Public-key Authenticated Encryption with Keyword Search: Cryptanalysis, Enhanced Security, and Quantum-resistant Instantiation*

Zi-Yuan Liu†
Department of Computer Science
National Chengchi University
Taipei 11605, Taiwan
zyliu@cs.nccu.edu.tw

Yi-Fan Tseng
Department of Computer Science
National Chengchi University
Taipei 11605, Taiwan
yftseng@cs.nccu.edu.tw

Raylin Tso‡§
Department of Computer Science
National Chengchi University
Taipei 11605, Taiwan
raylin@cs.nccu.edu.tw

Masahiro Mambo
Institute of Science and Engineering
Kanazawa University
Kanazawa 920-1192, Japan
mambo@ec.t.kanazawa-u.ac.jp

Yu-Chi Chen
Department of Computer Science and Engineering
Yuan Ze University
Taoyuan 32003, Taiwan
wycchen@saturn.yzu.edu.tw

ABSTRACT

With the rapid development of cloud computing, an increasing number of companies are adopting cloud storage technology to reduce overhead. However, to ensure the privacy of sensitive data, the uploaded data need to be encrypted before being outsourced to the cloud. The concept of public-key encryption with keyword search (PEKS) was introduced by Boneh *et al.* to provide flexible usage of the encrypted data. Unfortunately, most of the PEKS schemes are not secure against inside keyword guessing attacks (IKGA), so the keyword information of the trapdoor may be leaked to the adversary. To solve this issue, Huang and Li presented public key authenticated encryption with keyword search (PAEKS) in which the trapdoor generated by the receiver is only valid for authenticated ciphertexts. With their seminal work, many PAEKS schemes have been introduced for the enhanced security of PAEKS. Some of them further consider the upcoming quantum attacks. However, our cryptanalysis indicated that in fact, these schemes could not withstand IKGA. To fight against the attacks from quantum adversaries and support the privacy-preserving search functionality, we first introduce a novel generic PAEKS construction in this work. Then, we further present the first quantum-resistant PAEKS instantiation based on lattices. The security proofs show that our instantiation not only satisfies the basic requirements but also achieves enhanced security models, namely the multi-ciphertext indistinguishability and trapdoor privacy. Furthermore, the comparative results indicate that with only some additional expenditure, the proposed instantiation provides more secure properties, making it suitable for more diverse application environments.

CCS CONCEPTS

• **Security and privacy** → *Cryptanalysis and other attacks*; **Public key encryption**; **Management and querying of encrypted data**; **Privacy-preserving protocols**.

KEYWORDS

Cryptanalysis; Generic construction; Keyword search; Public-key authenticated encryption; Post-quantum; Trapdoor privacy.

1 INTRODUCTION

In recent years, with the widespread development of cloud computing technology, the application of cloud storage has become increasingly popular. With the support of cloud storage, users and enterprises can easily reduce the cost of local maintenance and storage. In addition, combined with the Internet of Things devices, cloud storage systems can provide more meta-services and applications. However, as the uploaded data are usually critical and sensitive, ensuring that service providers can properly protect the privacy of data becomes an important issue. Therefore, to avoid privacy leakage, users need to encrypt data before outsourcing them to the cloud. Unfortunately, the encrypted data will lose the flexibility of use, such as search or modification. As the search function can considerably reduce the transmission demand, this function is extremely important for cloud storage.

To resolve this issue, the concept of searchable encryption was first introduced by Song *et al.* [56] and Boneh *et al.* [7]. In these primitives, encrypted data are uploaded along with multiple encrypted keywords by the sender, while the receiver can generate trapdoors for specific keywords. With the trapdoor, the cloud server can perform a search to find the matched encrypted keywords, *i.e.,* they are associated with the same keyword, and return the corresponding encrypted data to the receiver. With the distinction of whether the generation of encrypted keywords and trapdoors is symmetric or asymmetric, searchable encryption can be further divided into symmetric search encryption (SSE) and public-key encryption with keyword search (PEKS).

*In this version, we have fixed the issues pointed out by Keita Emura [20] in the version accepted by ASIACCS 2022 (ePrint - 20211123:100238).

†Also with Graduate School of Natural Science and Technology, Kanazawa University, Kanazawa 920-1192, Japan.

‡Also with Artificial Intelligence and E-learning Center, National Chengchi University, Taipei 11605, Taiwan.

§Corresponding author

The first SSE scheme was presented by Song *et al.* [56] in 2000. Because SSE has an advantage in efficiency, it has been extensively studied [17, 44, 46, 57]. However, in practical applications, SSE has the same problem as symmetric encryption—the key distribution problem. To resolve this problem, Boneh *et al.* [7] combined the concept of public-key encryption and searchable encryption to introduce the first PEKS scheme. In this scheme, the searchable ciphertext (*i.e.,* encrypted keyword) is generated by using the receivers' public keys, while a receiver can generate a trapdoor by using his/her private key and hand it to the cloud server to search for the matching searchable ciphertexts. In addition to proposing the notion of PEKS and its construction, Boneh *et al.* [7] also formalized the security requirement of the PEKS, namely ciphertext indistinguishability (CI), *i.e.,* indistinguishability against chosen keyword attacks (CKA), which ensures that there exists no adversary who can obtain any keyword information from the ciphertext.

However, Byun *et al.* [8] pointed out that only considering CKA is insufficient. The adversary may retrieve the keyword information from the trapdoor by adaptively generating ciphertexts for guessing keywords and performing tests. To model this attack scenario, they further considered the notion of trapdoor privacy (TP), *i.e.,* indistinguishability against keyword guessing attacks (KGA) [53]. This security notion can be divided into outside KGA launched by an external adversary (*e.g.,* eavesdropper) and inside KGA (IKGA) launched by an internal adversary (*e.g.,* malicious cloud server). As discussed in Byun *et al.*'s work [8], the keyword space in PEKS schemes is small and limited, *e.g.,* only 225,000 ($\approx 2^{16}$) words in Merriam-Webster's collegiate dictionary [10]. Consequently, upon a brute force attack, there is a high probability ($\frac{1}{2^{16}}$) that the adversary can obtain the keyword information hidden by the trapdoor.

Although many KGA-secure PEKS schemes have been introduced [5, 12–14, 19, 21, 22, 30, 31, 52, 53, 58–60], it was not until the concept of public-key authenticated encryption with keyword search (PAEKS) was proposed by Huang and Li [27] that the IKGA was solved in the single-server context without the communication between the sender and receiver. In this notion, the trapdoor generated by the receiver is only valid to the ciphertexts that are authenticated by a specified sender. In this way, the adversaries cannot perform KGA by adaptively generating ciphertext for any keyword to test the trapdoors. As the concept of PAEKS solves the privacy concern, many variants PAEKS schemes [11, 26, 36, 38, 40–42, 45, 47–49] have been proposed to be suitable for various scenarios.

1.1 Motivation

MCI and TP Security. Among various PAEKS schemes, Qin *et al.* [49] first considered that each encrypted file is related to multiple searchable ciphertexts in practical scenarios. In this context, PAEKS needs to ensure that no adversary knows whether two searchable ciphertext tuples respectively exist ciphertexts that are related to the same keyword. Hence, they introduced an enhanced security notion called multi-ciphertext indistinguishability (MCI) to model this scenario. More concretely, compared with CI, the adversary in the security model of MCI outputs two keyword tuples and is given the challenge ciphertext tuple corresponding to one of the keyword tuples. The adversary's goal is to point out which keyword tuple generates the challenge ciphertext tuple. We also note that Pan and Li [48] followed this concept and introduced the notion called multi-trapdoor privacy (MTP) to ensure that no adversary knows whether two trapdoor tuples respectively exist trapdoors that are related to the same keyword. Unfortunately, Cheng and Meng [15] recently showed that Pan and Li's scheme [48] not only cannot satisfy MCI but also has flaws in the security proof of MTP.

Quantum-resistant PAEKS. As Shor [54, 55] has confirmed that there exists a quantum algorithm that can be used to crack the foundation of many cryptographic primitives—the discrete logarithm hard assumption, scholars have begun to explore how to construct quantum-resistant PEKS schemes [6, 61]. To further satisfy TP, Zhang *et al.* [62, 63] introduced two lattice-based PEKS schemes that are secure against IKGA by restricting the ciphertext to be authenticated by the sender. However, our cryptanalysis shows that their schemes contain flaws, and therefore, an adversary can directly obtain the keyword information of the trapdoor. In addition, Liu *et al.* [39] introduced a generic PAEKS construction and further presented an instantiation based on NTRU lattices. Unfortunately, their system model is not a "pure" public-key setting. More specifically, their construction requires a trusted authority to assist users in generating their private keys.

Hence, with the above description, it raises an urgent problem:

Can we obtain a secure quantum-resistant PAEKS without the assistance of trusted authorities?

1.2 Our Contribution

In this work, we first cryptanalyze Zhang *et al.*'s lattice-based PEKS schemes [62, 63] and show that their schemes cannot resist the attacks from inside adversary due to their security model exist flaws.

Then, to resolve the problem described in Section 1.1, we present a generic PAEKS construction by adopting smooth projection hash function (SPHF) and PEKS. As a high-level idea, to prevent adversaries from being able to adaptively generate ciphertexts for any keyword and further guess the keyword hidden in the trapdoor, we restrict that the trapdoor generated from a receiver is only valid to the ciphertext generated from a specific sender. To meet this requirement, our strategy is to enable the sender and the receiver to obtain high-entropy randomness without any interaction by utilizing (pseudo-random) SPHF. Through this randomness, both parties can obtain an extended keyword to generate a ciphertext and a trapdoor, respectively, instead of generating them through the original low-entropy keyword. As a result, the adversary cannot perform IKGA by randomly selecting keywords.

In addition, to further achieve the MCI properties, we provide a theoretical result in Theorem 3.2: if the PAEKS algorithm of a PAEKS scheme is probabilistic and the PAEKS scheme satisfies CI, then this PAEKS scheme also satisfies MCI. This interesting result can boost the security of many existing PAEKS schemes.

Eventually, we compile Behnia *et al.*'s PEKS [6], and Li and Wang's SPHF [37] by our generic construction and propose the first quantum-resistant PAEKS scheme based on lattices. In terms of the computational cost and the communication cost, the results show that our instantiation provides more secure properties with only a little additional expenditure.

2 PRELIMINARIES

This section introduces some requisite knowledge, including the background of lattices and the definitions of cryptographic primitives.

2.1 Background of Lattices

2.1.1 Lattices. Here, we briefly summarize the concept of lattices. Let $\mathbf{B} = [\mathbf{b}_1|\cdots|\mathbf{b}_n] \in \mathbb{R}^{m\times n}$, where $\mathbf{b}_1, \cdots, \mathbf{b}_n$ are n linear independent vectors. An m-dimensional lattice Λ generated by $\mathbf{B}$ is defined as $\Lambda(\mathbf{B}) := \{\sum_{i=1}^{n} \mathbf{b}_i a_i \mid a_i \in \mathbb{Z}\}$. Here, $\mathbf{B}$ is called the basis of Λ. In addition, given $n, m, q \in \mathbb{Z}$, $\mathbf{u} \in \mathbb{Z}_q^n$, and $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, we can define two q-ary lattices and a coset as follows:

- $\Lambda_q(\mathbf{A}) := \{\mathbf{y} \in \mathbb{Z}_q^m \mid \exists z \in \mathbb{Z}_q^n, \mathbf{y} = \mathbf{A}^\top \mathbf{z} \bmod q\}$;
- $\Lambda_q^\perp(\mathbf{A}) := \{\mathbf{e} \in \mathbb{Z}_q^m \mid \mathbf{A}\mathbf{e} = 0 \bmod q\}$;
- $\Lambda_q^{\mathbf{u}}(\mathbf{A}) := \{\mathbf{e} \in \mathbb{Z}^m \mid \mathbf{A}\mathbf{e} = \mathbf{u} \bmod q\}$.

2.1.2 Discrete Gaussian Distributions. For any positive real number σ, any center $\mathbf{c} \in \mathbb{Z}^m$, and any $\mathbf{x} \in \mathbb{Z}^m$, we define the Gaussian distribution of $\mathcal{D}_{\sigma,\mathbf{c}}$ by the probability distribution function $\rho_{\sigma,\mathbf{c}}(\mathbf{x}) := \exp(-\pi \cdot \|\mathbf{x} - \mathbf{c}\|^2/\sigma^2)$. Furthermore, for any lattice $\Lambda \subset \mathbb{Z}^m$, we define $\rho_{\sigma,\mathbf{c}}(\Lambda) := \sum_{\mathbf{x}\in\Lambda} \rho_{\sigma,\mathbf{c}}(\mathbf{x})$. Then, the discrete Gaussian distribution over lattice Λ with parameter $(\sigma, \mathbf{c})$ is defined as follows: For any $\mathbf{x} \in \Lambda$, $\mathcal{D}_{\Lambda,\sigma,\mathbf{c}}(\mathbf{x}) := \rho_{\sigma,\mathbf{c}}(\mathbf{x})/\rho_{\sigma,\mathbf{c}}(\Lambda)$.

2.1.3 Lattices with Trapdoors. Next, we introduce the preimage sampleable functions and lattice basis delegation technique.

(1) $\mathsf{TrapGen}(1^n, 1^m, q)$ [4, 43]: For input $n, m, q \in \mathbb{Z}$, this probabilistic polynomial time (PPT) algorithm outputs a pair $(\mathbf{A} \in \mathbb{Z}_q^{n\times m}, \mathbf{T}_\mathbf{A} \in \mathbb{Z}_q^{m\times m})$, where $\mathbf{T}_\mathbf{A}$ is a basis for $\Lambda_q^\perp(\mathbf{A})$, such that the following property holds:
$$\{\mathbf{A} : (\mathbf{A}, \mathbf{T}_\mathbf{A}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)\} \approx \{\mathbf{A} : \mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}\}.$$
Here, $\mathbf{T}_\mathbf{A}$ is called a trapdoor of $\mathbf{A}$.
(2) $\mathsf{SamplePre}(\mathbf{A}, \mathbf{T}_\mathbf{A}, \mathbf{u}, \sigma)$ [24]: For an input matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and its trapdoor $\mathbf{T}_\mathbf{A} \in \mathbb{Z}_q^{m\times m}$, a vector $\mathbf{u} \in \mathbb{Z}_q^n$, and parameter $\sigma \geq \|\widetilde{\mathbf{T}_\mathbf{A}}\| \cdot \omega(\sqrt{\log(m)})$, this PPT algorithm outputs a sample $\mathbf{t} \in \mathbb{Z}_q^m$ from a distribution that is statistically close to $\mathcal{D}_{\Lambda_q^{\mathbf{u}}(\mathbf{A}),\sigma}$ such that $\mathbf{A}\mathbf{t} = \mathbf{u} \bmod q$.
(3) $\mathsf{NewBasisDel}(\mathbf{A}, \mathbf{R}, \mathbf{T}_\mathbf{A}, \sigma)$ [3]: For an input matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, a $\mathbb{Z}_q$-invertible matrix $\mathbf{R}$ sampled from the distribution $\mathcal{D}_{m\times m}$, trapdoor $\mathbf{T}_\mathbf{A}$, and parameter $\sigma \in \mathbb{R}$, this PPT algorithm outputs a short lattice basis $\mathbf{T}_\mathbf{B}$ of $\Lambda_q^\perp(\mathbf{B})$, where $\mathbf{B} = \mathbf{A}\mathbf{R}^{-1}$.
(4) $\mathsf{SampleLeft}(\mathbf{A}, \mathbf{M}, \mathbf{T}_\mathbf{A}, \mathbf{u}, \sigma)$ [2]: For an input matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and its corresponding trapdoor $\mathbf{T}_\mathbf{A} \in \mathbb{Z}_q^{m\times m}$, a matrix $\mathbf{M} \in \mathbb{Z}_q^{n\times m_1}$, a vector $\mathbf{u} \in \mathbb{Z}_q^n$, and a parameter $\sigma \geq \|\widetilde{\mathbf{T}_\mathbf{A}}\| \cdot \omega(\sqrt{\log(m + m_1)})$, this PPT algorithm outputs a sample $\mathbf{t} \in \mathbb{Z}^{m+m_1}$ from a distribution statistically close to $\mathcal{D}_{\Lambda_q^{u}([\mathbf{A}|\mathbf{M}]),\sigma}$ such that $[\mathbf{A}|\mathbf{M}] \cdot \mathbf{t} = \mathbf{u} \bmod q$.

2.2 Public-key Encryption with Keyword Search

In this subsection, we recall the definition of PEKS defined by Boneh *et al.* [7]. A PEKS scheme PEKS consists of the following four algorithms:

- $\mathsf{KeyGen}(1^\lambda)$: Taking as input a security parameter λ, this PPT algorithm outputs a pair of keys $(\mathsf{pk}_{\mathsf{PEKS}}, \mathsf{sk}_{\mathsf{PEKS}})$, where $\mathsf{pk}_{\mathsf{PEKS}}$ is the public key and $\mathsf{sk}_{\mathsf{PEKS}}$ is the private key.
- $\mathsf{PEKS}(\mathsf{pk}_{\mathsf{PEKS}}, kw)$: Taking as input the public key $\mathsf{pk}_{\mathsf{PEKS}}$ and a keyword kw, this PPT algorithm outputs a searchable ciphertext $\mathsf{ct}_{\mathsf{PEKS},kw}$ related to the keyword kw.
- $\mathsf{Trapdoor}(\mathsf{sk}_{\mathsf{PEKS}}, kw')$: Taking as input the private key $\mathsf{sk}_{\mathsf{PEKS}}$ and a keyword kw', this PPT algorithm outputs a trapdoor $\mathsf{td}_{\mathsf{PEKS},kw'}$ related to keyword kw'.
- $\mathsf{Test}(\mathsf{ct}_{\mathsf{PEKS},kw}, \mathsf{td}_{\mathsf{PEKS},kw'})$: Taking as input the searchable ciphertext $\mathsf{ct}_{\mathsf{PEKS},kw}$ and trapdoor $\mathsf{td}_{\mathsf{PEKS},kw'}$, this deterministic algorithm outputs 1 if $\mathsf{ct}_{\mathsf{PEKS},kw}$ and $\mathsf{td}_{\mathsf{PEKS},kw'}$ are related to the same keyword (*i.e.,* $kw = kw'$); otherwise, it outputs 0.

Correctness. For any security parameter λ, any honestly generated key pairs $(\mathsf{pk}_{\mathsf{PEKS}}, \mathsf{sk}_{\mathsf{PEKS}})$, any keywords kw, kw', any ciphertext $\mathsf{ct}_{\mathsf{PEKS},kw} \leftarrow \mathsf{PEKS}(\mathsf{pk}_{\mathsf{PEKS}}, kw)$, and any trapdoor $\mathsf{td}_{\mathsf{PEKS},kw'} \leftarrow \mathsf{Trapdoor}(\mathsf{sk}_{\mathsf{PEKS}}, kw')$, then we have $\Pr[\mathsf{Test}(\mathsf{ct}_{\mathsf{PEKS},kw}, \mathsf{td}_{\mathsf{PEKS},kw'}) = 1] = 1 - \mathbf{negl}(\lambda)$ when $kw = kw'$ and $\Pr[\mathsf{Test}(\mathsf{ct}_{\mathsf{PEKS},kw}, \mathsf{td}_{\mathsf{PEKS},kw'}) = 0] = 1 - \mathbf{negl}(\lambda)$ when $kw \neq kw'$.

Ciphertext Indistinguishability of PEKS. The CI ensures that no PPT adversary can obtain any keyword information from the given challenge ciphertext, even if it can adaptively query the trapdoor oracle for any keyword, except for the challenge keywords. This security requirement is modeled by the following indistinguishability against the chosen keyword attack (IND-CKA) game of PEKS that is interacted by a challenger C and an adversary $\mathcal{A}$.

IND-CKA Game of PEKS:

- **Setup.** After receiving a security parameter λ, C generates $(\mathsf{pk}_{\mathsf{PEKS}}, \mathsf{sk}_{\mathsf{PEKS}})$ by performing the KeyGen algorithm. Then, it sends the public key $\mathsf{pk}_{\mathsf{PEKS}}$ to $\mathcal{A}$ and keeps the private key $\mathsf{sk}_{\mathsf{PEKS}}$ secret.
- **Phase 1.** In this phase, $\mathcal{A}$ is allowed to adaptively issue queries to the trapdoor oracle polynomially many times: for any keyword kw, C generates a trapdoor $\mathsf{td}_{\mathsf{PEKS},kw}$ by performing $\mathsf{Trapdoor}(\mathsf{sk}_{\mathsf{PEKS}}, kw)$ and returns $\mathsf{td}_{\mathsf{PEKS},kw}$ to $\mathcal{A}$.
- **Challenge.** After $\mathcal{A}$ terminates the **Phase 1**, it outputs two challenge keywords kw_0^*, kw_1^* to C. The restriction is that $\mathcal{A}$ never issues these two challenge keywords to the trapdoor oracle. C then randomly chooses a bit $b \in \{0, 1\}$ and returns the challenge ciphertext ct^* to $\mathcal{A}$ by performing $\mathsf{PEKS}(\mathsf{pk}_{\mathsf{PEKS}}, kw_b^*)$.

- **Phase 2.** $\mathcal{A}$ can continue to query the trapdoor oracle as in **Phase 1** for any keyword kw, except for the challenge keywords (*i.e.,* $kw \notin \{kw_0^*, kw_1^*\}$).
- **Guess.** Finally, $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$ as its answer, and wins the game if $b = b'$.

The advantage of $\mathcal{A}$ winning the above game is defined as

$$Adv_{\mathcal{A}}^{CI-PEKS}(\lambda) := \left|\Pr[b = b'] - \frac{1}{2}\right|.$$

Definition 2.1 (Ciphertext Indistinguishability of PEKS). A PEKS scheme is called CI (or IND-CKA secure) if, for any PPT adversary $\mathcal{A}$, $Adv_{\mathcal{A}}^{CI-PEKS}(\lambda)$ is negligible.

2.3 Labelled Public-key Encryption Scheme

A labelled public-key encryption (PKE) scheme can be viewed as the variant of PKE. As described in [1], a labelled PKE scheme PKE consists of the following three algorithms:

- $\mathsf{KeyGen}(1^\lambda)$: Taking as input a security parameter λ, this PPT algorithm outputs a pair of keys $(\mathsf{ek}_{\mathsf{PKE}}, \mathsf{dk}_{\mathsf{PKE}})$, where $\mathsf{ek}_{\mathsf{PKE}}$ is the public encryption key and $\mathsf{dk}_{\mathsf{PKE}}$ is the private decryption key.
- $\mathsf{Encrypt}(\mathsf{ek}_{\mathsf{PKE}}, \mathsf{label}, \mathsf{m}_{\mathsf{PKE}}; \rho)$: Taking as input the public encryption key $\mathsf{ek}_{\mathsf{PKE}}$, a label label, a plaintext $\mathsf{m}_{\mathsf{PKE}}$, and a randomness ρ, this PPT algorithm outputs a ciphertext $\mathsf{ct}_{\mathsf{PKE}}$.
- $\mathsf{Decrypt}(\mathsf{dk}_{\mathsf{PKE}}, \mathsf{label}, \mathsf{ct}_{\mathsf{PKE}})$: Taking as input the private decryption key $\mathsf{dk}_{\mathsf{PKE}}$, a label label, and a ciphertext $\mathsf{ct}_{\mathsf{PKE}}$, this deterministic algorithm outputs a plaintext $\mathsf{m}_{\mathsf{PKE}}$ or $\perp$.

In addition, it must to satisfy the following correctness and security.

- Correctness: For any security parameter λ, any pair of keys $(\mathsf{dk}_{\mathsf{PKE}}, \mathsf{ek}_{\mathsf{PKE}}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, any label label, any plaintext $\mathsf{m}_{\mathsf{PKE}}$, any randomness ρ, and any ciphertext $\mathsf{ct}_{\mathsf{PKE}} \leftarrow \mathsf{Encrypt}(\mathsf{ek}_{\mathsf{PKE}}, \mathsf{label}, \mathsf{m}_{\mathsf{PKE}}; \rho)$, a labelled PKE scheme is correct if
$$\Pr[\mathsf{Decrypt}(\mathsf{dk}_{\mathsf{PKE}}, \mathsf{label}, \mathsf{ct}_{\mathsf{PKE}}) = \mathsf{m}_{\mathsf{PKE}}] = 1 - \mathbf{negl}(\lambda).$$
- IND-CPA/IND-CCA1/IND-CCA2 security: Informally, we say that a labelled PKE scheme has indistinguishability against chosen-plaintext attacks (IND-CPA) if there is no adversary that can obtain any information about the challenge plaintext. Suppose that the adversary is allowed to query the decryption oracle for any ciphertext, except for the challenge ciphertext, then we call it indistinguishability against chosen-ciphertext attacks (IND-CCA2) security. Here we note that if the adversary cannot continuously query the oracles after obtaining the challenge ciphertext, we call it IND-CCA1 security.

2.4 Smooth Projective Hash Functions

The SPHF was first introduced by Cramer and Shoup [16] to transform an IND-CPA secure encryption scheme into IND-CCA2 security. Besides, various extended definitions of SPHF are also introduced to achieve password-based authenticated key exchange schemes [9, 18, 23, 25, 29, 32]. Informally, SPHF is defined for an NP language $\mathcal{L}$ over a domain $\mathcal{X}$ that contains two keyed algorithms, namely Hash and $\mathsf{ProjHash}$ that takes as input the hashing key hk and a projection key hp, respectively. The important property of SPHF is as follows: for a word $\chi \in \mathcal{L}$, the outputs of both algorithms are indistinguishable, while for a word $\chi \notin \mathcal{L}$, the outputs of Hash algorithms are statistically indistinguishable with a random element.

In this work, we focused on the stronger type of SPHF, called "word-independent" SPHF defined by Katz and Vaikuntanathan [33, 34]. Compared with general SPHF, the ProjKG algorithm in word-independent SPHF does not require a word as its input. The following formally define the languages and word-independent SPHF.

We first consider a family of languages $(\mathscr{L}_{\mathsf{lpar},\mathsf{ltrap}})_{\mathsf{lpar},\mathsf{ltrap}}$ indexed by some language parameter lpar and some language trapdoor ltrap, together with a family of NP language $(\widetilde{\mathscr{L}}_{\mathsf{lpar}})_{\mathsf{lpar}}$ indexed by some parameter lpar, with witness relation $\widetilde{\mathscr{R}}_{\mathsf{lpar}}$, such that

$$\widetilde{\mathscr{L}}_{\mathsf{lpar}} := \{\chi \in \mathcal{X}_{\mathsf{lpar}} \mid \exists\omega, \widetilde{\mathscr{R}}_{\mathsf{lpar}}(\chi,\omega) = 1\} \subseteq \mathscr{L}_{\mathsf{lpar},\mathsf{ltrap}} \subseteq \mathcal{X}_{\mathsf{lpar}},$$

where $(\mathcal{X}_{\mathsf{lpar}})_{\mathsf{lpar}}$ is a family of sets and the parameter lpar is generated by a polynomial-time algorithm $\mathsf{Setup.lpar}(1^\lambda)$ for some security parameter λ. We suppose that the membership in $\mathcal{X}_{\mathsf{lpar}}$ and $\widetilde{\mathscr{R}}_{\mathsf{lpar}}$ can be checked in polynomial time by the given lpar, and that the membership in $\mathscr{L}_{\mathsf{lpar},\mathsf{ltrap}}$ by the given lpar and ltrap.

Then, let $(\widetilde{\mathscr{L}}_{\mathsf{lpar}} \subseteq \mathscr{L}_{\mathsf{lpar},\mathsf{ltrap}} \subseteq \mathcal{X}_{\mathsf{lpar}})_{\mathsf{lpar},\mathsf{ltrap}}$ be the languages defined as above. An approximate word-independent SPHF scheme SPHF for these languages consists of the following four algorithms:

- $\mathsf{HashKG}(\mathsf{lpar})$: Taking as input a language parameter lpar, this PPT algorithm outputs a hashing key hk.
- $\mathsf{ProjKG}(\mathsf{hk}, \mathsf{lpar})$: Taking as input a hashing key hk and the language parameter lpar, this PPT algorithm outputs a projection key hp.
- $\mathsf{Hash}(\mathsf{hk}, \mathsf{lpar}, \chi)$: Taking as input a hashing key hk, the language parameter lpar, and a word $\chi \in \mathcal{X}_{\mathsf{lpar}}$, this deterministic algorithm outputs a hash value $\mathsf{H} \in \{0,1\}^\delta$ for some $\delta \in \mathbb{N}$.
- $\mathsf{ProjHash}(\mathsf{hp}, \mathsf{lpar}, \chi, \omega)$: Taking as input a projection key hp, the language parameter lpar, a word $\chi \in \widetilde{\mathscr{L}}_{\mathsf{lpar}}$, and a witness ω (*i.e.,* $\widetilde{\mathscr{R}}(\chi,\omega) = 1$), this deterministic algorithm outputs a projected hash value $\mathsf{pH} \in \{0,1\}^\delta$ for some $\delta \in \mathbb{N}$.

An approximate word-independent SPHF scheme has to fulfill the following properties:

- Approximate correctness: For a word $\chi \in \widetilde{\mathscr{L}}_{\mathsf{lpar}}$ and its corresponding witness ω, we say SPHF is ϵ-approximate correctness if $\Pr[\mathsf{HD}(\mathsf{Hash}(\mathsf{hk}, \mathsf{lpar}, \chi), \mathsf{ProjHash}(\mathsf{hp}, \mathsf{lpar}, \chi, \omega)) > \epsilon \cdot \delta] \leq \mathbf{negl}(\lambda)$, where $\mathsf{HD}(\cdot,\cdot)$ outputs the Hamming distance of two input values. In addition, if an approximate SPHF is 0-correct, then it is called SPHF.
- Smoothness: For a word $\chi \notin \widetilde{\mathscr{L}}_{\mathsf{lpar}}$, the hash value H is statistically indistinguishable from a random element chosen from $\{0,1\}^\delta$ for some $\delta \in \mathbb{N}$.

In addition to these two properties, to prove the security of the proposed generic construction, we need another property called pseudo-randomness:

- Pseudo-randomness: For a word $\chi \in \widetilde{\mathscr{L}}_{\mathsf{lpar}}$, the hash value H is indistinguishable from a random element chosen from $\{0,1\}^{\delta}$ for some $\delta \in \mathbb{N}$.

In fact, an (approximate word-independent) SPHF does not need this property or even satisfy it. However, if the language for the (approximate word-independent) SPHF is labelled CCA-secure ciphertext, it is easily satisfied because the ciphertexts are based on hard-on-average problems [35].

3 DEFINITION AND SECURITY MODELS OF PAEKS

Public-key authenticated encryption with keyword search (PAEKS), first introduced by Huang and Li [27], can be viewed as inheriting the existed PEKS scheme [7] but additionally satisfies TP. Next, we review the definition and security requirements of PAEKS defined in [27].

3.1 Definition of PAEKS

A PAEKS scheme PAEKS consists of the following six algorithms:

- $\mathsf{Setup}(1^{\lambda})$: Taking as input a security parameter λ, this PPT algorithm outputs a public parameter pp.
- $\mathsf{KeyGen}_S(\mathsf{pp})$: Taking as input the public parameter pp, this PPT algorithm outputs a pair of public/private keys $(\mathsf{pk}_S, \mathsf{sk}_S)$ of the sender.
- $\mathsf{KeyGen}_R(\mathsf{pp})$: Taking as input the public parameter pp, this PPT algorithm outputs a pair of public/private keys $(\mathsf{pk}_R, \mathsf{sk}_R)$ of the receiver.
- $\mathsf{PAEKS}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{sk}_S, \mathsf{pk}_R, kw)$: Taking as input the public parameter pp, the public key pk_S and private key sk_S of the sender, the public key pk_R of the receiver, and a keyword kw, this PPT algorithm outputs a searchable ciphertext ct_{kw} related to the keyword kw.
- $\mathsf{Trapdoor}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{pk}_R, \mathsf{sk}_R, kw')$: Taking as input the public parameter pp, the public key pk_S of the sender, the public key pk_R and the private key sk_R of the receiver, and a keyword kw', this PPT/deterministic algorithm outputs a trapdoor $\mathsf{td}_{kw'}$ related to the keyword kw'.
- $\mathsf{Test}(\mathsf{pp}, \mathsf{ct}_{kw}, \mathsf{td}_{kw'})$: Taking as input the public parameter pp, searchable ciphertext ct_{kw}, and trapdoor $\mathsf{td}_{kw'}$, this algorithm outputs 1 if ct_{kw} and $\mathsf{td}_{kw'}$ are related to the same keyword (*i.e.,* $kw = kw'$); otherwise, it outputs 0.

Correctness. For any security parameter λ, any honestly generated key pairs of the sender $(\mathsf{pk}_S, \mathsf{sk}_S)$ and receiver $(\mathsf{pk}_R, \mathsf{sk}_R)$, any keywords kw, kw', any ciphertext $\mathsf{ct}_{kw} \leftarrow \mathsf{PAEKS}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{sk}_S, \mathsf{pk}_R, kw)$, and any trapdoor $\mathsf{td}_{kw'} \leftarrow \mathsf{Trapdoor}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{pk}_R, \mathsf{sk}_R, kw')$, then we have

$$\begin{cases} \Pr[\mathsf{Test}(\mathsf{pp}, \mathsf{ct}_{kw}, \mathsf{td}_{kw'}) = 1] = 1 - \mathbf{negl}(\lambda) \text{ when } kw = kw'; \\ \Pr[\mathsf{Test}(\mathsf{pp}, \mathsf{ct}_{kw}, \mathsf{td}_{kw'}) = 0] = 1 - \mathbf{negl}(\lambda) \text{ when } kw \neq kw'. \end{cases}$$

3.2 Security Requirements of PAEKS

A secure PAEKS scheme should satisfy ciphertext indistinguishability (CI) and trapdoor privacy (TP). Informally, the notion of CI, first proposed by Boneh *et al.* [7], aims to ensure that no PPT adversary can obtain any knowledge of the keyword from the ciphertext. While the concept of TP, first introduced by Byun [8] in 2006, aims to ensure that there is no PPT (inside) adversary can obtain any knowledge of the keyword from the trapdoor.

These two requirements are formally modeled by the following IND-CKA game and indistinguishability against IKGA (IND-IKGA) game, respectively, interacted with a challenger C and an adversary $\mathcal{A}$.

IND-CKA Game of PAEKS:

- **Setup.** After receiving a security parameter λ, C generates the public parameter pp by executing the Setup algorithm. Then, it executes the KeyGen_S and KeyGen_R algorithms to obtain the public/private key pairs $(\mathsf{pk}_S, \mathsf{sk}_S)$ and $(\mathsf{pk}_R, \mathsf{sk}_R)$ of the sender and the receiver, respectively. Finally, it sends $(\mathsf{pp}, \mathsf{pk}_S, \mathsf{pk}_R)$ to $\mathcal{A}$.
- **Phase 1.** In this phase, $\mathcal{A}$ is allowed to adaptively issue queries to the following two oracles polynomially many times.
  - **Ciphertext Oracle** O_C: For any keyword kw, C generates a searchable ciphertext ct_{kw} by performing $\mathsf{PAEKS}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{sk}_S, \mathsf{pk}_R, kw)$ and returns ct_{kw} to $\mathcal{A}$.
  - **Trapdoor Oracle** O_T: For any keyword kw, C generates a trapdoor td_{kw} by performing $\mathsf{Trapdoor}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{pk}_R, \mathsf{sk}_R, kw)$ and returns td_{kw} to $\mathcal{A}$.
- **Challenge.** After $\mathcal{A}$ terminates **Phase 1**, it outputs two challenge keywords kw_0^*, kw_1^* to C. The restriction is that $\mathcal{A}$ never issues the queries to O_C and O_T for these two challenge keywords. C then randomly chooses a bit $b \in \{0,1\}$ and returns the challenge ciphertext ct^* to $\mathcal{A}$ by performing $\mathsf{PAEKS}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{sk}_S, \mathsf{pk}_R, kw_b^*)$.
- **Phase 2.** $\mathcal{A}$ can continue to query the oracles as in **Phase 1** for any keyword kw, except for the challenge keywords (*i.e.,* $kw \notin \{kw_0^*, kw_1^*\}$).
- **Guess.** Finally, $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$ as its answer and wins the game if $b = b'$.

The advantage of $\mathcal{A}$ winning the above game is defined as

$$Adv_{\mathcal{A}}^{CI-PAEKS}(\lambda) := \left| \Pr[b = b'] - \frac{1}{2} \right|.$$

Definition 3.1 (Ciphertext Indistinguishability of PAEKS). A PAEKS scheme is called CI (or IND-CKA secure) if, for any PPT adversary $\mathcal{A}$, $Adv_{\mathcal{A}}^{CI-PAEKS}(\lambda)$ is negligible.

Here, we introduce Theorem 3.2 to show that if the PAEKS algorithm of a secure PAEKS scheme is probabilistic, then this scheme satisfies MCI.

Theorem 3.2. *Suppose that a PAEKS scheme satisfies CI and its* PAEKS *algorithm is probabilistic, then the PAEKS scheme satisfies MCI.*

Proof. Suppose that an adversary $\mathcal{A}$ can break the MCI of a PAEKS scheme, then there is a challenger C who can use $\mathcal{A}$ as the black box algorithm to break the CI of the same PAEKS scheme.

- **Setup.** Given a tuple of public information $(\mathsf{pp}, \mathsf{pk}_S, \mathsf{pk}_R)$, C passes this information to $\mathcal{A}$.
- **Phase 1.** On receiving any ciphertext query or trapdoor query for a keyword kw from $\mathcal{A}$, C queries O_C for the ciphertext query and queries O_T for trapdoor query. Then, it returns the answer to $\mathcal{A}$.
- **Challenge.** After receiving two tuples of challenge keywords $(kw^*_{0,1}, \cdots, kw^*_{0,n})$ and $(kw^*_{1,1}, \cdots, kw^*_{1,n})$, C performs the following steps. It randomly chooses a tuple $(kw^*_{0,i}, kw^*_{1,i})$ for some i such that $kw^*_{0,i} \neq kw^*_{1,i}$. Then, it takes this tuple as its challenge keyword and receives a challenge ciphertext ct^*. In addition, it randomly chooses $n-1$ elements $(r_1, \cdots, r_{i-1}, r_{i+1}, \cdots, r_n)$ from the output space of the PAEKS algorithm. Finally, it returns $(r_1, \cdots, r_{i-1}, \mathsf{ct}^*, r_{i+1}, \cdots, r_n)$ as the challenge ciphertext for $\mathcal{A}$.
- **Phase 2.** $\mathcal{A}$ can continue to query the oracles as in **Phase 1** for any keyword kw, except for the challenge keywords (*i.e.,* $kw \neq kw^*_{i,j}$) for $i \in \{0,1\}$ and $j \in \{1,n\}$.
- **Guess.** $\mathcal{A}$ finally outputs a bit b', then C takes $\mathcal{A}$'s answer as its answer.

As ct^* is C's challenge ciphertext and the PAEKS scheme satisfies CI and the PAEKS algorithm is probabilistic, for the view of $\mathcal{A}$, $(r_1, \cdots, r_{i-1}, \mathsf{ct}^*, r_{i+1}, \cdots, r_n)$ is the same as the n truly ciphertext. Therefore, suppose $\mathcal{A}$'s answer is right, then C can use $\mathcal{A}$'s answer to break the CI of the PAEKS scheme. □

IND-IKGA Game of PAEKS:

- **Setup.** Like the **IND-CKA game**, C generates the public parameter pp and public/private key pairs $(\mathsf{pk}_S, \mathsf{sk}_S)$ and $(\mathsf{pk}_R, \mathsf{sk}_R)$ of the sender and the receiver. Then, it sends $(\mathsf{pp}, \mathsf{pk}_S, \mathsf{pk}_R)$ to $\mathcal{A}$.
- **Phase 1.** Like the **IND-CKA game**, $\mathcal{A}$ is allowed to adaptively issue queries to O_C and O_T polynomially many times.
- **Challenge.** After $\mathcal{A}$ terminates **Phase 1**, it outputs two challenge keywords kw^*_0, kw^*_1 to C. The restriction is that $\mathcal{A}$ never issues the queries to O_C and O_T for these two challenge keywords. C then randomly chooses a bit $b \in \{0,1\}$ and returns the challenge trapdoor td^* to $\mathcal{A}$ by performing $\mathsf{Trapdoor}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{pk}_R, \mathsf{sk}_R, kw^*_b)$.
- **Phase 2.** $\mathcal{A}$ can continue to query the oracles as in **Phase 1** for any keyword kw, except for the challenge keywords (*i.e.,* $kw \notin \{kw^*_0, kw^*_1\}$).
- **Guess.** Finally, $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$ as its answer and wins the game if $b = b'$.

The advantage of $\mathcal{A}$ winning the above game is defined as

$$Adv^{TP-PAEKS}_{\mathcal{A}}(\lambda) := \left|\Pr[b = b'] - \frac{1}{2}\right|.$$

Definition 3.3 (Trapdoor Privacy of PAEKS). A PAEKS scheme is called TP (or IND-IKGA secure) if, for any PPT adversary $\mathcal{A}$, $Adv^{TP-PAEKS}_{\mathcal{A}}(\lambda)$ is negligible.

To enhance the security requirements of PAEKS, Qin et al. [49] introduced the notion called multi-ciphertext indistinguishability (MCI). This notion aims to ensure that no PPT adversary can distinguish two tuples of challenge ciphertexts. In addition, Pan and Li [48] considered a concept similar to TP, called multi-trapdoor privacy (MTP), to ensure that no PPT adversary can distinguish two tuples of challenge trapdoors. For the security games of MCI and MTP, please refer to [49] and [48], respectively.

4 CRYPTANALYSIS OF PREVIOUS TRAPDOOR PRIVACY SCHEMES

In this section, we cryptanalyze two lattice-based (variant) PEKS schemes proposed by Zhang *et al.* [62] at Inf. Sci. in 2019 and Zhang *et al.* [63] at IEEE Trans. Dependable Secur. Comput. in 2021, respectively. The core idea of these schemes against IKGA is to restrict the malicious adversary from adaptively generating ciphertexts for any keyword and to further test the trapdoor generated from the receiver. Although these schemes have been proven to satisfy TP (*i.e.,* the schemes are secure against IKGA), the security models in [62] and [63] do not capture the IKGA in a real scenario. More concretely, the adversary is considered to have won the game if and only if the adversary is able to generate a valid searchable ciphertext, rather than just obtain the information about the keyword from the challenge trapdoor. In the following, we directly present our cryptanalysis by two lemmas to show that there exists an adversary that can easily break the TP in polynomial time by randomly choosing keywords because the trapdoor directly leaks the keyword information. Please refer to Appendix A for Zhang *et al.*'s schemes).

Lemma 4.1. *Zhang et al.'s [63] forward-secure PEKS scheme is vulnerable to IKGA.*

Proof. Here, we show how an inside adversary $\mathcal{A}$ can retrieve the keyword information hidden in the trapdoor. Suppose that $\mathcal{A}$ has received a trapdoor $\mathsf{td}_j := \mathbf{t}_{kw\|j}$ related to some time j and keyword kw. It tries to obtain the keyword information via the following steps:

(1) Since $\mathbf{t}_{kw\|j}$ is generated from the receiver by performing $\mathsf{SamplePre}(\mathbf{A}_{R\|j}\boldsymbol{\beta}_j^{-1}, \mathbf{T}_{kw\|j}, \boldsymbol{\mu}, \sigma)$, we know that $\boldsymbol{\mu} = \mathbf{A}_{R\|j}\boldsymbol{\beta}_j^{-1} \cdot \mathbf{t}_{kw\|j} = \mathbf{A}_{R\|j} \cdot H_2(kw\|j)^{-1} \cdot \mathbf{t}_{kw\|j}$.
(2) Then, $\mathcal{A}$ randomly selects a guessed keyword kw' to test whether $\boldsymbol{\mu} \stackrel{?}{=} \mathbf{A}_{R\|j} \cdot H_2(kw'\|j)^{-1} \cdot \mathbf{t}_{kw\|j}$.
(3) If the equation in Step 2 holds, $\mathcal{A}$ outputs kw' as its guess; otherwise, $\mathcal{A}$ returns to Step 2 and continues to select and test other keywords.

Therefore, as the keyword space is limited, there is a high probability that $\mathcal{A}$ can obtain the keyword related to the trapdoor by the brute force attack. □

Lemma 4.2. *Zhang et al.'s [62] proxy-oriented identity-based PEKS scheme is vulnerable to IKGA.*

Proof. Let id_P and id_R be two identities of the proxy and the receiver, respectively. Here, we show that how an inside adversary $\mathcal{A}$ can retrieve the keyword information hidden in the trapdoor. Suppose that $\mathcal{A}$ has received a trapdoor $\mathsf{td} := \mathbf{d}_{kw}$ related to some keyword kw. It tries to obtain the keyword information via the following steps:

(1) Since $\mathbf{d}_{kw}$ is generated from the receiver by performing SamplePre$(\mathbf{A}_{id_R}\boldsymbol{\gamma}^{-1}, \mathbf{D}_{kw}, \mathbf{v}, \delta)$, we know that $\mathbf{v} = \mathbf{A}_{id_R}\boldsymbol{\gamma}^{-1}\mathbf{d}_{kw}$, where $\boldsymbol{\gamma} \leftarrow H_4(id_P \| id_R \| kw)$.
(2) Then, $\mathcal{A}$ randomly selects a guessed keyword kw' and computes $\boldsymbol{\gamma}' \leftarrow H_4(id_P \| id_R \| kw')$.
(3) $\mathcal{A}$ tests whether $\mathbf{v} \stackrel{?}{=} \mathbf{A}_{id_R}\boldsymbol{\gamma}'^{-1}\mathbf{d}_{kw}$.
(4) If the equation in Step 3 holds, $\mathcal{A}$ outputs kw' as its guess; otherwise, $\mathcal{A}$ returns to Step 2 and continues to select and test other keywords.

Therefore, similarly, there is a high probability that $\mathcal{A}$ can obtain the keyword related to the trapdoor by a brute force attack. □

5 PROPOSED GENERIC PAEKS CONSTRUCTION

In this section, we propose a generic PAEKS construction based on a PEKS and SPHF with details as follows:

- A PEKS scheme PEKS = (KeyGen, PEKS, Trapdoor, Test) with keyword space $\mathcal{KS}_{\mathsf{PEKS}}$ that satisfies CI.
- An ϵ-approximate, word-independent, and pseudo-random SPHF scheme SPHF = (HashKG, ProjKG, Hash, ProjHash) with the output length $\{0,1\}^c$ for the language of the ciphertext of a CCA2-secure labelled PKE scheme PKE = (KeyGen, Encrypt, Decrypt) with public key space $\mathcal{PKS}_{\mathsf{PKE}}$ and plaintext space $\mathcal{PS}_{\mathsf{PKE}}$, where ϵ is negligible.

Language of Ciphertext. Let $(\mathsf{lpar}, \mathsf{ltrap}) = (\mathsf{ek}_{\mathsf{PKE}}, \mathsf{dk}_{\mathsf{PKE}})$, where $\mathsf{ek}_{\mathsf{PKE}} \in \mathcal{PKS}_{\mathsf{PKE}}$ and $\mathsf{dk}_{\mathsf{PKE}}$ is its corresponding decryption key. We define the language of ciphertext as: $\widetilde{\mathscr{L}} := \{(\mathsf{label}, \mathsf{ct}_{\mathsf{PKE}}, \mathsf{m}_{\mathsf{PKE}}) \mid \exists \rho, \mathsf{ct}_{\mathsf{PKE}} \leftarrow \mathsf{Encrypt}(\mathsf{ek}_{\mathsf{PKE}}, \mathsf{label}, \mathsf{m}_{\mathsf{PKE}}; \rho)\}$ and $\mathscr{L} := \{(\mathsf{label}, \mathsf{ct}_{\mathsf{PKE}}, \mathsf{m}_{\mathsf{PKE}}) \mid \mathsf{Decrypt}(\mathsf{dk}_{\mathsf{PKE}}, \mathsf{label}, \mathsf{ct}_{\mathsf{PKE}}) = \mathsf{m}_{\mathsf{PKE}}\}$, where the witness relation $\widetilde{\mathscr{K}}$ is implicit defined as: $\widetilde{\mathscr{K}}((\mathsf{label}, \mathsf{ct}_{\mathsf{PKE}}, \mathsf{m}_{\mathsf{PKE}}), \rho) = 1$ if and only if $\mathsf{ct}_{\mathsf{PKE}} \leftarrow \mathsf{Encrypt}(\mathsf{ek}_{\mathsf{PKE}}, \mathsf{label}, \mathsf{m}_{\mathsf{PKE}}; \rho)$.

Construction. The whole construction is described as follows:

- Setup(1^λ): Given a security parameter λ, this algorithm runs the following steps:
  - Generates $(\mathsf{ek}_{\mathsf{PKE}}, \mathsf{dk}_{\mathsf{PKE}}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$.
  - Randomly chooses a plaintext $\mathsf{m}_{\mathsf{PKE}} \xleftarrow{\$} \mathcal{PS}_{\mathsf{PKE}}$ and a label $\mathsf{label} \xleftarrow{\$} \{0,1\}^*$.
  - Chooses two functions $H_1 : \mathcal{PKS}_{\mathsf{PKE}} \times \mathcal{PS}_{\mathsf{PKE}} \times \{0,1\}^* \to \mathcal{PKS}_{\mathsf{PKE}}$ and $H_2 : \mathcal{KS}_{\mathsf{PEKS}} \times \{0,1\}^* \to \mathcal{KS}_{\mathsf{PEKS}}$, which are modeled as random oracles.
  - Outputs the public parameter $\mathsf{pp} := (\lambda, \mathsf{mpk}, \mathsf{ek}_{\mathsf{PKE}}, \mathsf{m}_{\mathsf{PKE}}, \mathsf{label}, H_1, H_2)$.
- KeyGen$_S$(pp): Given the public parameter pp, this algorithm runs the following steps:
  - Checks whether $\mathsf{mpk} \stackrel{?}{=} H_1(\mathsf{ek}_{\mathsf{PKE}}, \mathsf{m}_{\mathsf{PKE}}, \mathsf{label})$. If the equation is not satisfied, it terminates.
  - Computes $\mathsf{hk}_S \leftarrow \mathsf{SPHF.HashKG}(\mathsf{mpk})$.
  - Computes $\mathsf{hp}_S \leftarrow \mathsf{SPHF.ProjKG}(\mathsf{hk}_S, \mathsf{mpk})$.
  - Generates $\mathsf{ct}_{\mathsf{PKE},S} \leftarrow \mathsf{PKE.Encrypt}(\mathsf{mpk}, \mathsf{label}, \mathsf{m}_{\mathsf{PKE}}; \rho_S)$, where ρ_S is the witness randomly selected such that $\widetilde{\mathscr{K}}((\mathsf{label}, \mathsf{ct}_{\mathsf{PKE},S}, \mathsf{m}_{\mathsf{PKE}}), \rho_S) = 1$ is satisfied.
  - Outputs the public key $\mathsf{pk}_S := (\mathsf{hp}_S, \mathsf{ct}_{\mathsf{PKE},S})$ and private key $\mathsf{sk}_S := (\mathsf{hk}_S, \rho_S)$ of the sender.
- KeyGen$_R$(pp): Given the public parameter pp, this algorithm runs the following steps:
  - Checks whether $\mathsf{mpk} \stackrel{?}{=} H_1(\mathsf{ek}_{\mathsf{PKE}}, \mathsf{m}_{\mathsf{PKE}}, \mathsf{label})$. If the equation is not satisfied, it terminates.
  - Computes $\mathsf{hk}_R \leftarrow \mathsf{SPHF.HashKG}(\mathsf{mpk})$.
  - Computes $\mathsf{hp}_R \leftarrow \mathsf{SPHF.ProjKG}(\mathsf{hk}_R, \mathsf{mpk})$.
  - Generates $\mathsf{ct}_{\mathsf{PKE},R} \leftarrow \mathsf{PKE.Encrypt}(\mathsf{mpk}, \mathsf{label}, \mathsf{m}_{\mathsf{PKE}}; \rho_R)$, where ρ_R is the witness randomly selected such that $\widetilde{\mathscr{K}}((\mathsf{label}, \mathsf{ct}_{\mathsf{PKE},R}, \mathsf{m}_{\mathsf{PKE}}), \rho_R) = 1$ is satisfied.
  - Generates $(\mathsf{pk}_{\mathsf{PEKS}}, \mathsf{sk}_{\mathsf{PEKS}}) \leftarrow \mathsf{PEKS.KeyGen}(1^\lambda)$.
  - Outputs the public key $\mathsf{pk}_R := (\mathsf{hp}_R, \mathsf{ct}_{\mathsf{PKE},R}, \mathsf{pk}_{\mathsf{PEKS}})$ and private key $\mathsf{sk}_R := (\mathsf{hk}_R, \rho_R, \mathsf{sk}_{\mathsf{PEKS}})$ of the receiver.
- PAEKS$(\mathsf{pp}, \mathsf{pk}_S, \mathsf{sk}_S, \mathsf{pk}_R, kw)$: Given the public parameter pp, the public key pk_S and the private key sk_S of the sender, the public key pk_R of the receiver, and a keyword $kw' \in \mathcal{KS}_{\mathsf{PEKS}}$, this algorithm runs the following steps:
  - Computes $\mathsf{H}_S \leftarrow \mathsf{SPHF.Hash}(\mathsf{hk}_S, \mathsf{mpk}, (\mathsf{ct}_{\mathsf{PKE},R}, \mathsf{m}_{\mathsf{PKE}}))$ and $\mathsf{pH}_S \leftarrow \mathsf{SPHF.ProjHash}(\mathsf{hp}_R, \mathsf{mpk}, (\mathsf{ct}_{\mathsf{PKE},S}, \mathsf{m}_{\mathsf{PKE}}), \rho_S)$.
  - Computes $\mathsf{der\text{-}kw}_S \leftarrow H_2(kw, \mathsf{H}_S \oplus \mathsf{pH}_S)$.
  - Generates $\mathsf{ct}_{\mathsf{PEKS},\mathsf{der\text{-}kw}_S} \leftarrow \mathsf{PEKS.PEKS}(\mathsf{pk}_{\mathsf{PEKS}}, \mathsf{der\text{-}kw}_S)$.
  - Outputs a searchable ciphertext $\mathsf{ct}_{kw} := \mathsf{ct}_{\mathsf{PEKS},\mathsf{der\text{-}kw}_S}$.
- Trapdoor$(\mathsf{pp}, \mathsf{pk}_S, \mathsf{pk}_R, \mathsf{sk}_R, kw')$: Given the public parameter pp, the public key pk_S of the sender, the public key pk_R and private key sk_R of the receiver, and a keyword $kw' \in \mathcal{KS}_{\mathsf{PEKS}}$, this algorithm runs the following steps:
  - Computes $\mathsf{H}_R \leftarrow \mathsf{SPHF.Hash}(\mathsf{hk}_R, \mathsf{mpk}, (\mathsf{ct}_{\mathsf{PKE},S}, \mathsf{m}_{\mathsf{PKE}}))$ and $\mathsf{pH}_R \leftarrow \mathsf{SPHF.ProjHash}(\mathsf{hp}_S, \mathsf{mpk}, (\mathsf{ct}_{\mathsf{PKE},R}, \mathsf{m}_{\mathsf{PKE}}), \rho_R)$.
  - Computes $\mathsf{der\text{-}kw}'_R \leftarrow H_2(kw', \mathsf{H}_R \oplus \mathsf{pH}_R)$.
  - Generates $\mathsf{td}_{\mathsf{PEKS},\mathsf{der\text{-}kw}'_R} \leftarrow \mathsf{PEKS.Trapdoor}(\mathsf{sk}_{\mathsf{PEKS}}, \mathsf{der\text{-}kw}'_R)$.
  - Outputs a trapdoor $\mathsf{td}_{kw'} := \mathsf{td}_{\mathsf{PEKS},\mathsf{der\text{-}kw}'_R}$.
- Test$(\mathsf{pp}, \mathsf{ct}_{kw}, \mathsf{td}_{kw'})$: Given the public parameter pp, the searchable ciphertext ct_{kw}, and the trapdoor $\mathsf{td}_{kw'}$, this algorithm outputs the result of $\mathsf{PEKS.Test}(\mathsf{ct}_{kw}, \mathsf{td}_{kw'})$.

Correctness. Suppose that the public parameter pp and the public/private key pairs $(\mathsf{pk}_S, \mathsf{sk}_S)$, $(\mathsf{pk}_R, \mathsf{sk}_R)$ are honestly generated. Let ct_{kw} be the searchable ciphertext related with the keyword kw generated by the sender, and $\mathsf{td}_{kw'}$ be the trapdoor related with the keyword kw' generated by the receiver.

As the underlying SPHF is ϵ-correct for some $\epsilon = \mathbf{negl}(\lambda)$, it follows that

$$
\begin{aligned}
\mathsf{H}_S &= \mathsf{SPHF.Hash}(\mathsf{hk}_S, \mathsf{mpk}, (\mathsf{ct}_{\mathsf{PKE},R}, \mathsf{m}_{\mathsf{PKE}})) \\
&= \mathsf{SPHF.ProjHash}(\mathsf{hp}_S, \mathsf{mpk}, (\mathsf{ct}_{\mathsf{PKE},R}, \mathsf{m}_{\mathsf{PKE}}), \rho_R) \\
&= \mathsf{pH}_R; \\
\mathsf{H}_R &= \mathsf{SPHF.Hash}(\mathsf{hk}_R, \mathsf{mpk}, (\mathsf{ct}_{\mathsf{PKE},S}, \mathsf{m}_{\mathsf{PKE}})) \\
&= \mathsf{SPHF.ProjHash}(\mathsf{hp}_R, \mathsf{mpk}, (\mathsf{ct}_{\mathsf{PKE},S}, \mathsf{m}_{\mathsf{PKE}}), \rho_S) \\
&= \mathsf{pH}_S.
\end{aligned}
$$

Therefore, $\mathsf{H}_S \oplus \mathsf{pH}_S = \mathsf{H}_R \oplus \mathsf{pH}_R$ holds. Clearly, if $kw = kw'$, then $\mathsf{der\text{-}kw}_S = H_2(kw, \mathsf{H}_S \oplus \mathsf{pH}_S) = H_2(kw', \mathsf{H}_R \oplus \mathsf{pH}_R)$, and therefore, $\mathsf{ct}_{\mathsf{PEKS,der\text{-}kw}_S}$ and $\mathsf{td}_{\mathsf{PEKS,der\text{-}kw}'_R}$ are related to the same extended keyword. As the underlying PEKS scheme is correct, $\mathsf{PAEKS.Test}(\mathsf{pp}, \mathsf{ct}_{kw}, \mathsf{td}_{kw'}) = 1$ holds with overwhelming probability. In contrast, since H_2 is modeled as a random oracle, if $kw \neq kw'$, then $\mathsf{der\text{-}kw}_S = H_2(kw, \mathsf{H}_S \oplus \mathsf{pH}_S) \neq H_2(kw', \mathsf{H}_R \oplus \mathsf{pH}_R) = \mathsf{der\text{-}kw}'_R$, and therefore, $\mathsf{ct}_{\mathsf{PEKS,der\text{-}kw}_S}$ and $\mathsf{td}_{\mathsf{PEKS,der\text{-}kw}'_R}$ are related to different extended keywords. Consequently, $\mathsf{PAEKS.Test}(\mathsf{pp}, \mathsf{ct}_{kw}, \mathsf{td}_{kw'}) = 0$ holds with overwhelming probability.

Security Analysis. Below, Theorem 5.1 and Theorem 5.2 indicate that the proposed construction satisfies CI and TP, respectively, by adopting the sequence-of-games strategy. More concretely, we construct a sequence of games: the first game is identical to the real attack game and $\mathcal{A}$ can only distinguish these games with a negligible advantage. For simplicity, let $Adv_{\mathcal{A}}^{\mathbf{Game}_i}(\lambda)$ denote the advantage of $\mathcal{A}$ in game $\mathbf{Game}_i$, where $i \in \{0, \cdots, 3\}$. Furthermore, by Theorem 5.3, we also show that the proposed construction satisfies MCI.

Theorem 5.1. *The proposed generic PAEKS construction satisfies CI if the underlying SPHF scheme satisfies pseudo-randomness and H_2 is modeled as random oracle.*

Proof. This proof consists of four games, illustrated as follows:

$\mathbf{Game}_0$: This game is identical to the real IND-CKA game defined in Section 3.2. Suppose that the advantage of $\mathcal{A}$ in this game is defined as $Adv_{\mathcal{A}}^{\mathbf{Game}_0}(\lambda) := \epsilon$. In addition, to simulate a real view for $\mathcal{A}$, on receiving the query for some keyword kw from $\mathcal{A}$, the challenger C responds as follows:

- O_C: For keyword kw, C computes $\mathsf{ct}_{kw} \leftarrow \mathsf{PAEKS}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{sk}_S, \mathsf{pk}_R, kw)$ and returns ct_{kw} to $\mathcal{A}$.
- O_T: For keyword kw, C computes $\mathsf{td}_{kw} \leftarrow \mathsf{Trapdoor}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{pk}_R, \mathsf{sk}_R, kw)$ and returns td_{kw} to $\mathcal{A}$.

$\mathbf{Game}_1$: This game is identical to $\mathbf{Game}_0$, except for the generation of the challenge ciphertext ct^* in the **Challenge** phase. More concretely, instead of generating $\mathsf{H}_S \leftarrow \mathsf{SPHF.Hash}(\mathsf{hk}_S, \mathsf{mpk}, (\mathsf{ct}_{\mathsf{PKE},R}, \mathsf{m}_{\mathsf{PKE}}))$, C randomly chooses H_S from the output space of the SPHF.Hash algorithm. Since the underlying SPHF scheme satisfies pseudo-randomness, $\mathcal{A}$ cannot distinguish the view between $\mathbf{Game}_0$ and $\mathbf{Game}_1$. Therefore, we obtain

$$\left|Adv_{\mathcal{A}}^{\mathbf{Game}_1}(\lambda) - Adv_{\mathcal{A}}^{\mathbf{Game}_0}(\lambda)\right| \leq \mathbf{negl}(\lambda).$$

$\mathbf{Game}_2$: This game further changes the generation of the challenge ciphertext ct^* in the **Challenge** phase. In this game, $\mathsf{der\text{-}kw}_S$ is randomly chosen from $\mathcal{KS}_{\mathsf{PEKS}}$, instead of by computing $\mathsf{der\text{-}kw}_S \leftarrow H_2(kw_b^*, \mathsf{H}_S \oplus \mathsf{pH}_S)$ for some $b \in \{0, 1\}$. As H_S is randomly chosen and H_2 is modeled as random oracle, the output of $H_2(kw_b^*, \mathsf{H}_S \oplus \mathsf{pH}_S)$ is random. Therefore, $\mathcal{A}$ cannot distinguish the view between $\mathbf{Game}_1$ and $\mathbf{Game}_2$. Consequently, we obtain

$$\left|Adv_{\mathcal{A}}^{\mathbf{Game}_2}(\lambda) - Adv_{\mathcal{A}}^{\mathbf{Game}_1}(\lambda)\right| \leq \mathbf{negl}(\lambda).$$

$\mathbf{Game}_3$: This game is the last game. Because the challenge ciphertext $\mathsf{ct}^* = \mathsf{ct}_{\mathsf{PEKS,der\text{-}kw}_S}$ is generated from $\mathsf{PEKS.PEKS}(\mathsf{pk}_{\mathsf{PEKS}}, \mathsf{der\text{-}kw}_S)$ and $\mathsf{der\text{-}kw}_S$ is now randomly chosen from $\mathcal{KS}_{\mathsf{PEKS}}$, the challenge ciphertext does not contain any information about the challenge keywords (kw_0^*, kw_1^*) given by $\mathcal{A}$. The only way for $\mathcal{A}$ is to guess. Therefore, we have

$$Adv_{\mathcal{A}}^{\mathbf{Game}_3}(\lambda) = 0.$$

Finally, combining the above games, we have $\epsilon \leq \mathbf{negl}(\lambda)$. The proof is completed. □

Theorem 5.2. *The proposed generic PAEKS construction satisfies TP if the underlying SPHF scheme satisfies pseudo-randomness and H_2 is modeled as random oracle.*

Proof. This proof is similar to the proof of Theorem 5.1, again with four games.

$\mathbf{Game}_0$: This game is identical to the real IND-IKGA game defined in Section 3.2. Suppose that the advantage of $\mathcal{A}$ in this game is defined as $Adv_{\mathcal{A}}^{\mathbf{Game}_0}(\lambda) := \epsilon$. In addition, the view simulated by the challenger C is the same as that in $\mathbf{Game}_0$ in the proof of Theorem 5.1.

$\mathbf{Game}_1$: This game is identical to $\mathbf{Game}_0$, except for the generation of the challenge trapdoor td^* in the **Challenge** phase. More concretely, instead of generating $\mathsf{H}_R \leftarrow \mathsf{SPHF.Hash}(\mathsf{hk}_R, \mathsf{mpk}, (\mathsf{ct}_{\mathsf{PKE},S}, \mathsf{m}_{\mathsf{PKE}}))$, C randomly chooses H_R from the output space of the SPHF.Hash algorithm. Since the underlying SPHF scheme satisfies pseudo-randomness, $\mathcal{A}$ cannot distinguish the view between $\mathbf{Game}_0$ and $\mathbf{Game}_1$. Therefore, we obtain

$$\left|Adv_{\mathcal{A}}^{\mathbf{Game}_1}(\lambda) - Adv_{\mathcal{A}}^{\mathbf{Game}_0}(\lambda)\right| \leq \mathbf{negl}(\lambda).$$

$\mathbf{Game}_2$: This game further changes the generation of the challenge trapdoor td^* in the **Challenge** phase. In this game, $\mathsf{der\text{-}kw}_R$ is randomly chosen from $\mathcal{KS}_{\mathsf{PEKS}}$, instead of by computing $\mathsf{der\text{-}kw}_R \leftarrow H_2(kw_b^*, \mathsf{H}_R \oplus \mathsf{pH}_R)$ for some $b \in \{0, 1\}$. As H_R is randomly chosen, the output of $H_2(kw_b^*, \mathsf{H}_R \oplus \mathsf{pH}_R)$ is random. Therefore, $\mathcal{A}$ cannot distinguish the view between $\mathbf{Game}_1$ and $\mathbf{Game}_2$. Consequently, we obtain

$$\left|Adv_{\mathcal{A}}^{\mathbf{Game}_2}(\lambda) - Adv_{\mathcal{A}}^{\mathbf{Game}_1}(\lambda)\right| \leq \mathbf{negl}(\lambda).$$

$\mathbf{Game}_3$: This game is the last game. Because the challenge trapdoor $\mathsf{td}^* = \mathsf{td}_{\mathsf{PEKS,der\text{-}kw}_R}$ is generated from $\mathsf{PEKS.PEKS}(\mathsf{pk}_{\mathsf{PEKS}}, \mathsf{der\text{-}kw}_R)$ and $\mathsf{der\text{-}kw}_R$ is now randomly chosen from $\mathcal{KS}_{\mathsf{PEKS}}$, the challenge trapdoor does not contain any information about the challenge keywords (kw_0^*, kw_1^*) given by $\mathcal{A}$. The only way for $\mathcal{A}$ is to guess. Therefore, we have

$$Adv_{\mathcal{A}}^{\mathbf{Game}_3}(\lambda) = 0.$$

Finally, combining the above games, we have $\epsilon \leq \mathbf{negl}(\lambda)$. The proof is now complete. □

Theorem 5.3. *The proposed generic PAEKS construction further satisfies MCI if Theorem 3.2 and Theorem 5.1 holds, and* PEKS *algorithm of the underlying PEKS scheme is probabilistic.*

Proof. In the proposed construction, the PAEKS algorithm actually perform the PEKS algorithm of the underlying PEKS scheme. To the best of our knowledge, for the current well-known PEKS schemes (*e.g.*, [6, 7, 28]), the PEKS is probabilistic. Hence, by combining the result of Theorem 5.1 and Theorem 3.2, the proposed construction satisfies MCI. ◻

6 LATTICE-BASED INSTANTIATION

In this section, we propose the first quantum-resistant PAEKS instantiation based on lattices. This instantiation leverages three lattice-based primitives as the building blocks and inherits their securities to be secure against quantum attacks. More concretely, we adopt the word-independent SPHF scheme introduced by Li and Wang [37] based on the labelled IND-CCA1 PKE scheme introduced by Micciancio and Peikert [43], and the PEKS scheme introduce by Behnia *et al.* [6]. Note that, since labelled IND-CCA1 PKE can be transferred to IND-CCA2 PKE by combining a one-time signature scheme; for simplicity, we only describe the weaker version (IND-CCA1) of the PKE scheme [43] in the following instantiation.

Before introducing our instantiation, we define some important notations. Let $\mathcal{R}$ be a ring and $\mathcal{U}$ be a subset of $\mathcal{R}^\times$ of invertible elements. In addition, let $\mathbf{G} := \mathbf{I}_n \otimes \mathbf{g}^\top$ be the gadget matrix defined in [43], where $\mathbf{g}^\top := [1, 2, \cdots, 2^k]$ and $k := \lceil \log q \rceil - 1$. Finally, we also define the encoding function $\mathsf{Encode}(\mu \in \{0,1\}) := \mu \cdot (0, \cdots, 0, \lceil q/2 \rceil)^\top$ and the deterministic rounding function $\mathsf{R}(x) := \lfloor 2x/q \rceil \bmod 2$. Finally, the notations $[\mathbf{A}|\mathbf{B}]$ and $[\mathbf{A};\mathbf{B}] = [\mathbf{A}^\top|\mathbf{B}^\top]^\top$ denote the horizontal concatenation and vertical concatenation of matrices $\mathbf{A}$ and $\mathbf{B}$, respectively.

The whole instantiation is described as follows:

- $\mathsf{Setup}(1^\lambda)$: Given a security parameter λ and the parameters $q, n, m, \sigma_1, \sigma_2, \alpha$ (set as instructed in the following parameter selection part), this algorithm runs the following steps:
  - Sets $\kappa, \rho, \ell \leftarrow \mathsf{poly}(n)$ and randomly chooses $\mathbf{m} = m_1 m_2 \cdots m_\kappa \xleftarrow{\$} \{0,1\}^\kappa$.
  - Computes $(\mathbf{A}_0, \mathbf{T}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)$.
  - Sets $\mathsf{ek}_{\mathsf{PKE}} := \mathbf{A}_0$, $\mathsf{dk}_{\mathsf{PKE}} := \mathbf{T}$, and $\mathsf{m}_{\mathsf{PKE}} := \mathbf{m}$.
  - Randomly chooses element $u \xleftarrow{\$} \mathcal{U}$ and sets label $:= u$
  - Choose two secure hash functions $H_1 : \mathbb{Z}_q^{n\times m} \times \{0,1\}^\kappa \times \mathcal{U} \to \mathbb{Z}_q^{n\times m}$, $H_2 : \{1,-1\}^\ell \times \{0,1\}^\kappa \to \{1,-1\}^\ell$, and an injective ring homomorphism $h : \mathcal{R} \to \mathbb{Z}_q^{n\times n}$.
  - Computes $\mathbf{A} \leftarrow H_1(\mathbf{A}_0, \mathbf{m}, u) \in \mathbb{Z}_q^{n\times m}$ and sets $\mathsf{mpk} := \mathbf{A}$.
  - Outputs $\mathsf{pp} := (\lambda, n, m, q, \sigma_1, \sigma_2, \kappa, \rho, \ell, \mathsf{ek}_{\mathsf{PKE}} := \mathbf{A}_0, \mathsf{mpk} := \mathbf{A}, \mathsf{m}_{\mathsf{PKE}} := \mathbf{m}, \mathsf{label} := u, H_1, H_2, h)$.
- $\mathsf{KeyGen}_S(\mathsf{pp})$: Given the public parameter pp, this algorithm runs the following steps:
  - Checks whether $\mathbf{A} \stackrel{?}{=} H_1(\mathbf{A}_0, \mathbf{m}, u)$.
  - Computes $\mathbf{A}_u = \mathbf{A} + [\mathbf{0}; \mathbf{G}h(u)]$, randomly chooses a matrix $\mathsf{hk}_S := \mathbf{k}_S \xleftarrow{\$} D^m_{\mathbb{Z},s}$, and computes $\mathsf{hp}_S := \mathbf{p}_S = \mathbf{A}_u^\top \cdot \mathbf{k}_S \in \mathbb{Z}_q^n$, where $s \geq \eta_\epsilon(\Lambda^\perp(\mathbf{A}_u))$ for some $\epsilon = \mathbf{negl}(n)$.
  - For $i = 1, \cdots, \kappa$, randomly chooses vectors $\mathbf{s}_{S,i} \xleftarrow{\$} \mathbb{Z}_q^n$ as well as $\mathbf{e}_{S,i} \xleftarrow{\$} D^m_{\mathbb{Z},t}$ (re-select $\mathbf{e}_{S,i}$ if $\|\mathbf{e}_{S,i}\| > 2t\sqrt{m}$), and computes $\mathbf{c}_{S,i} = \mathbf{A}_u^\top \cdot \mathbf{s}_{S,i} + \mathbf{e}_{S,i} + \mathsf{Encode}(m_i) \bmod q$, where $t = \sigma_1\sqrt{m} \cdot \omega(\sqrt{\log n}$.
  - Outputs the public key $\mathsf{pk}_S := (\mathsf{hp}_S := \mathbf{p}_S, \mathsf{ct}_{\mathsf{PKE},S} := \{\mathbf{c}_{S,i}\}_{i=1}^\kappa)$ and the private key $\mathsf{sk}_S := (\mathsf{hk}_S := \mathbf{k}_S, \rho_S := \{\mathbf{s}_{S,i}\}_{i=1}^\kappa)$ of the sender.
- $\mathsf{KeyGen}_R(\mathsf{pp})$: Given the public parameter pp, this algorithm runs the following steps:
  - Checks whether $\mathbf{A} \stackrel{?}{=} H_1(\mathbf{A}_0, \mathbf{m}, u)$.
  - Computes $\mathbf{A}_u = \mathbf{A} + [\mathbf{0}; \mathbf{G}h(u)]$, randomly chooses a matrix $\mathsf{hk}_R := \mathbf{k}_R \leftarrow D^m_{\mathbb{Z},s}$, and computes $\mathsf{hp}_R := \mathbf{p}_R = \mathbf{A}_u^\top \cdot \mathbf{k}_R \in \mathbb{Z}_q^n$, where $s \geq \eta_\epsilon(\Lambda^\perp(\mathbf{A}_u))$ for some $\epsilon = \mathbf{negl}(n)$.
  - For $i = 1, \cdots, \kappa$, randomly chooses vectors $\mathbf{s}_{R,i} \xleftarrow{\$} \mathbb{Z}_q^n$ as well as $\mathbf{e}_{R,i} \leftarrow D^m_{\mathbb{Z},t}$ (re-select $\mathbf{e}_{R,i}$ if $\|\mathbf{e}_{R,i}\| > 2t\sqrt{m}$), and computes $\mathbf{c}_{R,i} = \mathbf{A}_u^\top \cdot \mathbf{s}_{R,i} + \mathbf{e}_{R,i} + \mathsf{Encode}(m_i) \bmod q$, where $t = \sigma_1\sqrt{m} \cdot \omega(\sqrt{\log n}$.
  - Generates $(\mathbf{B}_R, \mathbf{S}_R) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)$.
  - Selects $\ell + 1$ random matrices $\mathbf{B}_{R,1}, \cdots, \mathbf{B}_{R,\ell}, \mathbf{C}_R \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$ and a random vector $\mathbf{r}_R \xleftarrow{\$} \mathbb{Z}_q^n$.
  - Outputs the public key $\mathsf{pk}_R := (\mathsf{hp}_S := \mathbf{p}_R, \mathsf{ct}_{\mathsf{PKE},R} := \{\mathbf{c}_{R,i}\}_{i=1}^\kappa, \mathsf{pk}_{\mathsf{PEKS}} := \{\mathbf{B}_R, \{\mathbf{B}_{R,i}\}_{i=1}^\ell, \mathbf{C}_R, \mathbf{r}_R\})$ and the private key $\mathsf{sk}_R := (\mathsf{hk}_R := \mathbf{k}_R, \rho_R := \{\mathbf{s}_{R,i}\}_{i=1}^\kappa, \mathsf{sk}_{\mathsf{PEKS}} := \mathbf{S}_R)$ of the receiver.
- $\mathsf{PAEKS}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{sk}_S, \mathsf{pk}_R, kw)$: Given the public parameter pp, the public key pk_S and the private key sk_S of the sender, the public key pk_R of the receiver, and a keyword $kw \in \{1,-1\}^\ell$, this algorithm run as follows.
  - For $i = 1, \cdots, \kappa$, computes $h_{S,i} \leftarrow \mathsf{R}(\mathbf{c}_{R,i}^\top \cdot \mathbf{k}_S \pmod q)$, $p_{S,i} \leftarrow \mathsf{R}(\mathbf{s}_{S,i}^\top \cdot \mathbf{p}_R \pmod q)$ and $y_{S,i} = h_{S,i} \cdot p_{S,i}$.
  - Sets $\mathbf{y}_S = y_{S,1} y_{S,2} \cdots y_{S,\kappa} \in \{0,1\}^\kappa$.
  - Computes $\mathsf{der\text{-}kw}_S := \mathbf{dk}_S = dk_{S,1} dk_{S,2} \cdots dk_{S,\ell} \leftarrow H_2(kw, \mathbf{y}_S) \in \{1,-1\}^\ell$.
  - Computes $\mathbf{B}_{dk} = \mathbf{C}_R + \sum_{i=1}^\ell dk_{S,i}\mathbf{B}_{R,i}$ and $\mathbf{F}_{dk} = [\mathbf{B}_R|\mathbf{B}_{dk}] \in \mathbb{Z}_q^{n\times 2m}$.
  - For $j = 1, \cdots, \rho$, performs the following steps:
    - Chooses $b_j \xleftarrow{\$} \{0,1\}$, a random $\mathbf{s}_j \xleftarrow{\$} \mathbb{Z}_q^n$, and matrices $\mathbf{R}_{i_j} \xleftarrow{\$} \{1,-1\}^{m\times m}$ for $i = 1, \cdots, \ell$.
    - Sets $\bar{\mathbf{R}}_j = \sum_{i=1}^\ell dk_{S,i}\mathbf{R}_{i_j} \in \{-\ell, \cdots, \ell\}^{m\times m}$.
    - Chooses noise vectors $x_j \xleftarrow{\bar{\Psi}_\alpha} \mathbb{Z}_q$ and $\mathbf{y}_j \xleftarrow{\bar{\Psi}_\alpha^m} \mathbb{Z}_q^m$.
    - Sets $\mathbf{z}_j \leftarrow \bar{\mathbf{R}}_j^\top \mathbf{y}_j \in \mathbb{Z}_q^m$, $c_{0_j} = \mathbf{r}_R^\top \mathbf{s}_j + x_j + b_j\lfloor q/2 \rfloor \in \mathbb{Z}_q$, and $\mathbf{c}_{1_j} = \mathbf{F}_{dk}^\top \mathbf{s}_j + [\mathbf{y}_j; \mathbf{z}_j] \in \mathbb{Z}_q^{2m}$.
  - Outputs a searchable ciphertext $\mathsf{ct}_{kw} := (\mathsf{ct}_{\mathsf{PEKS},\mathsf{der\text{-}kw}_S} := \{c_{0_j}, \mathbf{c}_{1_j}, b_j\}_{j=1}^\rho)$.
- $\mathsf{Trapdoor}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{pk}_R, \mathsf{sk}_R, kw')$: Given the public parameter pp, the public key pk_S of the sender, the public key pk_R and private key sk_R of the receiver, and a keyword $kw' \in \{1,-1\}^\ell$, this algorithm runs as follows.
  - For $i = 1, \cdots, \kappa$, computes $h_{R,i} \leftarrow \mathsf{R}(\mathbf{c}_{S,i}^\top \cdot \mathbf{k}_R \pmod q)$ and $p_{R,i} \leftarrow \mathsf{R}(\mathbf{s}_{R,i}^\top \cdot \mathbf{p}_S \pmod q)$, and $y_{R,i} = h_{R,i} \cdot p_{R,i}$.
  - Sets $\mathbf{y}_R = y_{R,1} y_{R,2} \cdots y_{R,\kappa} \in \{0,1\}^\kappa$.
  - Computes $\mathsf{der\text{-}kw}'_R := \mathbf{dk}_R = dk_{R,1} dk_{R,2} \cdots dk_{R,\ell} \leftarrow H_2(kw', \mathbf{y}_R)$.
  - Computes $\mathbf{B}_{dk} = \mathbf{C}_R + \sum_{i=1}^\ell dk_{R,i}\mathbf{B}_{R,i}$ and samples $\mathsf{td}_{\mathsf{PEKS},\mathsf{der\text{-}kw}'_R} := \mathbf{t}_{dk} \leftarrow \mathsf{SampleLeft}(\mathbf{B}_R, \mathbf{B}_{dk}, \mathbf{S}_R, \mathbf{r}_R, \sigma_2)$.
  - Outputs $\mathsf{td}_{tw'} := (\mathsf{td}_{\mathsf{PEKS},\mathsf{der\text{-}kw}'_R} := \mathbf{t}_{dk})$.

- Test(pp, ct_{kw}, $\mathsf{td}_{kw'}$): Given the public parameter pp, the searchable ciphertext ct_{kw}, and the trapdoor $\mathsf{td}_{kw'}$, this algorithm runs as follows.
  - For $j = 1, \cdots, \rho$, computes $v_j = c_{0_j} - \mathbf{t}_{dk}\mathbf{c}_{1_j} \in \mathbb{Z}_q$.
  - Checks whether $|v_j - \lfloor q/2 \rfloor| < \lfloor q/4 \rfloor$; sets $v_j = 1$ if the equation holds; otherwise, sets $v_j = 0$.
  - If $v_j = b_j$ for all $j = 1, \cdots, \rho$, outputs 1; otherwise, outputs 0.

Correctness. To ensure that the proposed construction works correctly, there are two conditions that need to be satisfies:

- If $kw = kw'$, the sender and the receiver obtain the same derived keyword (*i.e.,* der-kw$_S$ = der-kw$'_R$).
- If ct_{kw} and $\mathsf{td}_{kw'}$ are related to the same derived keyword, then the Test algorithm outputs 1.

We first consider the first condition by Lemma 6.1 followed by the description in [37]. That is, if the norm of the first error term is less than $\epsilon/2 \cdot q/4$ and $kw = kw'$, then $\mathbf{dk}_S = \mathbf{dk}_R$.

Lemma 6.1. *Suppose the norm of the first error term ($\mathbf{e}_{R,i}^\top \cdot \mathbf{k}_{S,i}$ and $\mathbf{e}_{S,i}^\top \cdot \mathbf{k}_{R,i}$) is less than $\epsilon/2 \cdot q/4$ and $kw = kw'$, then $\mathbf{dk}_S = \mathbf{dk}_R$*

Proof. For $i = 1, \cdots, \kappa$, we have

$$\begin{aligned} h_{S,i} &= \mathsf{R}(\mathbf{c}_{R,i}^\top \cdot \mathbf{k}_{S,i} \pmod q) \\ &= \mathsf{R}((\mathbf{s}_{R,i}^\top \cdot \mathbf{A}_u) \cdot \mathbf{k}_{S,i} + \underbrace{\mathbf{e}_{R,i}^\top \cdot \mathbf{k}_{S,i}}_{\text{first error term}} \pmod q) \\ &= \mathsf{R}((\mathbf{s}_{R,i}^\top \cdot \mathbf{A}_u) \cdot \mathbf{k}_{S,i} \pmod q) \\ &= p_{R,i}; \end{aligned}$$

$$\begin{aligned} h_{R,i} &= \mathsf{R}(\mathbf{c}_{S,i}^\top \cdot \mathbf{k}_{R,i} \pmod q) \\ &= \mathsf{R}((\mathbf{s}_{S,i}^\top \cdot \mathbf{A}_u) \cdot \mathbf{k}_{R,i} + \underbrace{\mathbf{e}_{S,i}^\top \cdot \mathbf{k}_{R,i}}_{\text{first error term}} \pmod q) \\ &= \mathsf{R}((\mathbf{s}_{S,i}^\top \cdot \mathbf{A}_u) \cdot \mathbf{k}_{R,i} \pmod q) \\ &= p_{S,i}. \end{aligned}$$

Since $y_{S,i} = h_{S,i} \cdot p_{R,i} = h_{R,i} \cdot p_{S,i} = y_{R,i}$ for $i = 1, \cdots, \kappa$, we have $\mathbf{y}_S = \mathbf{y}_R$. Furthermore, as $\mathbf{y}_S = \mathbf{y}_R$ and $kw = kw'$, we have der-kw$_S$ = $\mathbf{dk}_S = H_2(kw, \mathbf{y}_S) = H_2(kw', \mathbf{y}_R) = \mathbf{dk}_R$ = der-kw$'_R$. □

Then, we consider the second condition in which the Test algorithm will output a correct answer: For all $j = 1, \cdots, \rho$, we have

$$\begin{aligned} v_j &= c_{0_j} - \mathbf{t}_{dk}\mathbf{c}_{1_j} \\ &= \mathbf{r}_R^\top \mathbf{s}_j + x_j + b_j \lfloor q/2 \rfloor - \mathbf{t}_{dk}(\mathbf{F}_{dk}^\top \mathbf{s}_j + [\mathbf{y}_j; \mathbf{z}_j]) \\ &= b_j \lfloor q/2 \rfloor + \underbrace{x_j - \mathbf{t}_{dk}[\mathbf{y}_j; \mathbf{z}_j]}_{\text{second error term}}. \end{aligned}$$

According to Lemma 22 in [2], if the norm of the second error term is bounded by $q \cdot \sigma_2 \cdot \ell \cdot m \cdot \alpha \cdot \omega(\sqrt{\log m}) + O(\ell\sigma_2 m^{3/2}) \le q/5$, then b_j can be obtained correctly. Hence, we have $v_j = b_j$ for $j = 1, \cdots, \rho$ if the derived keywords are the same.

Parameter Selection. To make the system work properly, the parameters have the following restrictions [2, 37, 43]:

(1) $m > 5n \log q$ so TrapGen can operate [43].
(2) $q > \sigma_1 m^{3/2} \omega(\sqrt{\log n})$ so that the first error term is bounded by $\epsilon/2 \cdot q/4$ and therefore $\mathbf{y}_S = \mathbf{y}_R$.
(3) $\alpha < [\sigma_2 \ell m \omega(\sqrt{\log m})]^{-1}$ and $q = \Omega(\sigma_2 m^{3/2})$ so that the second error term is bounded by $q/5$.
(4) $\sigma_1 = 2\sqrt{n}$ and $q > 2\sqrt{n}/\alpha$ so that Regev's reduction [50, 51] can operate.
(5) $\sigma_2 > \ell \cdot m \cdot \omega(\sqrt{\log m})$ such that the security proof in [2] and SampleLeft work correctly.

To achieve these requirements, the parameters are set as follows.

$$\begin{aligned} m &= 6n^{1+\delta}; & q &= m^{2.5} \cdot \omega(\sqrt{\log n}); \\ \sigma_1 &= 2\sqrt{n}; & \sigma_2 &= m\ell \cdot \omega(\sqrt{\log n}); \\ n^\delta &> \lceil \log q \rceil; & \alpha &= [\ell^2 m^2 \cdot \omega(\sqrt{\log n})]^{-1}. \end{aligned}$$

Security. The security of the proposed instantiation is directly based on the underlying schemes. As the language of Li and Wang's word-independent SPHF scheme [37] is for the ciphertext of the labelled IND-CCA2 PKE scheme [43], the word of the scheme is a ciphertext. Therefore, this SPHF trivially satisfies pseudorandomness. In addition, the PEKS algorithm in Behnia *et al.*'s PEKS scheme [6] is probabilistic. On the basis of Theorem 5.1, Theorem 5.2, and Theorem 5.3, we obtain the following theorem:

Theorem 6.2. *The proposed lattice-based PAEKS scheme satisfies MCI and TP.*

7 COMPARISON

In this section, we present a comparison of our lattice-based instantiation with other PEKS/PAEKS schemes (*i.e.,* BOC+04 [7], HL17 [27], ZTW+19 [62], QCH+20 [49], BOY20 [6], ZXW+21 [63], and LTT+21 [39]) in terms of security properties, computational complexity, computational cost, and communication cost. Table 1 presents a comparison of the seven properties of each scheme, namely CI, MCI, TP, quantum-resistance (QR), and no trusted authority (NTA). As we have cryptanalyzed ZTW+19 [62] and ZTW+21 [63] in the previous section, there are only the QCH+20's [49] and LTT+21's [39] schemes satisfy TP. In addition, only LTT+21 [39] provides quantum-resistant instantiation based on the NTRU lattices. However, their solution requires an additional trusted authority to help users generate their private keys, which increases the difficulty of use in practice. To provide higher-level security, we removed this requirement. In general, our instantiation is the first quantum-resistant PAEKS scheme that satisfies MCI as well as TP and does not require a trusted authority.

We subsequently conducted two comparisons with three lattice-based schemes (*i.e.,* ZTW+19, BOY20, and ZXW+21) in terms of computational complexity and communication cost in Table 2 and Table 3, respectively. For simplicity, only five types of time-consuming operations are considered, namely general multiplication (T_M), general hash function (T_H), SamplePre function (T_{SP}), BasisDel function (T_{BD}), and SampleLeft function (T_{SL}). In addition, Fig. 1 presents the results of the experimental simulation, where the simulation was carried out in the MATLAB language on Windows 10 Enterprise Version 1909 with Inter(R) Core(TM) i7-9700 CPU with 3.00 GHz and

Table 1: Comparison of security properties with those of PAEKS schemes

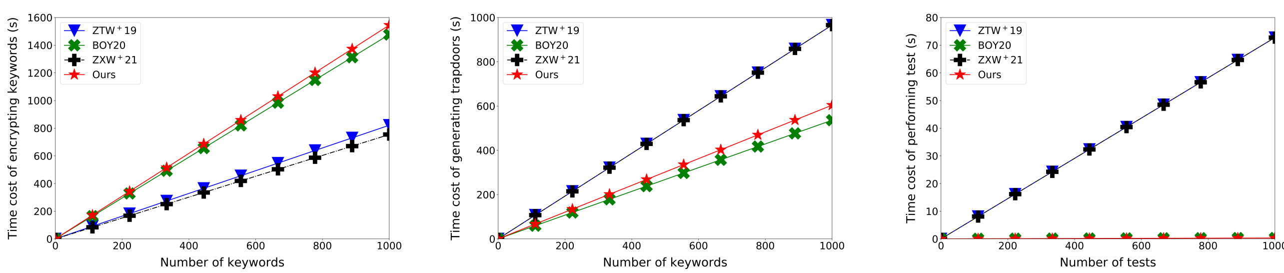
Schemes	CI	MCI	TP	QR	NTA
BOC⁺04 [7]	✓	✓	✗	✗	✓
HL17 [27]	✗	✗	✗	✗	✓
ZTW⁺19 [62]	✓	✓	✗	✓	✗
QCH⁺20 [49]	✓	✓	✓	✗	✓
BOY20 [6]	✓	✓	✗	✓	✓
ZXW⁺21 [63]	✓	✓	✗	✓	✓
LTT⁺21 [39]	✓	✓	✓	✓	✗
Ours	✓	✓	✓	✓	✓

✓: The scheme supports the corresponding feature; ✗: The scheme fails in supporting the corresponding feature; QR: Quantum-resistant; NTA: No trusted authority.

Table 2: Comparison of Required Operations with those for other Lattice-based PEKS Schemes

Schemes	Ciphertext Generation	Trapdoor Generation	Testing
ZTW⁺19 [62]	$2T_H + (\rho n + nm^2 + \rho nm + \rho)T_M + T_{SP}$	$T_H + nm^2 T_M + T_{BD} + T_{SP}$	$T_H + (\ell m + nm)T_M$
BOY20 [6]	$\rho(m^2 + 2nm + n + \ell + 1)T_M$	$\ell T_M + T_{SL}$	$2\rho m T_M$
ZXW⁺21 [63]	$T_H + (\rho n + nm^2 + \rho nm + \rho)T_M + T_{SP}$	$T_H + nm^2 T_M + T_{BD} + T_{SP}$	$T_H + (\ell m + nm)T_M$
Ours	$T_H + (\kappa(m + n + 1) + \rho(m^2 + 2nm + n + \ell + 1))T_M$	$T_H + (\kappa(m + n + 1) + \ell)T_M + T_{SL}$	$2\rho m T_M$

κ, ρ: The parameters related to security parameter λ; ℓ: The length of the keyword; T_M, T_H, T_{SP}, T_{BD}, and T_{SL}: The running time of a general multiplication, general hash function, SamplePre function, BasisDel function, and SampleLeft function, respectively.



(a) Encrypting keywords (b) Generating trapdoors (c) Performing tests

Figure 1: Comparison of Computational Costs with other Lattice-based PEKS Schemes

Table 3: Comparison of Communication Costs with other Lattice-based PEKS Schemes

Schemes	Ciphertext	Trapdoor
ZTW⁺19 [62]	$(\ell + m\ell + m)\lvert q\rvert$	$m\lvert q\rvert$
BOY20 [6]	$\kappa(\lvert q\rvert + 2m\lvert q\rvert + 1)$	$2m\lvert q\rvert$
ZXW⁺21 [63]	$(\ell + m\ell + m)\lvert q\rvert$	$m\lvert q\rvert$
Ours	$\kappa(\lvert q\rvert + 2m\lvert q\rvert + 1)$	$2m\lvert q\rvert$

n: The parameter related to security parameter; m: Dimension; q: Modules; κ: The parameter related to security parameter; ℓ: The length of the keyword.

32GB of system memory. To achieve the 80-bit security level, we set the parameters with $n = 256, m = 9753, q = 4096, \rho = 10, \kappa = 10, \ell = 10, \sigma_1 = 8, \sigma_2 = 8$, where ρ, κ are the parameters related to the security parameter (*i.e.*, $\kappa, \rho \leftarrow \mathrm{poly}(\lambda)$) and ℓ is the length of the keyword. In addition, we adopted the internal .net classes of MATLAB, namely `System.Security.Cryptography.HashAlgorithm` to implement the SHA256 hash function.

As our instantiation adopted BOY20 [6] as the building block, we first analyzed the differences with BOY20 [6]. The results indicated that our instantiation only required some extra cost in terms of computational cost. In terms of the communication cost, as our instantiation did not require additional elements to meet the required securities (*e.g.*, TP), the communication cost was the same as that for BOY20 [6]. In contrast, although our instantiation took approximately twice as long as ZTW$^+$19 [62] and ZXW$^+$21 [63] to generate ciphertexts, the time it took to generate trapdoors and perform tests decreased by approximately 40% and 99%, respectively. In terms of the communication cost, the ciphertext size and the trapdoor size of our instantiation were both approximately twice larger than those for ZTW$^+$19 [62] and ZXW$^+$21 [63]. Although the communication cost increased, we believe that this additional cost is acceptable under the trade-offs of more security and efficiency.

8 CONCLUSION

In this work, we proposed a generic PAEKS construction that could transform a PEKS scheme to a PAEKS scheme by equipping a pseudo-random SPHF scheme. Our security proofs demonstrated that the proposed construction satisfied two basic security notations—CI and TP. In addition, based on our theoretical result (Theorem 3.2), we demonstrated that the proposed construction further satisfied MCI if the PEKS algorithm of the underlying PEKS scheme were probabilistic. Furthermore, we introduced the first quantum-resistant PAEKS instantiation that not only offered privacy-preserving keyword search but also satisfied MCI and TP. Compared with the existing quantum-resistant PEKS schemes, the results indicated that our instantiation was safer and more suitable for environments with security concerns.

ACKNOWLEDGMENTS

The authors thank Keita Emura [20] for his detailed analysis of the previous version (accepted by ASIACCS 2022) and for pointing out issues and corresponding solutions. The authors thank the anonymous reviewers of ASIACCS 2022 for their insightful suggestions on this work. This research was supported by the Ministry of Science and Technology, Taiwan (ROC), under project numbers MOST 108-2218-E-004-002-MY2, MOST 109-2628-E-155-001-MY3, MOST 109-2221-E-004-011-MY3, MOST 109-3111-8-004-001-, MOST 110-2218-E-004-001-MBK, and MOST 110-2221-E-004-003-.

REFERENCES

[1] Michel Abdalla, Fabrice Benhamouda, and David Pointcheval. 2015. Public-key Encryption Indistinguishable under Plaintext-checkable Attacks. In *PKC*.
[2] Shweta Agrawal, Dan Boneh, and Xavier Boyen. 2010. Efficient Lattice (H)IBE in the Standard Model. In *EUROCRYPT*.
[3] Shweta Agrawal, Dan Boneh, and Xavier Boyen. 2010. Lattice Basis Delegation in Fixed Dimension and Shorter-ciphertext Hierarchical IBE. In *CRYPTO*.
[4] Joël Alwen and Chris Peikert. 2011. Generating Shorter Bases for Hard Random Lattices. *Theory Comput. Syst.* 48, 3 (2011), 535–553.
[5] Joonsang Baek, Reihaneh Safavi-Naini, and Willy Susilo. 2008. Public Key Encryption with Keyword Search Revisited. In *ICCSA*.
[6] Rouzbeh Behnia, Muslum Ozgur Ozmen, and Attila Altay Yavuz. 2020. Lattice-based Public Key Searchable Encryption from Experimental Perspectives. *IEEE Trans. Dependable Secur. Comput.* 17, 6 (2020), 1269–1282.
[7] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. 2004. Public Key Encryption with Keyword Search. In *EUROCRYPT*.
[8] Jin Wook Byun, Hyun Suk Rhee, Hyun-A Park, and Dong Hoon Lee. 2006. Off-line Keyword Guessing Attacks on Recent Keyword Search Schemes over Encrypted Data. In *SDM*.
[9] Ran Canetti, Dana Dachman-Soled, Vinod Vaikuntanathan, and Hoeteck Wee. 2012. Efficient Password Authenticated Key Exchange via Oblivious Transfer. In *PKC*.
[10] Yan-Cheng Chang and Michael Mitzenmacher. 2005. Privacy Preserving Keyword Searches on Remote Encrypted Data. In *ACNS*.
[11] Biwen Chen, Libing Wu, Sherali Zeadally, and Debiao He. 2019. Dual-server Public-key Authenticated Encryption with Keyword Search. *IEEE Trans. Cloud Comput.* (2019).
[12] Rongmao Chen, Yi Mu, Guomin Yang, Fuchun Guo, Xinyi Huang, Xiaofen Wang, and Yongjun Wang. 2016. Server-aided Public Key Encryption with Keyword Search. *IEEE Trans. Inf. Forensics Secur.* 11, 12 (2016), 2833–2842.
[13] Rongmao Chen, Yi Mu, Guomin Yang, Fuchun Guo, and Xiaofen Wang. 2015. Dual-server Public-key Encryption with Keyword Search for Secure Cloud Storage. *IEEE Trans. Inf. Forensics Secur.* 11, 4 (2015), 789–798.
[14] Rongmao Chen, Yi Mu, Guomin Yang, Fuchun Guo, and Xiaofen Wang. 2015. A New General Framework for Secure Public Key Encryption with Keyword Search. In *ACISP*.
[15] Leixiao Cheng and Fei Meng. 2021. Security Analysis of Pan et al.'s "Public-key Authenticated Encryption with Keyword Search Achieving Both Multi-ciphertext and Multi-trapdoor Indistinguishability". *J. Syst. Archit.* (2021), 102248.
[16] Ronald Cramer and Victor Shoup. 2002. Universal Hash Proofs and a Paradigm for Adaptive Chosen Ciphertext Secure Public-key Encryption. In *EUROCRYPT*.
[17] Ioannis Demertzis, Javad Ghareh Chamani, Dimitrios Papadopoulos, and Charalampos Papamanthou. 2020. Dynamic Searchable Encryption with Small Client Storage. In *NDSS*.
[18] Pierre-Alain Dupont, Julia Hesse, David Pointcheval, Leonid Reyzin, and Sophia Yakoubov. 2018. Fuzzy Password-authenticated Key Exchange. In *EUROCRYPT*.
[19] Keita Emura. 2017. A Generic Construction of Secure-channel Free Searchable Encryption with Multiple Keywords. In *NSS*.
[20] Keita Emura. 2022. Generic Construction of Public-key Authenticated Encryption with Keyword Search Revisited: Stronger Security and Efficient Construction. Cryptology ePrint Archive, Report 2022/072. https://ia.cr/2022/072.
[21] Keita Emura, Atsuko Miyaji, Mohammad Shahriar Rahman, and Kazumasa Omote. 2015. Generic Constructions of Secure-channel Free Searchable Encryption with Adaptive Security. *Secur. Commun. Networks* 8, 8 (2015), 1547–1560.
[22] Liming Fang, Willy Susilo, Chunpeng Ge, and Jiandong Wang. 2009. A Secure Channel Free Public Key Encryption with Keyword Search Scheme without Random Oracle. In *CANS*.
[23] Rosario Gennaro and Yehuda Lindell. 2003. A Framework for Password-based Authenticated Key Exchange. In *EUROCRYPT*.
[24] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. 2008. Trapdoors for Hard Lattices and New Cryptographic Constructions. In *STOC*.
[25] Adam Groce and Jonathan Katz. 2010. A New Framework for Efficient Password-based Authenticated Key Exchange. In *CCS*.
[26] Debiao He, Mimi Ma, Sherali Zeadally, Neeraj Kumar, and Kaitai Liang. 2017. Certificateless Public Key Authenticated Encryption with Keyword Search for Industrial Internet of Things. *IEEE Trans. Ind. Informatics* 14, 8 (2017), 3618–3627.
[27] Qiong Huang and Hongbo Li. 2017. An Efficient Public-key Searchable Encryption Scheme Secure against Inside Keyword Guessing Attacks. *Inf. Sci.* 403 (2017), 1–14.
[28] Yong Ho Hwang and Pil Joong Lee. 2007. Public Key Encryption with Conjunctive Keyword Search and Its Extension to a Multi-user System. In *Pairing*.
[29] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. 2018. OPAQUE: An Asymmetric PAKE Protocol Secure against Pre-computation Attacks. In *EUROCRYPT*.
[30] Ik Rae Jeong, Jeong Ok Kwon, Dowon Hong, and Dong Hoon Lee. 2009. Constructing PEKS Schemes Secure against Keyword Guessing Attacks is Possible? *Comput. Commun.* 32, 2 (2009), 394–396.
[31] Peng Jiang, Yi Mu, Fuchun Guo, Xiaofen Wang, and Qiaoyan Wen. 2016. Online/offline Ciphertext Retrieval on Resource Constrained Devices. *Comput. J.* 59, 7 (2016), 955–969.
[32] Jonathan Katz and Vinod Vaikuntanathan. 2009. Smooth Projective Hashing and Password-based Authenticated Key Exchange from Lattices. In *ASIACRYPT*.
[33] Jonathan Katz and Vinod Vaikuntanathan. 2011. Round-optimal Password-based Authenticated Key Exchange. In *TCC*.
[34] Jonathan Katz and Vinod Vaikuntanathan. 2013. Round-optimal Password-based Authenticated Key Exchange. *J. Cryptol.* 26, 4 (2013), 714–743.
[35] Franziskus Kiefer and Mark Manulis. 2014. Distributed Smooth Projective Hashing and Its Application to Two-server Password Authenticated Key Exchange. In *ACNS*.
[36] Hongbo Li, Qiong Huang, Jian Shen, Guomin Yang, and Willy Susilo. 2019. Designated-server Identity-based Authenticated Encryption with Keyword Search for Encrypted Emails. *Inf. Sci.* 481 (2019), 330–343.
[37] Zengpeng Li and Ding Wang. 2019. Achieving One-round Password-based Authenticated Key Exchange over Lattices. *IEEE Trans. Serv. Comput.* (2019).
[38] Zi-Yuan Liu, Yi-Fan Tseng, Raylin Tso, Yu-Chi Chen, and Masahiro Mambo. 2021. Identity-certifying Authority-aided Identity-based Searchable Encryption Framework in Cloud System. *IEEE Syst. J.* (2021).
[39] Zi-Yuan Liu, Yi-Fan Tseng, Raylin Tso, Masahiro Mambo, and Yu-Chi Chen. 2021. Public-key Authenticated Encryption with Keyword Search: A Generic Construction and Its Quantum-resistant Instantiation. *Comput. J.* (2021).
[40] Yang Lu and Jiguo Li. 2021. Lightweight Public Key Authenticated Encryption with Keyword Search against Adaptively-chosen-targets Adversaries for Mobile Devices. *IEEE Trans. Mob. Comput.* (2021).
[41] Yang Lu, Jiguo Li, and Fen Wang. 2020. Pairing-free Certificate-based Searchable Encryption Supporting Privacy-preserving Keyword Search Function for IIoTs. *IEEE Trans. Ind. Informatics* 17, 4 (2020), 2696–2706.

[42] Yang Lu, Jiguo Li, and Yichen Zhang. 2019. Secure Channel Free Certificate-based Searchable Encryption withstanding Outside and Inside Keyword Guessing Attacks. *IEEE Trans. Serv. Comput.* (2019).
[43] Daniele Micciancio and Chris Peikert. 2012. Trapdoors for Lattices: Simpler, Tighter, Faster, Smaller. In *EUROCRYPT*.
[44] Tarik Moataz and Abdullatif Shikfa. 2013. Boolean Symmetric Searchable Encryption. In *ASIACCS*.
[45] Mahnaz Noroozi and Ziba Eslami. 2019. Public Key Authenticated Encryption with Keyword Search: Revisited. *IET Inf. Secur.* 13, 4 (2019), 336–342.
[46] Simon Oya and Florian Kerschbaum. 2021. Hiding the Access Pattern is not Enough: Exploiting Search Pattern Leakage in Searchable Encryption. In *USENIX Security*.
[47] Nasrollah Pakniat, Danial Shiraly, and Ziba Eslami. 2020. Certificateless Authenticated Encryption with Keyword Search: Enhanced Security Model and a Concrete Construction for Industrial IoT. *J. Inf. Secur. Appl.* 53 (2020), 102525.
[48] Xiangyu Pan and Fagen Li. 2021. Public-key Authenticated Encryption with Keyword Search Achieving Both Multi-ciphertext and Multi-trapdoor Indistinguishability. *J. Syst. Archit.* 115 (2021), 102075.
[49] Baodong Qin, Yu Chen, Qiong Huang, Ximeng Liu, and Dong Zheng. 2020. Public-key Authenticated Encryption with Keyword Search Revisited: Security Model and Constructions. *Inf. Sci.* 516 (2020), 515–528.
[50] Oded Regev. 2005. On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. In *STOC*.
[51] Oded Regev. 2009. On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. *J. ACM* 56, 6 (2009), 34:1–34:40.
[52] Hyun Sook Rhee, Jong Hwan Park, Willy Susilo, and Dong Hoon Lee. 2009. Improved Searchable Public Key Encryption with Designated Tester. In *ASIACCS*.
[53] Hyun Sook Rhee, Jong Hwan Park, Willy Susilo, and Dong Hoon Lee. 2010. Trapdoor Security in a Searchable Public-key Encryption Scheme with a Designated Tester. *J. Syst. Softw.* 83, 5 (2010), 763–771.
[54] Peter W Shor. 1994. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In *FOCS*.
[55] Peter W Shor. 1999. Polynomial-time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Rev.* 41, 2 (1999), 303–332.
[56] Dawn Xiaodong Song, David A. Wagner, and Adrian Perrig. 2000. Practical Techniques for Searches on Encrypted Data. In *IEEE S&P*.
[57] Shi-Feng Sun, Xingliang Yuan, Joseph K Liu, Ron Steinfeld, Amin Sakzad, Viet Vo, and Surya Nepal. 2018. Practical Backward-secure Searchable Encryption from Symmetric Puncturable Encryption. In *CCS*.
[58] Tatsuya Suzuki, Keita Emura, and Toshihiro Ohigashi. 2018. A Generic Construction of Integrated Secure-channel Free PEKS and PKE. In *ISPEC*.
[59] Tatsuya Suzuki, Keita Emura, and Toshihiro Ohigashi. 2019. A Generic Construction of Integrated Secure-channel Free PEKS and PKE and its Application to EMRs in Cloud Storage. *J. Medical Syst.* 43, 5 (2019), 1–15.
[60] Qiang Tang and Liqun Chen. 2009. Public-Key Encryption with Registered Keyword Search. In *EuroPKI*.
[61] Lei Xu, Xingliang Yuan, Ron Steinfeld, Cong Wang, and Chungen Xu. 2019. Multi-writer Searchable Encryption: An LWE-based Realization and Implementation. In *ASIACCS*.
[62] Xiaojun Zhang, Yao Tang, Huaxiong Wang, Chunxiang Xu, Yinbin Miao, and Hang Cheng. 2019. Lattice-based Proxy-oriented Identity-based Encryption with Keyword Search for Cloud Storage. *Inf. Sci.* 494 (2019), 193–207.
[63] Xiaojun Zhang, Chunxiang Xu, Huaxiong Wang, Yuan Zhang, and Shixiong Wang. 2021. FS-PEKS: Lattice-based Forward Secure Public-key Encryption with Keyword Search for Cloud-assisted Industrial Internet of Things. *IEEE Trans. Dependable Secur. Comput.* 18, 3 (2021), 1019–1032.

A ZHANG *ET AL.*'S PEKS SCHEMES

A.1 Forward-secure PEKS

Here, we briefly review Zhang *et al.*'s lattice-based forward-secure PEKS scheme [63], which consists of five algorithms.

- $\mathsf{Setup}(1^\lambda)$: Taking as input a security parameter λ, this algorithm runs the following steps:
  - Randomly selects $\boldsymbol{\mu} \xleftarrow{\$} \mathbb{Z}_q^n$ and three secure hash functions $H_1 : \mathbb{Z}_q^{n\times m} \times \{0, \cdots, \eta\} \to \mathbb{Z}_q^{m\times m}$, $H_2 : \{0,1\}^{\ell_1} \times \{0, \cdots, \eta\} \to \mathbb{Z}_q^{n\times m}$, $H_3 : \mathbb{Z}_q^{m\times \ell} \times \{0,1\}^{\ell} \to \mathbb{Z}_q^n$.
  - Respectively generates $(\mathbf{A}_{S\|0}, \mathbf{T}_{S\|0})$ and $(\mathbf{A}_{R\|0}, \mathbf{T}_{R\|0})$ by performing $\mathsf{TrapGen}(1^n, 1^m, q)$.
  - Outputs the public parameter $\mathsf{pp} := (\boldsymbol{\mu}, H_1, H_2, H_3)$, the public/private key pairs of the sender $(\mathsf{pk}_{S,0} := \mathbf{A}_{S\|0}, \mathsf{sk}_{S,0} := \mathbf{T}_{S\|0})$ and the receiver $(\mathsf{pk}_{R,0} := \mathbf{A}_{R\|0}, \mathsf{sk}_{R,0} := \mathbf{T}_{R\|0})$ for time period 0.
- $\mathsf{KeyUpdate}(\mathsf{pk}_{R,i}, \mathsf{sk}_{R,i}, i, j)$: Taking as input an input public/private key pair $(\mathsf{pk}_{R,i} := \mathbf{A}_{R\|i}, \mathsf{sk}_{R,i} := \mathbf{T}_{R\|i})$ of the receiver in the previous time period i and the current time period j, this algorithm runs the following steps:
  - Computes $\mathbf{R}_{R\|i\to j} = H_1(\mathbf{A}_R\|j) + \cdots + H_1(\mathbf{A}_R\|i+1) \in \mathbb{Z}_q^{m\times m}$.
  - Computes $\mathbf{T}_{R\|j} \leftarrow \mathsf{NewBasisDel}(\mathbf{A}_{R\|i}, \mathbf{R}_{R\|i\to j}, \mathbf{T}_{R\|i}, \delta_j)$, where $\mathbf{A}_{R\|j} = \mathbf{A}_{R\|i}(\mathbf{R}_{R\|i\to j})^{-1} = \mathbf{A}_R(\mathbf{R}_{R\|j})^{-1} \in \mathbb{Z}_q^{n\times m}$.
  - Outputs the public/private key pair $(\mathsf{pk}_{R,j} := \mathbf{A}_{R\|j}, \mathsf{sk}_{R,j} := \mathbf{T}_{R\|j})$ of the receiver for time period j.

  Note that the sender can use the same steps to generate his/her public/private key pair $(\mathsf{pk}_{S,j} := \mathbf{A}_{S\|j}, \mathsf{sk}_{S,j} := \mathbf{T}_{S\|j})$ for time period j.
- $\mathsf{PEKS}(\mathsf{pk}_{S,j}, \mathsf{sk}_{S,j}, \mathsf{pk}_{R,j}, j, kw)$: Taking as input a public/private key pair $(\mathsf{pk}_{S,j} := \mathbf{A}_{S\|j}, \mathsf{sk}_{\mathsf{S,j}} := \mathbf{T}_{S\|j})$ of the sender for time period j, the public key $\mathsf{pk}_{R,j} := \mathbf{A}_{R\|j}$ of the receiver for time period j, the current time period j, and keyword $kw \in \{0,1\}^{\ell_1}$, the sender runs the following steps:
  - Chooses a random binary string $\boldsymbol{\gamma}_j = \gamma_{j_1}\gamma_{j_2}\cdots\gamma_{j_\ell} \xleftarrow{\$} \{0,1\}^{\ell}$, uniform matrix $\mathbf{B}_j \xleftarrow{\$} \mathbb{Z}_q^{n\times\ell}$, noise $\mathbf{e}_j = e_{j_1}e_{j_2}\cdots e_{j_\ell})$, and noise $\mathbf{V}_j = \mathbf{v}_{j_1}\mathbf{v}_{j_2}\cdots\mathbf{v}_{j_\ell}$, where $e_{j_1}, \cdots, e_{j_\ell} \xleftarrow{\$} \mathbb{Z}_q$ and $\mathbf{v}_{j_1}, \cdots, \mathbf{v}_{j_\ell} \xleftarrow{\$} \mathbb{Z}_q^m$.
  - Computes $\boldsymbol{\beta}_j \leftarrow H_2(kw\|j), \mathbf{c}_{j_1} = \boldsymbol{\mu}^\top\mathbf{B}_j + e_j + (\gamma_{j_1}, \cdots, \gamma_{j_\ell})\lfloor q/2\rfloor, \mathbf{c}_{j_2} = (\mathbf{A}_{R\|j}\boldsymbol{\beta}_j^{-1})\mathbf{B}_j + \mathbf{V}_j$.
  - Computes $\mathbf{h}_j \leftarrow H_3(\mathbf{c}_{j_2}\|\boldsymbol{\gamma}_j) \in \mathbb{Z}_q^n$ and generates $\boldsymbol{\zeta}_j \leftarrow \mathsf{SamplePre}(\mathbf{A}_{S\|j}, \mathbf{T}_{S\|j}, \mathbf{h}_j, \sigma_j)$.
  - Outputs a searchable ciphertext $\mathsf{ct}_j := (\mathbf{c}_{j_1}, \mathbf{c}_{j_2}, \boldsymbol{\zeta}_j)$.
- $\mathsf{Trapdoor}(\mathsf{pk}_{R,j}, \mathsf{sk}_{R,j}, j, kw)$: Taking as input a public/private key pair $(\mathsf{pk}_{R,j} := \mathbf{A}_{R\|j}, \mathsf{sk}_{R,j} := \mathbf{T}_{R\|j})$ of the receiver for time period j, current time period j and keyword $kw \in \{0,1\}^{\ell_1}$, the receiver runs the following steps:
  - Computes $\boldsymbol{\beta}_j \leftarrow H_2(kw\|j), \mathbf{T}_{kw\|j} \leftarrow \mathsf{NewBasisDel}(\mathbf{A}_{R\|j}, \boldsymbol{\beta}_j, \mathbf{T}_{R\|j}, \delta_j) \in \mathbb{Z}_q^{m\times m}$.
  - Computes $\mathbf{t}_{kw\|j} \leftarrow \mathsf{SamplePre}(\mathbf{A}_{R\|j}\boldsymbol{\beta}_j^{-1}, \mathbf{T}_{w\|j}, \boldsymbol{\mu}, \sigma_j) \in \mathbb{Z}_q^m$.
  - Outputs a trapdoor $\mathsf{td}_j := \mathbf{t}_{kw\|j}$.
- $\mathsf{Test}(\mathsf{ct}_j, \mathsf{td}_j)$: Taking as input a ciphertext $\mathsf{ct}_j := (\mathbf{c}_{j_1}, \mathbf{c}_{j_2}, \boldsymbol{\zeta}_j)$ for time period j and trapdoor $\mathsf{td}_j := \mathbf{t}_{kw\|j}$ for time period j, the cloud server runs the following steps:
  - Computes $\boldsymbol{\gamma}_j = \gamma_{j_1}\gamma_{j_2}\cdots\gamma_{j_\ell} = \mathbf{c}_{j_1} - \mathbf{t}_{w\|j}^\top\mathbf{c}_{j_2}$.
  - For $k = 1, \cdots, \ell$, if $|\gamma_{j_k} - \lfloor q/2\rfloor| < \lfloor q/4\rfloor$, sets $\gamma_{j_k} = 1$; otherwise, sets $\gamma_{j_k} = 0$.
  - Updates $\boldsymbol{\gamma}_j$ and computes $\mathbf{h}_j \leftarrow H_3(\mathbf{c}_{j_2}\|\boldsymbol{\gamma}_j) \in \mathbb{Z}_q^n$.
  - If $\mathbf{A}_{S\|j}\boldsymbol{\zeta}_j = \mathbf{h}_j$ and $\boldsymbol{\zeta}_j$ is distributed in $\mathcal{D}_{\Lambda_q^{\mathbf{h}_j}(\mathbf{A}_{S\|j}),\sigma_j}$, outputs 1; otherwise, outputs 0.

A.2 Proxy-oriented Identity-based PEKS

In this subsection, we review Zhang *et al.*'s proxy-oriented identity-based PEKS scheme [62], which consists of six algorithms.

- $\mathsf{Setup}(1^\lambda)$: Taking as input a security parameter λ, the key generator center runs the following steps:

- Generates $(\mathbf{A}, \mathbf{T_A}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)$.
- Selects a uniform random vector $\mathbf{v} \xleftarrow{\$} \mathbb{Z}_q^n$ and five secure cryptographic hash functions: $H_1 : \{0,1\}^{\ell_1} \to \mathbb{Z}_q^{m\times m}$, $H_2 : \{0,1\}^{\ell_1} \times \{0,1\}^{\ell_1} \times \{0,1\}^{\ell_2} \times \mathbb{Z}_q^n \to \mathbb{Z}_q^n$, $H_3 : \{0,1\}^{\ell_1} \times \{0,1\}^{\ell_1} \times \{0,1\}^{\ell_2} \times \mathbb{Z}_q^m \to \mathbb{Z}_q^{m\times m}$, $H_4 : \{0,1\}^{\ell_1} \times \{0,1\}^{\ell_1} \times \{0,1\}^{\ell_3} \to \mathbb{Z}_q^{m\times m}$, and $H_5 : \{0,1\}^{\ell} \times \mathbb{Z}_q^{m\times \ell} \to \mathbb{Z}_q^n$.
- Outputs the public parameters $\mathsf{pp} := (\mathbf{A}, \mathbf{v}, H_1, H_2, H_3, H_4, H_5)$ and master private key $\mathsf{msk} := \mathbf{T_A}$.

- $\mathsf{KeyExtract}(\mathsf{msk}, id)$: Taking as input the master secret key $\mathsf{msk} := \mathbf{T_A}$ and an identity $id \in \{0,1\}^{\ell_1}$, key generator center runs the following steps:
  - Computes $\mathbf{R}_{id} \leftarrow H_1(id)$ and $\mathbf{A}_{id} = \mathbf{A}(\mathbf{R}_{id})^{-1} \in \mathbb{Z}_q^{n\times m}$.
  - Generates $\mathbf{T}_{id} \leftarrow \mathsf{NewBasisDel}(\mathbf{A}, \mathbf{R}_{id}, \mathbf{T_A}, \sigma)$.
  - Outputs the secret key $\mathsf{sk}_{id} := \mathbf{T}_{id}$ for identity id.
- Proxy-oriented key generation: This interactive PPT algorithm between a data owner id_O and a proxy id_P. id_O first runs the following steps:
  - Generates a warrant $w \in \{0,1\}^{\ell}$ according to its requirements.
  - Selects a uniform random vector $\mathbf{r} \xleftarrow{\$} \mathbb{Z}_q^n$ and computes $\boldsymbol{\mu} \leftarrow H_2(id_O \| id_P \| w \| \mathbf{r})$.
  - Computes $\boldsymbol{\beta}_w \leftarrow \mathsf{SamplePre}(\mathbf{A}_{id_O}, \mathbf{T}_{id_O}, \boldsymbol{\mu}, \delta) \in \mathbb{Z}_q^m$.
  - Sends $(w, \mathbf{r}, \boldsymbol{\beta}_w)$ directly to id_P.

  After receiving the data send from id_O, id_P runs the following steps:
  - Computes $\mathbf{R}_w \leftarrow H_3(id_O \| id_P \| w \| \boldsymbol{\beta}_w)$ and $\mathbf{T}_{pro} \leftarrow \mathsf{NewBasisDel}(\mathbf{A}_{id_P}, \mathbf{R}_w, \mathbf{T}_{id_P}, \sigma)$.
  - Sets $(\mathsf{pk}_{pro} := \mathbf{A}_{pro}, \mathsf{sk}_{pro} := \mathbf{T}_{pro})$ as the proxy-oriented public/private key pair, where $\mathbf{A}_{pro} = \mathbf{A}_{id_P}(\mathbf{R}_w)^{-1} \in \mathbb{Z}_q^{n\times m}$.
- $\mathsf{IBEKS}(\mathsf{pk}_{pro}, \mathsf{sk}_{pro}, kw, id_R)$: Taking as input the public/private key pair $(\mathsf{pk}_{pro} := \mathbf{A}_{pro}, \mathsf{sk}_{pro} := \mathbf{T}_{pro})$ of the proxy-oriented a keyword $kw \in \{0,1\}^{\ell_3}$, and receiver's identity id_R, the proxy id_P runs the following steps:
  - Randomly chooses $\mathbf{F} \xleftarrow{\$} \mathbb{Z}_q^{n\times \ell}$ and a binary string $\boldsymbol{\tau} = \tau_1\tau_2\cdots\tau_\ell \xleftarrow{\$} \{0,1\}^{\ell}$.
  - Samples a noise vector $\boldsymbol{\eta} = \eta_1\eta_2\cdots\eta_\ell \xleftarrow{\$} \chi$ and a noise matrix $\mathbf{S} = \mathbf{s}_1\mathbf{s}_2\cdots\mathbf{s}_\ell \xleftarrow{\$} \mathbb{Z}_q^{m\times \ell}$, where χ is a Gaussian distribution.
  - Computes $\boldsymbol{\gamma} \leftarrow H_4(id_P \| id_R \| kw)$, $\boldsymbol{\xi} = (\mathbf{A}_{id_R}\boldsymbol{\gamma}^{-1})^{\top}\mathbf{F} + \mathbf{S}$, $\boldsymbol{\zeta} = \mathbf{v}^{\top}\mathbf{F} + \boldsymbol{\eta} + (\tau_1, \tau_2, \cdots, \tau_\ell)\lfloor q/2 \rfloor$.
  - Computes $\mathbf{h} \leftarrow H_5(\boldsymbol{\tau} \| \boldsymbol{\xi})$ and $\boldsymbol{\theta} \leftarrow \mathsf{SamplePre}(\mathbf{A}_{pro}, \mathbf{T}_{pro}, \mathbf{h}, \delta) \in \mathbb{Z}_q^m$.
  - Outputs a searchable ciphertext $\mathsf{ct} := (\boldsymbol{\xi}, \boldsymbol{\zeta}, \boldsymbol{\theta})$.
- $\mathsf{Trapdoor}(\mathsf{sk}_{id_R}, kw)$: Taking as input the private key $\mathsf{sk}_{id_R} := \mathbf{T}_{id_R}$ of the receiver id_R and a keyword $kw \in \{0,1\}^{\ell_3}$, id_R runs the following steps:
  - Computes $\boldsymbol{\gamma} \leftarrow H_4(id_P \| id_R \| kw)$ and $\mathbf{D}_{kw} \leftarrow \mathsf{NewBasisDel}(\mathbf{A}_{id_R}, \boldsymbol{\gamma}, \mathbf{T}_{id_R}, \sigma) \in \mathbb{Z}_q^{m\times m}$.
  - Generates $\mathbf{d}_{kw} \leftarrow \mathsf{SamplePre}(\mathbf{A}_{id_R}\boldsymbol{\gamma}^{-1}, \mathbf{D}_{kw}, \mathbf{v}, \delta) \in \mathbb{Z}_q^m$, where $\mathbf{A}_{id_R}\boldsymbol{\gamma}^{-1}\mathbf{d}_{kw} = \mathbf{v}$ is satisfied.
  - Outputs a trapdoor $\mathsf{td} := \mathbf{d}_{kw}$.
- $\mathsf{Test}(\mathsf{pk}_{pro}, \mathsf{ct}, \mathsf{td})$: Taking as input the proxy-oriented public key $\mathsf{pk}_{pro} := \mathbf{A}_{\mathrm{pro}}$, a searchable ciphertext $\mathsf{ct} := (\boldsymbol{\xi}, \boldsymbol{\zeta}, \boldsymbol{\theta})$, and a trapdoor $\mathsf{td} := \mathbf{d}_{kw}$, the cloud server runs the following steps:
  - Computes $\boldsymbol{\tau} = \tau_1\tau_2\cdots\tau_\ell = \boldsymbol{\zeta} - \mathbf{d}_{kw}^{\top}\boldsymbol{\xi} \in \mathbb{Z}_q^{\ell}$.
  - For $j = 1, \cdots, \ell$, if $|\tau_j - \lfloor q/2 \rfloor| < \lfloor q/4 \rfloor$, sets $\tau_j = 1$; otherwise, sets $\tau_j = 0$.
  - Updates $\boldsymbol{\tau}$ and further computes $\mathbf{h} \leftarrow H_5(\boldsymbol{\tau} \| \boldsymbol{\xi})$.
  - Checks whether the equation $\mathbf{A}_{pro}\boldsymbol{\theta} \stackrel{?}{=} \mathbf{h}$ holds. If the equation holds, outputs 1; otherwise, outputs 0.