrow \mathbf{xtoken}'[c] \cdot m$
For $c = \mathsf{SP}[i] + 1, \ldots, T$ do
  $\mathbf{xtoken}'[c] \xleftarrow{\$} \mathbb{Z}_{p_1}[x]/(\phi)$
  $\mathbf{xtoken}[c] \leftarrow \mathbf{xtoken}'[c] \cdot m$
$\mathsf{Res} \leftarrow \mathsf{ServerSearch}(\mathbf{EDB}, (\mathsf{stag}, \mathbf{xtoken}))$
Return $((\mathsf{stag}, \mathbf{xtoken}), \mathsf{Res}, \mathsf{RP}[i])$

Table 6: The Simulator

same as done in XSet.SetUp routine of game G_6. From this observation we can say -

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

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 conditions 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 -

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

The 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 $\mathcal{L}_{\mathrm{NTRU-OQXT}}$.

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 [CJJ+13]. 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 $H[\mathsf{id}, \mathsf{w}] \xleftarrow{\$} \mathbb{Z}_p[x]/(\phi)$
2. For $\mathsf{w} \in \bar{s}$ do $\mathsf{WPerms}[\mathsf{w}] \xleftarrow{\$} \mathsf{Perm}([\mathsf{SP[i]}])$.
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 $h \xleftarrow{\$} \mathbb{Z}_p[x]/(\phi)$; $\mathsf{XSet} \leftarrow \mathsf{XSet} \cup \{h\}$

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 $((\mathsf{stag}, \mathbf{xtoken}), \mathsf{Res}, \mathsf{RP}[i])$. Based on the similar argument as shown in OXT [CJJ+13] we can claim that the output of $\mathcal{S}$ has the same distribution as the output of the Initialize routine of G_7.

6 NTRU-TWINSSE

In this section, we discuss the construction of the first *lattice-based*, quantum-safe instantiation of TWINSSE [BTR+23] using our optimized implementation of NTRU-OQXT. We call this quantum-safe instantiation NTRU-TWINSSE . We begin by giving a brief overview of the original scheme as proposed by Bag et al. in [BTR+23].

TWINSSE: Brief Overview. TWINSSE is a fast and highly scalable SSE scheme with support for conjunctive, disjunctive and more general Boolean queries (in both CNF and DNF forms), that uses a conjunctive SSE scheme as a black box. The main contribution of the work is to support conjunctive, disjunctive and general Boolean queries with *linear* storage overhead that outperforms the existing SSE schemes [KM17] with similar query expressiveness. The core technical contribution of TWINSSE is generation of a "meta-database" that consists of specially designed "meta-keywords" (mkw), which is a disjunctive combination of specific keywords from the database (for more details on the construction of "meta-keywords" see [BTR+23]). The motivation for generating meta-keywords is to

use a conjunctive SSE as a black box and support richer queries including conjunctions, disjunctions, and more general Boolean queries over encrypted data.

For example, a query of the form $query = \mathsf{w}_1 \vee \mathsf{w}_2 \vee \ldots \vee \mathsf{w}_\mathsf{n}$ is processed by TWINSSE as follows - The original disjunctive query *query* is transformed into a corresponding meta-query $query_{\mathsf{mkw}} = \mathsf{mkw}_1 \wedge \mathsf{mkw}_2 \wedge \ldots \wedge \mathsf{mkw}_{\mathsf{n}'}$ (where $\mathsf{n}' \geq \mathsf{n}$) consisting of a conjunction of meta-keywords corresponding to the keywords present in *query*. The transformed meta-query is then given to the conjunctive SSE oracle. The result returned by the SSE oracle is a superset of the result corresponding to the original disjunctive query *query*. Thus, TWINSSE provides a consolidated framework by deploying a conjunctive SSE as a black box that provides support for general Booelan queries over encrypted database.

The meta-keywords scale linearly with the number of keywords in the original database. The storage overhead of TWINSSE hence scales linearly with the plaintext database size. This results in a significant improvement from the quadratic storage overheads incurred by state-of-the-art SSE [KM17] supporting disjunctive and Boolean queries. The search time complexity of TWINSSE depends upon the underlying conjunctive SSE. In [BTR$^+$23] the authors show an instantiation of TWINSSE from OXT [CJJ$^+$13] (TWINSSE$_{\text{OXT}}$). The experimental results depict, TWINSSE$_{\text{OXT}}$ exhibits sublinear search time complexity for both conjunctive and disjunctive queries, which makes it amenable to scale to large real-world databases.

NTRU-TWINSSE. In this work we propose a quantum-safe instantitation of TWINSSE by replacing the underlying conjunctive SSE used in [BTR$^+$23] with our highly optimized, latticed-based conjunctive SSE scheme NTRU-OQXT. Our construction exploits the black box instantiation of the conjunctive SSE feature of TWINSSE and requires no further modifications to the original algorithm as proposed in [BTR$^+$23]. The resultant scheme NTRU-TWINSSE, supports conjunctive, disjunctive, and more general Boolean queries in sub-linear time complexity while ensuring post-quantum security guarantees derived from NTRU-OQXT. The complexity overheads of NTRU-TWINSSE scale with that of TWINSSE$_{\text{OXT}}$. Since the storage overhead of NTRU-OQXT is extremely optimized and compact, NTRU-TWINSSE scales linearly (in a similar way as TWINSSE$_{\text{OXT}}$) with the number of keywords in the database. This adds to the high scalability of our construction to large real-world databases with millions of keyword-document pairs. The search complexity for both conjunctive and disjunctive queries depends on the underlying SSE scheme, as such NTRU-TWINSSE shows extremely efficient search performance. It significantly outperforms TWINSSE$_{\text{OXT}}$ due to the fast and optimized NTT and *Fast Fourier Sampling* based trapdoor functions used in NTRU-OQXT. A more detailed analysis of the performance of NTRU-TWINSSE is presented in Section 7.2.

6.1 Correctness and Security Analysis of NTRU-TWINSSE

The correctness and security guarantees of NTRU-TWINSSE follows directly from the underlying conjunctive SSE, NTRU-OQXT (from [BTR$^+$23]). We give a brief overview of the correctness proof and security analysis of NTRU-TWINSSE in this section. Leakage-based cryptanalysis of SSE schemes evaluating performance on Enron or any other real-world dataset has been performed previously [PM21, BTR$^+$23]. We conjecture empirical analysis

of leakage-based cryptanalysis of NTRU-OQXT and NTRU-TWINSSE will follow a similar trend.

Correctness. The proof of correctness for NTRU-TWINSSE follows from the correctness of conjunctive SSE scheme i.e., the correctness of NTRU-OQXT (Section 4.4).

Theorem 4 (Correctness of NTRU-TWINSSE)**.** *Assuming that* NTRU-OQXT *satisfies correctness of search for conjunctive queries (Section 4.4) and Lemma 3.1 of [BTR+23] holds,* NTRU-TWINSSE *satisfies correctness for both conjunctive and disjunctive queries.*

Proof. Consider a disjunctive query as stated below -

$$query = \mathsf{w}_1 \vee \cdots \vee \mathsf{w}_\mathsf{n}$$

The equivalent conjunctive expression of meta-keywords can be expressed as below (Refer to [BTR+23] for details).

$$query_{\mathsf{mkw}} = \mathsf{mkw}_{i_0,j_0} \wedge \mathsf{mkw}_{i_1,j_1} \wedge \ldots \wedge \mathsf{mkw}_{i_\mathsf{n},j_\mathsf{n}}$$

We write the following relation (as given in Lemma 3.1 in [BTR+23]) -

$$\mathbf{DB}(query_{\mathsf{mkw}}) = \mathbf{DB}(query) \cup \Big(\bigcap_{k=0}^{\mathsf{n}} \Big(\bigcup_{\substack{l \in [N] \setminus (\{[i_k,j_k]\} \\ \cup \{\ell_r : r \in [\mathsf{n}]\})}} \mathbf{DB}(\mathsf{w}_l) \Big) \Big)$$

From the above equation it can be inferred that $\mathbf{DB}(query) \subseteq \mathbf{DB}(query_{\mathsf{mkw}})$. Hence, all ids of the actual result set of disjunctive query *query* is included in the result set obtained from the query using the search routine of NTRU-TWINSSE , which proves the correctness of the NTRU-TWINSSE .

Security Analysis. The security analysis of TWINSSE follows from the security notions of the underlying conjunctive SSE (as discussed in [BTR+23]). Similarly, the security of NTRU-TWINSSE can be formalized in terms of the leakage profile of NTRU-OQXTwith no additional assumptions necessitated by the aforementioned integration into the TWINSSE framework. We claim the security guarantees of NTRU-TWINSSE by stating the following theorem. The proof of security follows analogously from the original TWINSSE scheme [BTR+23] and is not detailed here. For more details on the security proof, we would like to refer the readers to the original TWINSSE security proofs in [BTR+23].

Theorem 5 (Security of NTRU-TWINSSE)**.** *Assuming that* NTRU-OQXT *is an (adaptively) secure SSE scheme with respect to* $\mathcal{L}_{\mathrm{NTRU-OQXT}} = (\mathcal{L}^{\mathsf{Setup}}_{\mathrm{NTRU-OQXT}}, \mathcal{L}^{\mathsf{Search}}_{\mathrm{NTRU-OQXT}})$, NTRU-TWINSSE *is an (adaptively) secure SSE scheme with respect to*

$$\mathcal{L}_{\mathrm{NTRU-TWINSSE}} = (\mathcal{L}^{\mathsf{Setup}}_{\mathrm{NTRU-TWINSSE}}, \mathcal{L}^{\mathsf{Search}}_{\mathrm{NTRU-TWINSSE}})$$

where for any plaintext database $\mathbf{DB}$*, any search query "query", and any pair of bucketization parameters* (n', n_{B}) *(See [BTR+23] for details on bucketization parameters), we have*

$$\mathcal{L}^{\mathsf{Setup}}_{\text{NTRU}-\text{TWINSSE}}(\mathbf{DB}) = \left(\mathcal{L}^{\mathsf{Setup}}_{\text{NTRU}-\text{OQXT}}(\widehat{\mathbf{DB}}), n', n_{\mathsf{B}}\right)$$

$$\mathcal{L}^{\mathsf{Search}}_{\text{NTRU}-\text{TWINSSE}}(query) = \begin{cases} \mathcal{L}^{\mathsf{Search}}_{\text{NTRU}-\text{OQXT}}(query); \\ \text{if } query \text{ is conjunctive} \\ \left\{\mathcal{L}^{\mathsf{Search}}_{\text{NTRU-OQXT}}(query_{\mathsf{mkw},k})\right\}_{k\in[n_{\mathsf{B}}]} \\ \text{if } query \text{ is disjunctive} \end{cases}$$

7 Experimental Results

In this section, we describe the experimental setup used to evaluate the performance of NTRU-OQXT and NTRU-TWINSSE, as well as the parameter choices and other low-level details of our prototype implementation(s). We provide a prototype implementation of our proposed scheme here - `https://github.com/debadrita05/NTRU-OQXT.git`.

Data Set and Platform. We used the Enron email dataset[13] 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.[14] The entire implementation of NTRU-OQXT and NTRU-TWINSSE 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. The parameter choices are as follows:

(i) n : the degree of the polynomial ring; Falcon uses two variations of $n = 512/1024$. Our implementation uses $n = 512$, however, the experiments can easily be performed with $n = 1024$ without significant modifications/overheads.

(ii) p_1 : Falcon uses carefully chosen modulus for maximum efficiency of *Number Theoretic Transforms* (NTT) operations. p_1 needs to be a prime of the form $p_1 = k \cdot 2n+1$. We use, $p_1 = (23835 \cdot (2 \cdot 512)) + 1$.

(iii) (q, p) : ring-LWR parameters. We use, $q = 2^{32}, p = 2^5$.

We remark that our parameter choice is motivated by the Falcon specifications [PFH+17, GJK24]. As a result, our proposed system inherits similar resistance to cryptanalytic attacks on lattice-based hardness assumptions [How07, MW16, ADH+19].

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

[14] 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.

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 NTRU-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 NTRU-OQXT and NTRU-TWINSSE as an SSE scheme for encrypted document collections proves to be the natural choice for prototyping and experimentation.

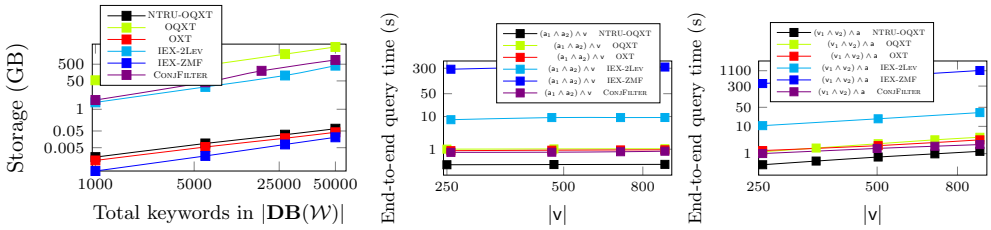
Concretely, we build the TSet and XSet data structures for the Enron dataset using Setup routine [1]. The NTRU-OQXT.Setup routine incorporates specialized lattice-based operations to generate the elements that are stored in the TSet and XSet. The KeyGen function computes the public matrix and the secret trapdoor in the NTRU lattice for every keyword-id pair. The Sign function samples a *short* vector (signatures) $\mathbf{y}_{\mathsf{id}}$ from a specific coset of a discrete Gaussian distribution. These two algorithms are directly incorporated from [PFH+17]. Due to use of extremely efficient implementation as well as algorithmic optimizations used in Falcon our implementation of both NTRU-OQXT and NTRU-TWINSSE shows extremely optimized and efficient performance overheads.

7.1 Experimental Evaluation of NTRU-OQXT

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

Evaluation of Storage Overhead. To achieve post-quantum security, lattice-based constructions of OQXT rely upon matrices of large dimensions due to which its storage overhead is significantly higher as we scale it to larger databases. We optimize our new construction NTRU-OQXT by incorporating NTRU lattice which is more compact due to the use of a polynomial ring structure. The result of such optimization is reflected in our experimental evaluations. Figure 2a compares the storage overhead of NTRU-OQXT and OQXT on Enron database. We vary the number of keywords in the database and observe the change in the storage pattern. It is observed that the storage overhead of NTRU-OQXT is significantly reduced to megabytes of memory usage (from several gigabytes required by OQXT). NTRU-OQXT scales linearly with the size of the database and competes with the classically secure OXT [CJJ+13], slightly exceeds the storage requirements of IEX-ZMF [KM17] which is a highly optimized construction (at the cost of increased search latency), while significantly reducing the quadratic storage overheads incurred by IEX-2Lev and ConjFilter [KM17, PPSY21].

Evaluation of End-to-End Search Latency. Figure 2b and Figure 2c compares the end-to-end search latency of NTRU-OQXT with that of OQXT, OXT, IEX-2Lev, IEX-ZMF and ConjFilter for conjunctive queries. Search time required for a conjunctive query of NTRU-OQXT is reduced by $3-4\times$ from that of OQXT, while it is reduced by half when compared to OXT and ConjFilter, around $20\times$ when compared with IEX-2Lev and



(a) (b) (c)

Figure 2: (2a) Server Storage (both axes are in log scale); (2b) End-to-End Search latency with a constant frequency of the least frequent term (both axes are in log scale); (2c) End-to-End Search latency with variable frequency of the least frequent term (both axes are in log scale).

around 600× compared to IEX-ZMF. To validate this, we consider a three-keyword query of the form $query = (\mathsf{a}_1 \wedge \mathsf{a}_2) \wedge \mathsf{v}$, where $\mathsf{a}_1, \mathsf{a}_2$ and v are three keywords from **DB**. Without loss of generality, we consider the first term of *query* (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 whereas the frequency of a is kept constant (constant term). Figure 2b shows a constant time overhead for conjunctive queries of this form with NTRU-OQXT, which is significantly faster than OQXT, OXT, IEX-2Lev, IEX-ZMF and ConjFilter. This happens because the conjunctive search time of OQXT, 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 optimized quantum-safe construction, NTRU-OQXT we not only preserve the sub-linear search complexity of the existing schemes but improve the performance by $3 - 4\times$.

Another set of experiments is carried out on queries of the form $query = (\mathsf{v}_1 \wedge \mathsf{v}_2) \wedge \mathsf{a}$ (Figure 2c). This time we vary the frequency of the 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 $(\mathsf{v}_1 \wedge \mathsf{v}_2)$ and the constant term as a in Figure 2c). It is observed that the search time of OQXT, OXT and ConjFilter increases linearly with the increase in the frequency of the least frequent keyword. NTRU-OQXT preserves similar linear dependency while significantly reducing the search time of OQXT by $3 - 4\times$ and of both OXT and ConjFilter by half. The search time complexity of IEX-2Lev and IEX-ZMF follows a similar trend but is significantly (20× and 600× respectively) more as compared to NTRU-OQXT.

Performance Improvement. NTRU-OQXT incorporates highly efficient constructions from Falcon that is built on NTRU lattices (in a polynomial ring structure), over which efficient NTT transformations are performed. The polynomial multiplications for generating ring-LWR samples are extremely fast since NTT transforms a polynomial $f(x) \in \mathbb{Z}[x]/\phi(x)$ from its coefficient representation into a vector of evaluations at n-th roots of unity modulo ϕ. The transformation itself handles the modular reduction in $\mathbb{Z}[x]/\phi(x)$, so (any operation) multiplication of polynomials in the NTT domain respects the cyclic structure mod ϕ, and is hence simplified to coefficient-wise multiplication. The trapdoor sampler used in Falcon's Sign function (during a *short* signature generation) uses the randomized variant of fast

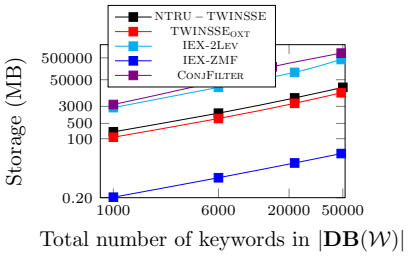


Figure 3: Server storage overhead with database size (|**DB**|) for the Enron database.

Fourier nearest plane algorithm of [DP16], called *Fast Fourier Sampling*. This highly optimized implementation makes the trapdoor sampling operation in NTRU-OQXT extremely efficient. All these operations reduce the overall time required by NTRU-OQXT making it highly efficient and amenable for practical deployment while scaling over large real-world databases.

7.2 Evaluation of NTRU-TWINSSE

In this section, we present the results of our experimental evaluation of NTRU-TWINSSE, and compare its performance overheads with TWINSSE$_{\text{OXT}}$, IEX-2Lev, IEX-ZMF and ConjFilter.

Storage Overhead. Figure 3 compares the storage overhead of NTRU-TWINSSE, TWINSSE$_{\text{OXT}}$, IEX-2Lev, IEX-ZMF and
ConjFilter on Enron database. TWINSSE$_{\text{OXT}}$ scales linearly with the number of keywords in the database because of its inherent reliance on the underlying OXT scheme (which shows similar behavior). Our new optimized construction of NTRU-TWINSSE shows a similar linear increase in storage with the increase in plaintext database size while significantly reducing the quadratic storage requirements of IEX-2Lev and ConjFilter. It requires more storage than the highly optimized construction of IEX-ZMF which we consider as a trade-off since NTRU-TWINSSE significantly outperforms IEX-ZMF for both conjunctive and disjunctive queries. Despite being a lattice-based implementation NTRU-OQXT is significantly storage efficient as it involves compact NTRU lattices, and is competitive with the classically secure OXT scheme and smoothly scales to large real-world databases.

End-to-End Latency of Conjunctive queries. We point out that the search performance of TWINSSE$_{\text{OXT}}$ is inherited from OXT and is therefore identical to OXT as shown in Figure 4a and 4b. In a similar way, the search performance of NTRU-TWINSSE is dependent on that of NTRU-OQXT. To validate this, we perform two sets of experiments. First, we consider a two-keyword query of the form $query = \mathsf{a} \wedge \mathsf{v}$, where a and v are two keywords from **DB** (Figure 4a). The results demonstrate constant time overhead for conjunctive queries of this form with NTRU-TWINSSE, which is identical with NTRU-OQXT. The conjunctive search time required by all the schemes, TWINSSE$_{\text{OXT}}$, IEX-2Lev, IEX-ZMF and ConjFilter is slower than our optimized quantum-safe version

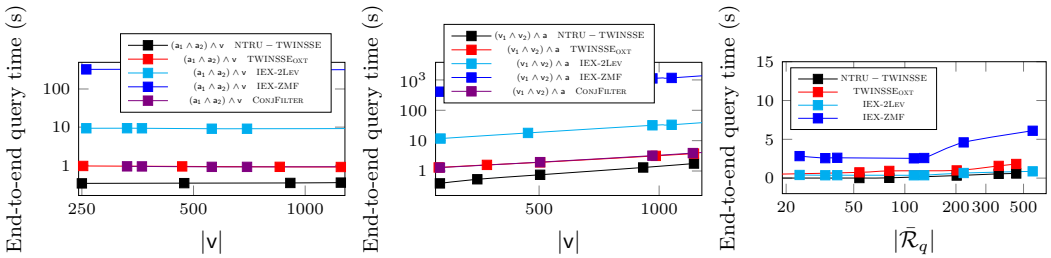


Figure 4: (4a) End-to-End Search latency with a constant frequency of the least frequent term (both axes are in log scale); (4b) End-to-End Search latency with variable frequency of the least frequent term (both axes are in log scale); (4c) Comparison of end-to-end search latency vs final result size for disjunctive queries of the form $query = \mathsf{w}_1 \vee \mathsf{w}_2$.

of NTRU-TWINSSE. For the second set of experiments, Figure 4b demonstrates the conjunctive search time comparison of NTRU-TWINSSE with TWINSSE$_{\text{OXT}}$ when carried out on queries of the form $query = (\mathsf{v}_1 \wedge \mathsf{v}_2) \wedge \mathsf{a}$. We vary the frequency of the least frequent keyword a with different queries and the frequency of v is kept constant. It is observed that the search time of NTRU-TWINSSE increases linearly with the increase in the frequency of the least frequent keyword/conjunct similar to TWINSSE$_{\text{OXT}}$ and ConjFilter while significantly reducing their search time by half and search time of IEX-2Lev and IEX-ZMF by 20× and 600× respectively. The search time results are directly dependent on the underlying conjunctive SSE scheme, hence it shows a similar trend when compared to the conjunctive search time of NTRU-OQXT.

End-to-End Latency of Disjunctive queries. Finally, we also provide an end-to-end disjunctive query performance comparison of NTRU-TWINSSE with TWINSSE$_{\text{OXT}}$ and IEX-2Lev in Figure 4c. For queries with smaller result sizes, our proposed NTRU-TWINSSE exhibits faster end-to-end query processing than TWINSSE$_{\text{OXT}}$, IEX-2Lev and IEX-ZMF. This is because NTRU-TWINSSE builds on top of NTRU-OQXT, hence, its search performance efficiency of is significantly improved and outperforms the existing state-of-the-art SSE TWINSSE$_{\text{OXT}}$, IEX-2Lev by $2-3\times$ and IEX-ZMF by around $15\times$. In terms of practical performance across a wide range of Boolean queries and scalability to large databases, NTRU-TWINSSE outperforms all the schemes - TWINSSE$_{\text{OXT}}$, IEX-2Lev and IEX-ZMF.

8 Concluding Remarks

In this paper, we presented the first practically efficient SSE scheme with fast conjunctive *and* disjunctive keyword searches, compact storage, and security based on the (plausible) quantum-hardness of well-studied *lattice-based* assumptions. We described NTRU-OQXT – a highly compact NTRU lattice-based conjunctive SSE scheme supporting extremely fast searches. We then presented an extension of this scheme, called NTRU-TWINSSE , that

additionally supports disjunctive queries. We also presented prototype implementations of both schemes, and experimentally validated that they support extremely fast query processing while scaling smoothly to large real-world datasets, and are either competitive with or even better than the best quantum-unsafe conjunctive and/or disjunctive SSE schemes in terms of search latency.

References

[ADH+19] Martin R. Albrecht, Léo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn W. Postlethwaite, and Marc Stevens. The general sieve kernel and new records in lattice reduction. In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2019.

[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.

[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+23] 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. *Proc. Priv. Enhancing Technol.*, 2023.

[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.

[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.

[DLP14] Léo Ducas, Vadim Lyubashevsky, and Thomas Prest. Efficient identity-based encryption over NTRU lattices. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, 2014.

[DP16] Léo Ducas and Thomas Prest. Fast fourier orthogonalization. In *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2016, Waterloo, ON, Canada*, 2016.

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

[GJK24] Phillip Gajland, Jonas Janneck, and Eike Kiltz. A closer look at falcon. *IACR Cryptol. ePrint Arch.*, 2024.

[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.

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

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *ACM STOC 2008*, pages 197–206, 2008.

[How07] Nick Howgrave-Graham. A hybrid lattice-reduction and meet-in-the-middle attack against NTRU. In *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference*, 2007.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In *Algorithmic Number Theory, Third International Symposium, ANTS-III,Portland, Oregon, USA*, 1998.

[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.

[Kle00] Philip N. Klein. Finding the closest lattice vector when it's unusually close. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, 2000.

[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.

[KM19] Seny Kamara and Tarik Moataz. Computationally volume-hiding structured encryption. In *EUROCRYPT 2019*, pages 183–213, 2019.

[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.

[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.

[MW16] Daniele Micciancio and Michael Walter. Practical, predictable lattice basis reduction. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2016.

[OK21] Simon Oya and Florian Kerschbaum. Hiding the access pattern is not enough: Exploiting search pattern leakage in searchable encryption. In *USENIX Security 2021*, 2021.

[PFH+17] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon: Fast-fourier lattice-based compact signatures over ntru, 2017.

[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.

[PPYY19] 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.

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

[SS11] Damien Stehlé and Ron Steinfeld. Making NTRU as secure as worst-case problems over ideal lattices. In *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, 2011.

[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.

[TPM23] Debadrita Talapatra, Sikhar Patranabis, and Debdeep Mukhopadhyay. Conjunctive searchable symmetric encryption from hard lattices. In *8th IEEE European Symposium on Security and Privacy, EuroS&P*, 2023.