PunSearch: Enabling Puncturable Encrypted Search over Lattice for Cloud Storage Systems

Yibo Cao, Shiyuan Xu, Gang Xu, Xiu-Bo Chen, Tao Shang, Yuling Chen, and Zongpeng Li

***Abstract*—Searchable encryption (SE) has been widely studied for cloud storage systems, allowing data encrypted search and retrieval. However, existing SE schemes can not support the fine-grained searchability revocation, making it impractical for real applications. Puncturable encryption (PE) [Oakland'15] can revoke the decryption ability of a data receiver for a specific message, which can potentially alleviate this issue. Moreover, the threat of quantum computing remains an important and realistic concern, potentially leading to data privacy leakage for cloud storage systems. Consequently, designing a post-quantum puncturable encrypted search scheme is still far-reaching. In this paper, we propose PunSearch, the first puncturable encrypted search scheme over lattice for outsourced data privacy-preserving in cloud storage systems. PunSearch provides a fine-grained searchability revocation while enjoying quantum safety. Different from existing PE schemes, we construct a novel trapdoor generation mechanism through evaluation algorithms and lattice pre-image sampling technique. We then design a search permission verification method to revoke the searchability for specific keywords. Furthermore, we formalize a new IND-Pun-CKA security model, and utilize it to analyze the security of PunSearch. Comprehensive performance evaluation indicates that the computational overheads of Encrypt, Trapdoor, Search, and Puncture algorithms in PunSearch are just 0.06, 0.005, 0.05, and 0.31 times of other prior arts, respectively under the best cases. These results demonstrate that PunSearch is effective and secure for cloud storage systems.**

***Index Terms*—Puncturable encrypted search, lattice-based cryptography, cloud storage, privacy-preserving.**

I. Introduction

CLOUD storage has become an indispensable component of current data management (e.g., Google Drive, Aliyun) with prominent benefits, such as efficient storage and service elasticity [1], [2], [3], [4]. Specifically, data senders have the ability to outsource their data to the cloud, thereby reducing local maintenance costs. In addition, data receivers can search data from cloud servers, facilitating convenient data sharing and retrieval. Meanwhile, cloud servers are usually semi-honest, causing data privacy leakage. Although the encryption-before-outsourcing technique can protect data privacy to some extent, it heavily limits data availability, which can be detrimental to data search and sharing [5]. To ensure data privacy without losing its usability, a cryptographic primitive named searchable encryption (SE) was formalized. As shown in Fig. 1, through the SE technique, data owners can encrypt their data files before uploading to the cloud, while cloud servers can search over the encrypted data directly. In recent years, numerous researchers have endeavored to apply SE in cloud storage, thereby enhancing the functionality and security of SE, e.g., conjunctive keyword SE [6], multi-receiver SE [7], [8], authenticated SE [9], [10], etc.

Nevertheless, the afore-mentioned schemes only provide immutable searchability for data receivers, which renders them unsuitable for real-world applications. To address this problem, an intuitive approach is to revoke the searchability promptly. For example, multiple departments affiliated within an enterprise often outsource numerous private data to the cloud, and the data accessibility to each department is subject to change over time. When a department is no longer engaged in these activities, the enterprise could revoke its searchability to limit its data access permission, which is essential for cloud storage systems. Furthermore, the advent of quantum computing presents significant obstacles to data privacy-preserving in cloud storage systems [11], [12]. In particular, quantum computers can break the searchable encryption schemes based on the classical cryptographic assumption (e.g. discrete logarithm). Consequently, a substantial amount of encrypted private data will be disclosed, resulting in serious breaches of data confidentiality and integrity.

In a nutshell, there still exist two challenges to be addressed in encrypted search for cloud storage systems. The first challenge is how to construct a SE scheme with searchability revocation. A straightforward approach is to introduce the receiver revocation [13], [14] and forward security [15], [16], [17] into SE, which is designed to revoke a data receiver's searchability. However, this mechanism is all-or-nothing, and cannot realize fine-grained searchability revocation for specific keywords. To bridge the gap, puncturable encryption (PE) provides the revocation of decryption ability for a specific message, which motivates this work to design a puncturable encrypted search scheme. Subsequently, the second challenge is how to ensure a puncturable encrypted search that can withstand quantum computing attacks. To address this con-

Y. Cao and X.-B. Chen are with the Information Security Center, State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China. (E-mail: caoyibo@bupt.edu.cn, flyover100@163.com).

Y. Cao and S. Xu are with the Department of Computer Science, School of Computing and Data Science, The University of Hong Kong, Pok Fu Lam, Hong Kong. (E-mail: syxu2@cs.hku.hk).

G. Xu is with the School of Information Science and Technology, North China University of Technology, Beijing, China, and also with the Information Security Center, State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China. (E-mail: gx@ncut.edu.cn).

T. Shang is with the School of Cyber Science & Technology, Beihang University, Beijing, China. (E-mail: shangtao@buaa.edu.cn).

Y. Chen is with the State Key Laboratory of Public Big Data, College of Computer Science and Technology, Guizhou University, Guiyang, China. (E-mail: ylchen3@gzu.edu.cn).

Z. Li is with the Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing, China. (E-mail: zongpeng@tsinghua.edu.cn).

cern, lattice-based cryptography has been widely adopted with quantum-resistance. Many researchers have designed various lattice-based SE primitives [18], [19], [20], [21], and these schemes are used for the data privacy-preserving. As far as we know, there is no post-quantum puncturable encrypted search scheme, which has become another motivation for us. This state of affairs guides us to a question of this work:

Can we propose a post-quantum puncturable encrypted search primitive for cloud data privacy-preserving?

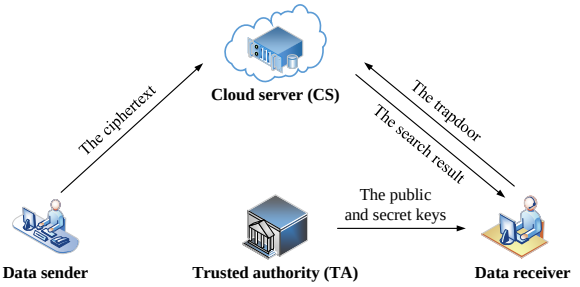


Fig. 1. The cloud data encrypted search procedure.

In this paper, we provide the answer affirmatively through the following results. We propose **PunSearch**, the first lattice-based puncturable encrypted search scheme for outsourced data privacy-preserving in cloud storage systems. PunSearch achieves the fine-grained searchability revocation for specific keywords while resisting quantum computing attacks. For puncturing the secret key, we introduce lattice basis generation algorithms ExtendLeft and RandBasis to obtain a lattice basis as a punctured secret key. Based on this, how to generate a search trapdoor from a punctured secret key and realize the search is our main technical challenge. Existing lattice-based SE schemes have utilized the SampleLeft or SamplePre algorithms to generate the trapdoor [15], [19], [20], [21], but this punctured secret key cannot be used as input for these algorithms. Thus, addressing this challenge is not trivial. Different from the existing schemes, we leverage the evaluation algorithm Eval_{pk} with the GenSamplePre technique to design a novel trapdoor generation mechanism. Specifically, we first invoke the Eval_{pk} algorithm to map a punctured tag list P into several matrices. Then, we utilize these matrices and the punctured secret key as the input of GenSamplePre technique to obtain a trapdoor. For the search phase, we need to verify the search permission to determine whether this trapdoor has the searchability for a specific ciphertext. If so, we execute the ciphertext search and return a result to the data receiver.

We enumerate our fourfold contributions as follows.

- We first present a puncturable encrypted search scheme over lattice for cloud storage systems, named PunSearch. Our scheme not only supports the fine-grained revocation of searchability for specific keywords, but also resists quantum computing attacks, which adapts for data privacy-preserving in cloud storage systems.
- We design a novel architecture for a searchable encryption scheme by introducing a secret key puncture procedure and constructing a designated trapdoor generation mechanism. Specifically, we first introduce the lattice basis generation algorithm to revoke the searchability of a data receiver for specific keywords, and then utilize the evaluation algorithm and lattice pre-image sampling technique to generate the trapdoor. Moreover, we devise a permission verification method to determine the searchability of this trapdoor.
- We also formalize the security notation of PunSearch for the first time. Then, we give rigorous security analysis to demonstrate that PunSearch provides IND-Pun-CKA security in the random oracle model (ROM), which can be reduced to the Learning With Errors (LWE) assumption.
- Comprehensive performance evaluation indicates that PunSearch is more efficient than prior arts [15], [19], [20], [21], [22], [23] in the context of computational overhead. In particular, the time cost of **Encrypt**, **Trapdoor**, and **Search** algorithms in PunSearch are just 0.06, 0.005, and 0.05 times compared to other lattice-based SE schemes [15], [19], [20], [21], respectively for the best cases. Besides, the time cost of our **Puncture** algorithm is only 0.31 times of other lattice-based PE schemes [22], [23], which is practical for cloud storage systems.

The remainder is structured as follows. Section II summarizes the literature review, and Section III presents the basic concepts. In Section IV, we present the problem formulation. The concrete design of our PunSearch scheme is elaborated in Section V. Sections VI and VII cover the security analysis and performance evaluation and comparison, respectively. Eventually, we conclude this paper in Section VIII.

II. Related Works

A. Searchable Encryption

Searchable encryption (SE) technique supports encrypted search for cloud storage systems, which has garnered widespread attention. Boneh et al. [24] formalized the SE scheme in a public key setting, named public key encryption with keyword search (PEKS). Since then, various SE schemes have been proposed for cloud data privacy-preserving. Xu et al. [25] presented an authorized SE primitive to protect the privacy of user identity and encrypted data. For cloud e-mail servers, a more practical multi-keyword SE scheme with hidden structures (PMSEHS) was designed by Xu et al. [26] to achieve the encrypted e-mail search as fast as possible. Zhang et al. [27] presented a subversion-resistant and consistent attribute-based SE system, which is designed to resist several external attacks. Nevertheless, these schemes cannot support the searchability revocation of the data receiver.

To alleviate this, many researchers proposed receiver revocation SE [13], [14]. Sun et al. [13] utilized proxy-based encryption to put forward an attribute-based SE scheme supporting user revocation. Zhang et al. [14] designed a novel secure search method under a multi-owner model, which provides ranked multi-keyword search and efficient receiver revocation. Alternatively, forward security SE [28], [16] can also be a good solution for this issue. To extend the functionality, a forward authenticated SE with fieldless concatenated keyword was constructed by [16], which can resist keyword guessing attacks and unauthorized ciphertext search.

Moreover, lattice-based SE primitives were proposed to cope with quantum computers [15], [18], [19], [20], [21], [29]. Concretely, Zhang et al. [15] presented a forward secure SE method over lattice, namely FS-PEKS, applied in the cloud-assisted industrial Internet of Things (IIoT). They [18] then devised a biometric identity-based SE scheme supporting multi-keyword search (BIB-MKS) over lattice. Following this, Luo et al. designed a lattice-based attribute-based authenticated SE (ABAEKS) [19] and proxy-based authenticated SE (Re-PAEKS) [20] schemes, they are resistant to internal keyword guessing attacks. For the multi-keyword search scenarios, Lin et al. [21] constructed three SE schemes that support disjunctive, conjunctive, and range keyword searches, respectively. However, the above-mentioned solutions forgot to consider the fine-grained searchability revocation for specific keywords, which limits the practicality of cloud storage systems.

B. Puncturable Encryption

Puncturable encryption (PE), formalized by Green et al. [30], can revoke the decryption ability for a specific message. Phuong et al. [31] combined PE and attribute-based encryption (ABE) scheme to propose a new punctured keys generation method. To enable the encrypted data with decryption ability revocation, many PE schemes were introduced in the cloud environment [32], [33]. For instance, a revocable ABE scheme incorporated PE to realize the data receiver revocation [32] in 2023. Meanwhile, Cui et al. [33] presented a PE primitive to support the secret key self-update, enhancing its practicability.

However, the above-mentioned schemes forgot to consider quantum computing attacks. To address this, many researchers have constructed serval PE schemes based on lattice hardness [34], [22], [23], [35], [36]. Susilo et al. [34] pointed out the puncturing property can be constructed through efficiently computable functions, and presented the first lattice-based PE scheme. Following this direction, Dutta et al. [22] offered puncturable identity-based encryption (PIBE) over lattice, and extended it to the puncturable key-policy ABE scheme. Subsequently, Dutta et al. also [23] put forward a lattice-based hierarchical PIBE scheme that supports more general key updates and flexible secret key puncture. For cloud storage systems, to avoid unauthorized access, Yang et al. designed an innovative lattice-based puncturable ciphertext-policy attribute-based encryption (CP-PABE) scheme [35] and a puncturable attribute-based matchmaking encryption (PM-ABE) [36] scheme, achieving the privacy-preserving for cloud data in the post-quantum era.

Unfortunately, none of the existing PE schemes provide search property. Thus, it is necessary to design a puncturable encrypted search scheme for cloud storage systems while resisting quantum computing attacks.

III. PRELIMINARIES

Definition 1: We define the basis of lattice Λ is a matrix $\mathbf{A} =(\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_m) \in \mathbb{R}^{n\times m}$ with linearly independent columns vectors, s.t. $\Lambda = \Lambda(\mathbf{A}) = \{x_1 \cdot \mathbf{a}_1 + x_2 \cdot \mathbf{a}_2 + \cdots + x_m \cdot \mathbf{a}_m | x_i \in \mathbb{Z}\}$.

Definition 2: Given three integers $n, m, q \in \mathbb{Z}$ and a matrix $\mathbf{M} \in \mathbb{Z}_q^{n\times m}$, we give the definition of the q-ary lattice: $\Lambda_q^{\perp}(\mathbf{M}) := \{\mathbf{r} \in \mathbb{Z}^m | \mathbf{Mr} = \mathbf{0} \bmod q\}$, $\Lambda_q^{\mathbf{u}}(\mathbf{M}) := \{\mathbf{r} \in \mathbb{Z}^m | \mathbf{Mr} = \mathbf{u} \bmod q\}$.

Definition 3: Given a parameter $\sigma \in \mathbb{R}^+$, two vectors $\mathbf{c} \in \mathbb{Z}^m$ and $\mathbf{x} \in \mathbb{Z}^m$, we define that the $\forall \mathbf{x} \in \Lambda, \mathcal{D}_{\sigma,\mathbf{c}} = \frac{\rho_{\sigma,\mathbf{c}}(\mathbf{x})}{\rho_{\sigma,\mathbf{c}}(\Lambda)}$ is the discrete Gaussian distribution over lattice Λ, where $\rho_{\sigma,\mathbf{c}}(\mathbf{x}) = \exp(-\pi\frac{\|\mathbf{x}-\mathbf{c}\|^2}{\sigma^2})$ and $\rho_{\sigma,\mathbf{c}}(\Lambda) = \sum_{\mathbf{x}\in\Lambda} \rho_{\sigma,\mathbf{c}}(\mathbf{x})$.

Definition 4: Given two integers $q \geq 2$ and $n \geq 1$, we define a gadget matrix as $\mathbf{G} = \mathbf{I}_n \otimes \mathbf{g}^\top \in \mathbb{Z}^{n\times n\lceil \log q\rceil}$, $\mathbf{g}^\top = [1, 2, \cdots, 2^{\lceil \log q\rceil -1}] \in \mathbb{Z}_q^{\lceil \log q\rceil}$. Moreover, $\mathbf{G}$ can be extended to a matrix over $\mathbb{Z}_q^{n\times m}$, where $m > n\lceil \log q\rceil$.

Given a vector $\mathbf{s} \in \mathbb{Z}_q^n$, the Learning With Errors (LWE) distribution over $\mathbb{Z}_q^n \times \mathbb{Z}_q$ is sampled by randomly selecting a vector $\mathbf{a} \in \mathbb{Z}_q^n$ and an error vector $e \leftarrow \chi$, where χ is a B-bounded noise distribution s.t. $|e| \leq B$ with non-negligible probability, and returning $(\mathbf{a}, b) =(\mathbf{a}, \mathbf{a}^\top\mathbf{s} + e \bmod q)$.

Definition 5: Given m independent pairs $(\mathbf{a}_i, b_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, where each sample is governed by the following either one to define the decisional $\mathrm{LWE}_{n,m,q,\chi}$ assumption:

1) Pseudo-random sample: $(\mathbf{a}_i, b_i) = (\mathbf{a}_i, \mathbf{a}_i^\top\mathbf{s}+e_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, where $\mathbf{s}$ is a randomly vector, e_i is an error vector, and $\mathbf{a}_i$ is an uniform vector.
2) Random sample: Randomly samples from $\mathbb{Z}_q^n \times \mathbb{Z}_q$.

Moreover, the decisional $\mathrm{LWE}_{n,m,q,\chi}$ assumption has proven to be as hard as the worst-case SIVP and GapSVP according to the reference [37].

Definition 6: [38], [34] Given a positive $\delta > 0$, a function family $F = \{f : \mathbb{Z}_q^d \to \mathbb{Z}_q\}$ and a function $\alpha_F : \mathbb{Z} \to \mathbb{Z}$, we define three evaluation algorithms as follows:

- $\mathbf{M}_f \leftarrow \mathsf{Eval}_{pk}(f, \{\mathbf{M}_i\}_{i=1}^d)$: Given a function $f \in F$ and matrices $\{\mathbf{M}_i\}_{i=1}^d \in \mathbb{Z}_q^{n\times m}$, this algorithm returns a matrix $\mathbf{M}_f \in \mathbb{Z}_q^{n\times m}$.
- $\mathbf{c}_f \leftarrow \mathsf{Eval}_{ct}(f, \{\mathbf{M}_i, \mathbf{c}_i, b_i\}_{i=1}^d\})$: Given a function $f \in F$ and tuples $\{\mathbf{M}_i \in \mathbb{Z}_q^{n\times m}, \mathbf{c}_i \in \mathbb{Z}_q^m, b_i \in \mathbb{Z}_q\}_{i=1}^d$, this algorithm returns a vector $\mathbf{c}_f \in \mathbb{Z}_q^m$, s.t. $\mathbf{c}_f = (\mathbf{M} + f(\mathbf{b})\mathbf{G})^\top\mathbf{s} + \mathbf{e}_f \in \mathbb{Z}_q^m$, where $\mathbf{s} \in \mathbb{Z}_q^n$, $\mathbf{b} = (b_1, b_2, \cdots, b_d)$, $\mathbf{M}_f \leftarrow \mathsf{Eval}_{pk}(f, \{\mathbf{M}_i\}_{i=1}^d)$, $\mathbf{c}_i = (\mathbf{M}_i + b_i\mathbf{G})^\top\mathbf{s} + \mathbf{e}_i$, $\|\mathbf{e}_f\| \leq \Delta$, $\|\mathbf{e}_i\| < \delta$, and $\Delta < \delta\alpha_F(n)$.
- $\mathbf{S}_f \leftarrow \mathsf{Eval}_{sim}(f, \mathbf{M}, \{\mathbf{S}_i, b_i^*\}_{i=1}^d\})$: Given a function $f \in F$, a matrix $\mathbf{M} \in \mathbb{Z}_q^{n\times m}$, and tuples $\{\mathbf{S}_i \in \{-1, 1\}^{m\times m}, b_i^* \in \mathbb{Z}_q\}_{i=1}^d$, this algorithm returns a matrix $\mathbf{S}_f \in \mathbb{Z}_q^{m\times m}$, s.t. $\mathbf{MS}_f - f(\mathbf{b}^*)\mathbf{G} = \mathbf{M}_f$, where $\mathbf{b}^* = (b_1^*, b_2^*, \cdots, b_d^*)$, $\mathbf{M}_f \leftarrow \mathsf{Eval}_{pk}(f, \{\mathbf{MS}_i - b_i^*\mathbf{G}\}_{i=1}^d)$, and $\|\mathbf{S}_f\|_2 < \alpha_F(n)$.

For two list $T := (t_1, \cdots, t_d) \in \mathbb{Z}_q^d$, $P := (t'_1, \cdots, t'_\psi) \in \mathbb{Z}_q^\psi$ and a function $f_{t',j} \in F$, we define $f_{t',j}(T) \neq 0$ iff $t'_j \in T$ for $j \in \{1, \cdots, \psi\}$. Otherwise, $f_{t',j}(T) = 0$.

Lemma 1: [39] Given three integers $n, m, q \in \mathbb{Z}$, where $m \geq 2n \log q$, the $\mathsf{TrapGen}(n, m, q)$ algorithm returns a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and a basis $\mathbf{T}_\mathbf{A} \in \mathbb{Z}^{m\times m}$, where $\mathbf{A}$ is a uniform matrix and $\|\widetilde{\mathbf{T}_\mathbf{A}}\| = \mathcal{O}(\sqrt{n \log q})$.

Lemma 2: [40] Given three integers $n, m, q \in \mathbb{Z}$, two matrices $\mathbf{N}_1, \mathbf{N}_2 \in \mathbb{Z}_q^{n\times m}$, and a basis $\mathbf{T}_{\mathbf{N}_1}$ of $\Lambda_q^{\perp}(\mathbf{N}_1)$, the $\mathsf{ExtendRight}(\mathbf{N}_1, \mathbf{T}_{\mathbf{N}_1}, \mathbf{N}_2)$ algorithm returns a basis $\mathbf{T}_{(\mathbf{N}_1|\mathbf{N}_2)} \in \mathbb{Z}^{2m\times 2m}$ of $\Lambda_q^{\perp}(\mathbf{N}_1|\mathbf{N}_2)$, s.t. $\|\widetilde{\mathbf{T}_{\mathbf{N}_1}}\| = \|\widetilde{\mathbf{T}_{\mathbf{N}_1|\mathbf{N}_2}}\|$.

TABLE I
GLOSSARY

Acronym	Definition
λ	The security parameter
pp	The public parameter
$pk_R, sk_{R,\emptyset}$	The data receiver's public and initial secret keys
d	The number of tags
t	The tag embedded in ciphertext
T	The tag list, where $T = (t_1, \cdots, t_d)$
$\mathbf{ck}, \mathbf{tk}$	The keyword
CT	The keyword ciphertext
ψ	The number of punctured tags
t'_ψ	The ψ-th punctured tag
P	The punctured tag list, where $P = \{t'_1, \cdots, t'_\psi\}$
$sk_{R,\psi-1}$	The secret key with the punctured tag $t'_{\psi-1}$
$\mathbf{td}_\psi$	The trapdoor with the punctured tag t'_ψ

Lemma 3: [41] Given three integers $n, m, q \in \mathbb{Z}$, four matrices $\mathbf{N}_1, \mathbf{R} \in \mathbb{Z}_q^{n\times m}, \mathbf{N}_2 \in \mathbb{Z}_q^{m\times m}, \mathbf{G} \in \mathbb{Z}_q^{n\times m}$, and a basis $\mathbf{T}_\mathbf{G}$ of $\Lambda_q^\perp(\mathbf{G})$, the $\mathsf{ExtendLeft}(\mathbf{N}_1, \mathbf{G}, \mathbf{T}_\mathbf{G}, \mathbf{N}_2)$ algorithm returns a basis $\widetilde{\mathbf{T}_{(\mathbf{N}_1|\mathbf{N}_1\mathbf{N}_2+\mathbf{G})}} \in \mathbb{Z}^{2m\times 2m}$ of $\Lambda_q^\perp(\mathbf{N}_1|\mathbf{N}_1\mathbf{N}_2+\mathbf{G})$, s.t. $\|\widetilde{\mathbf{T}_{(\mathbf{N}_1|\mathbf{N}_1\mathbf{N}_2+\mathbf{G})}}\| \le \|\widetilde{\mathbf{T}_\mathbf{G}}\|(1+\|\mathbf{N}_2\|_2)$.

Lemma 4: [40] Given four integers $n, m, m', q \in \mathbb{Z}$, a matrix $\mathbf{M} \in \mathbb{Z}_q^{n\times m}$, a basis $\mathbf{T}_\mathbf{M} \in \mathbb{Z}^{m\times m}$ of $\Lambda^\perp(\mathbf{M})$, and a parameter $\sigma' \ge \|\widetilde{\mathbf{T}_\mathbf{M}}\|\omega(\sqrt{\log m})$, the $\mathsf{RandBasis}(\mathbf{M}, \mathbf{T}_\mathbf{M}, \sigma')$ algorithm returns a basis $\mathbf{T}'_\mathbf{M} \in \mathbb{Z}^{m\times m}$ of $\Lambda^\perp(\mathbf{M})$, s.t. $\|\widetilde{\mathbf{T}'_\mathbf{M}}\| \le \sigma'\sqrt{m}$.

Lemma 5: [40] Given four positive integers $n, m, q, k \in \mathbb{Z}$, where $q \ge 2$, and $m \ge 2n\log q$, a matrix $\mathbf{M} \in \mathbb{Z}_q^{n\times km}$, a basis $\mathbf{T}_{\mathbf{M}_\mathcal{N}}$ of $\Lambda_q^\perp(\mathbf{M}_\mathcal{N})$, a set $\mathcal{N} \subseteq [k]$, a vector $\mathbf{u} \in \mathbb{Z}_q^n$, and a parameter $\sigma \ge \|\widetilde{\mathbf{T}_{\mathbf{M}_\mathcal{N}}}\| \cdot \omega(\sqrt{\log km})$, the $\mathsf{GenSamplePre}(\mathbf{M}, \mathbf{T}_{\mathbf{M}_\mathcal{N}}, \mathcal{N}, \mathbf{u}, \sigma)$ algorithm returns a vector $\mathbf{e} \in \mathbb{Z}^{km}$ over $\mathcal{D}_{\Lambda_q^\mathbf{u}(\mathbf{M}),\sigma}$, s.t. $\mathbf{Me} = \mathbf{u} \bmod q$.

Lemma 6: [38] Given five integers $n, m, k, q \in \mathbb{Z}$, a parameter $\sigma > 0$ and two matrices $\mathbf{M} \in \mathbb{Z}_q^{n\times m}$, $\mathbf{U} \in \mathbb{Z}_q^{n\times k}$, if a matrix $\mathbf{K} \in \mathbb{Z}^{m\times k}$ is sampled from $\mathcal{D}_\sigma(\Lambda_q^\mathbf{U}(\mathbf{M}))$ and $\mathbf{S}$ is sampled uniformly in $\{-1, 1\}^{m\times m}$, then $\|\mathbf{K}^\top\| \le \sigma\sqrt{mk}$ and $\|\mathbf{S}^\top\| \le 20\sqrt{m}$.

Lemma 7: [41] Given a prime $q > 2$, two integers $m > (n+1)\log q + \omega(\log n)$ and $k = k(n)$, and three matrices $\mathbf{K} \in \mathbb{Z}_q^{n\times m}$, $\mathbf{E} \in \mathbb{Z}_q^{n\times k}$ and $\mathbf{F} \in \{-1, 1\}^{m\times k}$, the distribution $(\mathbf{K}, \mathbf{KF}, \mathbf{F}^\top\mathbf{r})$ is close to $(\mathbf{K}, \mathbf{E}, \mathbf{F}^\top\mathbf{r})$, $\forall \mathbf{r} \in \mathbb{Z}_q^m$.

IV. PROBLEM FORMULATION

In this section, we show the problem formulation of PunSearch, and the acronyms are defined in Tab. I.

A. System Model

Our PunSearch scheme involved four entities, Trusted authority (TA), Data sender, Data receiver, and Cloud server (CS), as presented in Fig. 2.

- **Trusted authority (TA)** is in charge of initializing the entire system and calculating the public and initial secret keys of the data receiver. When TA maintains a punctured tag list $P = \{t_1\}$, it has the ability to generate a punctured secret key promptly based on dynamic data search requirements, which is used to revoke the searchability of the data receiver for a specific keyword encrypted associated with t_1.
- **Data sender** can extract the keyword from a collection of data files. After receiving a keyword, a data receiver's public key, and a tag list T, the data sender calculates the ciphertext, and outsources it to CS.
- **Data receiver** is accountable for generating a trapdoor through a keyword together with its secret key, and uploads it to CS. If the permission verification is valid (i.e. $P \cap T = \emptyset$) and the trapdoor is matched with the corresponding ciphertext, the data receiver will obtain a search result from CS.
- **Cloud server (CS)** is designed to store the ciphertext with a tag list T outsourced by the data sender. After obtaining a trapdoor sent by the data receiver, CS performs the ciphertext search operations if the permission verification is valid (i.e. $P \cap T = \emptyset$).

B. Formal Definition

Our PunSearch scheme $\Pi_{\mathrm{PunSearch}}$ includes six algorithms, i.e., **Setup**, **KeyGen**$_R$, **Encrypt**, **Puncture**, **Trapdoor**, **Search**, as described below.

- $pp \leftarrow$ **Setup**(1^λ): TA inputs a security parameter λ, this algorithm calculates a public parameter pp.
- $(pk_R, sk_{R,\emptyset}) \leftarrow$ **KeyGen**$_R(pp)$: TA inputs a public parameter pp, this algorithm calculates the public and initial secret keys $(pk_R, sk_{R,\emptyset})$ for a data receiver.
- $\mathrm{CT} \leftarrow$ **Encrypt**$(pp, pk_R, \mathbf{ck}, T)$: The data sender inputs a public parameter pp, a public key pk_R of data receiver, a keyword $\mathbf{ck}$, and a tag list T, this algorithm calculates a ciphertext CT with $\mathbf{ck}$ and T.
- $sk_{R,\psi} \leftarrow$ **Puncture**$(pp, sk_{R,\psi-1}, t'_\psi)$: TA inputs a public parameter pp, a secret key $sk_{R,\psi-1}$ of data receiver with a punctured tag $t'_{\psi-1}$, and a punctured tag t'_ψ, this algorithm calculates a secret key $sk_{R,\psi}$ with a punctured t'_ψ.
- $\mathrm{TD}_\psi \leftarrow$ **Trapdoor**$(pp, pk_R, sk_{R,\psi}, \mathbf{tk})$: The data receiver inputs a public parameter pp, the public key pk_R and secret key $sk_{R,\psi}$ of data receiver with a punctured tag t'_ψ, and a keyword $\mathbf{tk}$, this algorithm calculates a trapdoor TD_ψ with $\mathbf{tk}$ and t'_ψ.
- "Success" or "Failure" $\leftarrow$ **Search**$(pp, \mathrm{CT}, T, \mathrm{TD}_\psi, P)$: CS inputs a public parameter pp, a ciphertext CT, a tag list T, and a trapdoor TD_ψ and a punctured tag list P, this algorithm outputs "Success" if the trapdoor meets the permission verification and matches the ciphertext. Otherwise, this algorithm outputs "Failure".

C. Security Model

We provide a novel security model for our PunSearch scheme, named PunSearch for ciphertext indistinguishability against chosen keyword attacks (IND-Pun-CKA), which includes several interactions of an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$, the specific model $\mathbf{Exp}^{\text{IND-Pun-CKA}}_{\text{PunSearch},\mathcal{A}}(\lambda)$ is defined as follows:

1) **Initialize**: A challenge query index $q^* \in \mathbb{N}^+$, a challenge tag list $T^* = (t_1^*, \cdots, t_d^*)$, the hash function $H : \mathbb{Z}_q^n \to \mathbb{Z}_q^{n\times n}$, and two empty sets P and C are given. For each query q_i, $\mathcal{C}$ maintains a set $\mathcal{Q}$, which is empty initially.

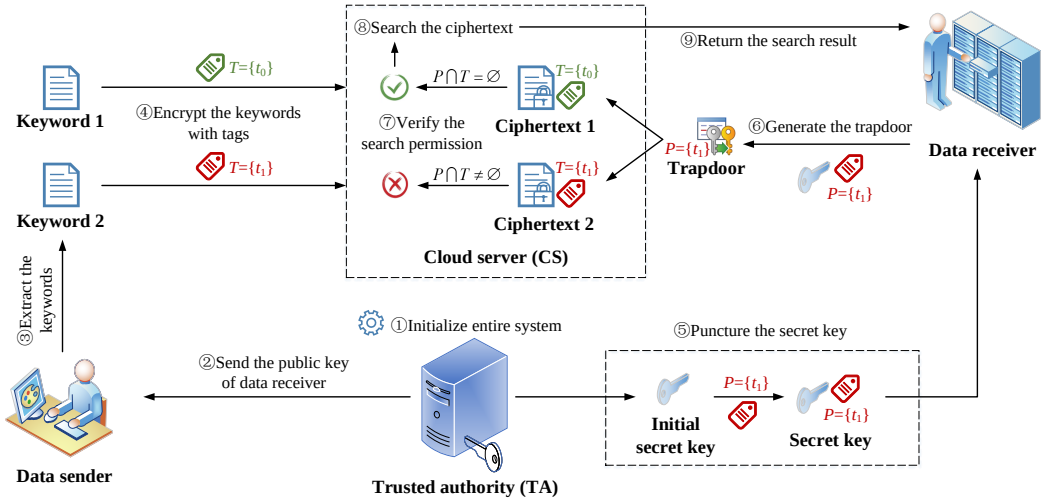


Fig. 2. System model of our PunSearch scheme for cloud storage.

2) **Setup**: Given a security parameter λ, $\mathcal{C}$ invokes the **Setup**(1^λ) and **KeyGen**$_R(pp)$ algorithms to obtain a public parameter pp and the public and initial secret keys $(pk_R, sk_{R,\emptyset})$ of data receiver. Then, $\mathcal{C}$ maintains a set $\mathcal{Q}$ used to record the tuple $(q_i, sk_{R,\psi}, P, C)$. Finally, $\mathcal{C}$ returns pp and pk_R to $\mathcal{A}$.
3) **Phase 1**: $\mathcal{A}$ is responsible for querying these oracles.
   a) Hash Queries $\mathcal{O}_H$: Given a keyword $\mathbf{ck}$ from $\mathcal{A}$, $\mathcal{C}$ keeps a list $\mathcal{H}$ and searches $\mathbf{ck}$ in it, and then sends $H(\mathbf{ck})$ to $\mathcal{A}$.
   b) Ciphertext Queries $\mathcal{O}_{\mathrm{CT}}$: Given a keyword $\mathbf{ck}$, a public key pk_R of data receiver, and a tag list $T = (t_1, \cdots, t_d)$ from $\mathcal{A}$, $\mathcal{C}$ invokes the **Encrypt**$(pp, pk_R, \mathbf{ck}, T)$ algorithm to calculate the ciphertext CT, and then transmits it to $\mathcal{A}$.
   c) Puncture Queries $\mathcal{O}_{Pun}$: Given a query index q_i and a punctured tag t'_ψ from $\mathcal{A}$, $\mathcal{C}$ performs the following procedures. If there exists $(q_i, sk_{R,\psi-1}, P, C)$ in $\mathcal{Q}$, $\mathcal{C}$ calls the **Puncture**$(pp, sk_{R,\psi-1}, t'_\psi)$ algorithm to calculate the secret key $sk_{R,\psi}$ with the punctured tag t'_ψ, where $P = P \cup \{t'_\psi\}$, and replaces $\{q_i, sk_{R,\psi-1}, P, C\}$ to $\{q_i, sk_{R,\psi}, P, C\}$ in $\mathcal{Q}$. Otherwise, $\mathcal{C}$ invokes **KeyGen**$_R(pp)$ and **Puncture**$(pp, sk_{R,\emptyset}, t'_\psi)$ to calculate the secret key $sk_{R,\psi}$ with $P = \{t'_\psi\}$, and creates a new tuple $(q_i, sk_{R,\psi}, P, C)$ in $\mathcal{Q}$, where $C = \emptyset$.
   d) Corrupt Queries $\mathcal{O}_{Cor}$: Given a query index q_i from $\mathcal{A}$, $\mathcal{C}$ executes the following procedures:
      - $q_i \neq q^*$: If there exists $(q_i, sk_{R,\psi-1}, P, C)$ in $\mathcal{Q}$, $\mathcal{C}$ sends $sk_{R,\psi-1}$ to $\mathcal{A}$, and assigns $C = P$. Otherwise, $\mathcal{C}$ invokes **KeyGen**$_R(pp)$ algorithm to calculate the initial secret key $sk_{R,\emptyset}$, and sends it to $\mathcal{A}$. Then, $\mathcal{C}$ assigns $C = P$, and creates a new tuple $(q_i, sk_{R,\emptyset}, P, C)$ in $\mathcal{Q}$.
      - $q_i = q^*$: If it exists $(q^*, sk_{R,\psi-1}, P, C)$ in $\mathcal{Q}$, $\mathcal{C}$ checks whether $P \cap T^* = \emptyset$. If so, $\mathcal{C}$ sends $\perp$ to $\mathcal{A}$. Otherwise, $\mathcal{C}$ sends the most novel secret key $sk_{R,\psi-1}$ of data receiver to $\mathcal{A}$. If there does not exist $(q^*, sk_{R,\psi-1}, P, C)$ in $\mathcal{Q}$, $\mathcal{C}$ sends $\perp$ to $\mathcal{A}$.

      In subsequent procedures, $\mathcal{C}$ returns $\perp$ for all the $\mathcal{O}_{Cor}$ queries from $\mathcal{A}$.
   e) Trapdoor Queries $\mathcal{O}_{\mathrm{TD}}$: Given a keyword $\mathbf{tk}$ and a public key pk_R of data receiver, $\mathcal{C}$ invokes the **Trapdoor**$(pp, pk_R, sk_{R,\psi}, \mathbf{tk})$ to calculate a trapdoor TD_ψ, and then transmits it to $\mathcal{A}$.
4) **Challenge**: $\mathcal{A}$ selects two challenge keywords $\mathbf{ck}_0^*, \mathbf{ck}_1^* \in \mathbb{Z}_q^n$ which have not been queried in **Phase 1**, and transmits them to $\mathcal{C}$. Then, $\mathcal{C}$ chooses a random bit $b \in \{0, 1\}$, and calculates the ciphertext CT_b^* using **Encrypt**$(pp, pk_R, \mathbf{ck}_b^*, T^*)$ algorithm. Finally, $\mathcal{C}$ returns it to $\mathcal{A}$.
5) **Phase 2**: $\mathcal{C}$ responds all queries from $\mathcal{A}$ as showed in **Phase 1**, but either $\mathbf{ck}_0^*$ or $\mathbf{ck}_1^*$ cannot be queried in $\mathcal{O}_{\mathrm{CT}}$ and $\mathcal{O}_{\mathrm{TD}}$.
6) **Guess**: $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$. If $b' = b$, $\mathcal{A}$ wins this game.

The advantage of the adversary $\mathcal{A}$ to win the above-mentioned $\mathbf{Exp}^{\text{IND-Pun-CKA}}_{\text{PunSearch},\mathcal{A}}(\lambda)$ is defined as follows:

$$\mathsf{Adv}^{\text{IND-Pun-CKA}}_{\text{PunSearch},\mathcal{A}}(\lambda) = |\Pr[b' = b] - \frac{1}{2}|.$$

Definition 7: Our PunSearch scheme enjoys IND-Pun-CKA security, if the advantage of a PPT adversary $\mathcal{A}$ to win the above-mentioned $\mathbf{Exp}^{\text{IND-Pun-CKA}}_{\text{PunSearch},\mathcal{A}}(\lambda)$ is negligible.

V. The Design of PunSearch

In this section, we first provide the design rationale of PunSearch, and then describe our design in detail. After that, we give the parameter settings and correctness analysis.

A. Design Rationale

Traditional lattice-based SE schemes can provide the encrypted search for cloud-assisted data sharing, but they can not support the fine-grained searchability revocation for specific keywords, which is impractical for cloud storage systems. PE is a novel primitive formalized in [Oakland'15], offering the decryption revocation mechanism through puncturing the secret key with a tag list [30]. Thus, the puncture property is expected to be integrated into the SE scheme to improve its practicality, which motivates this work to design a puncturable encrypted search scheme (abbr. PunSearch).

To achieve it, we introduce the $\mathsf{ExtendLeft}$ and $\mathsf{RandBasis}$ algorithm to puncture a secret key of the data receiver. As mentioned in Section I, how to generate a search trapdoor from a punctured secret key and realize the search is our main technical challenge. An intuitive approach is to combine the PE with a lattice-based SE scheme. However, most existing SE schemes invoke the $\mathsf{SamplePre}$ or $\mathsf{SampleLeft}$ algorithms to generate a search trapdoor. After directly mapping the keyword to a matrix $\mathbf{A_{tk}}$, we are unable to find a suitable basis of $\Lambda_q^{\perp}(\mathbf{A} \mid \mathbf{A_{tk}} \mid \mathbf{A}_{f_{t',1}} \mid \cdots \mid \mathbf{A}_{f_{t',\psi}})$ to invoke the $\mathsf{SamplePre}$ (or $\mathsf{SampleLeft}$) algorithm, because this punctured secret key is usually a basis of $\Lambda_q^{\perp}(\mathbf{A} \mid \mathbf{A}_{f_{t',1}} \mid \cdots \mid \mathbf{A}_{f_{t',\psi}})$. The trapdoor cannot be generated validly, therefore combining the PE and SE primitives directly to design an encrypted search scheme becomes unrealistic.

To address this problem, we first introduce the gadget matrix $\mathbf{G}$ to embed the keyword $\mathbf{tk}$ into a matrix $\mathbf{A_{tk}}$. Then, we generate serval matrices $\mathbf{A}_{f_{t',1}}, \cdots, \mathbf{A}_{f_{t',\psi}}$ with punctured tag list P through evaluation algorithm. After that, we leverage the $\mathsf{GenSamplePre}$ technique to sample a vector as the trapdoor, by inputting a matrix $\mathbf{A} \mid \mathbf{A_{tk}} \mid \mathbf{A}_{f_{t',1}} \mid \cdots \mid \mathbf{A}_{f_{t',\psi}}$ together with a basis of $\Lambda_q^{\perp}(\mathbf{A} \mid \mathbf{A}_{f_{t',1}} \mid \cdots \mid \mathbf{A}_{f_{t',\psi}})$ (i.e. a punctured secret key). Till now, we have addressed the trapdoor generation problem. Moreover, we design a novel permission verification method for the search procedure to determine whether this trapdoor has the searchability for a specific ciphertext. In this way, if a secret key has been punctured by a tag t', the trapdoor generated from it can not search the ciphertext with tag list T. Otherwise, the ciphertext search will be executed to return a result for the data receiver. Consequently, the fine-grained searchability revocation for specific keywords has been realized.

B. System Initialization

To begin with, TA inputs a security parameter 1^{λ}, and invokes $\mathbf{Setup}(1^{\lambda})$ to obtain a public parameter pp, which will be distributed to other entities.

TA first initializes several integers $n, m, q, d \in \mathbb{Z}$, a parameter σ, and a gadget matrix $\mathbf{G} \in \mathbb{Z}_q^{n \times m}$. Then, TA selects a vector $\mathbf{u} \in \mathbb{Z}_q^n$ uniformly, and defines a collision-resistant hash function $H : \mathbb{Z}_q^n \to \mathbb{Z}_q^{n \times n}$. Finally, TA returns the public parameter as $pp := (n, m, q, \sigma, d, \mathbf{G}, \mathbf{u}, H)$, and transmits it to other entities.

C. Key Generation

This phase is used for the data receiver's key generation. After input the public parameter pp, TA calls the $\mathbf{KeyGen}_R(pp)$ algorithm to obtain the public and initial secret keys $(pk_R, sk_{R,\emptyset})$, and transmits them to the data receiver.

Firstly, TA invokes $(\mathbf{A}, \mathbf{T_A}) \leftarrow \mathsf{TrapGen}(n, m, q)$ to generate a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and a basis $\mathbf{T_A} \in \mathbb{Z}^{m \times m}$ of lattice $\Lambda_q^{\perp}(\mathbf{A})$. After that, it selects $d+1$ random matrices $\mathbf{A}_0, \mathbf{A}_1, \cdots, \mathbf{A}_d \in \mathbb{Z}_q^{n \times m}$ used to calculate the ciphertext. The public and initial secret keys are defined as:

$$pk_R := (\mathbf{A}, \mathbf{A}_0, \mathbf{A}_1, \cdots, \mathbf{A}_d), sk_{R,\emptyset} := \mathbf{T_A}.$$

Ultimately, TA transmits the key pair $(pk_R, sk_{R,\emptyset})$ to the data receiver through a confidential channel.

D. Ciphertext Generation

The data owner selects a keyword $\mathbf{ck} \in \mathbb{Z}_q^n$ for subsequent ciphertext search. Before generating the ciphertext, the data owner initially defines a tag list $T := (t_1, \cdots, t_d)$ and uses the public parameter pp, the data receiver's public key pk_R, and the keyword $\mathbf{ck}$ as inputs to execute the $\mathbf{Encrypt}(pp, pk_R, \mathbf{ck}, T)$ algorithm, generating keyword ciphertext CT with T.

First of all, the data sender calculates a matrix $\mathbf{A_{ck}} = \mathbf{A}_0 + H(\mathbf{ck})\mathbf{G} \in \mathbb{Z}_q^{n \times m}$ to embed the keyword $\mathbf{ck}$. Subsequently, the data sender selects several vectors $\mathbf{s} \in \mathbb{Z}_q^n$ and $\mathbf{e}_0 \leftarrow \chi^m$, a value $e_2 \leftarrow \chi$, and many matrices $\mathbf{R_{ck}}, \mathbf{R}_{t,1}, \cdots, \mathbf{R}_{t,d} \in \{-1, 1\}^{m \times m}$, and calculates a matrix $\mathbf{A}_{\mathbf{ck},T} = (\mathbf{A} \mid \mathbf{A_{ck}} \mid \mathbf{A}_1 + t_1\mathbf{G} \mid \cdots \mid \mathbf{A}_d + t_d\mathbf{G}) \in \mathbb{Z}_q^{n \times (d+2)m}$ for the tag list $T = (t_1, \cdots, t_d)$. After that, the data sender calculates

$$\mathbf{c}_1 = \mathbf{A}_{\mathbf{ck},T}^{\top}\mathbf{s} + \mathbf{e}_1 \in \mathbb{Z}_q^{(d+2)m}, c_2 = \mathbf{u}^{\top}\mathbf{s} + e_2 \in \mathbb{Z}_q,$$

where the error term $\mathbf{e}_1$ as

$$\begin{aligned}\mathbf{e}_1 &= (\mathbf{I}_m \mid \mathbf{R_{ck}} \mid \mathbf{R}_{t,1} \mid \cdots \mid \mathbf{R}_{t,d})^{\top}\mathbf{e}_0 \\ &= (\mathbf{e}_0^{\top} \mid (\mathbf{R}_{\mathbf{ck}}^{\top}\mathbf{e}_0)^{\top} \mid (\mathbf{R}_{t,1}^{\top}\mathbf{e}_0)^{\top} \mid \cdots \mid (\mathbf{R}_{t,d}^{\top}\mathbf{e}_0)^{\top})^{\top} \\ &:= (\mathbf{e}_0^{\top} \mid \mathbf{e}_{\mathbf{ck}}^{\top} \mid \mathbf{e}_{t,1}^{\top} \mid \cdots \mid \mathbf{e}_{t,d}^{\top})^{\top} \in \mathbb{Z}_q^{(d+2)m}.\end{aligned}$$

In the end, the data sender defines the ciphertext $\mathrm{CT} := (\mathbf{c}_1, c_2)$ with the keyword $\mathbf{ck}$ and the tag list T, and then outsources it to CS.

E. Puncture Phase

The puncture phase is performed by the TA, and is dedicated to generating a new secret key with a punctured tag for the data receiver. TA maintains a list P used to record the punctured tags. After input a public parameter pp, a secret key $sk_{R,\psi-1}$ with the punctured tag t'_{ψ}, TA calls $\mathbf{Puncture}(pp, sk_{R,\psi-1}, t'_{\psi})$ algorithm, and then outputs a secret key $sk_{R,\psi}$ with the punctured tag t'_{ψ}. Specifically, this phase has been divided into two cases.

1) If $P = \emptyset$: TA evaluates $\mathbf{A}_{f_{t',1}} \leftarrow \mathsf{Eval}_{pk}(\{\mathbf{A}_i\}_{i=1}^d, f_{t',1})$ to generate a matrix $\mathbf{A}_{f_{t',1}} \in \mathbb{Z}_q^{n \times m}$. Subsequently, TA invokes $\mathsf{ExtendRight}$ and $\mathsf{RandBasis}$ algorithm to generate and randomize a basis as

$$\begin{aligned}\mathbf{T}_{t',1} &\leftarrow \mathsf{ExtendRight}(\mathbf{A}, \mathbf{T_A}, \mathbf{A}_{f_{t',1}}), \\ \widehat{\mathbf{T}_{t',1}} &\leftarrow \mathsf{RandBasis}(\mathbf{A} \mid \mathbf{A}_{f_{t',1}}, \mathbf{T}_{t',1}, \sigma_1),\end{aligned}$$

where $\mathbf{T}_{t',1} \in \mathbb{Z}^{2m \times 2m}$ is a basis of lattice $\Lambda_q^{\perp}(\mathbf{A} \mid \mathbf{A}_{f_{t',1}})$, $\widehat{\mathbf{T}_{t',1}} \in \mathbb{Z}^{2m \times 2m}$ is a randomizing basis of lattice $\Lambda_q^{\perp}(\mathbf{A} \mid$

$\mathbf{A}_{f_{t',1}}$) generated from $\mathbf{T}_{t',1}$, and $\sigma_1 = \omega(\alpha_F(n)\sqrt{\log m})$. As a result, TA updates the punctured list $P = P \cup \{t'_1\}$, and then return a new secret key $sk_{R,1} = \widehat{\mathbf{T}_{t',1}}$ with the punctured tag t'_1 to the data receiver.

2) If $P \neq \emptyset$: Assume that $P = \{t'_1, \cdots, t'_{\psi-1}\}$, TA evaluates $\mathbf{A}_{f_{t',\psi}} \leftarrow \mathsf{Eval}_{pk}(f_{t',\psi}, \{\mathbf{A}_i\}_{i=1}^{d})$ to generate a matrix $\mathbf{A}_{f_{t',\psi}} \in \mathbb{Z}_q^{n\times m}$. The ExtendRight and RandBasis algorithms are invoked by TA to generate a basis as

$$\mathbf{T}_{t',\psi} \leftarrow \mathsf{ExtendRight}(\mathbf{A} \mid \mathbf{A}_{f_{t',1}} \mid \cdots \mid \mathbf{A}_{f_{t',\psi-1}}, \widehat{\mathbf{T}_{t',\psi-1}}, \mathbf{A}_{f_{t',\psi}}),$$
$$\widehat{\mathbf{T}_{t',\psi}} \leftarrow \mathsf{RandBasis}(\mathbf{A} \mid \mathbf{A}_{f_{t',1}} \mid \cdots \mid \mathbf{A}_{f_{t',\psi}}, \mathbf{T}_{t',\psi}, \sigma_\psi),$$

where $\mathbf{T}_{t',\psi} \in \mathbb{Z}^{(\psi+1)m\times(\psi+1)m}$ is a basis of lattice $\Lambda_q^{\perp}(\mathbf{A} \mid \mathbf{A}_{f_{t',1}} \mid \cdots \mid \mathbf{A}_{f_{t',\psi}})$, $\widehat{\mathbf{T}_{t',\psi}} \in \mathbb{Z}^{(\psi+1)m\times(\psi+1)m}$ is a randomizing basis of lattice $\Lambda_q^{\perp}(\mathbf{A} \mid \mathbf{A}_{f_{t',1}} \mid \cdots \mid \mathbf{A}_{f_{t',\psi}})$ generated from $\mathbf{T}_{t',\psi}$, and $\sigma_\psi = \sigma_1(\sqrt{m\log m})^{\psi-1}$. Eventually, TA updates the punctured list $P = P \cup \{t'_\psi\}$, and then sends a new secret key $sk_{R,\psi} = \widehat{\mathbf{T}_{t',\psi}}$ with the punctured tag t'_ψ to the data receiver.

For $\forall t' \in P = \{t'_1, \cdots, t'_\psi\}$, the trapdoor calculated by $sk_{R,1}$ (or $sk_{R,\psi}$) is unable to search for ciphertext with the tag list T if $t' \in T$, which can revoke this trapdoor's searchability for specific keywords.

F. Trapdoor Generation

In order to generate a trapdoor utilized own secret key $sk_{R,\psi}$ with a punctured tag t'_ψ, the data receiver has the ability to call $\mathbf{Trapdoor}(pp, pk_R, sk_{R,\psi}, \mathbf{tk})$ algorithm after inputting the public parameter pp, the public and secret keys $(pk_R, sk_{R,\psi})$ and a keyword $\mathbf{tk} \in \mathbb{Z}_q^n$ to be searched. The specific procedure is as follows.

At the beginning, the data receiver calculates a matrix $\mathbf{A}_{\mathbf{tk}} = \mathbf{A}_0 + H(\mathbf{tk})\mathbf{G} \in \mathbb{Z}_q^{n\times m}$ to embed the keyword $\mathbf{tk}$. For $j \in P = \{t'_1, \cdots, t'_\psi\}$, $\mathbf{A}_{f_{t',j}} \leftarrow \mathsf{Eval}_{pk}(\{\mathbf{A}_i\}_{i=1}^{d}, f_{t',j})$ is evaluated by the data receiver to generate a matrix $\mathbf{A}_{f_{t',j}} \in \mathbb{Z}_q^{n\times m}$. Following that, the data receiver samples a vector $\mathbf{td}_\psi \in \mathbb{Z}_q^{(\psi+2)m}$ as

$$\mathbf{td}_\psi \leftarrow \mathsf{GenSamplePre}(\mathbf{A} \mid \mathbf{A}_{\mathbf{tk}} \mid \mathbf{A}_{f_{t',1}} \mid \cdots \mid \mathbf{A}_{f_{t',\psi}}, \widehat{\mathbf{T}_{t',\psi}}, \{1, 3, \cdots, \psi+2\}, \mathbf{u}, \sigma),$$

such that $(\mathbf{A} \mid \mathbf{A}_{\mathbf{tk}} \mid \mathbf{A}_{f_{t',1}} \mid \cdots \mid \mathbf{A}_{f_{t',\psi}})\mathbf{td}_\psi = \mathbf{u}(\mod q)$.

At last, the data receiver uploads this trapdoor $\mathrm{TD}_\psi := \mathbf{td}_\psi$ with the keyword $\mathbf{tk}$ and the punctured tag t'_ψ to CS.

G. Search Phase

For each ciphertext CT with the tag list T, CS inputs the parameter pp, a trapdoor TD_ψ with the punctured tag t'_ψ, and calls the $\mathbf{Search}(pp, \mathrm{CT}, T, \mathrm{TD}_\psi, P)$ algorithm to execute two-level procedures and then finds the search results.

1) The permission verification: For the punctured tag list P, if $P \cap T \neq \emptyset$, i.e., if there exists $j \in \{1, \cdots, \psi\}$, such that

$$t'_j \in T \iff f_{t',j}(T) \neq 0,$$

this algorithm outputs "Failure" to the data receiver meaning that the trapdoor does not have the searchability for the keyword CT. Otherwise, CS executes the following procedure.

2) The ciphertext search: Parse $\mathrm{CT} = (\mathbf{c}_1, c_2)$, where $\mathbf{c}_1 = (\mathbf{c}^\top \mid \mathbf{c}_{\mathbf{ck}}^\top \mid \mathbf{c}_{t,1}^\top \mid \cdots \mid \mathbf{c}_{t,d}^\top)^\top$. For $j \in \{1, \cdots, \psi\}$, CS evaluates Eval_{ct} algorithm to generate a vector $\mathbf{c}_{f_{t',j}} \in \mathbb{Z}_q^m$ as

$$\mathbf{c}_{f_{t',j}} \leftarrow \mathsf{Eval}_{ct}(f_{t',j}, \{\mathbf{A}_i, \mathbf{c}_i, t_i\}_{i=1}^{d}).$$

After that, CS calculates a vector $\mathbf{c}'_1 = (\mathbf{c}^\top \mid \mathbf{c}_{\mathbf{ck}}^\top \mid \mathbf{c}_{f_{t',1}}^\top \mid \cdots \mid \mathbf{c}_{f_{t',\psi}}^\top)^\top \in \mathbb{Z}_q^{(\psi+2)m}$, and computes a value as

$$\eta = c_2 - \mathbf{td}_\psi^\top \mathbf{c}'_1.$$

If $|\eta| < \lfloor \frac{q}{4} \rfloor$, this algorithm outputs "Success" to the data receiver, which means that the ciphertext CT and the trapdoor TD_ψ correspond to the same keyword. Otherwise, this algorithm outputs "Failure" to the data receiver.

H. Correctness Analysis and Parameters Setting

Given a tag list $T = (t_1, \cdots, t_d)$ and a punctured list $P = \{t'_1, \cdots, t'_\psi\}$, assume that a data receiver's public key is $pk_R = (\mathbf{A}, \mathbf{A}_0, \mathbf{A}_1, \cdots, \mathbf{A}_d)$, a ciphertext $\mathrm{CT} = (\mathbf{c}_1, c_2)$ with $\mathbf{ck}$ and a trapdoor is $\mathrm{TD}_\psi = \mathbf{td}_\psi$ with $\mathbf{tk}$.

If $f_{t',j}(T) = 0$ where $j \in \{1, \cdots, \psi\}$ and $\mathbf{ck} = \mathbf{tk}$:

$$\begin{aligned}
\eta &= c_2 - \mathbf{td}_\psi^\top \mathbf{c}'_1 = \mathbf{u}^\top \mathbf{s} + e_2 - \mathbf{td}_\psi^\top (\mathbf{c}^\top \mid \mathbf{c}_{\mathbf{ck}}^\top \mid \mathbf{c}_{f_{t',1}}^\top \mid \cdots \mid \mathbf{c}_{f_{t',\psi}}^\top)^\top \\
&= \mathbf{u}^\top \mathbf{s} + e_2 - \mathbf{td}_\psi^\top ((\mathbf{A}^\top \mathbf{s} + \mathbf{e}_0)^\top \mid (\mathbf{A}_{\mathbf{ck}}^\top \mathbf{s} + \mathbf{e}_{\mathbf{ck}})^\top \mid ((\mathbf{A}_{f_{t',1}} \\
&\quad + f_{t',1}(T)\mathbf{G})^\top \mathbf{s} + \mathbf{e}_{f_{t',1}})^\top \mid \cdots \mid ((\mathbf{A}_{f_{t',\psi}} + f_{t',\psi}(T)\mathbf{G})^\top \mathbf{s} \\
&\quad + \mathbf{e}_{f_{t',\psi}})^\top)^\top \\
&= \mathbf{u}^\top \mathbf{s} + e_2 - ((\mathbf{A} \mid \mathbf{A}_{\mathbf{ck}} \mid \mathbf{A}_{f_{t',1}} \mid \cdots \mid \mathbf{A}_{f_{t',\psi}})\mathbf{td}_\psi)^\top \mathbf{s} \\
&\quad - \mathbf{td}_\psi^\top (\mathbf{e}_0^\top \mid \mathbf{e}_{\mathbf{ck}}^\top \mid \mathbf{e}_{f_{t',1}}^\top \mid \cdots \mid \mathbf{e}_{f_{t',\psi}}^\top)^\top \\
&= \mathbf{u}^\top \mathbf{s} + e_2 - \mathbf{u}^\top \mathbf{s} - \mathbf{td}_\psi^\top (\mathbf{e}_0^\top \mid \mathbf{e}_{\mathbf{ck}}^\top \mid \mathbf{e}_{f_{t',1}}^\top \mid \cdots \mid \mathbf{e}_{f_{t',\psi}}^\top)^\top \\
&= e_2 - \mathbf{td}_\psi^\top (\mathbf{e}_0^\top \mid \mathbf{e}_{\mathbf{ck}}^\top \mid \mathbf{e}_{f_{t',1}}^\top \mid \cdots \mid \mathbf{e}_{f_{t',\psi}}^\top)^\top.
\end{aligned}$$

Then, we have that:

$$\begin{aligned}
\|\eta\| &\leq \|e_2\| + \|\mathbf{td}_\psi^\top\| \cdot \|(\mathbf{e}_0^\top \mid \mathbf{e}_{\mathbf{ck}}^\top \mid \mathbf{e}_{f_{t',1}}^\top \mid \cdots \mid \mathbf{e}_{f_{t',\psi}}^\top)^\top\| \\
&\leq B + \sigma\sqrt{(\psi+2)m}(B + 20\sqrt{m}B + \psi\alpha_F(n)B).
\end{aligned}$$

Based on this, we provide the parameter settings as:

- $B \geq \sqrt{n}\omega(\log n)$ for LWE assumption.
- $m \geq \lceil 2n\log q \rceil$ for TrapGen lemma.
- $\alpha_F(n) > \sqrt{n\log m}$ for evaluation algorithms.
- $\sigma_1 = \omega(\alpha_F(n)\sqrt{\log m})$ and $\{\sigma_i = \sigma_1(\sqrt{m\log m})^{i-1}\}_{i=2}^{\psi}$ for ExtendRight and RandBasis lemmas.
- $\sigma \geq (\psi+1)m \cdot \omega(\log(\psi+1)m)$ for GenSamplePre lemma.
- $B + \sigma\sqrt{(\psi+2)m}(B + 20\sqrt{m}B + \psi\alpha_F(n)B) < \frac{q}{4}$ for correctness.

VI. Security Analysis

In this section, we prove the IND-Pun-CKA security of our PunSearch scheme.

Theorem 1: If the $\mathrm{LWE}_{n,m,q,\chi}$ assumption holds, our PunSearch scheme satisfies IND-Pun-CKA security in the ROM. For any PPT adversary $\mathcal{A}$, if $\mathcal{A}$ disrupts our scheme with a non-negligible advantage ϵ, then we can build a PPT challenger $\mathcal{C}$ to address the $\mathrm{LWE}_{n,m,q,\chi}$ assumption with a non-negligible probability.

Proof Given a challenge query index $q^* \in \mathbb{N}^+$, a challenge tag list $T^* = (t_1^*, \cdots, t_d^*)$, the hash function $H : \mathbb{Z}_q^n \to \mathbb{Z}_q^{n\times n}$,

and two empty sets P and C. For each query q_i, $\mathcal{C}$ maintains a set $\mathcal{Q}$, which is empty initially.

$\widehat{\textbf{Game 0}}$: This is equivalent to the IND-Pun-CKA security model defined in Section IV. C. Specifically, $\mathcal{C}$ invokes **Setup**(1^λ) algorithm to initialize this system, and responds to all the queries from $\mathcal{A}$. Then, $\mathcal{A}$ chooses two challenge keyword $\mathbf{ck}_0^*, \mathbf{ck}_1^* \in \mathbb{Z}_q^n$ which have not been queried in **Phase 1**, and transmits them to $\mathcal{C}$. After that, $\mathcal{C}$ selects a bit $b \in \{0,1\}$, invokes **Encrypt**$(pp, pk_R, \mathbf{ck}_b^*, T^*)$ algorithm to compute a challenge ciphertext CT_b, and then returns to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs $b' \in \{0,1\}$ and wins this game if $b' = b$.

$\widehat{\textbf{Game 1}}$: The $\widehat{\textbf{Game 1}}$ is equivalent to $\widehat{\textbf{Game 0}}$, except that the calculation way of $\mathbf{A}_0, \mathbf{A}_1, \cdots, \mathbf{A}_d$. In $\widehat{\textbf{Game 1}}$, $\mathcal{C}$ selects many matrices $\mathbf{R}_{\mathbf{ck}}^*, \mathbf{R}_{t,1}^*, \cdots, \mathbf{R}_{t,d}^* \in \{-1,1\}^{m\times m}$ and $\mathbf{h}^* \leftarrow \mathbb{Z}_q^{n\times n}$ randomly, and then calculates $\mathbf{A}_0 = \mathbf{A}\mathbf{R}_{\mathbf{ck}}^* - \mathbf{h}^*\mathbf{G}$, $\mathbf{A}_1 = \mathbf{A}\mathbf{R}_{t,1}^* - t_1^*\mathbf{G}$, $\cdots$, $\mathbf{A}_d = \mathbf{A}\mathbf{R}_{t,d}^* - t_d^*\mathbf{G}$. According to Lemma 7, $\mathbf{A}\mathbf{R}_{\mathbf{ck}}^*, \mathbf{A}\mathbf{R}_{t,1}^*, \cdots, \mathbf{A}\mathbf{R}_{t,d}^*$ are indistinguishable with uniform distribution. Consequently, $\widehat{\textbf{Game 0}}$ and $\widehat{\textbf{Game 1}}$ cannot be distinguished statistically. Based on above-mentioned settings, when $\mathcal{A}$ executes the queries in **Phase 1**, $\mathcal{C}$ executes the following procedures to obtain the answers:

- Hash Queries $\mathcal{O}_H$: Assume q_H represents the maximum number of hash queries performed by $\mathcal{A}$, and $\mathcal{C}$ selects a random value $\omega^* \in [q_H]$ and maintains a list $\mathcal{H}$. For $i \in [q_H]$, $\mathcal{A}$ submits a keyword $\mathbf{ck}_i$ to query its hash value $H(\mathbf{ck}_i)$. If $i = \omega^*$, C sets $H(\mathbf{ck}_i) = \mathbf{h}^*$, updates $\mathcal{H} = \mathcal{H} \cup \{\mathbf{ck}_i, H(\mathbf{ck}_i)\}$, and returns $H(\mathbf{ck}_i)$ to $\mathcal{A}$. Otherwise, if there exists $\{\mathbf{ck}_i, H(\mathbf{ck}_i)\}$ in $\mathcal{H}$, $\mathcal{C}$ sends $H(\mathbf{ck}_i)$ to $\mathcal{A}$. Otherwise, $\mathcal{C}$ selects a random matrix $\mathbf{h} \leftarrow \mathbb{Z}_q^{n\times n}$ which is not included in $\mathcal{H}$, sets $H(\mathbf{ck}_i) = \mathbf{h}$, updates $\mathcal{H} = \mathcal{H} \cup \{\mathbf{ck}_i, H(\mathbf{ck}_i)\}$, and returns $H(\mathbf{ck}_i)$ to $\mathcal{A}$.
- Ciphertext Queries $\mathcal{O}_{\mathrm{CT}}$: After receiving a keyword $\mathbf{ck}$ and a tag list $T = (t_1, \cdots, t_d)$ from $\mathcal{A}$, $\mathcal{C}$ calls $\mathrm{CT} \leftarrow$ **Encrypt**$(pp, pk_R, \mathbf{ck}, T)$ to calculate a ciphertext CT with $\mathbf{ck}$ and T, and transmits it to $\mathcal{A}$.
- Puncture Queries $\mathcal{O}_{Pun}$: After obtaining a query index q_i and a punctured tag t'_ψ from $\mathcal{A}$, $\mathcal{C}$ proceeds as follow.
  - $q_i \neq q^*$: If there exists $(q_i, sk_{R,\psi-1}, P, C)$ in $\mathcal{Q}$, $\mathcal{C}$ calls $sk_{R,\psi} \leftarrow$ **Puncture**$(pp, sk_{R,\psi-1}, t'_\psi)$ to obtain the secret key $sk_{R,\psi}$ with the punctured tag t'_ψ, where $P = P \cup \{t'_\psi\}$, and replaces $\{q_i, sk_{R,\psi-1}, P, C\}$ to $\{q_i, sk_{R,\psi}, P, C\}$ in $\mathcal{Q}$. Otherwise, $\mathcal{C}$ invokes $(\mathbf{A}, \mathbf{T}_{\mathbf{A}}) \leftarrow \mathsf{TrapGen}(n, m, q)$ to obtain the initial secret key $sk_{R,\emptyset} = \mathbf{T}_{\mathbf{A}}$ of data receiver. Then, $\mathcal{C}$ calls $sk_{R,\psi} \leftarrow$ **Puncture**$(pp, sk_{R,\emptyset}, t'_\psi)$, and sets $P = \{t'_\psi\}$ and $C = \emptyset$. For further queries, $\mathcal{C}$ calls **Puncture** algorithm accordingly, and returns $sk_{R,\psi-1}$ to $\mathcal{A}$.
  - $q_i = q^*$: If there exists $(q^*, -, P, C)$, $\mathcal{C}$ sets $P = P \cup \{t'_\psi\}$, and replaces $(q^*, -, P, C)$ to the new tuple. Otherwise, $\mathcal{C}$ sets $P = P \cup \{t'_\psi\}$ and $C = \emptyset$, and constructs $(q^*, -, P, C)$ as a new tuple. In this circumstance, $\mathcal{C}$ does not generate the punctured secret key to $\mathcal{A}$.
- Corrupt Queries $\mathcal{O}_{Cor}$: After receiving a query index q_i from $\mathcal{A}$, $\mathcal{C}$ executes the following procedures.
  - $q_i \neq q^*$: If there exists $(q_i, sk_{R,\psi-1}, P, C)$, $\mathcal{C}$ returns $sk_{R,\psi-1}$ to $\mathcal{A}$, and assigns $C = P$. Otherwise, $\mathcal{C}$ sends the initial secret key $sk_{R,\emptyset}$ to $\mathcal{A}$, and assigns $C = P = \emptyset$. In subsequent procedures, $\mathcal{C}$ returns $\perp$ for all the queries from $\mathcal{A}$.
  - $q_i = q^*$: If there exists $(q^*, -, P, C)$, $\mathcal{C}$ checks if $P \cap T^* = \emptyset$. If $P \cap T^* = \emptyset$, $\mathcal{C}$ returns $\perp$ to $\mathcal{A}$. Otherwise, it means that there exists a punctured tag $t'_k \in P$, such that $f_{t',k}(T^*) \neq 0$. We assume that $P = \{t'_1, \cdots, t'_k\}$, if $k > 1$, the t_1 and t_k need to be swapped to construct $P = \{t'_k, t'_2, \cdots, t'_{k-1}, t'_1\}$. Then, $\mathcal{C}$ calculates $\mathbf{R}_{f_{t',k}}^* \leftarrow \mathsf{Eval}_{sim}(f_{t',k}, \mathbf{A}, \{\mathbf{R}_j^*, t_j^*\}_{j=1}^d)$, and sets $\mathbf{A}_{f_{t',k}} = \mathbf{A}\mathbf{R}_{f_{t',k}}^* - f_{t',k}(T^*)\mathbf{G}$. After that, $\mathcal{C}$ invokes $\mathbf{T}_{\mathbf{A}|\mathbf{A}_{f_{t',k}}} \leftarrow \mathsf{ExtendLeft}(\mathbf{A}, -f_{t',k}(T^*)\mathbf{G}, \mathbf{T}_{\mathbf{G}}, \mathbf{R}_{f_{t',k}}^*)$, $\mathbf{T}_{t',k} \leftarrow \mathsf{ExtendRight}(\mathbf{A} \mid \mathbf{A}_{f_{t',k}}, \mathbf{T}_{\mathbf{A}|\mathbf{A}_{f_{t',k}}}, \mathbf{A}_{f_{t',2}} \mid \cdots \mid \mathbf{A}_{f_{t',1}})$ and $\widehat{\mathbf{T}_{t',k}} \leftarrow \mathsf{RandBasis}(\mathbf{A} \mid \mathbf{A}_{f_{t',k}} \mid \cdots \mid \mathbf{A}_{f_{t',1}}, \mathbf{T}_{t',k}, \sigma_k)$. Finally, $\mathcal{C}$ outputs $sk_{R,\psi} = \widehat{\mathbf{T}_{t,k}}$ to $\mathcal{A}$. Otherwise, $\mathcal{C}$ sets $P = \emptyset$, and outputs $\perp$ to $\mathcal{A}$. In subsequent procedures, $\mathcal{C}$ returns $\perp$ for all the queries from $\mathcal{A}$.
- Trapdoor Queries $\mathcal{O}_{\mathrm{TD}}$: After receiving a query index q_i, a keyword $\mathbf{tk}$ and a public key pk_R of data receiver from $\mathcal{A}$, $\mathcal{C}$ executes the following procedures.
  - $q_i \neq q^*$: $\mathcal{C}$ obtains the secret key $sk_{R,\psi}$ using the same way in $\mathcal{O}_{Pun}$, calls **Trapdoor**$(pp, pk_R, sk_{R,\psi}, \mathbf{tk})$ algorithm to obtain a trapdoor $\mathrm{TD}_\psi = \mathbf{td}_\psi$, and transmits it to $\mathcal{A}$.
  - $q_i = q^*$: If $H(\mathbf{tk}) = \mathbf{h}^*$, $\mathcal{C}$ aborts this game, and the probability of abort is at most $\frac{1}{q_H}$. We assume that there exists a query index $\omega^* \in [q_H]$, due to the collision-resistance property of H, for the query $i \in [q_H] \subset \{\omega^*\}$, we can hold that $H(\mathbf{tk}) \neq \mathbf{h}^*$. Otherwise, for $P = \{t'_k, t'_2, \cdots, t'_{k-1}, t'_1\}$, $\mathcal{C}$ invokes $\mathbf{td}_k \leftarrow \mathsf{GenSamplePre}(\mathbf{A} \mid \mathbf{A}_{\mathbf{tk}} \mid \mathbf{A}_{f_{t',k}} \mid \cdots \mid \mathbf{A}_{f_{t',1}}, \widehat{\mathbf{T}_{t,k}}, \{1, 3, \cdots, \psi+2\}, \sigma)$ to generate the trapdoor $\mathbf{td}_k$. Finally, $\mathcal{C}$ returns $\mathrm{TD}_\psi = \mathbf{td}_k$ to $\mathcal{A}$.

$\widehat{\textbf{Game 2}}$: The $\widehat{\textbf{Game 2}}$ is equivalent to $\widehat{\textbf{Game 1}}$, except that the calculation way of the challenge ciphertext CT^*. In $\widehat{\textbf{Game 2}}$, $\mathcal{C}$ selects CT^* from $\mathbb{Z}^{(d+2)m} \times \mathbb{Z}_q$ randomly. Thus, $\mathcal{A}$ can not have the advantage in $\widehat{\textbf{Game 2}}$.

Reduction to LWE: Assume that an adversary $\mathcal{A}$ can distinguish $\widehat{\textbf{Game 1}}$ and $\widehat{\textbf{Game 2}}$ with non-negligible probability, we can construct an algorithm $\mathcal{B}$ to solve $\mathrm{LWE}_{n,m,q,\chi}$ assumption with non-negligible probability.

- **LWE instance**: $\mathcal{B}$ initializes two LWE instances $(\mathbf{A}, \mathbf{c_A}) \in \mathbb{Z}_q^{n\times m} \times \mathbb{Z}_q^m$, and $(\mathbf{u}, c_{\mathbf{u}}) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, which are either random (i.e. $\mathbf{c_A} \xleftarrow{\$} \mathbb{Z}_q^m$, $c_{\mathbf{u}} \xleftarrow{\$} \mathbb{Z}_q$) or pseudo-random (i.e. satisfying $\mathbf{c_A} = \mathbf{A}^\top\mathbf{s} + \mathbf{e}_0$, $c_{\mathbf{u}} = \mathbf{u}^\top\mathbf{s} + e_2$), where $\mathbf{s} \in \mathbb{Z}_q^n$, $\mathbf{e}_0 \in \chi^m$, and $e_2 \in \chi$. After receiving the answers from $\mathcal{A}$, $\mathcal{B}$ needs to distinguish these two afore-mentioned cases.
- **Initialize**: A challenge query index $q^* \in \mathbb{N}^+$, a challenge tag list $T^* = (t_1^*, \cdots, t_d^*)$, the hash function $H : \mathbb{Z}_q^n \rightarrow$

$\mathbb{Z}_q^{n\times n}$, and two empty sets P and C are given. For each query q_i, $\mathcal{C}$ maintains a set $\mathcal{Q}$, which is empty initially.

- **Setup**: $\mathcal{B}$ invokes $(\mathbf{A}, \mathbf{T_A}) \leftarrow \mathsf{TrapGen}(n, m, q)$ to obtain a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and a basis $\mathbf{T_A} \in \mathbb{Z}^{m\times m}$ firstly. Then, $\mathcal{B}$ chooses many matrices $\mathbf{R}^*_{\mathbf{ck}}, \mathbf{R}^*_{t,1}, \cdots, \mathbf{R}^*_{t,d} \in \{-1, 1\}^{m\times m}$ and a hash value $\mathbf{h}^* \in \mathbb{Z}_q^{n\times n}$ randomly, and calculates $\mathbf{A}_0 = \mathbf{A}\mathbf{R}^*_{\mathbf{ck}} - \mathbf{h}^*\mathbf{G}, \mathbf{A}_1 = \mathbf{A}\mathbf{R}^*_{t,1} - t^*_1\mathbf{G}, \cdots, \mathbf{A}_d = \mathbf{A}\mathbf{R}^*_{t,d} - t^*_d\mathbf{G}$. Finally, $\mathcal{B}$ sends $pp = \{n, m, q, \sigma, d, \mathbf{G}, \mathbf{u}, H\}$ and $pk_R = \{\mathbf{A}, \mathbf{A}_0, \mathbf{A}_1, \cdots, \mathbf{A}_d\}$ to $\mathcal{A}$, and keeps $sk_{R,\emptyset} = \mathbf{T_A}$.
- **Phase 1**: $\mathcal{B}$ responds all queries from $\mathcal{A}$ as showed in $\widehat{\textbf{Game 1}}$.
- **Challenge**: After receiving two challenge keywords $\mathbf{ck}^*_0, \mathbf{ck}^*_1 \in \mathbb{Z}_q^n$ which have not been queried in **Phase 1**, $\mathcal{B}$ selects a random bit $b \in \{0, 1\}$, and calculates the ciphertext of $\mathbf{ck}^*_b$ as follows:

$$\mathbf{c}^*_1 = (\mathbf{I}_m \mid \mathbf{R}_{\mathbf{ck}^*} \mid \mathbf{R}^*_{t,1} \mid \cdots \mid \mathbf{R}^*_{t,d})^\top \mathbf{c}_{\mathbf{A}}, c^*_2 = c_{\mathbf{u}}.$$

  Then, $\mathcal{B}$ returns the challenge ciphertext $\mathrm{CT}^*_b = (\mathbf{c}^*_1, c^*_2)$ with the challenge tag list T^* to $\mathcal{A}$.
- **Phase 2**: $\mathcal{B}$ responds all queries from $\mathcal{A}$ as showed in **Phase 1**, but either $\mathbf{ck}^*_0$ or $\mathbf{ck}^*_1$ cannot be queried in $\mathcal{O}_{\mathrm{CT}}$ and $\mathcal{O}_{\mathrm{TD}}$.
- **Guess**: $\mathcal{A}$ outputs a guess as to whether it interacts with $\widehat{\textbf{Game 1}}$ and $\widehat{\textbf{Game 2}}$. After that, $\mathcal{B}$ returns the guess from $\mathcal{A}$ to solve the $\mathrm{LWE}_{n,m,q,\chi}$ assumption.

Analysis: If the LWE instances are pseudorandom, we have:

$$\begin{aligned}
\mathbf{c}^*_1 &= (\mathbf{I}_m \mid \mathbf{R}_{\mathbf{ck}^*} \mid \mathbf{R}^*_{t,1} \mid \cdots \mid \mathbf{R}^*_{t,d})^\top (\mathbf{A}^\top\mathbf{s} + \mathbf{e}_0) \\
&= (\mathbf{A} \mid \mathbf{A}^*_{\mathbf{ck}} \mid \mathbf{A}_1 + t^*_1\mathbf{G} \mid \cdots \mid \mathbf{A}_d + t^*_d\mathbf{G})^\top\mathbf{s} \\
&\quad + (\mathbf{e}_0^\top \mid \mathbf{e}^{*\top}_{\mathbf{ck}} \mid \mathbf{e}^{*\top}_{t,1} \mid \cdots \mid \mathbf{e}^{*\top}_{t,d}) \\
&= \mathbf{H}^{*\top}\mathbf{s} + \mathbf{e}^*_1 \in \mathbb{Z}_q^{(d+2)m}, \\
c^*_2 &= c_{\mathbf{u}} = \mathbf{u}^\top\mathbf{s} + e_2 \in \mathbb{Z}_q,
\end{aligned}$$

where $\mathbf{H}^* = (\mathbf{A} \mid \mathbf{A}^*_{\mathbf{ck}} \mid \mathbf{A}_1 + t^*_1\mathbf{G} \mid \cdots \mid \mathbf{A}_d + t^*_d\mathbf{G})$ and $\mathbf{e}^*_1 = (\mathbf{e}_0^\top \mid \mathbf{e}^{*\top}_{\mathbf{ck}} \mid \mathbf{e}^{*\top}_{t,1} \mid \cdots \mid \mathbf{e}^{*\top}_{t,d})$. Since $\mathbf{A}\mathbf{R}^*_{\mathbf{ck}} = \mathbf{A}_0 + \mathbf{h}^*\mathbf{G} = \mathbf{A}_0 + H(\mathbf{ck}^*_b)\mathbf{G} = \mathbf{A}^*_{\mathbf{ck}}$ holds if and only if $H(\mathbf{ck}^*_b) = \mathbf{h}^*$, the distribution of CT^*_b can correspond to $\widehat{\textbf{Game 1}}$ with probability $\frac{1}{q_H}$.

If the LWE instances are random, we have $\mathbf{c}^*_1$ and c^*_2 are uniform over $\mathbb{Z}^{(d+2)m}$ and $\mathbb{Z}_q$, respectively according to Lemma 7. Thus, the distribution of CT^*_b corresponds to $\widehat{\textbf{Game 2}}$. Consequently, if the advantage of the adversary $\mathcal{A}$ to distinguish between $\widehat{\textbf{Game 1}}$ and $\widehat{\textbf{Game 2}}$ is a non-negligible value ϵ, the advantage of the algorithm $\mathcal{B}$ to solve $\mathrm{LWE}_{n,m,q,\chi}$ assumption is $\frac{\epsilon}{q_H}$, which is also non-negligible. □

VII. Performance Evaluation and Comparison

We now evaluate the computational and communication overhead of PunSearch. To ensure fairness, we compare our results with other lattice-based schemes [15], [19], [20], [21], [22], [23]. All experiments are implemented in Python language on a MacOS system with an Apple M2 CPU, 8GB RAM, and 256GB SSD. Each round of the experiment is carried out independently.

A. Computational Overhead Analysis

To begin with, we perform a theoretical analysis of the computational overhead of **Encrypt**, **Trapdoor**, and **Search** algorithms in our PunSearch scheme together with other lattice-based SE schemes (FS-PEKS [15], ABAEKS [19], Re-PAEKS [20], and IBEDKS [21]), as summarized in Table II.

Concretely, considering the **Encrypt** algorithm, T_H denotes the time cost of $H(\mathbf{ck})$, n^2mT_{Mul} indicates the time cost of $\mathbf{A}_{\mathbf{ck}}$, $(d+2)m^2T_{Mul}$ corresponds to the time cost of $\mathbf{e}_1$, $dnmT_{Mul}$ reflects the time cost of $\mathbf{A}_{\mathbf{ck},T}$, $(d+2)nmT_{Mul}$ and nT_{Mul} represent the time cost of $\mathbf{c}_1$, and c_2, respectively. Therefore, the total cost for encrypting a keyword in PunSearch is $T_H + [n^2m + (d+2)m^2 + (2d+2)nm + n]T_{Mul}$. It indicates that PunSearch is more efficient than others as it only requires hash and matrix multiplication operations, without the time-consuming matrix inversion and lattice basis sampling operations. Due to $T_{NBD} >> T_{SP} + T_{SL} > T_{GSP}$, the computational overhead of our $\mathbf{Trapdoor}$ algorithm is considerably lower than FS-PEKS and IBEDKS, and similar to ABAEKS and Re-PAEKS. Note that T_{Ect} is essentially a constant-level multiplication operation. With regard to the **Search** algorithm, the number of multiplications in these five schemes is proportional to m. As we additionally provide the puncture property, our **Search** algorithm relies on the number of punctured tags ψ. When ψ is small (the normal case), the time cost of our **Search** algorithm is even more efficient than others ([15], [19], [20], [21]) that do not support puncture. Not only the theoretical analysis, but also the results of simulation experiments validate the same, as depicted in Fig. 3(c).

Furthermore, we conduct the experimental simulation analysis as follows. For fairness and security, we configure the parameters $q = 4097$, $n = 16$, $m = \lceil 2n \log q \rceil = 385$, $|att| = 10$, $N = 10$, $d = 10$ and $\psi = 1$, for simulation of our PunSearch scheme and other prior arts.

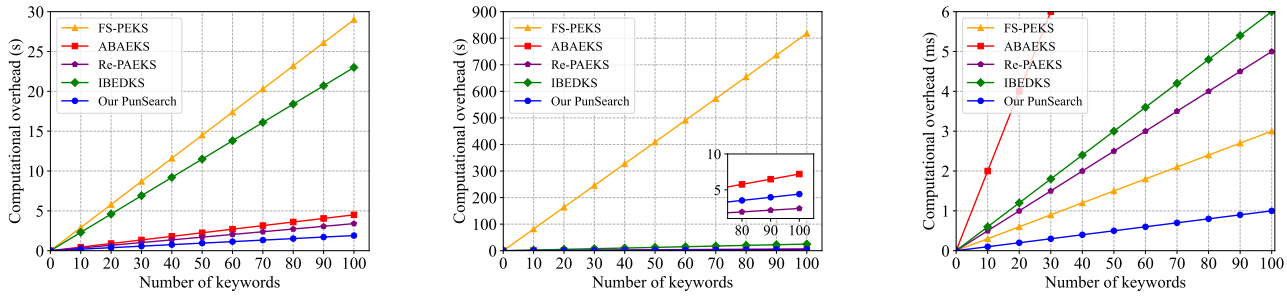
On the one hand, we compare the computational overhead of the **Encrypt**, **Trapdoor** and **Search** algorithms in PunSearch with others (FS-PEKS [15], ABAEKS [19], Re-PAEKS [20] and IBEDKS [21]). Table III illustrates these results for the number of keywords $k = 1$. Specifically, the time costs of these three algorithms in PunSearch are $18.79\mathrm{ms}$, $44.35\mathrm{ms}$, and $0.01\mathrm{ms}$, which are just $0.060\times$, $0.005\times$ and $0.050\times$ compared to [15], [19], [20], [21], respectively for the best cases. Fig. 3 depicts the overhead of **Encrypt**, **Trapdoor** and **Search** algorithms in these schemes, with k ranging from 1 to 100. It observes that the computational overheads of the three algorithms in PunSearch are proportional to k, with the **Encrypt** and **Search** algorithms showing a more gradual increase compared to prior arts. Due to the larger matrix size inputted to GenSamplePre algorithm, the time cost to generate a trapdoor in PunSearch is only higher than one scheme [20]. As trapdoor generation is a one-time procedure, this result is acceptable in practice, which is a trade-off between functionality and practicality.

On the other hand, we also provide the computational overheads of the **Puncture** algorithm in our design together with other PE schemes over lattice (PIBE [22] and PHIBE [23]), as depicted in Fig. 4, for comparing the puncture

TABLE II
THEORETICAL COMPUTATIONAL OVERHEAD COMPARISON

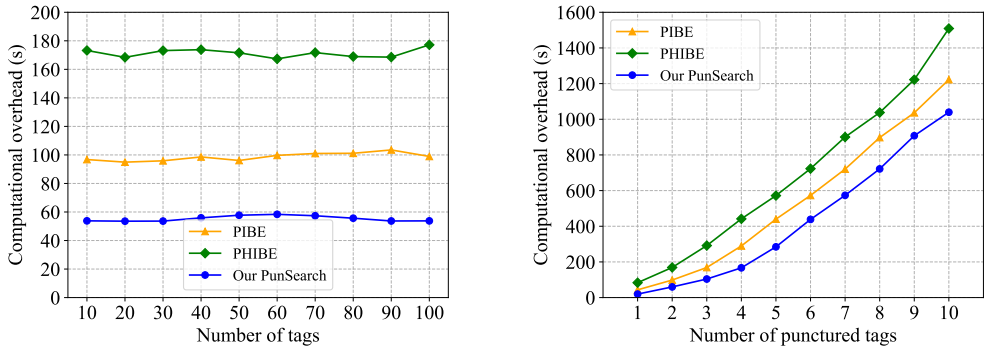
Schemes	Encrypt	Trapdoor	Search
FS-PEKS [15]	$T_H + T_{Inv} + (nm^2 + nml + nl)T_{Mul}$	$T_H + T_{Inv} + T_{NBD} + T_{SP} + nm^2 T_{Mul}$	mlT_{Mul}
ABAEKS [19]	$T_H + T_{SL} + [(2\|att\| + 3)nm + n^2 + (\|att\| + 1)m^2]T_{Mul}$	$T_H + T_{Epk} + T_{SL} + (n^2 + m^2 + 3nm)T_{Mul}$	$T_{Ect} + 5mT_{Mul}$
Re-PAEKS [20]	$T_H + T_{SL} + (n^2 + 3m^2 + 5nm)T_{Mul}$	$T_H + T_{SL} + (n^2 + 3m^2 + 5nm)T_{Mul}$	$8mT_{Mul}$
IBEDKS [21]	$2T_H + T_{Inv} + [n^2m + nm^2 + (n + m)^2 + n + N - 1]T_{Mul}$	$(l + 1)T_H + T_{Inv} + T_{SP} + T_{SL} + (nm^2 + N^2 + N - 2)T_{Mul}$	$(2m + 1)T_{Mul}$
Our PunSearch	$T_H + [n^2m + (d + 2)m^2 + (2d + 2)nm + n]T_{Mul}$	$T_H + \psi T_{Epk} + T_{GSP} + nm^2 T_{Mul}$	$\psi T_{Ect} + (\psi + 2)mT_{Mul}$

Note: l: The security-level of testing; $|att|$: The length of attributes; N: The maximum amount of keyword defined in IBEDKS [21]; d: The number of tags; ψ: The number of punctured tags; k: The number of keywords; T_H: The time cost of hash function; T_{Inv}: The time cost of matrix inversion; T_{Mul}: The time cost of multiplication; T_{NBD}: The time cost of NewBasisDel algorithm; T_{SP}: The time cost of SamplePre algorithm; T_{SL}: The time cost of SampleLeft algorithm; T_{Epk}: The time cost of Eval_{pk} algorithm; T_{Ect}: The time cost of Eval_{ct} algorithm; T_{GSP}: The time cost of GenSamplePre algorithm.



(a) **Encrypt** algorithm (b) **Trapdoor** algorithm (c) **Search** algorithm

Fig. 3. Computational overhead comparison between our PunSearch scheme and other PEKS schemes [15], [19], [20], [21] with the number of keywords k.



(a) **Puncture** algorithm (w.r.t. d) (b) **Puncture** algorithm (w.r.t. ψ)

Fig. 4. Computational overhead evaluation of the **Puncture** algorithm in our PunSearch scheme and other PE schemes [22], [23].

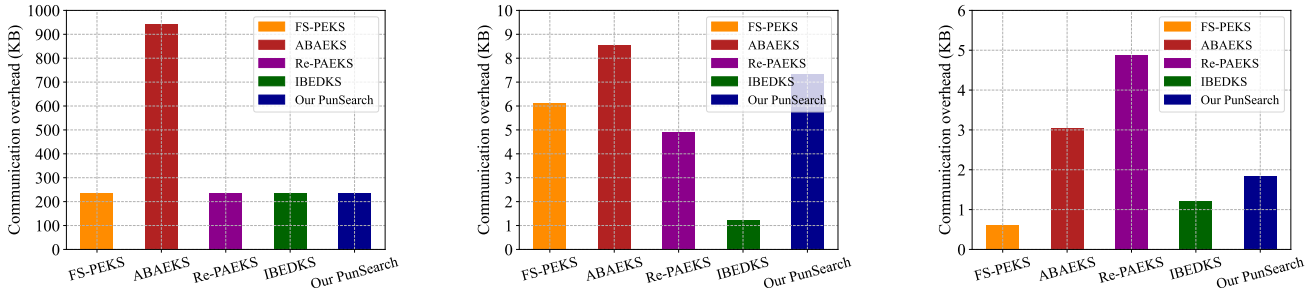
TABLE III
COMPUTATIONAL OVERHEAD COMPARISON IN $k = 1$ (ms)

Schemes	Encrypt	Trapdoor	Search
FS-PEKS [15]	292.11	8181.90	0.03
ABAEKS [19]	45.37	72.41	0.20
Re-PAEKS [20]	33.80	24.06	0.05
IBEDKS [21]	229.40	245.98	0.06
Our PunSearch	18.79	44.35	0.01

property. Specifically, Fig. 4(a) illustrates the time cost with respect to d when $\psi = 2$, while Fig. 4(b) presents the time cost in relation to ψ when $d = 10$. In our design, the matrix sizes used in the ExtendRight and RandBasis algorithms are smaller compared to those in PIBE and PHIBE. This difference makes the PunSearch scheme markedly more efficient than the other in the context of **Puncture** algorithm. For instance, the time costs for this algorithm in PIBE, PHIBE, and PunSearch are 96.77ms, 173.23ms, and 53.80ms, respectively, when $d = 10$ and $\psi = 2$. As ψ increases, the time cost of ExtendRight and RandBasis algorithms rises dramatically, Consequently, the efficiency of the **Puncture** algorithm becomes more drastically impacted by the number of punctured tags ψ.

B. Communication Overhead Analysis

In Table IV, we compare the communication overhead of PunSearch with other lattice-based SE schemes [15], [19],



(a) Secret key size (b) Ciphertext size (c) Trapdoor size

Fig. 5. Communication overhead comparison between our PunSearch scheme and other PEKS schemes [15], [19], [20], [21].

TABLE IV
THEORETICAL COMMUNICATION OVERHEAD COMPARISON

Schemes	Secret key	Ciphertext	Trapdoor
FS-PEKS [15]	$m^2\|\mathbb{Z}_q\|$	$l(m+1)\|\mathbb{Z}_q\|$	$m\|\mathbb{Z}_q\|$
ABAEKS [19]	$4m^2\|\mathbb{Z}_q\|$	$(\|att\|+4)m\|\mathbb{Z}_q\|$	$5m\|\mathbb{Z}_q\|$
Re-PAEKS [20]	$m^2\|\mathbb{Z}_q\|$	$8m\|\mathbb{Z}_q\|$	$8m\|\mathbb{Z}_q\|$
IBEDKS [21]	$m^2\|\mathbb{Z}_q\|$	$(2m+1)\|\mathbb{Z}_q\|$	$(2m+1)\mathbb{Z}_q$
Our PunSearch	$m^2\|\mathbb{Z}_q\|$	$[(d+2)m+1]\|\mathbb{Z}_q\|$	$(\psi+2)m\|\mathbb{Z}_q\|$

Note: $|\mathbb{Z}_q|$: The size of an element in $\mathbb{Z}_q$.

[20], [21]. In the **KeyGen**$_R$ algorithm, the secret key size in PunSearch is close to [15], [20], and [21], and smaller than that in [19]. Since the secret key must be transmitted to a data receiver after executing the **KeyGen**$_R$ algorithm, a smaller secret key size is more convenient for storage in practical scenarios. For the **Encrypt** algorithm, the ciphertext size in PunSearch relies on d. The ciphertext sizes are related to parameters such as l and $|att|$ in FS-PEKS and ABAEKS, while Re-PAEKS and IBEDKS have relatively constant costs. Although our design does not outperform the other schemes in terms of ciphertext size, it additionally offers fine-grained searchability revocation, which is an acceptable trade-off. Compared to ciphertext, PunSearch incurs a lower communication overhead for transmitting a trapdoor, and its size increases linearly as the number of punctured tags ψ grows.

We set the parameters $|\mathbb{Z}_q| = \lceil \log q \rceil = 13$, $l = 10$, $|att| = 10$, $N = 10$, $d = 10$, $\psi = 1$, and give an experimental comparison of communication overhead for **KeyGen**$_R$, **Encrypt** and **Trapdoor** algorithms in PunSearch and others ([15], [19], [20], [21]), as depicted in Fig. 5. In particular, in Fig. 5(a), we evaluate the key size of different schemes as the communication overhead of the **KeyGen**$_R$ algorithm, where our design is more efficient than other schemes. In Fig. 5(b), the ciphertext sizes of the five schemes are 6.13KB, 8.55KB, 4.89KB, 1.22KB and 7.33KB, respectively. Since our scheme adds tags to the data ciphertext, the ciphertext size of PunSearch does not have an advantage over the other schemes, but this is acceptable. Fig. 5(c) displays the communication overhead of the **Trapdoor** algorithm in PunSearch and these four schemes. The trapdoor size of PunSearch is only 1.83KB, significantly lower than that of ABAEKS and Re-PAEKS, and it also possesses the searchability revocation property for specific keywords compared to FS-PEKS and IBEDKS. Obviously, our PunSearch scheme can effectively reduce the network transmission burden during the communication between the data receiver and the CS.

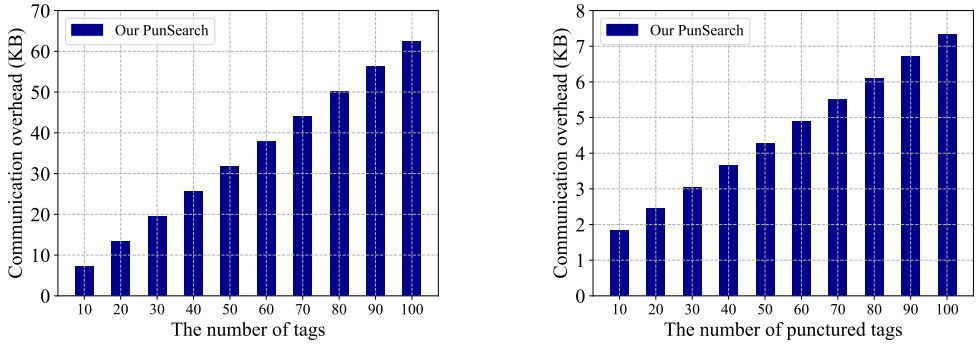
Fig. 5 provides strong support for the theoretical results presented in Table IV. Kindly note that the ciphertext and trapdoor size of PunSearch are $[(d+2)m+1]|\mathbb{Z}_q|$ and $(\psi+2)m|\mathbb{Z}_q|$, respectively, both of which are proportional to d and ψ. Fig. 6 displays the details of our evaluation.

VIII. CONCLUSION

In this paper, we present the first puncturable encrypted search scheme over lattice for outsourced data privacy-preserving in cloud storage systems, named PunSearch. Inspired by the PE primitive, we puncture the secret key of data receivers through the Puncture algorithm. After that, we design a novel trapdoor generation and search algorithm to match a ciphertext with a trapdoor with searchability. Our scheme provides fine-grained searchability revocation for specific keywords and resists quantum computing attacks. The rigorous security analysis reveals that PunSearch enjoys IND-Pun-CKA security in the ROM. Comprehensive performance results illustrate that PunSearch is more efficient than other prior arts in the context of computational overheads. As a future work, we acknowledge that introducing an authenticated encryption to PunSearch to defend against insider keyword guessing attacks is an interesting direction.

REFERENCES

[1] M. Wang, J. Yu, W. Shen, and R. Hao, "Privacy-preserving time-based auditing for secure cloud storage," *IEEE Transactions on Information Forensics and Security*, 2024.
[2] H. Yu, H. Zhang, Z. Yang, and S. Yu, "Edasvic: Enabling efficient and dynamic storage verification for clouds of industrial internet platforms," *IEEE Transactions on Information Forensics and Security*, 2024.
[3] S. Xu, X. Chen, Y. Guo, S.-M. Yiu, S. Gao, and B. Xiao, "Efficient and secure post-quantum certificateless signcryption with linkability for iomt," *IEEE Transactions on Information Forensics and Security*, pp. 1–1, 2024.
[4] G. Xu, D.-l. Kong, K. Zhang, S. Xu, Y. Cao, Y. Mao, J. Duan, J. Kang, and X.-B. Chen, "A model value transfer incentive mechanism for federated learning with smart contracts in aiot," *IEEE Internet of Things Journal*, 2024.
[5] M. Wang, Y. Miao, Y. Guo, H. Huang, C. Wang, and X. Jia, "Aesm 2 attribute-based encrypted search for multi-owner and multi-user distributed systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 1, pp. 92–107, 2022.



(a) Ciphertext size (b) Trapdoor size

Fig. 6. Communication overhead evaluation of the **Puncture** algorithm in PunSearch scheme with the number of tags d and punctured tags ψ.

[6] D. Zhang, S. Wang, Q. Zhang, and Y. Zhang, "Attribute based conjunctive keywords search with verifiability and fair payment using blockchain," *IEEE Transactions on Services Computing*, 2023.

[7] N. Yang, C. Tang, Q. Zhou, and D. He, "Dynamic consensus committee-based for secure data sharing with authorized multi-receiver searchable encryption," *IEEE Transactions on Information Forensics and Security*, 2023.

[8] W. Li, "Multi-receiver data authorization with data search for data sharing in cloud-assisted iov," *IEEE Transactions on Intelligent Transportation Systems*, 2024.

[9] L. Chen, J. Li, J. Li, and J. Weng, "Paess: Public-key authentication encryption with similar data search for pay-per-query," *IEEE Transactions on Information Forensics and Security*, 2024.

[10] L. Cheng and F. Meng, "Server-aided public key authenticated searchable encryption with constant ciphertext and constant trapdoor," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 1388–1400, 2023.

[11] X. Chen, S. Xu, S. Gao, Y. Guo, S.-M. Yiu, and B. Xiao, "Fs-llrs: Lattice-based linkable ring signature with forward security for cloud-assisted electronic medical records," *IEEE Transactions on Information Forensics and Security*, 2024.

[12] G. Tang, B. Pang, L. Chen, and Z. Zhang, "Efficient lattice-based threshold signatures with functional interchangeability," *IEEE Transactions on Information Forensics and Security*, 2023.

[13] W. Sun, S. Yu, W. Lou, Y. T. Hou, and H. Li, "Protecting your right: Verifiable attribute-based keyword search with fine-grained owner-enforced search authorization in the cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 4, pp. 1187–1198, 2014.

[14] W. Zhang, Y. Lin, S. Xiao, J. Wu, and S. Zhou, "Privacy preserving ranked multi-keyword search for multiple data owners in cloud computing," *IEEE Transactions on Computers*, vol. 65, no. 5, pp. 1566–1577, 2015.

[15] X. Zhang, C. Xu, H. Wang, Y. Zhang, and S. Wang, "Fs-peks: Lattice-based forward secure public-key encryption with keyword search for cloud-assisted industrial internet of things," *IEEE Transactions on dependable and secure computing*, vol. 18, no. 3, pp. 1019–1032, 2019.

[16] Y. Lu and J. Li, "Privacy-preserving and forward public key encryption with field-free multi-keyword search for cloud encrypted data," *IEEE Transactions on Cloud Computing*, 2023.

[17] S. Xu, Y. Cao, X. Chen, Y. Zhao, and S.-M. Yiu, "Post-quantum public-key authenticated searchable encryption with forward security: General construction, and applications," in *International Conference on Information Security and Cryptology*. Springer, 2023, pp. 274–298.

[18] X. Zhang, C. Huang, D. Gu, J. Zhang, and H. Wang, "Bib-mks: post-quantum secure biometric identity-based multi-keyword search over encrypted data in cloud storage systems," *IEEE Transactions on Services Computing*, vol. 16, no. 1, pp. 122–133, 2021.

[19] F. Luo, H. Wang, C. Lin, and X. Yan, "Abaeks: Attribute-based authenticated encryption with keyword search over outsourced encrypted data," *IEEE Transactions on Information Forensics and Security*, 2023.

[20] F. Luo, H. Wang, and X. Yan, "Re-paeks: Public-key authenticated re-encryption with keyword search," *IEEE Transactions on Mobile Computing*, 2024.

[21] Z. Lin, H. Li, X. Chen, M. Xiao, and Q. Huang, "Identity-based encryption with disjunctive, conjunctive and range keyword search from lattices," *IEEE Transactions on Information Forensics and Security*, 2024.

[22] P. Dutta, W. Susilo, D. H. Duong, and P. S. Roy, "Puncturable identity-based encryption from lattices," in *Information Security and Privacy: 26th Australasian Conference (ACISP)*. Springer, 2021, pp. 571–589.

[23] P. Dutta, M. Jiang, D. H. Duong, W. Susilo, K. Fukushima, and S. Kiyomoto, "Hierarchical identity-based puncturable encryption from lattices with application to forward security," in *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, 2022, pp. 408–422.

[24] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Advances in Cryptology-EUROCRYPT*. Springer, 2004, pp. 506–522.

[25] L. Xu, W. Li, F. Zhang, R. Cheng, and S. Tang, "Authorized keyword searches on public key encrypted data with time controlled keyword privacy," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 2096–2109, 2019.

[26] P. Xu, S. Tang, P. Xu, Q. Wu, H. Hu, and W. Susilo, "Practical multi-keyword and boolean search over encrypted e-mail in cloud server," *IEEE Transactions on Services Computing*, vol. 14, no. 6, pp. 1877–1889, 2019.

[27] K. Zhang, Z. Jiang, J. Ning, and X. Huang, "Subversion-resistant and consistent attribute-based keyword search for secure cloud storage," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 1771–1784, 2022.

[28] Y. Cao, S. Xu, X. Chen, Y. He, and S. Jiang, "A forward-secure and efficient authentication protocol through lattice-based group signature in vanets scenarios," *Computer Networks*, vol. 214, p. 109149, 2022.

[29] S. Xu, Y. Cao, X. Chen, Y. Guo, Y. Yang, F. Guo, and S.-M. Yiu, "Post-quantum searchable encryption supporting user-authorization for outsourced data management," in *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, 2024, pp. 2702–2711.

[30] M. D. Green and I. Miers, "Forward secure asynchronous messaging from puncturable encryption," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 305–320.

[31] T. V. X. Phuong, R. Ning, C. Xin, and H. Wu, "Puncturable attribute-based encryption for secure data delivery in internet of things," in *IEEE INFOCOM 2018-IEEE conference on computer communications*. IEEE, 2018, pp. 1511–1519.

[32] D. Ghopur, J. Ma, X. Ma, J. Hao, T. Jiang, and X. Wang, "Puncturable key-policy attribute-based encryption scheme for efficient user revocation," *IEEE Transactions on Services Computing*, 2023.

[33] H. Cui and X. Yi, "Secure internet of things in cloud computing via puncturable attribute-based encryption with user revocation," *IEEE Internet of Things Journal*, 2023.

[34] W. Susilo, D. H. Duong, H. Q. Le, and J. Pieprzyk, "Puncturable encryption: a generic construction from delegatable fully key-homomorphic encryption," in *European Symposium on Research in Computer Security (ESORICS)*. Springer, 2020, pp. 107–127.

[35] M. Yang, H. Wang, and D. He, "Puncturable attribute-based encryption from lattices for classified document sharing," *IEEE Transactions on Information Forensics and Security*, 2024.

[36] M. Yang, H. Wang, and D. He, "Pm-abe: Puncturable bilateral fine-

grained access control from lattices for secret sharing," *IEEE Transactions on Dependable and Secure Computing*, 2024.

[37] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *Journal of the ACM (JACM)*, vol. 56, no. 6, pp. 1–40, 2009.

[38] D. Boneh, C. Gentry, S. Gorbunov, S. Halevi, V. Nikolaenko, G. Segev, V. Vaikuntanathan, and D. Vinayagamurthy, "Fully key-homomorphic encryption, arithmetic circuit abe and compact garbled circuits," in *Advances in Cryptology–EUROCRYPT*. Springer, 2014, pp. 533–556.

[39] D. Micciancio and C. Peikert, "Trapdoors for lattices: Simpler, tighter, faster, smaller," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2012, pp. 700–718.

[40] D. Cash, D. Hofheinz, E. Kiltz, and C. Peikert, "Bonsai trees, or how to delegate a lattice basis," *Journal of cryptology*, vol. 25, pp. 601–639, 2012.

[41] S. Agrawal, D. Boneh, and X. Boyen, "Efficient lattice (h) ibe in the standard model," in *Advances in Cryptology–EUROCRYPT*. Springer, 2010, pp. 553–572.