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 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 insider 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.

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. [8]. 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 [18,44,46,57]. However, in practical applications, SSE has the same problem as symmetric encryption—the key distribution problem. To resolve this problem, Boneh et al. [8] 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. [8] 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. [9] 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 [9], the keyword space in PEKS schemes is small and limited, e.g., only 225,000 ($\approx 2^{19}$) words in Merriam-Webster’s collegiate dictionary [11]. Consequently, upon a brute force attack, there is a high probability ($2^{19}$) that the adversary can obtain the keyword information hidden by the trapdoor.

Although many KGA-secure PEKS schemes have been introduced [5,13–15,20–22,30,31,52,53,58–60,62], 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 [12,26,36–38,40–42,45,47–49] 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 [16] 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. [63,64] introduced two lattice-based PEKS schemes that are secure against IKGA by restricting the ciphertext to be authenticated by the sender. However, their 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 [63,64] 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 a 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 IGKA 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 are 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 Benhamouda et al.’s SPHF [7] 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 \( B = [b_1|\cdots|b_n] \in \mathbb{R}^{m \times n} \), where \( b_1, \ldots, b_n \) are \( n \) linear independent vectors. An \( m \)-dimensional lattice \( \Lambda \) generated by \( B \) is defined as \( \Lambda(B) := \{ \sum_{i=1}^{n} a_i b_i : a_i \in \mathbb{Z} \} \). Here, \( B \) is called the basis of \( \Lambda \). In addition, given \( n, m, q \in \mathbb{Z} \), let \( \mathbb{Z}_q \) and \( \mathbb{Z}_q^m \) be two \( q \)-ary lattices and a coset as follows:

- \( \Lambda_q(A) := \{ y \in \mathbb{Z}_q^m : \exists z \in \mathbb{Z}_q^m, y = A^T z \ mod \ q \} \);
- \( \Lambda_q^0(A) := \{ e \in \mathbb{Z}_q^m : Ae = 0 \ mod \ q \} \);
- \( \Lambda_q^0(A) := \{ e \in \mathbb{Z}_q^m : Ae = u \ mod \ q \} \).

2.1.2 Discrete Gaussian Distributions. For any positive real number \( \sigma \), any center \( c \in \mathbb{Z}_q^m \), and any \( x \in \mathbb{Z}_q^m \), 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 \( \Lambda \in \mathbb{Z}_q^{m} \), we define \( \rho_{\sigma,c}(\Lambda) := \sum_{x \in \Lambda} \rho_{\sigma,c}(x) \). Then, the discrete Gaussian distribution over lattice \( \Lambda \) with parameter \( (\sigma, c) \) is defined as follows: For any \( x \in \Lambda \), \( \mathcal{D}_{\Lambda,\sigma,c}(x) := \rho_{\sigma,c}(x)/\rho_{\sigma,c}(\Lambda) \).

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

1. \( \text{TrapGen}(1^\lambda, 1^m, q) [4, 43] \): For input \( n, m, q \in \mathbb{Z} \), this probabilistic polynomial time (PPT) algorithm outputs a pair \( (A \in \mathbb{Z}_q^{m \times n}, T_A \in \mathbb{Z}_q^{2m \times n}) \), where \( T_A \) is a basis for \( \Lambda_q^0(A) \), such that the following property holds:

\[ \{ A : (A, T_A) := \text{TrapGen}(1^\lambda, 1^m, q) \} \approx \{ A : A \leftarrow \mathbb{Z}_q^{m \times n} \}. \]

Here, \( T_A \) is called a trapdoor of \( A \).

2. \( \text{SamplePre}(A, T_A, u, \sigma) [24] \): For an input matrix \( A \in \mathbb{Z}_q^{m \times n} \) and its trapdoor \( T_A \in \mathbb{Z}_q^{2m \times n} \), a vector \( u \in \mathbb{Z}_q^m \), and parameter \( \sigma \geq \| T_A \| \cdot \omega(\sqrt{\log(m + m)}) \), this PPT algorithm outputs a sample \( t \in \mathbb{Z}_q^m \) from a distribution that is statistically close to \( \mathcal{D}_{\Lambda_q^0(A),\sigma} \) such that \( At = u \mod q \).

3. \( \text{NewBaseDel}(A, R, T_A, \sigma) [3] \): For an input matrix \( A \in \mathbb{Z}_q^{m \times n} \) and its corresponding trapdoor \( T_A \in \mathbb{Z}_q^{2m \times n} \), a matrix \( M \in \mathbb{Z}_q^{m \times n} \), a vector \( u \in \mathbb{Z}_q^m \), and a parameter \( \sigma \geq \| T_A \| \cdot \omega(\sqrt{\log(m + m)}) \), this PPT algorithm outputs a sample \( t \in \mathbb{Z}_q^m \) from a distribution statistically close to \( \mathcal{D}_{\Lambda_q^0(A \cdot M),\sigma} \) such that \( [A \cdot M] \cdot t = \pi \mod q \).

2.2 Public-key Encryption with Keyword Search

In this subsection, we recall the definition of PEKS defined by Boneh et al. [8]. A PAEKS scheme PEKS consists of the following four algorithms:

- KeyGen(1^\lambda): Taking as input a security parameter \( \lambda \), this PPT algorithm outputs a pair of keys \( (pk_{\text{PEKS}}, sk_{\text{PEKS}}) \), where \( pk_{\text{PEKS}} \) is the public key and \( sk_{\text{PEKS}} \) is the private key.
- PEKS(pk_{\text{PEKS}}, kw): Taking as input the public key \( pk_{\text{PEKS}} \) and a keyword \( kw \), this PPT algorithm outputs a searchable ciphertext \( ct_{\text{PEKS}} \) related to the keyword \( kw \).
- Trapdoor(sk_{\text{PEKS}}, kw): Taking as input the private key \( sk_{\text{PEKS}} \) and a keyword \( kw \), this PPT algorithm outputs a trapdoor \( td_{\text{PEKS}} \) related to keyword \( kw \).
- Test(ct_{\text{PEKS}}, kw, td_{\text{PEKS}}): Taking as input the searchable ciphertext \( ct_{\text{PEKS}} \) and trapdoor \( td_{\text{PEKS}} \), this deterministic algorithm outputs 1 if \( ct_{\text{PEKS}} \) and \( td_{\text{PEKS}} \) are related to the same keyword (\( i.e., kw = kw' \)); otherwise, it outputs 0.

Correctness. For any security parameter \( \lambda \), any honestly generated key pairs \( (pk_{\text{PEKS}}, sk_{\text{PEKS}}) \), any keywords \( kw, kw' \), any ciphertext \( ct_{\text{PEKS}} \leftarrow \text{PEKS}(pk_{\text{PEKS}}, kw) \), and any trapdoor \( td_{\text{PEKS}} \leftarrow \text{Trapdoor}(sk_{\text{PEKS}}, kw) \), then we have

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

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 \( A \).

IND-CKA Game of PEKS:

- Setup. After receiving a security parameter \( \lambda \), \( C \) generates \( (pk_{\text{PEKS}}, sk_{\text{PEKS}}) \) by performing the KeyGen algorithm. Then, it sends the public key \( pk_{\text{PEKS}} \) to \( A \) and keeps the private key \( sk_{\text{PEKS}} \) secret.
A labelled public-key encryption (PKE) scheme can be viewed as a variant of PKE. As described in [1], a labelled PKE scheme consists of the following three algorithms:

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

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

- **Correctness**: For any security parameter \(\lambda\), any pair of keys \((dk\text{PKE}, ek\text{PKE}) \leftarrow \text{KeyGen}(\lambda)\), any label, any plaintext \(mp\text{PKE}\), any randomness \(\rho\), and any ciphertext \(ct\text{PKE}\), the following holds: \(\text{Decrypt}(ek\text{PKE}, label, mp\text{PKE}; \rho) = \text{KeyGen}(\lambda)\).
- **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.

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 [17] 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 [10, 19, 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 \(x \in L\), the outputs of both algorithms are indistinguishable, while for a word \(x \notin L\), 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 defines the languages and word-independent SPHF.

We first consider a family of languages \((\mathcal{L}_{\text{Ipar}}, \mathcal{L}_{\text{Itrap}}, \mathcal{L}_{\text{Itrap}})\) indexed by some parameter \(\text{Ipar}\) and some language \(\mathcal{L}_{\text{Itrap}}\) indexed by \(\text{Ipar}\), with witness relation \(\mathcal{R}_{\text{Ipar}}\), such that:

\[
\mathcal{L}_{\text{Ipar}} := \{x \in \mathcal{X}_{\text{Ipar}} \mid \exists\omega, \mathcal{R}_{\text{Ipar}}(x, \omega) = 1\} \subseteq \mathcal{L}_{\text{Ipar}}, \mathcal{L}_{\text{Itrap}} \subseteq \mathcal{X}_{\text{Ipar}},
\]

where \((\mathcal{X}_{\text{Ipar}}, \mathcal{L}_{\text{Ipar}})\) is a family of sets and the parameter \(\text{Ipar}\) is generated by a polynomial-time algorithm \(\text{Setup.Ipar}(\lambda)\) for some security parameter \(\lambda\). We suppose that the membership in \(\mathcal{X}_{\text{Ipar}}\) and \(\mathcal{L}_{\text{Ipar}}\) can be checked in polynomial time by the given \(\text{Ipar}\), and that the membership in \(\mathcal{R}_{\text{Ipar}}\) is by the given \(\text{Ipar}\) and \(\text{Itrap}\).

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

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

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

- **Approximate correctness**: For a word \(x \in \mathcal{X}_{\text{Ipar}}\) and its corresponding witness \(\omega\), we
say SPHF is $\epsilon$-approximate correctness if $\Pr[HHD(\text{Hash}(hk, lpar, \gamma), \text{ProjHash}(hp, lpar, \gamma, o)) > \epsilon \cdot \delta] \leq \negl(\lambda)$, where $HHD(\cdot, \cdot)$ 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 $\chi \notin \mathcal{D}_{lpar}$, the hash value $H$ is statistically indistinguishable from a random element chosen from $\{0, 1\}^d$ for some $d \in \mathbb{N}$.

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 $\chi \in \mathcal{D}_{lpar}$, the hash value $H$ is indistinguishable from a random element chosen from $\{0, 1\}^d$ for some $d \in \mathbb{N}$.

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 [26], can be viewed as inheriting the existed PEKS scheme [8] 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($\lambda^3$):** Taking as input a security parameter $\lambda$, this PPT algorithm outputs a parameter public pp.
- **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 sender.
- **KeyGen$_G$(pp):** Taking as input the public parameter pp, this PPT algorithm outputs a pair of public/private keys $(pk_G, sk_G)$ of the receiver.
- **PAEKS(pp, pk$_G$, sk$_G$, pk$_R$, sk$_R$, kw):** Taking as input the public parameter pp, the public key $pk_G$ and private key $sk_G$ 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 $kw$.
- **Trapdoor(pp, pk$_G$, pk$_R$, sk$_G$, kw$’$):** Taking as input the public parameter pp, the public key $pk_G$ of the sender, the public key $pk_R$ and private key $sk_R$ of the receiver, and a keyword $kw’$, this PPT/deterministic algorithm outputs a trapdoor $td_{kw’}$ related to the keyword $kw’$.
- **Test(pp, ct$_{kw}$, td$_{kw’}$):** Taking as input the public parameter pp, searchable ciphertext $ct_{kw}$, and trapdoor $td_{kw’}$, this algorithm outputs 1 if $ct_{kw}$ and $td_{kw’}$ are related to the same keyword (i.e., $kw = kw’$); otherwise, it outputs 0.

**Correctness.** For any security parameter $\lambda$, any honestly generated key pairs of the sender $(pk_R, sk_R)$ and receiver $(pk_G, sk_G)$, any keywords $kw, kw’$, any ciphertext $ct_{kw} \leftarrow \text{PAEKS}(pp, pk_G, sk_G, pk_R, kw)$, and any trapdoor $td_{kw’} \leftarrow \text{Trapdoor}(pp, pk_G, pk_R, sk_G, kw’)$, then we have

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

#### 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. [8], 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 et al. [9] 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 $A$.

**IND-CKA Game of PAEKS:**

- **Setup.** After receiving a security parameter $\lambda$, $C$ generates the public parameter pp by executing the Setup algorithm. Then, it executes the KeyGen$_R$ and KeyGen$_G$ algorithms to obtain the public/private key pairs $(pk_G, sk_G)$ and $(pk_R, sk_R)$ of the sender and the receiver, respectively. Finally, it sends $(pp, pk_G, pk_R)$ to $A$.
- **Phase 1.** In this phase, $A$ 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 PAEKS$(pp, pk_G, sk_G, pk_R, kw)$ and returns $ct_{kw}$ to $A$.
  - **Trapdoor Oracle $O_T$:** For any keyword $kw$, $C$ generates a trapdoor $td_{kw}$ by performing Trapdoor$(pp, pk_G, pk_R, sk_G, kw)$ and returns $td_{kw}$ to $A$.
- **Challenge.** After $A$ terminates Phase 1, it outputs two challenge keywords $kw_0, kw_1$ to $C$. The restriction is that $C$ 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 $ct^*_{kw}$ to $A$ by performing PAEKS$(pp, pk_G, sk_G, pk_R, kw^*_b)$.
- **Phase 2.** $A$ can continue to query the oracles as in Phase 1 for any keyword $kw$, except for the challenge keywords (i.e., $kw \notin \{kw_0, kw_1\}$).
- **Guess.** Finally, $A$ outputs a bit $b’ \in \{0, 1\}$ as its answer and wins the game if $b = b’$.

The advantage of $A$ winning the above game is defined as

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

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

**IND-IKGA 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 \( kw_i \) and \( kw_j \) 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 \( td_i \) to \( \mathcal{A} \) by performing Trapdoor \((pp, pk_S, pk_R, sk_R, kw_i) \). (Note: Known as the trapdoor privacy of PAEKS, \( \mathcal{A} \) can query \( O_C \) and \( O_T \) for trapdoors \( td_i \) or \( td_j \).)

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., \( kw \notin \{kw_i, kw_j\} \)).

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

\[
Adv_{\mathcal{A}}^{\text{IND-PAEKS}}(\lambda) = Pr[b = b'] - \frac{1}{2}. 
\]

Definition 3.2 (Trapdoor Privacy of PAEKS). A PAEKS scheme is called TP (or IND-IGKA secure) if, for any PPT adversary \( \mathcal{A} \), \( Adv_{\mathcal{A}}^{\text{TP-PAEKS}}(\lambda) \) 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. 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 \( \mathcal{A} \) can break the MCI of a PAEKS scheme, then there is a challenger \( \mathcal{C} \) 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 \((pp, pk_S, pk_R) \), \( \mathcal{C} \) passes this information to \( \mathcal{A} \).

Phase 1. On receiving any ciphertext query or trapdoor query for a keyword \( kw \) from \( \mathcal{A} \), \( \mathcal{C} \) 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 \((kw_{i1}, \ldots, kw_{in}) \) and \((kw'_{i1}, \ldots, kw'_{in}) \), \( \mathcal{C} \) performs the following steps. It randomly chooses a tuple \((kw_{i0}, kw'_{i0}) \) for some \( i \) such that \( kw_{i0} \neq kw'_{i0} \). Then, it takes this tuple as its challenge keyword and receives a challenge ciphertext \( ct' \). In addition, it randomly chooses \( n - 1 \) elements \((r_1, \ldots, r_{n-1}, r_{n+1}, \ldots, r_n) \) from the output space of the PAEKS algorithm. Finally, it returns \((r_1, \ldots, r_{n-1}, ct', r_{n+1}, \ldots, r_n) \) 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., \( kw \notin \{kw_{i0}, kw'_{i0}\} \)).

Guess. Finally, \( \mathcal{A} \) outputs a bit \( b' \), then \( \mathcal{C} \) takes \( \mathcal{A} \)'s answer as its answer.

As \( ct' \) is \( \mathcal{C} \)'s challenge ciphertext and the PAEKS algorithm is probabilistic, for the view of \( \mathcal{A} \), \((r_1, \ldots, r_{n-1}, ct', r_{n+1}, \ldots, r_n) \) is the same as the \( n \) truly ciphertext. Therefore, suppose \( \mathcal{A} \)'s answer is right, then \( \mathcal{C} \) can use \( \mathcal{A} \)'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, \( \mathcal{C} \) is given a challenge trapdoor \( td' \), instead of a challenge ciphertext \( ct' \). In addition, \( \mathcal{C} \) returns \((r_1, \ldots, r_{n-1}, td', r_{n+1}, \ldots, r_n) \) to \( \mathcal{A} \), where \((r_1, \ldots, r_{n-1}, r_{n+1}, \ldots, r_n) \) are randomly chosen from the output space of the Trapdoor algorithm. Based on the above description, with the answer of \( \mathcal{A} \), \( \mathcal{C} \) can also take \( \mathcal{A} \)'s answer as its answer. Therefore, the proof is completed.

4 CRYPTANALYSIS OF PREVIOUS TRAPDOOR PRIVACY SCHEMES

In this section, we cryptanalyze two lattice-based (variant) PEKS schemes proposed by Zhang et al. [63] at Inf. Sci. in 2019 and Zhang et al. [64] at IEEE Trans. Dependable Secur. Comput. in 2021, respectively. The core idea of these schemes against IGKA 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 IGKA), the security models in [63] and [64] do not capture the IGKA 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 [64] forward-secure PEKS scheme is vulnerable to IGKA.

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 \( td_j := tk_{kw_j} \) related to some time \( j \) and keyword \( kw \). It tries to obtain the keyword information via the following steps:

(1) Since \( tk_{kw_j} \) is generated from the receiver by performing \( \text{SamplePre}(A_{R||j}^\beta_j^{-1}, H_2(kw_j)^{-1} \cdot tk_{kw_j}) \), we know that \( \mu = A_{R||j}^\beta_j^{-1} \cdot tk_{kw_j} = A_{R||j} \cdot H_2(kw_j)^{-1} \cdot tk_{kw_j} \).
(2) Then, $A$ randomly selects a guessed keyword $kw' \in$ to test whether $y = y' \in \{0, 1\}^*$. 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 [63] 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 how an inside adversary $A$ can retrieve the keyword information hidden in the trapdoor. Suppose that $A$ has received a trapdoor $td := d_{kw}$ 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 SamplePre($A_{id_p}y^{-1}, D_{kw}, v, \delta$), we know that $v = A_{id_p}y^{-1}d_{kw}$, where $y = H_4(id_p, id_R, kw)$.
2. Then, $A$ randomly selects a guessed keyword $kw'$ and computes $y' = H_4(id_p, id_R, kw')$.
3. $A$ tests whether $v = A_{id_p}y'^{-1}d_{kw}$.
4. If the equation in Step 3 holds, $A$ outputs $kw'$ as its guess; otherwise, $A$ returns to Step 2 and continues to select and test other keywords.

Therefore, for the same reason, there is a high probability that $A$ 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 secure PEKS and SPHF for the language of the labelled CCA2-secure ciphertext. In particular, for the correctness and required securities, we require that the underlying PEKS scheme satisfy CL and the SPHF scheme is word-independent, $\epsilon$-correct for some negligible $\epsilon$, and pseudo-random. The following describes how to obtain this generic construction.

Let PEKS = (KeyGen, PEKS, Trapdoor, Test) be a PEKS with keyword space $KS_{PEKS}$, let PKE = (KeyGen, Encrypt, Decrypt) be a labelled PKE scheme with public key space $PKS_{PKE}$ and plaintext space $PS_{PKE}$, and let SPHF = (HashKG, ProjKG, Hash, Proj(Hash)) be the approximate word-independent SPHF for the language of the ciphertext defined below.

Language of Ciphertext. Let $(\Gamma, \lambda) = (ek_{pke}, dk_{pke})$, where $ek_{pke} \in PKS_{PKE}$ and $dk_{pke}$ is its corresponding decryption key. We define the language of ciphertext as follows:

$\mathcal{F} = \{(\lambda, c_{pke}, m_{pke}) \mid \exists p, c_{pke} \leftarrow \text{Encrypt}(ek_{pke}, label, m_{pke}; \rho)\}$

$\mathcal{L} = \{(\lambda, c_{pke}, m_{pke}) \mid \text{Decrypt}(dk_{pke}, label, c_{pke}) = m_{pke}\}$

where the witness relation $\mathcal{F}$ is implicitly defined as:

$\mathcal{F}((\lambda, c_{pke}, m_{pke}), \rho) = 1$ if and only if $c_{pke} \leftarrow \text{Encrypt}(ek_{pke}, label, m_{pke}; \rho)$.

Construction. The whole construction is described as follows:

- Setup($\lambda$): Given a security parameter $\lambda$, this algorithm runs the following steps:
  - Generates $(ek_{pke}, dk_{pke}) \leftarrow \text{PKE.KeyGen}(\lambda)$.
  - Randomly chooses a plaintext $m_{pke} \leftarrow \mathcal{P}_{PKS}$ and a label $\rho \in \{0, 1\}^*$. Chooses two secure hash functions $H_1 : PKS_{PKE} \times PS_{PKE} \times \{0, 1\}^* \rightarrow PKS_{PKE}$ and $H_2 : PKS_{PKE} \times \{0, 1\}^* \rightarrow KS_{PKE}$. Let $mpk := (\lambda, ek_{pke}, m_{pke}, label, H_1, H_2)$.
  - Generates $mpk := (\lambda, ek_{pke}, m_{pke}, label, H_1, H_2)$.
- KeyGen_{PS}($pp$): Given the public parameter $pp$, this algorithm runs the following steps:
  - Outputs the public parameter $pp := (\lambda, mpk, ek_{pke}, m_{pke}, label, H_1, H_2)$.
- KeyGen_{PS}($pp$): Given the public parameter $pp$, this algorithm runs the following steps:
  - Checks whether $mpk \leftarrow H_1(ek_{pke}, m_{pke}, label)$, if the equation is not satisfied, it terminates.
  - Computes $hp_{S} := \text{SPHF.ProjKG}(hk_{S}, mpk)$. Generates $ct_{pke} := \text{PKE.Encrypt}(mpk, label, m_{pke}; \rho_{S})$, where $\rho_{S}$ is the witness randomly selected such that $(ct_{pke}, mpk, m_{pke}, label) \leftarrow H_1(ek_{pke}, m_{pke}, label)$. Outputs the public key $pk_{S} := (hp_{S}, ct_{pke})$ and private key $sk_{S} := (hk_{S}, \rho_{S})$ of the sender.
- PEKS($pp, pk_{S}, sk_{S}, k_{p}, k_{R}$): 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 $k_{w} \in KS_{PKE}$, this algorithm runs the following steps:
  - Computes $H_{S} := \text{SPHF.Hash}(hk_{S}, mpk, ct_{pke}, mpke, m_{pke})$. Computes $H_{R} := \text{SPHF.ProjHash}(hp_{R}, mpk, ct_{pke}, m_{pke}, \rho_{R})$.
  - Computes $der-kws := H_{2}(k_{w}, H_{S} \oplus H_{R})$.
  - Generates $ct_{pke} := \text{PKE.Encrypt}(ek_{pke}, label, m_{pke}; \rho_{R})$. Outputs a searchable ciphertext $ct_{kw} := ct_{pke}$.
- Trapdoor($pp, pk_{S}, pk_{R}, sk_{S}, k_{w}$): Given the public parameter $pp$, the public key $pk_{S}$ of the sender, the public key $pk_{R}$ and private key $sk_{S}$ of the receiver, and a keyword $k_{w} \in KS_{PKE}$, this algorithm runs the following steps:
  - Computes $H_{R} := \text{SPHF.Hash}(hk_{R}, mpk, ct_{pke}, m_{pke})$. Computes $H_{S} := \text{SPHF.ProjHash}(hp_{S}, mpk, ct_{pke}, m_{pke}, \rho_{S})$.
  - Computes $der-kw' := H_{2}(k_{w}, H_{R} \oplus H_{R})$.
- Generates $td_{\text{PEKS,der-kw}'}_R$ \\
  $\leftarrow$ PEKS.Trapdoor$(sk_{\text{PEKS,der-kw}}')$. \\
- Outputs a trapdoor $td_{kw} := td_{\text{PEKS,der-kw}}'$.

- $\text{Test}(pp, ct_{kw}, td_{kw})$: Given the public parameter $pp$, the searchable ciphertext $ct_{kw}$, and the trapdoor $td_{kw}$, this algorithm outputs the result of PEKS.$\text{Test}(ct_{kw}, td_{kw})$.

**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

$$H_S = \text{SPHF}\text{-Hash}(hk_S, mpk, (ct_{\text{PKE,der}}, mpk_{\text{PKE}}))$$

$$= \text{SPHF}\text{-ProjHash}(hp_R, mpk, (ct_{\text{PKE}}, mpk_{\text{PKE}}), \rho_R)$$

$$= pH_R;$$

$$H_R = \text{SPHF}\text{-Hash}(hk_R, mpk, (ct_{\text{PKE,der}}, mpk_{\text{PKE}}))$$

$$= \text{SPHF}\text{-ProjHash}(hp_R, mpk, (ct_{\text{PKE}}, mpk_{\text{PKE}}), \rho_S)$$

$$= pH_S.$$

Therefore, $H_R \oplus pH_R \oplus pH_S$ holds. Clearly, if $kw = kw'$, then der-$kw_S = H_S(kw, H_S \oplus pH_S) = H_S(kw', H_R \oplus pH_R) = \text{der-$kw'$}_R$, and therefore, $ct_{\text{PEKS,der-$kw$}_S}$ and $td_{\text{PEKS,der-$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, if $kw \neq kw'$, then der-$kw_S = H_S(kw, H_S \oplus pH_S) \neq H_S(kw', H_R \oplus pH_R) = \text{der-$kw'$}_R$, and therefore, $ct_{\text{PEKS,der-$kw$}_S}$ and $td_{\text{PEKS,der-$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 under the standard model, 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 $Adv^i_{\mathcal{A}}(\lambda)$ denote the advantage of $\mathcal{A}$ in game $Game_i$, where $i \in \{0, \ldots, 3\}$. 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 under the standard model if the underlying SPHF scheme satisfies pseudo-randomness.

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

$\text{Game}_0$: 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 $Adv^0_{\mathcal{A}}(\lambda) := \epsilon$. 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:

- $Q_C$: For keyword $kw$, $C$ computes $ct_{kw}$ $\leftarrow$ PAEKS$(pp, pk_S, sk_S, pk_R, sk_R, kw)$ and returns $ct_{kw}$ to $\mathcal{A}$.

$\bullet \; O_T$: For keyword $kw$, $C$ computes $td_{kw}$ $\leftarrow$ Trapdoor$(pp, pk_S, pk_R, sk_G, kw)$ and returns $td_{kw}$ to $\mathcal{A}$.

$\text{Game}_1$: This game is identical to $\text{Game}_0$, except for the generation of the challenge ciphertext $ct^*$ in the Challenge phase. More concretely, instead of generating $H_S \leftarrow \text{SPHF.Hash}(hk_R, mpk, (ct_{\text{PKE,der}}, mpk_{\text{PKE}}))$, $C$ 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 $\text{Game}_0$ and $\text{Game}_1$. Therefore, we obtain

$$\left| Adv^1_{\mathcal{A}}(\lambda) - Adv^0_{\mathcal{A}}(\lambda) \right| \leq \text{negl}(\lambda).$$

$\text{Game}_2$: This game further changes the generation of the challenge ciphertext $ct^*$ in the Challenge phase. In this game, $der-kw_S$ is chosen from $KS_{\text{PEKS}}$, instead of by computing $der-kw_S \leftarrow H_S(kw'_0, H_S \oplus pH_S)$ for some $b \in \{0, 1\}$. As $H_S$ is randomly chosen, the output of $H_S(kw'_0, H_S \oplus pH_S)$ is random. Therefore, $\mathcal{A}$ cannot distinguish the view between $\text{Game}_1$ and $\text{Game}_2$. Consequently, we obtain

$$\left| Adv^2_{\mathcal{A}}(\lambda) - Adv^1_{\mathcal{A}}(\lambda) \right| \leq \text{negl}(\lambda).$$

$\text{Game}_3$: This game is the last game. Because the challenge ciphertext $ct^* = ct_{\text{PKE,der-$kw$}_S} = \text{PEKS}(pk_{\text{PKE,der-$kw$}_S})$ and $der-kw_S$ is now randomly chosen from $KS_{\text{PEKS}}$, the challenge ciphertext does not contain any information about the challenge keywords $(kw'_0, kw'_1)$. The only way for $\mathcal{A}$ to be is guessed. Therefore, we have

$$Adv^3_{\mathcal{A}}(\lambda) = 0.$$ 

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 under the standard model if the underlying SPHF scheme satisfies pseudo-randomness.

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

$\text{Game}_0$: 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 $Adv^0_{\mathcal{A}}(\lambda) := \epsilon$. In addition, the view simulated by the challenger $C$ is the same as that in $\text{Game}_0$ in the proof of Theorem 5.1.

$\text{Game}_1$: This game is identical to $\text{Game}_0$, except for the generation of the challenge trapdoor $td^*$ in the Challenge phase. More concretely, instead of generating $H_R \leftarrow \text{SPHF.Hash}(hk_R, mpk, (ct_{\text{PKE},mpk_{\text{PKE}}}))$, $C$ 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 $\text{Game}_0$ and $\text{Game}_1$. Therefore, we obtain

$$\left| Adv^1_{\mathcal{A}}(\lambda) - Adv^0_{\mathcal{A}}(\lambda) \right| \leq \text{negl}(\lambda).$$

$\text{Game}_2$: This game further changes the generation of the challenge trapdoor $td^*$ in the Challenge phase. In this game, $der-kw_R$ is chosen from $KS_{\text{PEKS}}$, instead of by computing $der-kw_R' \leftarrow H_S(kw'_0, H_R \oplus pH_R)$ for some $b \in \{0, 1\}$. As $H_S$ is randomly chosen, the output of $H_S(kw'_0, H_R \oplus pH_R)$ is random. Therefore, $\mathcal{A}$ cannot distinguish the view between $\text{Game}_1$ and $\text{Game}_2$. Consequently, we obtain

$$\left| Adv^2_{\mathcal{A}}(\lambda) - Adv^1_{\mathcal{A}}(\lambda) \right| \leq \text{negl}(\lambda).$$

Finally, combining the above games, we have $\epsilon \leq \text{negl}(\lambda)$. The proof is completed. □
$H((k_{i,1}''', H_{1} \oplus \phi_{1}p_{1})$ for some $b \in \{0, 1\}$. As $H_{1}$ is randomly chosen, the output of $H((k_{i,1}''', H_{1} \oplus \phi_{1}p_{1})$ is random. Therefore, $\mathcal{A}$ cannot distinguish the view between $\mathcal{G}_{1}$ and $\mathcal{G}_{2}$. Consequently, we obtain

$$\left| Ad_{\mathcal{G}_{1}}(\lambda) - Ad_{\mathcal{G}_{2}}(\lambda) \right| \leq \text{neg}(|\lambda|).$$

**Game 3:** This game is the last game. Because the challenge trapdoor $t' = t'_{\text{PEKS,der-kws}}$ is generated from PEKS, PEKS, and der-kws and der-kws is now randomly chosen from $KS_{\text{PEKS}}$, the challenge trapdoor does not contain any information about the challenge keywords $(k'_{w,1}, k'_{w,2})$ given by $\mathcal{A}$. The only way for $\mathcal{A}$ is to guess, therefore we have

$$Ad_{\mathcal{G}_{3}}(\lambda) = 0.$$

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

**Theorem 5.3.** The proposed generic PAEKS construction further satisfies MCI and MTP if the PEKS and 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 the best of our knowledge, for the current well-known PEKS schemes (e.g., [6, 8, 28]), the PEKS and Trapdoor algorithms are probabilistic. Hence, by combining the result of Theorem 3.3, the proposed construction satisfies MCI and MTP. \hfill $\square$

### 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 labelled IND-CCA1-secure PKE scheme introduced by Micciancio and Peikert [43], the word-independent SPHF scheme introduced by Benhamouda et al. [7], and the PEKS scheme introduced by Behnia et al. [6]. Note that, 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 $G = I_{g} \oplus g^{T}$ be the gadget matrix defined in [43], where $g^{T} = [1, 2, \ldots, 2^{k}]$ and $k = [\log q] - 1$. Finally, we also define the encoding function $\text{Encode}(\mu \in \{0, 1\}) = \mu \cdot (0, \ldots, 0, [q/2])^{T}$ and the deterministic rounding function $\text{R}(x) = [2x/q] \bmod 2$. 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 ($\ell'$):** Given a security parameter $\lambda$ and the parameters $n, m, \sigma_{1}, \sigma_{2}, a$ (set as in the following parameter selection part), this algorithm runs the following steps:
  - Sets $\rho, \ell' \leftarrow \text{poly}(n)$ and randomly chooses $m \leftarrow m_{1}, m_{2}, \ldots, m_{k} \leftarrow \{0, 1\}^{8}$.
  - Computes $(A_{0}, T) \leftarrow \text{TrapGen}(1^{n}, 1^{m}, q)$.
  - Sets $\text{ek}_{\text{PKE}} := A_{0}, \text{dk}_{\text{PKE}} := T$, and $\text{mp}_{\text{PKE}} := m$.
  - Randomly chooses element $u \leftarrow \mathcal{U}$ and sets label $\leq u$.
  - Choose two secure hash functions $H_{1} : \{0, 1\}^{n} \times \{0, 1\}^{m} \rightarrow \{0, 1\}^{n}$, and $H_{2} : \{0, 1\}^{n} \times \{0, 1\}^{m} \rightarrow \{0, 1\}^{n}$, and an injective ring homomorphism $h : \mathcal{R} \rightarrow \{0, 1\}^{n}$.
  - Computes $A \leftarrow H_{1}(A_{0}, m, u) \leftarrow \{0, 1\}^{n}$ and sets $\text{mp}_{\text{PKE}} := A$.
  - Outputs $\text{pp} := (\lambda, n, m, q, \sigma_{1}, \sigma_{2}, \rho, \ell', \text{ek}_{\text{PKE}} := A_{0}, \text{mp}_{\text{PKE}} := m, \text{label} \leftarrow u, H_{1}, H_{2}, h)$.

- **KeyGen$_{\text{p}}$(pp):** Given the public parameter pp, this algorithm runs the following steps:
  - Checks whether $A = H_{1}(A_{0}, m, u)$.
  - Computes $A_{u} = A + \{0, \text{G}(\text{u})\}$, randomly chooses a matrix $h_{k_{S}} := k_{S} \cdot \text{D}_{\mathcal{S}}^{m}$ and computes $p_{S} := \text{A}_{u} \cdot k_{S} \leftarrow \{0, 1\}^{n}$, where $s \geq \eta_{\ell}(\Lambda^{2}(\text{A}_{u}))$ for some $\epsilon = \text{neg}(\ell(n))$.
  - For $i = 1, \ldots, \kappa$, randomly chooses vectors $s_{S,i} \leftarrow \{0, 1\}^{n}$ as well as $e_{S,i} \leftarrow \text{D}_{\mathcal{S}}^{m}$ (re-select $e_{S,i}$ if $||e_{S,i}|| > 2\sqrt{m}$), and computes $c_{S,i} = A_{u}^{t} \cdot s_{S,i} + e_{S,i} \cdot \text{Encode}(m_{i}) \mod q$, where $t = \sigma_{1} \cdot \sqrt{m} \cdot \omega(\sqrt{\log n})$.
  - Outputs the public key $\text{pk}_{S} := (h_{S} := k_{S}, \rho_{S} := \{s_{S,i} \mid \varepsilon = \varepsilon\}$) and the private key $\text{sk}_{S} := (h_{k_{S}} := k_{S}, c_{S} := \{s_{S,i} \mid \varepsilon = \varepsilon\})$ of the sender.

- **KeyGen$_{\text{p}}$(pp):** Given the public parameter pp, this algorithm runs the following steps:
  - Checks whether $A = H_{1}(A_{0}, m, u)$.
  - Computes $A_{u} = A + \{0, \text{G}(\text{u})\}$, randomly chooses a matrix $h_{k_{S}} := k_{S} \cdot \text{D}_{\mathcal{S}}^{m}$ and computes $p_{S} := \text{A}_{u} \cdot k_{S} \leftarrow \{0, 1\}^{n}$, where $s \geq \eta_{\ell}(\Lambda^{2}(\text{A}_{u}))$ for some $\epsilon = \text{neg}(\ell(n))$.
  - For $i = 1, \ldots, \kappa$, randomly chooses vectors $s_{R,i} \leftarrow \{0, 1\}^{n}$ as well as $e_{R,i} \leftarrow \text{D}_{\mathcal{S}}^{m}$ (re-select $e_{R,i}$ if $||e_{R,i}|| > 2\sqrt{m}$), and computes $c_{R,i} = A_{u}^{t} \cdot s_{R,i} + e_{R,i} + \text{Encode}(m_{i}) \mod q$, where $t = \sigma_{1} \cdot \sqrt{m} \cdot \omega(\sqrt{\log n})$.
  - Generates $(B_{R}, B_{S}) \leftarrow \text{TrapGen}(1^{n}, 1^{m}, q)$.
  - Selects $\ell + 1$ random matrices $B_{R,1}, \ldots, B_{R,\ell}, C_{R} \leftarrow \{0, 1\}^{n \times m}$ and a random vector $r_{\mathcal{R}} \leftarrow \{0, 1\}^{n}$.
  - Outputs the public key $\text{pk}_{R} := (h_{S} := k_{S}, \rho_{R} := \{s_{R,i} \mid \varepsilon = \varepsilon\})$ of the receiver.

- **PAEKS$(\text{pp}, \text{sk}_{\mathcal{S}}, \text{sk}_{\mathcal{R}}, \text{pk}_{\mathcal{R}})$:** Given the public parameter pp, the public key $\text{pk}_{\mathcal{S}}$ and the private key $\text{sk}_{\mathcal{S}}$ of the sender, the public key $\text{pk}_{\mathcal{R}}$ of the receiver, and a keyword $k_{w} \in \{0, 1\}^{\ell}$, this algorithm runs as follows.
  - For $i = 1, \ldots, \kappa$, computes $h_{S,i} \leftarrow R(e_{R,i} \cdot k_{S} \text{ (mod q)}), p_{S,i} \leftarrow R(s_{R,i} \cdot p_{R} \text{ (mod q)})$ and $y_{S,i} = h_{S,i} \cdot p_{S,i}$.
  - Sets $y_{S} := y_{S,1} \cdot y_{S,2} \cdot \ldots \cdot y_{S,\kappa} \leftarrow \{0, 1\}^{n}$.
  - Computes $\text{der-kws} := d_{k_{S}} := d_{k_{S},1} \cdot d_{k_{S},2} \cdot \ldots d_{k_{S},\ell} := H_{2}(k_{w}, y_{S})$, where $k_{w} \in \{0, 1\}^{\ell}$.
  - Computes $B_{R} := C_{R} \cdot \text{D}_{\mathcal{S}}^{m} \cdot d_{k_{S},i} B_{R,i}$ and $f_{d_{k}} := [B_{R}]_{d_{k}} \leftarrow \{0, 1\}^{n \times m}$.
  - For $j = 1, \ldots, \rho$, performs the following steps:
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_\text{s} = der-kw_\text{p}).
- If ct_{kw} and \( td_{kw'} \) 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 [7]. That is, if the norm of the first error term is less than \( \varepsilon/2 \cdot q/4 \) 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 \( \varepsilon/2 \cdot q/4 \) and \( kw = kw' \), then \( dk_{s} = dk_{r} \).

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

\[
h_{S,i} = R(e_{S,i}^T \cdot k_{S,i}) \mod q
\]

\[
= R(s_{R,i}^T \cdot A_k) \cdot k_{S,i} + e_{R,i}^T \cdot k_{S,i} \mod q
\]

\[
= R(s_{R,i}^T \cdot A_k) \cdot k_{S,i} \mod q
\]

\[
= pr_{i};
\]

\[
h_{R,i} = R(e_{R,i}^T \cdot k_{R,i}) \mod q
\]

\[
= R(s_{S,i}^T \cdot A_k) \cdot k_{R,i} + e_{S,i}^T \cdot k_{R,i} \mod q
\]

\[
= R(s_{S,i}^T \cdot A_k) \cdot k_{R,i} \mod q
\]

\[
= pr_{i};
\]

Since \( y_{S,i} \) and \( y_{R,i} \) are related to the same derived keyword, we have \( y_{S,i} = y_{R,i} \).

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

\[
v_j = c_{0,j} - t_{dk} e_{1,j} \in \mathbb{Z}_q
\]

\[
r_{dk} s_j + x_j + b_j [q/2] - t_{dk} (F_{dk} s_j + y_j [z_j])
\]

where \( x_j = t_{dk} y_j [z_j] \).

According to Lemma 22 in [2], if the norm of the second error term is bounded by \( q \cdot \sigma_7 \cdot t \cdot m \cdot \omega(\sqrt{\log m}) + O(t \sigma_2 m^{3/2}) \leq q/5 \), then \( h_{S,i} \) can be obtained correctly. Hence, we have \( v_j = b_j \) for \( j = 1, \ldots, \rho \) if the derived keywords are the same.

**Parameter Selection.** To make the system work properly, the parameters have the following restrictions [2, 7, 43]:

1. \( m > 5n \log q \) so TrapGen can operate [43].
2. \( q > \sigma_1 m^{1/2} \omega(\sqrt{\log m}) \) so that the first error term is bounded by \( \varepsilon/2 \cdot q/4 \) and therefore \( y_S = y_R \).
3. \( \alpha < [\sigma_2 t \omega(\sqrt{\log m})]^{-1} \) 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 Revèg’s reduction [50, 51] can operate.
5. \( \sigma_2 > t \cdot m \cdot \omega(\sqrt{\log m}) \) such that the security proof in [2] and SampleLeft work correctly.

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

\[
m = 6n^{1+\delta}; \quad \sigma_1 = 2\sqrt{n}; \quad \sigma_2 = mt \cdot \omega(\sqrt{\log n});
\]

\[
m = m^{2.5} \cdot \omega(\sqrt{\log n}) \alpha = \left(t^2 m^2 \cdot \omega(\sqrt{\log n})\right)^{-1}; n^\delta \geq q^{\log q}.
\]

**Security.** The security of the proposed instantiation is directly based on the underlying schemes. As the language of Benhamouda et al.’s word-independent SPHF scheme [7] 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 pseudo-randomness. In addition, the PEKS and Trapdoor algorithms in Behnia et al.’s PEKS scheme [6] are probabilistic. On the basis of Theorem 5.3, we obtain the following theorem:

**Theorem 6.2.** The proposed lattice-based PAEKS scheme satisfies MCI and MTP under the standard model.
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>SM</th>
<th>NTA</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOC’04 [8]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>HL17 [27]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ZTW’19 [63]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>QCH*20 [49]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BOY20 [6]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ZXW*21 [64]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LTT*21 [39]</td>
<td>✓</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>
<td>✓</td>
</tr>
</tbody>
</table>

✓: The scheme supports the corresponding feature; ✗: The scheme fails to support the corresponding feature; SM: Standard model; 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 [63]</td>
<td>$(\ell + m\ell + m)[q]$</td>
<td>$m</td>
<td>q</td>
</tr>
<tr>
<td>BOY20 [6]</td>
<td>$\kappa(q + 2m</td>
<td>q</td>
<td>+ 1)$</td>
</tr>
<tr>
<td>ZXW*21 [64]</td>
<td>$(\ell + m\ell + m)[q]$</td>
<td>$m</td>
<td>q</td>
</tr>
<tr>
<td>Ours</td>
<td>$\kappa(</td>
<td>q</td>
<td>+ 2m</td>
</tr>
</tbody>
</table>

$k, \rho$: The parameters related to security parameter $\lambda$; $\ell$: The length of the keyword; $T_H$, $T_M$, $T_{SP}$, $T_{BD}$, and $T_{SL}$: The running time of a general multiplication, general hash function, SamplePre function, BasisDel function, and SampleLeft function, respectively.

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

7 COMPARISON

In this section, we present a comparison of our lattice-based instantiation with other PEKS/PAEKS schemes (i.e., BOC’04 [8], HL17 [27], ZTW*19 [63], QCH*20 [49], BOY20 [6], ZXW*21 [64], 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, MTP, quantum-resistance (QR), standard model (SM), and no trusted authority (NTA). As we have cryptanalyzed ZTW*19 [63] and ZTW*21 [64] 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 under the standard model 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 2009 with Inter(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 $n = 256$, $m = 9753$, $q = 4096$, $\rho = 10$, $\kappa = 10$, $\ell = 10$, $\sigma_1 = 8$, $\sigma_2 = 8$, where $\rho, \kappa$ 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 securities (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 [63] and ZXW*21 [64] 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 [63] and ZXW'21 [64]. 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 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 108-2218-E-004-002-MY2, MOST 109-2221-E-004-001-MBK, and MOST 110-2221-E-004-003-.

REFERENCES

PAEKS: Cryptanalysis, Enhanced Security, and Quantum-resistant Instantiation
ASIA CCS ’22, May 30–June 3, 2022, Nagasaki, Japan


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 [64], which consists of five algorithms.

- Setup(1)∗: Taking as input a security parameter λ, this algorithm runs the following steps:
  - Randomly selects μ ← ℤ_q and three secure hash functions
    H_1 : ℤ_q ⊄ ℤ_q, H_2 : ℤ_q × ℤ_q ⊄ ℤ_q, H_3 : ℤ_q × ℤ_q ⊄ ℤ_q
    respectively generates (A_S, T_S) and (A_R, T_R) by performing TrapGen(1^n, 1^n, q).
  - Outputs the public parameter pp = (μ, H_1, H_2, H_3), the public/private key pairs of the sender (pk_S, sk_S) and the receiver (pk_R, sk_R) for time period 0.
- KeyUpdate(pk_R, sk_R, i, j): Taking as input an public/private key pair (pk_R, sk_R): A_R[i], sk_R[j] = 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 R_{i−j} = H_1(A_R[i] j + · · · + H_1(A_R[i+1] j) × ℤ_q)
  - Computes T_R[i] ← NewBasicDel(A_R[i], R_{i−j} × T_R[i]), δ_j, where A_R[j] = A_R[j] × (R_{i−j} × T_R[i]) × ℤ_q.
  - Outputs the public/private key pair (pk_R[j] = A_R[j], sk_R[j] = 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 (pk_S[j] = A_S[j], sk_S[j] = T_S[j]) for time period j.

- PEKS(pk_S, sk_S, pk_R, j, k): Taking as input a public/private key pair (pk_S[j] = A_S[j], sk_S[j] = T_S[j]) of the receiver for time period j, the current time period j, and keyword kw ∈ {0, 1}^t, the sender runs the following steps:
  - Chooses a random binary string y_j = y_{j_1}y_{j_2} · · · y_{j_t}, uniform matrix B_j ∈ ℤ_q^{t×j_t}, noise e_j = e_{j_1}e_{j_2} · · · e_{j_t}, and noise v_j = v_{j_1}v_{j_2} · · · v_{j_t}, where e_{j_1}, · · · , e_{j_t} ∼ ℤ_q and v_{j_1}, · · · , v_{j_t} ∼ ℤ_q.
• Outputs a searchable ciphertext ct := (c_j, c_{jl}, z_j).

Trapdoor (pk_{R,j}, sk_{R,j}, j, kw): Taking as input a public/private key pair (pk_{R,j} := A_{R,j}, sk_{R,j} := T_{R,j}) of the receiver for time period j, current time period j and keyword kw ∈ {0, 1}^l, the receiver runs the following steps:

- Computes \( \beta_j \leftarrow H_2(kw||j), c_j := \mu^T(B_j + e_j + (y_{jl}, \ldots, y_{j,l}||j/2) \cdot T_j, c_j := (A_{R,j})^{-1}B_j + V_j. \)
- Computes \( h_j \leftarrow H_3(c_j||y_j) \in Z_q^m \) and generates \( z_j \leftarrow \frac{\text{SamplePre}(A_{ij}||j, T_{ij}||j, h_j, \sigma)}{z_j}. \)
- Outputs a searchable ciphertext ct := (c_j, c_{jl}, z_j).

- Generates a warrant w ∈ {0, 1}^l according to its requirements.
- Selects a uniform random vector \( r \in Z_q^m \) and computes \( \mu \leftarrow H_2(id_i||id_p||w||r). \)
- Computes \( \beta_w \leftarrow \frac{\text{SamplePre}(A_{id_i}, T_{id_i}, \mu, \delta)}{z_q}. \)
- Sends \((w, r, \beta_w)\) directly to id_p.

After receiving the data send from id_p, id_p runs the following steps:

- Computes \( R_w \leftarrow H_5(id_i||id_p||w||beta) \) and \( T_{pro} \leftarrow \text{NewBasisDel}(A_{id_p}, R_w, T_{id_p}, \sigma). \)
- Sets \((pk_{pro} := A_{pro}, sk_{pro} := T_{pro})\) as the proxy-oriented public/private key pair, where \( A_{pro} = A_{id_p}(R_w)^{-1} \in Z_q^{mn}. \)

IBEKS (pk_{pro}, sk_{pro}, kw, id_p): Taking as input the public/private key pair \((pk_{pro} := A_{pro}, sk_{pro} := T_{pro})\) of the proxy-oriented a keyword kw ∈ {0, 1}^l, and receiver’s identity id_p, the proxy ipd runs the following steps:

- Randomly chooses \( \eta \leftarrow \frac{\zeta}{\eta_q} \) and a binary string \( r = r_1r_2\cdots r_l \in \{0, 1\}^l. \)
- Samples a noise vector \( \eta = \eta_1\eta_2\cdots\eta_l \leftarrow \frac{\zeta}{\eta_q} \) and a noise matrix \( S = s_1s_2\cdots s_l \leftarrow \frac{\zeta}{\eta_q} \), where \( \zeta \) is a Gaussian distribution.

- Computes \( y \leftarrow H_4(id_p||id_k||kw), \xi = (A_{id_k}Y^{-1})^TF + S, \xi = v^T F + \eta + (r_1, r_2, \ldots, r_l) || j/2). \)
- Computes \( h \leftarrow H_5(r||\xi) \) and \( 0 \leftarrow \text{SamplePre}(A_{pro}, T_{pro}, h, \delta) \in Z_q^{mn}. \)
- Outputs a searchable ciphertext ct := (ξ, ξ, θ).

Trapdoor (sk_{id_p}, kw): Taking as input the private key \( sk_{id} := T_{id_p} \) of the receiver id_p and a keyword kw ∈ {0, 1}^l, id_p runs the following steps:

- Computes \( r \leftarrow r_1r_2\cdots r_l = H_5(r||\xi). \)
- Computes \( d_{id_p} \leftarrow \text{SamplePre}(A_{id_p}, Y^{-1}, D_{id_p}, v, \delta) \in Z_q^{mn}, \) where \( A_{id_p}Y^{-1}d_{id_p} = v \) is satisfied.
- Outputs a trapdoor td := d_{id_p}.

Test (pk_{pro}, ct, td): Taking as input the proxy-oriented public key \( pk_{pro} := A_{pro}, \) a searchable ciphertext \( ct := (ξ, ξ, θ), \) and a trapdoor td := d_{id_p}, the cloud server runs the following steps:

- Computes \( r = r_1r_2\cdots r_l \leftarrow \frac{\xi - d_{kw}^{T}r \xi}{Z_q^l}. \)
- For \( j = 1, \ldots, l, \) if \( |r_j - [q/2]| < [q/4], \) sets \( r_j = 1; \) otherwise, sets \( r_j = 0. \)
- Updates r and further computes \( h \leftarrow H_5(r||\xi). \)
- Checks whether the equation \( A_{pro}\theta \equiv h \) holds. If the equation holds, outputs 1; otherwise, outputs 0.