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
yftseeng@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 multi-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.

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).
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 (≈ 216) words in Merriam-Webster’s collegiate dictionary [10]. Consequently, upon a brute force attack, there is a high probability (1/216) 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 encrypted by a specified sender. In this way, the adversaries cannot perform IKGA 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 MTP 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.

In addition, 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 quantum-resistant PAEKS that satisfies both MCI and MTP (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 and MTP properties, we provide a theoretical result in Theorem 3.3: if the PAEKS and Trapdoor algorithms of a PAEKS scheme is probabilistic and the PAEKS scheme satisfies CI and TP, then this PAEKS scheme also satisfies MCI and MTP. 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 \( B = \{ b_1, \ldots, b_n \} \in \mathbb{R}^{n \times m} \), where \( b_1, \ldots, b_n \) are \( n \) linearly independent vectors. An \( m \)-dimensional lattice \( \mathcal{L} \) generated by \( B \) is defined as \( \mathcal{L}(B) := \{ \sum_{i=1}^{n} a_i b_i \mid a_i \in \mathbb{Z} \} \). Here, \( \mathcal{L} \) is called the basis of \( \Lambda \). In addition, given \( n, m, q \in \mathbb{Z} \), \( \Lambda \in \mathbb{Z}^n_q \), and \( \Lambda \in \mathbb{Z}^m_q \), we can define two \( q \)-ary lattices and a coset as follows:

- \( \Lambda_q(A) := \{ y \in \mathbb{Z}^m_q \mid 3 \mathbb{Z} \in \mathbb{Z}^n_q, y = A^T z \mod q \} \);
- \( \Lambda_q^\perp(A) := \{ e \in \mathbb{Z}^n_q \mid Ae = 0 \mod q \} \);
- \( \Lambda_q^\perp(A) := \{ e \in \mathbb{Z}^n_q \mid Ae = u \mod q \} \).

2.1.2 Discrete Gaussian Distributions. For any positive real number \( \sigma \), any center \( c \in \mathbb{Z}^m \), and any \( x \in \mathbb{Z}^m \), we define the Gaussian distribution of \( \mathcal{D}_{\sigma,c} \) by the probability distribution function

\[ \rho_{\sigma,c}(x) := \exp(- \pi \cdot |x - c|^2 / \sigma^2) \]

Furthermore, for any lattice \( \Lambda \subset \mathbb{Z}^m \), we define \( \rho_{\sigma,c} = \sum_{\xi \in \Lambda} \rho_{\sigma,c}(x) \). Then, the discrete Gaussian distribution over lattice \( \Lambda \) with parameter \( (\sigma, c) \) is defined as follows: For any \( x \in \Lambda \),

\[ \mathcal{D}_{\mathcal{L}_\Lambda,\sigma,c} := \rho_{\sigma,c}(x) / \rho_{\sigma,c}(A) \].

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

1. **TrapGen** \((1^n, 1^m, q) \) \([4, 43]\): For an input \( n, m, q \in \mathbb{Z} \), this probabilistic polynomial time (PPT) algorithm outputs a pair \((\mathbf{A} \in \mathbb{Z}^n_q \times \mathbb{Z}^m_q, \mathbf{T}_\mathbf{A} \in \mathbb{Z}^m_q) \), where \( \mathbf{T}_\mathbf{A} \) is a basis for \( \Lambda_q^\perp(\mathbf{A}) \), such that the following property holds:

\[ \{ x \in \mathbb{Z}^m_q \mid x = \mathbf{T}_\mathbf{A} \mathbf{A} \} \Rightarrow \{ x \in \mathbb{Z}^m_q \mid x = \mathbf{T}_\mathbf{A} \mathbf{A} \} \].

Here, \( \mathbf{T}_\mathbf{A} \) is called a trapdoor of \( \mathbf{A} \).

2. **SamplePre** \((\mathbf{A}, \mathbf{T}_\mathbf{A}, u, \sigma) \) \([24]\): For an input matrix \( \mathbf{A} \in \mathbb{Z}^n_q \times \mathbb{Z}^m_q \) and its trapdoor \( \mathbf{T}_\mathbf{A} \in \mathbb{Z}^m_q \), a vector \( u \in \mathbb{Z}^n_q \), and parameter \( \sigma \geq \| \mathbf{T}_\mathbf{A} \| \cdot \sqrt{\log(m)} \), this PPT algorithm outputs a sample \( t \in \mathbb{Z}^m_q \) from a distribution that is statistically close to \( \mathcal{D}_{\mathcal{L}_\mathbf{A}, u, \sigma} \) such that \( \mathbf{At} = u \mod q \).

3. **NewBasisDef** \((\mathbf{A}, \mathbf{T}_\mathbf{A}, \sigma) \) \([3]\): For an input matrix \( \mathbf{A} \in \mathbb{Z}^n_q \times \mathbb{Z}^m_q \) and its inverse \( \mathbf{A}^{-1} \), a vector \( u \in \mathbb{Z}^n_q \), and parameter \( \sigma \geq \| \mathbf{T}_\mathbf{A} \| \cdot \sqrt{\log(m + m)} \), this PPT algorithm outputs a sample \( t \in \mathbb{Z}^m_q \) from a distribution statistically close to \( \mathcal{D}_{\mathcal{L}_\mathbf{A}, \sigma} \) such that \( \mathbf{At} = u \mod q \).

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:

- **KeyGen** \((1^t) \): Taking as input a security parameter \( t \), this PPT algorithm outputs a pair of keys \((\mathbf{pK}, \mathbf{sK})\), where \( \mathbf{pK} \) is the public key and \( \mathbf{sK} \) is the private key.
- **PEKS** \((\mathbf{pK}, \mathbf{sK}, k) \): Taking as input the public key \( \mathbf{pK} \) and a keyword \( k \), this PPT algorithm outputs a searchable ciphertext \( \mathbf{ct} \) related to the keyword \( k \).
- **Trapdoor** \((\mathbf{sK}, k) \): Taking as input the private key \( \mathbf{sK} \) and a keyword \( k \), this PPT algorithm outputs a trapdoor \( \mathbf{td} \) related to keyword \( k \).
- **Test** \((\mathbf{ct}, \mathbf{td}, k) \): Taking as input the searchable ciphertext \( \mathbf{ct} \) and trapdoor \( \mathbf{td} \), this deterministic algorithm outputs 1 if \( \mathbf{ct} \) and \( \mathbf{td} \) are related to the same keyword (i.e., \( k = k' \)); otherwise, it outputs 0.

**Correctness.** For any security parameter \( t \), any honestly generated key pairs \((\mathbf{pK}, \mathbf{sK})\), keywords \( k, k' \), any ciphertext \( \mathbf{ct} \) and trapdoor \( \mathbf{td} \), the following properties must hold:

- **Setup.** After receiving a security parameter \( t \), \( C \) generates \((\mathbf{pK}, \mathbf{sK})\) by performing the KeyGen algorithm. Then, it sends the public key \( \mathbf{pK} \) to \( \mathcal{A} \) and keeps the private key \( \mathbf{sK} \) secret.
- **Phase 1.** In this phase, \( \mathcal{A} \) is allowed to adaptively issue queries to the trapdoor oracle polynomially many times: for any keyword \( k \), \( C \) generates a trapdoor \( \mathbf{td} \) by performing Trapdoor \((\mathbf{sK}, k) \) and returns \( \mathbf{td} \) to \( \mathcal{A} \).
- **Challenge.** \( \mathcal{A} \) terminates the **Phase 1**, it outputs two challenge keywords \( k_0, k_1 \). \( C \) then randomly chooses a bit \( b \in \{0, 1\} \) and returns the challenge ciphertext \( \mathbf{ct}^* \) to \( \mathcal{A} \) by performing PEKS \((\mathbf{pK}, k_b^*) \).
• **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

$$\text{Adv}_{\mathcal{A}}^{\text{C}l-\text{PEKS}}(\lambda) := \Pr[b = b'] - \frac{1}{2}.$$ 

**Definition 2.1 (Ciphertext Indistinguishability of PEKS)**. A PEKS scheme is called CI (or IND-CKA secure) if, for any PPT adversary $\mathcal{A}$, $\text{Adv}_{\mathcal{A}}^{\text{C}l-\text{PEKS}}(\lambda)$ 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 consists of the following three algorithms:

- **KeyGen**($\lambda^2$): Taking as input a security parameter $\lambda$, this PPT algorithm outputs a pair of keys $(ek_pKE, dk_pKE)$, where $ek_pKE$ is the public encryption key and $dk_pKE$ is the private decryption key.
- **Encrypt**$(ek_pKE, label, m_pKE; \rho)$: Taking as input the public encryption key $ek_pKE$, a label, a plaintext $m_pKE$, and a randomness $\rho$, this PPT algorithm outputs a ciphertext $ct_pKE$.
- **Decrypt**$(dk_pKE, label, ct_pKE)$: Taking as input the private decryption key $dk_pKE$, a label, and a ciphertext $ct_pKE$, this deterministic algorithm outputs a plaintext $m_pKE$ or $\bot$.

In addition, it must satisfy the following correctness and security:

- **Correctness**: For any security parameter $\lambda$, any pair of keys $(dk_pKE, ek_pKE) \leftrightarrow \text{KeyGen}(\lambda)$, any label, any plaintext $m_pKE$, any randomness $\rho$, and any ciphertext $ct_pKE \leftrightarrow \text{Encrypt}(ek_pKE, label, m_pKE; \rho)$, a labelled PKE scheme is correct if
  $$\Pr[\text{Decrypt}(dk_pKE, label, ct_pKE) = m_pKE] = 1 - \text{negl}(\lambda).$$

- **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 $L$ over a domain $X$ that contains two keyed algorithms, namely Hash and 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 $\chi \in L$, the outputs of both algorithms are indistinguishable, while for a word $\chi \notin L$, 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 $(X_{\text{par}, ltrap}, ltrap)$ indexed by some language parameter $lpar$ and some language trapdoor $ltrap$, together with a family of NP language $(X_{\text{par}}, lpar)$ indexed by some parameter $lpar$, with witness relation $R_{\text{par}},$ such that

$$\mathcal{L}_{\text{par}} := \{\chi \in X_{\text{par}} \mid \exists \omega, R_{\text{par}}(\chi, \omega) = 1\} \subseteq \mathcal{L}_{\text{par}, ltrap} \subseteq X_{\text{par}}.$$ 

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

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

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

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

- **Approximate correctness**: For a word $\chi \in \mathcal{L}_{\text{par}}$ and its corresponding witness $\omega$, we say SPHF is $\epsilon$-approximate correct if
  $$\Pr[\text{HD}(\text{Hash}(hk, lpar, \chi), \text{ProjHash}(hp, lpar, \chi, \omega)) > \epsilon] \leq \text{negl}(\lambda),$$
  where 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 $\chi \notin \mathcal{L}_{\text{par}}$, 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:
A PAEKS scheme should satisfy ciphertext 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:

- **Setup**(1\(^3\)): Taking as input a security parameter \( \lambda \), this PPT algorithm outputs a public parameter pp.
- **KeyGen\(_S\)(pp)**: Taking as input the public parameter pp, this PPT algorithm outputs a pair of public/private keys (pk\(_S\), sk\(_S\)) of the sender.
- **KeyGen\(_R\)(pp)**: Taking as input the public parameter pp, this PPT algorithm outputs a pair of public/private keys (pk\(_R\), sk\(_R\)) of the receiver.
- **PAEKS**(pp, pk\(_S\), sk\(_S\), pk\(_R\), kw\(_0\)): 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\(_0\), this PPT algorithm outputs a searchable ciphertext ct\(_{kw_0}\) related to the keyword kw\(_0\).
- **Trapdoor**(pp, pk\(_S\), pk\(_R\), sk\(_S\), sk\(_R\), kw\(_0\)): Taking as input the public parameter pp, the public key pk\(_S\), private key sk\(_R\) of the sender, private key sk\(_S\) of the receiver, and a keyword kw\(_0\), this PPT/deterministic algorithm outputs a trapdoor td\(_{kw_0}\) related to the keyword kw\(_0\).
- **Test**(pp, ct\(_{kw_0}\), td\(_{kw_0}\)): Taking as input the public parameter pp, searchable ciphertext ct\(_{kw_0}\), and trapdoor td\(_{kw_0}\), this algorithm outputs 1 if ct\(_{kw_0}\) and td\(_{kw_0}\) are related to the same keyword (i.e., kw = kw\(_0\)); otherwise, it outputs 0.

**Correctness.** For any security parameter \( \lambda \), any honestly generated key pairs of the sender (pk\(_S\), sk\(_S\)) and receiver (pk\(_R\), sk\(_R\)), any keywords kw, kw\(_0\), any ciphertext ct\(_{kw_0}\) \( \leftarrow \) PAEKS(pp, pk\(_S\), sk\(_S\), pk\(_R\), kw\(_0\)), and any trapdoor td\(_{kw_0}\) \( \leftarrow \) Trapdoor(pp, pk\(_S\), pk\(_R\), sk\(_R\), kw\(_0\)), then we have

\[
\begin{align*}
\Pr[\text{Test}(pp, ct_{kw_0}, td_{kw_0}) = 1] &= 1 - \text{negl}(\lambda) \text{ when } kw = kw_0; \\
\Pr[\text{Test}(pp, ct_{kw_0}, td_{kw_0}) = 0] &= 1 - \text{negl}(\lambda) \text{ when } kw \neq kw_0.
\end{align*}
\]

3.2 Security Requirements of PAEKS

A secure PAEKS scheme should satisfy ciphertext indistinguishable (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 IKG\(_A\) (IND-IKG\(_A\)) game, respectively, interacted with a challenger \( \mathcal{A} \) and an adversary \( \mathcal{A} \).

**IND-CKA Game of PAEKS:**

- **Setup.** After receiving a security parameter \( \lambda \), \( \mathcal{A} \) 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 (pk\(_S\), sk\(_S\)) and (pk\(_R\), sk\(_R\)) of the sender and the receiver, respectively. Finally, it sends (pp, pk\(_S\), 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, \( \mathcal{A} \) generates a searchable ciphertext ct\(_{kw_0}\) by performing PAEKS(pp, pk\(_S\), sk\(_S\), pk\(_R\), kw\(_0\)) and returns ct\(_{kw_0}\) to \( \mathcal{A} \).
  - **Trapdoor Oracle** \( O_T \): For any keyword kw, \( \mathcal{A} \) generates a trapdoor td\(_{kw_0}\) by performing Trapdoor(pp, pk\(_S\), pk\(_R\), sk\(_R\), kw\(_0\)) and returns td\(_{kw_0}\) to \( \mathcal{A} \).
- **Challenge.** After \( \mathcal{A} \) terminates Phase 1, it outputs two challenge keywords kw\(_0\), kw\(_1\) to \( \mathcal{C} \). The restriction is that \( \mathcal{A} \) never issues the queries to \( O_C \) and \( O_T \) for these two challenge keywords. \( \mathcal{C} \) then randomly chooses a bit \( b \in \{0,1\} \) and returns the challenge ciphertext ct\(_b\) to \( \mathcal{A} \) by performing PAEKS(pp, pk\(_S\), sk\(_S\), 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

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

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

**IND-IKG\(_A\) Game of PAEKS:**

- **Setup.** Like the IND-CKA game, \( \mathcal{C} \) generates the public parameter pp and public/private key pairs (pk\(_S\), sk\(_S\)) and (pk\(_R\), sk\(_R\)) of the sender and the receiver. Then, it sends (pp, pk\(_S\), 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 \( \mathcal{C} \). The restriction is that \( \mathcal{A} \) never issues the queries to \( O_C \) and \( O_T \) for these two challenge keywords. \( \mathcal{C} \) then randomly chooses a bit \( b \in \{0,1\} \)
and returns the challenge trapdoor td∗ to A by performing Trapdoor(pp, pkS, pkR, skg, kw∗).

- **Phase 2.** A can continue to query the oracles as in **Phase 1** for any keyword kw, except for the challenge keywords (i.e., kw /∈ {kw∗ i,j}).

- **Guess.** Finally, A outputs a bit b′ ∈ {0, 1} as its answer and wins the game if b = b′.

The advantage of A winning the above game is defined as

\[ \text{Adv}^\text{TP-PAEKS}_A(\lambda) = \Pr[b = b'] - \frac{1}{2}. \]

**Definition 3.2 (Trapdoor Privacy of PAEKS).** A PAEKS scheme is called TP (or IND-IKGA secure) if, for any PPT adversary A, \( \text{Adv}^\text{TP-PAEKS}_A(\lambda) \) 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. For the security games of MCI and privacy (MTP), 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. Here, we introduce Theorem 3.3 to show that if the PAEKS and Trapdoor algorithms of a secure PAEKS scheme are probabilistic, then this scheme satisfies MCI and MTP.

**Theorem 3.3.** Suppose that a PAEKS scheme satisfies CI and its PAEKS algorithm is probabilistic, then the PAEKS scheme satisfies MCI. Similarly, suppose that a PAEKS scheme satisfies TP and its Trapdoor algorithm is probabilistic, then the PAEKS scheme satisfies MTP.

**Proof.** As the part of TP is similar to CI, we only prove the part of CI. Suppose that an adversary A can break the MCI of a PAEKS scheme, then there is a challenger C who can use A as the black box algorithm to break the CI of the same PAEKS scheme.

- **Setup.** Given a tuple of public information (pp, pkS, pkR), C passes this information to A.

- **Phase 1.** On receiving any ciphertext query or trapdoor query for a keyword kw from A, C queries Ot for the ciphertext query and queries Ot for trapdoor query. Then, it returns the answer to A.

- **Challenge.** After receiving two tuples of challenge keywords (kw∗ i,1, · · · , kw∗ i,n) and (kw∗ j,1, · · · , kw∗ j,m), C performs the following steps. It randomly chooses a tuple (kw∗ i,l, kw∗ j,l) for some i such that kw∗ i,l + kw∗ j,l. Then, it takes this tuple as its challenge keyword and receives a challenge ciphertext ct∗. In addition, it randomly chooses n − 1 elements (r1, · · · , ri−1, ri+1, · · · , rn) from the output space of the PAEKS algorithm. Finally, it returns (r1, · · · , ri−1, ct∗, ri+1, · · · , rn) as the challenge ciphertext for A.

- **Phase 2.** A can continue to query the oracles as in **Phase 1** for any keyword kw, except for the challenge keywords (i.e., kw + kw∗ i,j) for i ∈ {0, 1} and j ∈ {1, n}.

- **Guess.** A finally outputs a bit b′, then C takes A’s answer as its answer.

As ct∗ is C’s challenge ciphertext and the PAEKS scheme satisfies CI, the PAEKS algorithm is probabilistic, for the view of A, (r1, · · · , ri−1, ct∗, ri+1, · · · , rn) is the same as the n truly ciphertext. Therefore, suppose A’s answer is right, then C can use A’s answer to break the CI of the PAEKS scheme.

The proof for TP is the similar to that for CI, except for the challenge part. More concretely, in the part of TP, C is given a challenge trapdoor td∗, instead of a challenge ciphertext ct∗. In addition, C returns (r1, · · · , ri−1, td∗, ri+1, · · · , rn) to A, where (r1, · · · , ri−1, ri+1, · · · , rn) are randomly chosen from the output space of the Trapdoor algorithm. Due to the PAEKS scheme satisfies TP, for the view of A, (r1, · · · , ri−1, td∗, ri+1, · · · , rn) is the same as the n truly trapdoors. Based on the above description, with the answer of A, C can also take A’s answer as its answer. Therefore, the proof is completed.

**4 CRYPTOANALYSIS 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 A can retrieve the keyword information hidden in the trapdoor. Suppose that A has received a trapdoor td j := tkw∥j related to some time j and keyword kw. It tries to obtain the keyword information via the following steps:

1. (Since tkw∥j is generated from the receiver by performing SamplePre(Akw∥jβ−1, tkw∥j, µ, σ), we know that µ = Akw∥jβ−1 · tkw∥j = H2(kw∥jβ −1 · tkw∥j).

2. Then, A randomly selects a guessed keyword kw′ to test whether µ = Akw∥j · H2(kw∥jβ −1 · tkw∥j).

3. If the equation in Step 2 holds, A outputs kw′ as its guess; otherwise, 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 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 IKG.

Proof. Let idp and idr be two identities of the proxy and the receiver, respectively. 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 \( td \coloneqq d_{kw} \) related to some keyword \( kw \). It tries to obtain the keyword information via the following steps:

1. Since \( d_{kw} \) is generated from the receiver by performing \( \text{SamplePre}(A_{idp}^{-1} \cdot D_{idw} \cdot v, \delta) \), we know that \( v = A_{idp}^{-1} \cdot d_{kw} \), where \( y = H_{I}(idp \|| idr \|| kw) \).
2. Then, \( \mathcal{A} \) randomly selects a guessed keyword \( kw' \) and computes \( y' = H_{I}(idp \|| idr \|| kw) \).
3. \( \mathcal{A} \) tests whether \( y = A_{idp}^{-1} \cdot d_{kw} \).
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. \( \square \)

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

Language of Ciphertext. Let \( (\text{lpar}, \text{ltrap}) = (ek_{\text{PKE}}, dk_{\text{PKE}}) \), where \( ek_{\text{PKE}} \in \mathcal{PS}_{\text{PKE}} \) and \( dk_{\text{PKE}} \) is its corresponding decryption key. We define the language of ciphertext as \( \mathcal{Z} = \{ (\text{label}, \text{ltrap}, \text{mpk}) \} \). For \( \mathcal{Z} \) to be a PEKS, the algorithm runs the following steps:

1. Generates \( (ek_{\text{PKE}}, dk_{\text{PKE}}) \); \( \text{KeyGen}(\lambda) \), this algorithm runs the following steps:
   - Checks whether \( \text{mpk} \|^{} H_{2}(ek_{\text{PKE}}, \text{mpk}, \text{label}) \).
   - \( \text{mpk} \|^{} H_{2}(ek_{\text{PKE}}, \text{mpk}, \text{label}) \).
   - Generates \( \text{ct}_{\text{PKE},\text{S}} \leftarrow \text{PEKS}.\text{Encrypt}(\text{mpk}, \text{label}, \text{mpk}, \rho) \).
   - Outputs the public key \( pk_{S} \leftarrow (\rho_{S}, \text{ct}_{\text{PKE},S}) \) and private key \( sk_{S} \leftarrow (\text{h}_{S}, \rho_{S}) \) of the sender.

2. PAEKSs(pp, pk_{S}, sk_{S}, 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 \( kw' \in \mathcal{KS}_{\text{PEKS}} \), this algorithm runs the following steps:
   - Checks whether \( \text{mpk} \|^{} H_{2}(ek_{\text{PKE}}, \text{mpk}, \text{label}) \).
   - \( \text{mpk} \|^{} H_{2}(ek_{\text{PKE}}, \text{mpk}, \text{label}) \).
   - Generates \( \text{ct}_{\text{PKE},\text{S}} \leftarrow \text{PEKS}.\text{Encrypt}(\text{mpk}, \text{label}, \text{mpk}, \rho_{R}) \).
   - Outputs the public key \( pk_{R} \leftarrow (\rho_{R}, \text{ct}_{\text{PKE},S}, \rho_{R}) \) and private key \( sk_{R} \leftarrow (\text{h}_{R}, \rho_{R}) \) of the receiver.

3. Trapdoor(pp, pk_{R}, sk_{R}, sk_{R}, kw'): Given the public parameter pp, the public key pk_{R} of the sender, and the private key sk_{R} of the receiver, and a keyword \( kw' \in \mathcal{KS}_{\text{PEKS}} \), this algorithm runs the following steps:
   - \( \text{mpk} \|^{} H_{2}(ek_{\text{PKE}}, \text{mpk}, \text{label}) \).
   - Generates \( \text{ct}_{\text{PKE},\text{S}} \leftarrow \text{PEKS}.\text{Encrypt}(\text{mpk}, \text{label}, \text{mpk}, \rho_{R}) \).
   - Outputs the searchable ciphertext \( ct_{kw'} \leftarrow \text{ct}_{\text{PKE},\text{S},\text{der}-kw'} \).

Correctness. Suppose that the public parameter pp and the public/private key pairs \( (pk_{S}, sk_{S}) \), \( (pk_{R}, sk_{R}) \) are honestly generated. Let \( ct_{kw} \) be the searchable ciphertext related with the keyword \( kw \) generated by the sender, and \( td_{kw} \) be the trapdoor related with the keyword \( kw' \) generated by the receiver.
As the underlying SPHF is $\epsilon$-correct for some $\epsilon = \text{negl}(\lambda)$, it follows that

\[ H_S = \text{SPHF.Hash}(hk_S, mpk, (ct_{PKE,R}, mpke)) \]
\[ = \text{SPHF.ProjHash}(hp_S, mpk, (ct_{PKE,R}, mpke), \rho_R) \]
\[ = pH_R; \]
\[ H_R = \text{SPHF.Hash}(hk_R, mpk, (ct_{PKE,R}, mpke)) \]
\[ = \text{SPHF.ProjHash}(hp_R, mpk, (ct_{PKE,R}, mpke), \rho_S) \]
\[ = pH_S. \]

Therefore, $H_S \oplus pH_S = H_R \oplus pH_R$ holds. Clearly, if $kw = kw'$, then der-$kw_S = H_2(kw, H_S \oplus pH_S) = H_2(kw', H_R \oplus pH_R)$, and therefore, $ct_{PKE,der-kws}$ and $td_{PKE,der-kw'_{R}}$ are related to the same extended keyword. As the underlying PEKS scheme is correct, $\text{PAEKS.Test}(pp, ct_{kw}, td_{kw'}) = 1$ holds with overwhelming probability. In contrast, since $H_2$ is modeled as a random oracle, if $kw \neq kw'$, then der-$kw_S = H_2(kw, H_S \oplus pH_S) \neq H_2(kw', H_R \oplus pH_R) = \text{der-kw'_{R}}$, and therefore, $ct_{PKE,der-kws}$ and $td_{PKE,der-kw'_{R}}$ are related to different extended keywords. Consequently, $\text{PAEKS.Test}(pp, ct_{kw}, td_{kw'}) = 0$ 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 attack game and $A$ can only distinguish these games with a negligible advantage. For simplicity, let $Adv_A^{Game_i}(\lambda)$ denote the advantage of $A$ in game $Game_i$, where $i \in \{0, 1, 2, 3\}$. Furthermore, by Theorem 5.3, we also show that the proposed construction satisfies MCI and MTP.

**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:

**Game0.** This game is identical to the real IND-CKA game defined in Section 3.2. Suppose that the advantage of $A$ in this game is defined as $Adv_A^{Game_0}(\lambda) := \epsilon$. In addition, to simulate a real view for $A$, on receiving the query for some keyword $kw$ from $A$, the challenger $C$ responds as follows:

- $O_C$: For keyword $kw$, $C$ computes $ct_{kw} \leftarrow \text{PEKS}(pp, pk_S, sk_S, pk_R, kw)$ and returns $ct_{kw}$ to $A$.
- $O_T$: For keyword $kw$, $C$ computes $td_{kw} \leftarrow \text{Trapdoor}(pp, pk_S, pk_R, sk_R, kw)$ and returns $td_{kw}$ to $A$.

**Game1:** This game is identical to $Game_0$, except for the generation of the challenge ciphertext $ct^*$ in the Challenge phase. More concretely, instead of generating $H_S \leftarrow \text{SPHF.Hash}(hk_S, mpk, (ct_{PKE,R}, mpke))$, $C$ randomly chooses $H_S$ from the output space of the SPHF.Hash algorithm. Since the underlying SPHF scheme satisfies pseudo-randomness, $A$ cannot distinguish the view between $Game_0$ and $Game_1$. Therefore, we obtain

\[ \left| Adv_A^{Game_0}(\lambda) - Adv_A^{Game_1}(\lambda) \right| \leq \text{negl}(\lambda). \]

**Game2:** This game further changes the generation of the challenge ciphertext $ct^*$ in the Challenge phase. In this game, der-$kw_S$ is randomly chosen from $\mathcal{KS}_{PEKS}$, instead of by computing der-$kw_S \leftarrow H_2(kw^*, H_S \oplus pH_S)$ for some $b \in \{0, 1\}$. As $H_S$ is randomly chosen and $H_2$ is modeled as a random oracle, the output of $H_2(kw^*, H_S \oplus pH_S)$ is random. Therefore, $A$ cannot distinguish the view between $Game_1$ and $Game_2$. Consequently, we obtain

\[ \left| Adv_A^{Game_1}(\lambda) - Adv_A^{Game_2}(\lambda) \right| \leq \text{negl}(\lambda). \]

Finally, combining the above games, we have $\epsilon \leq \text{negl}(\lambda)$. 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.

**Game0:** This game is identical to the real IND-IAKGA game defined in Section 3.2. Suppose that the advantage of $A$ in this game is defined as $Adv_A^{Game_0}(\lambda) := \epsilon$. In addition, the view simulated by the challenger $C$ is the same as that in $Game_0$ in the proof of Theorem 5.1.

**Game1:** This game is identical to $Game_0$, except for the generation of the challenge ciphertext $ct^*$ in the Challenge phase. More concretely, instead of generating $H_R \leftarrow \text{SPHF.Hash}(hk_R, mpk, (ct_{PKE,S}, mpke))$, $C$ randomly chooses $H_R$ from the output space of the SPHF.Hash algorithm. Since the underlying SPHF scheme satisfies pseudo-randomness, $A$ cannot distinguish the view between $Game_0$ and $Game_1$. Therefore, we obtain

\[ \left| Adv_A^{Game_0}(\lambda) - Adv_A^{Game_1}(\lambda) \right| \leq \text{negl}(\lambda). \]

**Game2:** This game further changes the generation of the challenge ciphertext $ct^*$ in the Challenge phase. In this game, der-$kw_R$ is randomly chosen from $\mathcal{KS}_{PEKS}$, instead of by computing der-$kw_R \leftarrow H_2(kw^*, H_R \oplus pH_R)$ for some $b \in \{0, 1\}$. As $H_R$ is randomly chosen, the output of $H_2(kw^*, H_R \oplus pH_R)$ is random. Therefore, $A$ cannot distinguish the view between $Game_1$ and $Game_2$. Consequently, we obtain

\[ \left| Adv_A^{Game_1}(\lambda) - Adv_A^{Game_2}(\lambda) \right| \leq \text{negl}(\lambda). \]
Games: This game is the last game. Because the challenge trapdoor $t^d = t^{dp_{PEKS,der-kw}}$ is generated from PEKS, $PEKS(pk_{PEKS,der-kw})$ and $der-kw$ is now randomly chosen from $KS_{PEKS}$, the challenge trapdoor does not contain any information about the challenge keywords $(kw_0^t, kw_1^t)$ given by $A$. The only way for $A$ to guess. Therefore, we have

$$\text{Adv}^{\text{Game3}}_{A}(\lambda) = 0.$$ 

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

Theorem 5.3. The proposed generic PAEKS construction further satisfies MCI and MTP if Theorem 3.3, Theorem 5.1 as well as Theorem 5.2 holds, and PEKS as well as Trapdoor algorithms of the underlying PEKS scheme are probabilistic.

Proof. In the proposed construction, the PAEKS and Trapdoor algorithms actually perform the PEKS and Trapdoor algorithms of the underlying PEKS scheme. To be the best of our knowledge, for the current well-known PEKS schemes (e.g., [6, 7, 28]), the PEKS and Trapdoor algorithms are probabilistic. Hence, by combining the result of Theorem 5.1, Theorem 5.2, and Theorem 3.3, the proposed construction satisfies MCI and MTP. □

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 Belenki 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 consider the weaker version (IND-CCA1) of the PKE scheme [43] in the following instantiation.

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

The whole instantiation is described as follows:

- Setup$(1^\lambda)$: Given a security parameter $\lambda$ and the parameters $q, n, m, s_1, s_2, a$ (set as instructed in the following parameter selection part), this algorithm runs the following steps:
  - Set $\ell := poly(n)$ and randomly chooses $m = m_1 m_2 \cdots m_k \in \{0, 1\}^k$.
  - Computes $(A_0, T) \leftarrow \text{TrapGen}(1^n, 1^m, q)$.
  - Sets $ek_{PKE} := A_0, dp_{PKE} := T$, and $mp_{PKE} := m$.
  - Randomly chooses element $u \in \mathcal{U}$ and sets label := $u$
- Choose two secure hash functions $H_1 : \mathbb{Z}_q^{\text{m} \cdot \text{c}} \times \{0, 1\}^\ell \rightarrow \mathbb{Z}_q^\text{m}$, $H_2 : \{1, -1\}^\ell \times \{0, 1\}^\ell \xrightarrow{} \{1, -1\}^\ell$, and an injective ring homomorphism $h : R \rightarrow \mathbb{Z}_q^n$.
- Computes $A \leftarrow H_1(A_0, m, u) \in \mathbb{Z}_q^{\text{m} \cdot \text{c}}$ and sets $mpk := A$.
- Outputs $pp := (A, n, m, q, s_1, s_2, a, p, t, ek_{PKE}) := A_0, mpk := m, label := u, H_1, H_2, h$.
- KeyGen$(p)$: Given the public parameter $pp$, this algorithm runs the following steps:
  - Checks whether $A \approx H_1(A_0, m, u)$.
  - Computes $A_u := A \oplus \{0; Gh(u)\}$, randomly chooses a matrix $hk := k_s \leftarrow D_{\mathbb{Z}_q}^m$, and computes $hp := ps = A_h \cdot k_s \in \mathbb{Z}_q^m$, where $s \geq 3\epsilon(A^2(A_u))$ for some $\epsilon = \text{neg}(n)$.
  - For $i = 1, \ldots, k$, randomly chooses vectors $s_{R,i} \leftarrow \mathbb{Z}_q^n$ as well as $e_{S,i} \leftarrow D_{\mathbb{Z}_q}^m$ (re-select $e_{S,i}$ if $\|e_{S,i}\| > 2\sqrt{m}$), and computes $e_{R,i} = A^i_* s_{R,i} + e_{S,i} + \text{Encode}(m)$ mod $q$, where $t = \sigma_1 \sqrt{m} \cdot \omega(\sqrt{\log n})$.
  - Outputs the public key $pk := (hp := ps, cd_{PKE} := \{s_{R,i}\}_{i=1}^k)$ and the private key $sk := (hk := k_s, ps) \leftarrow \{s_{S,i}\}_{i=1}^k$ of the sender.
- KeyGen$(p)$: Given the public parameter $pp$, this algorithm runs as follows:
  - Checks whether $A \approx H_1(A_0, m, u)$.
  - Computes $A_u := A \oplus \{0; Gh(u)\}$, randomly chooses a matrix $hk_r := k_r \leftarrow D_{\mathbb{Z}_q}^m$, and computes $hp := ps := A_h \cdot k_r \in \mathbb{Z}_q^m$, where $s \geq 3\epsilon(A^2(A_u))$ for some $\epsilon = \text{neg}(n)$.
  - For $i = 1, \ldots, k$, randomly chooses vectors $s_{R,i} \leftarrow \mathbb{Z}_q^n$ as well as $e_{R,i} \leftarrow D_{\mathbb{Z}_q}^m$ (re-select $e_{R,i}$ if $\|e_{R,i}\| > 2\sqrt{m}$), and computes $e_{R,i} = A^i_* s_{R,i} + e_{R,i} + \text{Encode}(m)$ mod $q$, where $t = \sigma_1 \sqrt{m} \cdot \omega(\sqrt{\log n})$.
  - Generates $(B_{R,S}, S_{G}) \leftarrow \text{TrapGen}(1^n, 1^m, q)$.
  - Selects $t + 1$ random matrices $B_{R_1}, \ldots, B_{R_t}, C_R \leftarrow \mathbb{Z}_q^{m \cdot n}$ and a random vector $r_{G} \leftarrow \mathbb{Z}_q^n$.
  - Outputs the public key $pk_R := (hp_S := ps, \text{cd}_{PEKS} := \{s_{R,i}\}_{i=1}^t, pk_{PEKS} := \{B_R, B_{R_1}, \ldots, B_{R_t}, C_R, r_{G}\})$ and the private key $sk_R := (hk_r := k_r, ps) \leftarrow \{s_{S,i}\}_{i=1}^k, sk_{PEKS} := S_{G}$ of the receiver.
- PAEKS$(pp, pk_{S}, sk_{S}, pk_{R}, kw)$: 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 $i = 1, \ldots, k$, computes $h_{S,i} := R(R_{k_r} \cdot k_s \mod{mod})$, and $y_{S,i} := h_{S,i} \cdot p_{i}$
  - Sets $y_S := Y_{S,1} Y_{S,2} \ldots Y_{S,k} \in \{0, 1\}^k$.
  - Computes $der-kws := dk_{S} := dk_{S,1} dk_{S,2} \ldots dk_{S,t}$.
  - Computes $B_{dk} := C_R + \sum_{i=1}^t dk_{S,i} B_{R,i}$ and $F_{dk} := \{B_R, B_{dk}\} \in \mathbb{Z}_q^{n \cdot m - \text{c}}$.
  - For $j = 1, \ldots, \rho$, performs the following steps:
    - Chooses $b_j \leftarrow \{0, 1\}$, a random $s_j \leftarrow \mathbb{Z}_q^n$, and matrices $R_{ij} \leftarrow \{1, -1\}^\text{m \cdot c}$ for $i = 1, \ldots, \ell$. 

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

- If \( kw = kw' \), the sender and the receiver obtain the same derived keyword (i.e., der-kw = der-kw').
- If \( ct_{ksw} \) and \( td_{ksw} \) 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 \( dk_S = dk_R \).

**Lemma 6.1.** Suppose the norm of the first error term \( (e_{R,i}^T \cdot k_{S,i} \) and \( e_{S,i}^T \cdot k_{R,i} \) is less than \( \epsilon/2 \cdot q/4 \) and \( kw = kw' \), then \( dk_S = dk_R \).

**Proof.** For \( i = 1, \ldots, \kappa \), we have

\[
\begin{align*}
  h_{S,i} &= R(e_{R,i}^T \cdot k_{S,i}) \pmod{q} \\
  &= R(s_{R,i}^T \cdot A_u) \cdot k_{S,i} + e_{R,i}^T \cdot k_{S,i} \pmod{q} \\
  &= \text{first error term} \\
  &= p_{R,i}; \\

  h_{R,i} &= R(e_{S,i}^T \cdot k_{R,i}) \pmod{q} \\
  &= R(s_{S,i}^T \cdot A_u) \cdot k_{R,i} + e_{S,i}^T \cdot k_{R,i} \pmod{q} \\
  &= \text{first error term} \\
  &= p_{S,i}.
\end{align*}
\]

Since \( y_{S,i} = h_{S,i} \cdot p_{R,i} = h_{R,i} \cdot p_{S,i} \) for \( i = 1, \ldots, \kappa \), we have
\( Y_S = Y_R \). Furthermore, as \( Y_S = Y_R \) and \( kw = kw' \), we have
\( \text{der-kw} = dk_S = H_2(kw, Y_S) = H_2(kw', Y_R) = dk_R = \text{der-kw}' \).

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

\[
\begin{align*}
  v_j &= c_{ij} - t_{dk}e_{ij} \\
  &= r_j^T s_j + b_j[q/2] - t_{dk}(F_j^T k_{ij} + [y_j; z_j]) \\
  &= b_j[q/2] + j - t_{dk}[y_j; z_j].
\end{align*}
\]

Therefore, the described algorithms in Behnia et al’s PEKS scheme [6] are probabilistic. On the basis of Theorem 5.1, Theorem 5.2, and Theorem 5.3, we obtain the following theorem:

**Theorem 5.6.** The proposed lattice-based PAEKS scheme satisfies MCI and MTP.

### 7. Comparison

In this section, we present a comparison of our lattice-based instantiation with other PEKS/PAEKS schemes (i.e., BOC’04 [7], HI17 [27], ZTW’19 [62], QCH’20 [49], BOY20 [6], ZWX’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 C, MCI,
Table 1: Comparison of security properties with those of PAEKS schemes

<table>
<thead>
<tr>
<th>Schemes</th>
<th>CI</th>
<th>MCI</th>
<th>TP</th>
<th>MTP</th>
<th>QR</th>
<th>NTA</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOC’04 [7]</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>HL17 [27]</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ZTW’19 [62]</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>QCH’20 [49]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BOY20 [6]</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ZXW’21 [63]</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LTT’21 [39]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Ours</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

✓: The scheme supports the corresponding feature; x: 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

<table>
<thead>
<tr>
<th>Schemes</th>
<th>Ciphertext Generation</th>
<th>Trapdoor Generation</th>
<th>Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZTW’19 [62]</td>
<td>$2T_H + (\rho n + \rho m)^2 + \rho n m + \rho)T_M + T_{SP}$</td>
<td>$T_H + nm^2T_M + T_{BD} + T_{SP}$</td>
<td>$T_H + (tm + nm)T_M$</td>
</tr>
<tr>
<td>BOY20 [6]</td>
<td>$\rho (m^2 + 2nm + n + \ell + 1)T_M$</td>
<td>$T_{M} + T_{SL}$</td>
<td>$2pmT_M$</td>
</tr>
<tr>
<td>ZXW’21 [63]</td>
<td>$T_H + (\rho n + \rho m)^2 + \rho n m + \rho)T_M + T_{SP}$</td>
<td>$T_H + nm^2T_M + T_{BD} + T_{SP}$</td>
<td>$T_H + (tm + nm)T_M$</td>
</tr>
<tr>
<td>Ours</td>
<td>$T_H + (\kappa (m + n + 1) + \rho (m^2 + 2nm + n + \ell + 1))T_M$</td>
<td>$T_H + (\kappa (m + n + 1) + \ell)T_M + T_{SL}$</td>
<td>$2pmT_M$</td>
</tr>
</tbody>
</table>

$\kappa, \rho$: The parameters related to security parameter $\lambda$; $\ell$: 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.

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

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

<table>
<thead>
<tr>
<th>Schemes</th>
<th>Ciphertext</th>
<th>Trapdoor</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZTW’19 [62]</td>
<td>$(\ell + m\ell + m)</td>
<td>q</td>
</tr>
<tr>
<td>BOY20 [6]</td>
<td>$\kappa(</td>
<td>q</td>
</tr>
<tr>
<td>ZXW’21 [63]</td>
<td>$(\ell + m\ell + m)</td>
<td>q</td>
</tr>
<tr>
<td>Ours</td>
<td>$\kappa(</td>
<td>q</td>
</tr>
</tbody>
</table>

$n$: The parameter related to security parameter; $m$: Dimension; $q$: Modules; $\kappa$: The parameter related to security parameter; $\ell$: The length of the keyword.

TP, MTP, 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 TP and MTP 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 2.
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 Intel(R) Core(TM) i7-9700 CPU with 3.00 GHz and 32GB of system memory. To achieve the 80-bit security level, we set the parameters with $n = 256$, $m = 9753$, $g = 4096$, $r = 10$, $k = 10$, $\ell = 10$, $\sigma_1 = 8$, $\sigma_2 = 8$, where $r, k$ are the parameters related to the security parameter (i.e., $\kappa, \rho \leftarrow \text{poly}(\lambda)$) and $\ell$ 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 security (e.g. TP and MTP), 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 notions—CI and TP. In addition, based on our theoretical result (Theorem 3.3), we demonstrated that the proposed construction further satisfied MCI and MTP if the PEKS algorithm and Trapdoor algorithms 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 MTP. 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

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.

• Setup(1)\(^\dagger\): Taking as input a security parameter \(\lambda\), this algorithm runs the following steps:

  1. Randomly selects \(\mu \leftarrow \mathbb{Z}_q^n\) and three secure hash functions \(H_1: \mathbb{Z}^{nxm} \times \{0, \ldots, \eta\} \rightarrow \mathbb{Z}^{mxm}, H_2: \{0, 1\}^{\ell} \rightarrow \mathbb{Z}^{nq}, H_3: \mathbb{Z}^{nxm} \times \{0, 1\}^{\ell} \rightarrow \mathbb{Z}^n\).

  2. Respectively generates \((A_{\mu, 0}, T_{S, 0})\) and \((A_{\mu, 0}, T_{R, 0})\) by performing TrapGen\((\ell, 1^n, q)\).

  3. Outputs the public parameter \(pp = (\mu, H_1, H_2, H_3)\), the public/private key pair of the sender \((pk_{R, 0}, sk_{R, 0}) = (T_{R, 0})\) for time period 0.

• KeyUpdate\((pk_{R, i}, sk_{R, i}, j, i)\): Taking as input an input public/private key pair \((pk_{R, i}, sk_{R, i}) = (T_{R, i})\) of the receiver in the previous time period \(i\) and the current time period \(j\), this algorithm runs the following steps:

  1. Computes \(R_{S, i,j} = H_1(AR_{i,j}) + \cdots + H_1(AR_{j-1, i} + 1) \in \mathbb{Z}^{nq}\).

  2. Computes \(T_{R, i,j} \leftarrow \text{NewBase} \text{al} \text{Del}(AR_{i,j}, R_{R, i,j} - j, T_{R, i,j})\), where \(AR_{i,j} = AR_{i,j}(R_{R, i,j} - 1) = AR_{R, i,j} - 1 \in \mathbb{Z}^{nxm}\).

  3. Outputs the public/private key pair \((pk_{R, i,j}, sk_{R, i,j}) = T_{R, i,j}\) of the receiver for time period \(j\).

Note that the sender can use the same steps to generate his/her public/private key pair \((pk_{S, i,j}, sk_{S, i,j}) = T_{S, i,j}\) for time period \(j\).

• PEKS\((pk_{S, j}, sk_{S, j}, pk_{R, j}, k, w)\): Taking as input a public/private key pair \((pk_{S, j}, sk_{S, j}) = T_{S, j}\) of the sender for time period \(j\), the public key \(pk_{R, j} = AR_{R, j}\) of the receiver for time period \(j\), the current time period \(j\), and keyword \(w \in \{0, 1\}^t\), the sender runs the following steps:

  1. Chooses a random binary string \(y_j = y_{j, 1}y_{j, 2}\cdots y_{j,t} \in \{0, 1\}^t\), uniform matrix \(B_j \in \mathbb{Z}^{nxq}\), noise \(c_j = c_{j, 1}c_{j, 2}\cdots c_{j,t}\), and noise \(V_j = v_{j, 1}v_{j, 2}\cdots v_{j,t}\), where \(c_{j, 1}, \ldots, c_{j,t} \in \mathbb{Z}^{nxq}\) and \(V_j \in \mathbb{Z}^{nxq}\).

  2. Computes \(\beta_j \leftarrow H_2(kw_j)\), \(c_j = \mu \cdot B_j + c_j + \{y_{j, 1}y_{j, 2}\cdots y_{j,t}\} = (AR_{S, j}B_j)^{\ell, 1} + v_j\).

  3. Computes \(h_j = H_3(\epsilon_jy_j) \in \mathbb{Z}^{nq}\), and generates \(\xi_j \leftarrow \text{SamplePre}(AR_{S, j}, T_{S, j}, h_j, \sigma_j)\).

  4. Outputs a searchable ciphertext \(ct_j = (c_{j}, c_j, \xi_j)\).

• Trapdoor\((pk_{R, j}, sk_{R, j}, k, w)\): Taking as input a public/private key pair \((pk_{R, j}, sk_{R, j}) = T_{R, j}\) of the receiver for time period \(j\), current time period \(j\) and keyword \(w \in \{0, 1\}^t\), the receiver runs the following steps:
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.

• Setup($\lambda$): Taking as input a security parameter $\lambda$, the key generator center runs the following steps:
  - Generates $(A, T_A)$ ← TrapGen($1^n, 1^m, q$).
  - Selects a uniform random vector $v \leftarrow Z_q^n$ and five secure cryptographic hash functions: $H_1 : \{0, 1\}^\ell \rightarrow Z_q^{m\times n}$, $H_2 : \{0, 1\}^{\ell} \times \{0, 1\}^5 \times \{0, 1\}^5 \times Z_q^n \rightarrow Z_q^n$, $H_3 : \{0, 1\}^{\ell} \times \{0, 1\}^5 \times \{0, 1\}^5 \times Z_q^{m\times n}$, $H_4 : \{0, 1\}^{\ell} \times \{0, 1\}^5 \times \{0, 1\}^5 \times Z_q^{m\times n}$, and $H_5 : \{0, 1\}^{\ell} \times Z_q^{m\times n} \rightarrow Z_q^n$.
  - Outputs the public parameters $p = (A, v, H_1, H_2, H_3, H_4, H_5)$ and master private key $msk = T_A$.

• KeyExtract(msk, id): Taking as input the master secret key $msk = T_A$ and an identity $id \in \{0, 1\}^\ell$, the key generator center runs the following steps:
  - Computes $H_1(id) = A(id) = A(R(id))^{-1} \in Z_q^{m\times n}$.
  - Generates $T_id$ ← NewBasisDel$(A, R(id), T_A, \sigma)$.
  - Outputs the secret key $sk_{id} = T_id$ for identity $id$.

• Trapdoor($sk_{id}$, kw): Taking as input the private key $sk_{id} = T_id$ of the receiver id and a keyword $kw \in \{0, 1\}^\ell$, $id_{kw}$ runs the following steps:
  - Computes $Y = H_4(id_{kw}) = A(id_{kw})Y^{-1}F + S, \xi = v^T F + \eta + (\tau_1, \tau_2, \ldots, \tau_\ell)q/2$.
  - Computes $h = H_5(r(\xi))$ and $\theta = \text{SamplePre}(A_{id_{kw}}, T_{id_{kw}}, h, \delta) \in Z_q^n$.
  - Computes $d_{kw} = \text{SamplePre}(A_{id_{kw}}, Y^{-1} F, \eta, \nu, \delta) \in Z_q^m$.
  - Outputs a trapdoor $td_{kw}$.

• Test(pk$_{id}$, ct, td): Taking as input the proxy-oriented public key $pk_{id} = A_{id}$, a searchable ciphertext $ct = (\xi, \rho, \theta)$, and a trapdoor $td = d_{kw}$, the cloud server runs the following steps:
  - Computes $r = \tau_1, \tau_2, \ldots, \tau_\ell = \xi - \text{d}_{kw}^T F \xi \in Z_q^n$.
  - For $j = 1, \ldots, t$, if $|r_j - q/2| < q/4$, sets $r_j = 1$; otherwise, sets $r_j = 0$.
  - Updates $r$ and further computes $h ← H_5(r(\xi))$.
  - Checks whether the equation $A_{id} \theta \equiv h$ holds. If the equation holds, outputs 1; otherwise, outputs 0.