Public-key Authenticated Encryption with Keyword Search: A Generic Construction and Its Quantum-resistant Instantiation

Zi-Yuan Liu\textsuperscript{1}, Yi-Fan Tzeng\textsuperscript{1*}, Raylin Tso\textsuperscript{1}, Masahiro Mambo\textsuperscript{2}, Yu-Chi Chen\textsuperscript{3}

\textsuperscript{1}Department of Computer Science, National Chengchi University, Taipei 11605, Taiwan
\{ziyliu, yftseng, raylin\}@cs.nccu.edu.tw
\textsuperscript{2}Institute of Science and Engineering, Kanazawa University, Kakuma-machi, Kanazawa 920-1192, Japan
\textsuperscript{3}Department of Computer Science and Engineering, Yuan Ze University, Taoyuan 32003, Taiwan

July 29, 2021

Abstract

The industrial Internet of Things (IIoT) integrates sensors, instruments, equipment, and industrial applications, enabling traditional industries to automate and intelligently process data. To reduce the cost and demand of required service equipment, IIoT relies on cloud computing to further process and store data. Public-key encryption with keyword search (PEKS) plays an important role, due to its search functionality, to ensure the privacy and confidentiality of the outsourced data and the maintenance of flexibility in the use of the data. Recently, Huang and Li proposed the “public-key authenticated encryption with keyword search” (PAEKS) to avoid the insider keyword guessing attacks (IKGA) in the previous PEKS schemes. However, all current PAEKS schemes are based on the discrete logarithm assumption and are therefore vulnerable to quantum attacks. In this study, we first introduce a generic PAEKS construction, with the assistance of a trusted authority, that enjoys the security against IKGA in the standard model, if all building blocks are secure under standard model. Based on the framework, we further propose a novel instantiation of quantum-resistant PAEKS that is based on NTRU assumption under random oracle. Compared with its state-of-the-art counterparts, the experiment result indicates that our instantiation is more efficient and secure.

Keywords—Public-key Authenticated Encryption with Keyword Search; Insider Keyword Guessing Attacks; Generic Construction; Quantum-resistant

1 Introduction

The Internet of Things (IoT) is a system that connects a large set of devices to a network, where these devices can communicate with each other over the network. Industrial IoT (IIoT) is a particular type of IoT that fully utilizes the advantages of IoT for remote detection, monitoring, and management in the industry. Because the volume of data and computation in the industry is very large, and long-term storage is required, IIoT is highly reliant on cloud computing technology to reduce the cost of storage and computing environments (Figure 1). Despite the numerous benefits of processing IIoT data through cloud computing, industrial data typically have commercial value and thus necessitate privacy protection when such sensitive data are offloaded to the cloud. Therefore, to ensure data confidentiality, sensitive data should be encrypted before being uploaded to the cloud.

In addition to data confidentiality, data sharing is indispensable in IIoT. For instance, in an industrial organization, the administrator in the information department (i.e., the data sender) must share the data collected from IoT devices with an administrator from another department (i.e., the data receiver). To ensure data confidentiality, the data sender encrypts the data by using the public key of the data receivers. However, in such a method, if the data receiver wants to retrieve the data from the ciphertext stored in the cloud, the data receiver must download all the ciphertext and further decrypt it, which consumes considerable time and resources.

*Corresponding author
Public-key encryption with keyword search (PEKS), first introduced by Boneh et al. [BCOP04], is highly suited to the aforementioned application environment because PEKS makes the ciphertext searchable. Furthermore, in PEKS, a data sender not only uploads encrypted data but also uploads the encrypted keywords related to the data using the data receiver’s public key. To download the data related to a specified keyword, the data receiver can use his/her private key to generate a corresponding trapdoor and submit the trapdoor to the cloud server. The cloud server can then identify encrypted keywords corresponding to the trapdoor and then returns the corresponding encrypted data to the data receiver. A secure PEKS scheme is required to ensure that the ciphertext and trapdoor will not leak any keyword information to malicious outsiders. However, Byun et al. [BRPL06] noted that having only the two aforementioned security requirements is insufficient because the cloud server may be malicious, where the malicious cloud server guesses the keyword hiding in the trapdoor—a type of attack called insider keyword guessing attacks (IKGA). More concretely, because the cloud server can adaptively generate a ciphertext for any keyword by using the data receiver’s public key, through trial and error, test for that self-made ciphertext that is matched with the trapdoor received from the data receiver. As mentioned in [BRPL06], because the keyword space is not large enough, the keyword-related information searched by the data receiver is very likely to be leaked to the malicious cloud server. Hence, if the malicious cloud server has the ability to perform encryption and test, such as [LLW21, ZCH21, ZLW*21, EIO20], then such PEKS schemes cannot resist IKGA.

To prevent IKGA, recently, Huang and Li [HL17] introduced a new cryptography primitive called public-key authenticated encryption with keyword search (PAEKS). In this primitive, the data sender not only generates but also authenticates ciphertext, whereas a trapdoor generated from the data receiver is only valid to the ciphertext authenticated by the specific data sender. Therefore, the cloud server cannot perform IKGA. In addition, for the same reason, the cloud server also cannot obtain any keyword information from the receiver’s search pattern [LZWT14] because he/she cannot generate ciphertext to test his/her guess. Furthermore, many PAEKS schemes [HMZ*18, LLY*19, NE19, PSE20, QCH*20, LHS*19, WZM*19, LTT*21] have been formulated for further application in IoT and IIoT as well as in cloud computing environments.

Shor [Sho99, Sho94] reported on quantum algorithms that can violate the traditional number-theoretic assumptions, such as the integer factoring assumption and discrete logarithm assumption. In particular, the advent of the 53-qubit quantum computer, proposed by Arute et al. [AAB*19], may improve quantum computing technology and affect the existing cryptographic systems. Because the security of existing PAEKS schemes [HMZ*18, LLY*19, NE19, PSE20, LHS*19, QCH*20, WZM*19] is based on the discrete logarithm assumption, quantum computers can come to pose a potential threat to existing schemes. Hence, the means of constructing a quantum-resistant PEAKS scheme is an emerging issue among scholars and practitioners.

1.1 Our Contribution

In this paper, we introduce a novel solution for constructing a quantum-resistant PAEKS scheme for IIoT. At a high level, the original keyword space is commonly found and easy to test. Our strategy is to allow a data sender and data receiver to generate an “extended keyword” from an original keyword without interacting with each other. Then, the ciphertext and trapdoor are generated using the extended keyword instead of the original keyword. Since the extended keyword is high-entropy, the malicious cloud server cannot generate a valid ciphertext to match the trapdoor to perform IKGA.

Accordingly, we provide a generic PAEKS construction by leveraging an identity-independent 2-tier
identity-based key encapsulation mechanism (IBKEM), a pseudorandom generator (PRG), and an anonymous identity-based encryption (IBE). Our generic construction is modeled by a variant PAEKS system model. That is, in addition to the same model as the original PAEKS system model, this model requires a trusted authority to help the data sender and data receiver to obtain their full private keys.

We also present two rigorous proofs to show that our construction satisfies the security requirements of PAEKS. These requirements are indistinguishability against chosen keyword attacks (IND-CKA) and indistinguishability against IKGA (IND-IKGA) under a multi-user setting in the standard model. Because our construction is IND-IKGA secure, there is no adversary who can infer any information about the queried keyword from the given trapdoor. Therefore, there is no search pattern privacy concern [LZWT14] in our construction.

Furthermore, we first employ Ducas et al.’s anonymous IBE [DLP14] to obtain an identity-independent 2-tier IBKEM under the NTRU assumption. We then combine the scheme with [DLP14] to obtain an instantiation of PAEKS. Because the security of [DLP14] is inherited, we obtain the first quantum-resistant instantiation of PAEKS.

The comparison results of our scheme with other state-of-the-art PAEKS schemes are presented in Table 2 and FIGURE 3; our instantiation was demonstrated to be not only more secure but also more efficient with respect to ciphertext generation, trapdoor generation, and testing.

1.2 Related Work

The PEKS schemes that are secure against IKGA can be separated into three categories: dual-server PEKS, PAEKS, and witness-based searchable encryption.

The concept of dual-server PEKS was first introduced by Chen et al. [CMY*15, CMY*16b, CMY*16a], which is secure against IKGA if the cloud server and key server do not collude with each other. However, Huang [HT17] indicates that [CMY*15, CMY*16b] are susceptible to IKGA. Recently, Chen et al. [CWZH19] introduced an efficient dual-server scheme that is resistant to IKGA without needing any pairing computations. In addition, Mao et al. [MFGW19] suggested a quantum-resistant dual-server PEKS scheme, which is also the first lattice-based PEKS that is protected from IKGA. Although dual-server PEKS is a successful solution to IKGA, the two servers in the system are related to the function of processing keywords, so they are not independent. Therefore, in many cases, it is difficult to guarantee that two servers will not collude.

Considering the above limitations, scholars thus began to study methods for constructing trapdoors that are only valid for certain ciphertexts. Fang et al. [FSGW09, FSGW13] first considered using a one-time signature to authenticate the ciphertext while having the trapdoor be valid only for the authenticated ciphertext, a method that improved resistance to IKGA. Huang and Li [HL17] formally defined the system model and security model for PAEKS. Noroozi and Eslami [NE19] first considered Huang and Li’s scheme [HL17] is not secure against IKGA and further improved [HL17] without incurring additional cost complexity. To resist quantum attacks, Zhang et al. [ZTW+19] proposed two lattice-based PAEKS schemes; however, Liu et al. [LTT20, LTTL21] recently demonstrated that the security models of these works are flawed and therefore cannot withstand IKGA. Pakniat et al. [PSE20] introduced the first certificateless PAEKS scheme for an IoT environment. Moreover, Qin et al. [QCH*20] and Li et al. [LHS*19] further prevented malicious adversary eavesdrops on the transmission channel of ciphertext and trapdoor, and executes the test algorithm to determine whether the two ciphertexts shared the same keyword. Although the aforementioned PAEKS schemes resist IKGA, these schemes are based on the discrete logarithm assumption, which makes them vulnerable to attacks from quantum computers.

Ma et al. [MMSY18] introduced a cryptographic primitive called “witness-based searchable encryption,” in which the trapdoor is valid only when the ciphertext has a witness relation to the trapdoor. Chen et al. [CXWT19] formulated an improvement to reduce the complexity of the trapdoor size. Inspired by [MMSY18], Liu et al. [LTTM21] introduced a new concept called “designated-ciphertext searchable encryption,” where the trapdoor is designated to a ciphertext; this concept affords users with a quantum-resistant instantiation. Despite their advantages, however, these schemes require the data sender to interact with the data receiver; moreover, they incur additional communication costs and are inapplicable to many scenarios.

1.3 Organization of the Paper

The rest of the paper is organized as follows. Section 2 introduces the preliminaries, and Section 3 recalls the definition of the building blocks used in our generic construction. Moreover, Section 4 provides the definition and security requirement of the PAEKS. Next, Sections 5 and 6 introduce our generic construction and the security proofs, respectively. Section 7 elaborates on the first quantum-resistant PAEKS instantiation, and
Table 1: Notations

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lambda )</td>
<td>Security parameter</td>
</tr>
<tr>
<td>( \Pi )</td>
<td>PAEKS</td>
</tr>
<tr>
<td>( \Psi )</td>
<td>IBE</td>
</tr>
<tr>
<td>( \Omega )</td>
<td>Identity-independent 2-tier IBKEM</td>
</tr>
<tr>
<td>( F )</td>
<td>Pseudorandom generator</td>
</tr>
<tr>
<td>( IDS )</td>
<td>Identity space</td>
</tr>
<tr>
<td>( CS )</td>
<td>Ciphertext space</td>
</tr>
<tr>
<td>( KS )</td>
<td>Shared key space</td>
</tr>
<tr>
<td>( PS )</td>
<td>Plaintext space</td>
</tr>
<tr>
<td>( W )</td>
<td>Keyword space</td>
</tr>
<tr>
<td>( N, Z, R )</td>
<td>Natural number, integer number, real number</td>
</tr>
<tr>
<td>( G_1, G_T )</td>
<td>Cyclic group</td>
</tr>
<tr>
<td>( v, V )</td>
<td>Vector, matrix</td>
</tr>
<tr>
<td>( a \parallel b )</td>
<td>Concatenation of element ( a ) and ( b )</td>
</tr>
<tr>
<td>( s \leftarrow S )</td>
<td>Sampling an element ( s ) from ( S )</td>
</tr>
<tr>
<td>( \tilde{T} )</td>
<td>Gram-Schmidt orthogonalization of ( T )</td>
</tr>
<tr>
<td>(</td>
<td>v</td>
</tr>
<tr>
<td>( \text{negl}(\cdot), \text{poly}(\cdot) )</td>
<td>Negligible function, polynomial function</td>
</tr>
<tr>
<td>( \text{PPT} )</td>
<td>Probabilistic polynomial-time</td>
</tr>
</tbody>
</table>

Section 8 details the analysis of the communication cost and computation cost incurred in the related PAEKS schemes. Finally, Section 9 concludes this study.

2 Preliminary

For simplicity and readability, we use the notations defined in Table 1 throughout the paper.

2.1 Lattices

We now introduce the basic concepts underlying lattices that are used in our instantiation. An \( m \)-dimension lattice \( \Lambda \) is an additive discrete subgroup of \( \mathbb{R}^m \), which can be defined as follows.

**Definition 1 (Lattices).** We say that a \( m \)-dimension lattice \( \Lambda \) generated by a basis \( B = [b_1 | \cdots | b_n] \in \mathbb{R}^{m \times n} \) is defined by

\[
\Lambda(B) = \Lambda(b_1, \cdots, b_n) = \left\{ \sum_{i=1}^{n} b_i a_i \mid a_i \in \mathbb{Z} \right\},
\]

where \( b_1, \cdots, b_n \in \mathbb{R}^m \) are \( n \) linear independent vectors.

In addition, for a prime \( q \), a matrix \( A \in \mathbb{Z}_q^{n \times m} \), and a vector \( u \in \mathbb{Z}_q^n \), we can define the following three sets [GPV08, ABB10]:

- \( \Lambda_q := \{ e \in \mathbb{Z}^m \mid \exists s \in \mathbb{Z}^n \text{ where } A^T s = e \text{ mod } q \} \).
- \( \Lambda_q^\perp := \{ e \in \mathbb{Z}^m \mid Ae = 0 \text{ mod } q \} \).
- \( \Lambda_q^u := \{ e \in \mathbb{Z}^m \mid Ae = u \text{ mod } q \} \).

2.2 Discrete Gaussian Distributions

For any vector \( x \in \mathbb{R}^n \) and any positive real number \( s \), we define the following two notations:

- \( \rho_{x \in \mathbb{R}^n} = \exp \left( -\pi \frac{||x - c||^2}{s^2} \right) \).
\[
\rho_{sc}(x) = \sum_{x \in \Lambda} \rho_{sc}(x).
\]

The discrete Gaussian distribution over the lattice \( \Lambda \) with center \( c \) and parameter \( s \) can then be defined as \( D_{\Lambda,sc}(x) = \rho_{sc}(x)/\rho_{sc}(\Lambda) \) for any \( x \in \Lambda \). Note that we usually omit \( c \) if \( c = 0 \).

## 2.3 Rings and NTRU Lattices

Here, we briefly introduce rings and NTRU lattices, as formulated in previous studies \([LPR10, LPR13]\). Let \( N \) be a power of 2. The ring can then be defined as \( \mathcal{R} = \mathbb{Z}[x]/\Phi(x) \), where \( \Phi(x) = x^N + 1 \). Furthermore, for some integer \( q \), we use \( \mathcal{R}_q \) to denote \( \mathcal{R}/q\mathcal{R} = \mathbb{Z}[x]/(q, \Phi(x)) \). For two polynomials \( f = \sum_{i=0}^{N-1} f_ix^i \) and \( g = \sum_{i=0}^{N-1} g_ix^i \), \( fg \) denotes polynomial multiplication in \( \mathbb{Q}[x] \) and \( f \ast g \) is defined as the convolution product of \( f \) and \( g \), i.e., \( f \ast g := fg \mod (x^N + 1) \). Additionally, \( [f] \) denotes the coefficient-wise rounding of \( f \).

The first NTRU-based public-key encryption is introduced in 1996 by Hoffstein et al. \([HPS98]\), and later Stehlé and Steinfield \([SS11]\) presents a new variant that has been proven to be secure in the worst-case lattice problem. Compared with integer lattices, the operations of NTRU are based on the ring of polynomials \( \mathcal{R} \), and can be defined as follows.

**Definition 2 (Anticirculant Matrix \([DLP14]\]).** An \( N \)-dimensional anticirculant matrix of \( f \) is the following Toeplitz matrix:

\[
\mathcal{A}_N(f) = \begin{pmatrix}
0 & f & \cdots & f_{N-1} \\
-f_{N-1} & 0 & \cdots & f_{N-2} \\
\vdots & \ddots & \ddots & \ddots \\
-f_1 & -f_2 & \cdots & 0
\end{pmatrix}
\]

**Definition 3 (NTRU Lattices \([BOY20]\)).** For prime integer \( q \) and \( f, g \in \mathcal{R} \), \( h = g \ast f^{-1} \mod q \), the NTRU lattices with \( h \) and \( q \) is defined as

\[
\Lambda_{h,q} = \{ (u, v) \in \mathcal{R}^2 \mid u + v \ast h = 0 \mod q \}.
\]

Here, \( \Lambda_{h,q} \) is a full-rank lattice generated by the rows of \( A_{h,q} = \left( -\mathcal{A}_N(h) \ I_N \ qI_N \ O_N \right) \), where \( I_N \) is an \( N \times N \) identity matrix and \( O_N \) is an \( N \times N \) zero matrix.

As mentioned by Hoffstein et al. \([HHP^*03]\), although one can generate the lattice from basis \( A_{h,q} \) by using a single polynomial \( h \in \mathcal{R}_q \), \( A_{h,q} \) has a large orthogonal defect and therefore inefficiency in standard lattice operation. Therefore, to solve the issue, they further showed that another short basis \( B_{f,g} \) generates the same lattice \( \Lambda_{h,q} \) as \( A_{h,q} \), where \( f, g, F, G \in \mathcal{R} \) and \( f \ast G - g \ast F = q \).

**Definition 4 (Statistical Distance \([ABB10]\)).** Given two random variables \( X \) and \( Y \) taking values in a finite set \( S \), the statistical distance is defined as:

\[
\Delta(X, Y) = \frac{1}{2} \sum_{s \in S} | \Pr[X = s] - \Pr[Y = s] | .
\]

Due to the efficient of NTRU, Ducas et al. \([DLP14]\) introduced a NTRU-based IBE scheme. In their scheme, they provided an algorithm that can efficiently obtain the pair of basis \( (h, B_{f,g}) \), as shown in Algorithm 1. Additionally, since \( B_{f,g} \) is a short basis, based on the results of \([DLP14, GPV08]\), there exists an algorithm \( \text{Gaussian Sampler}(B_{f,g}, \sigma, c) \) (Algorithm 2) that can sample a vector \( v \) without leaking any information of the basis \( B_{f,g} \) such that \( \Delta(D_{\Lambda(h),\sigma,c}, \text{Gaussian Sampler}(B_{f,g}, \sigma, c)) \leq 2^{-\lambda} \), where \( \sigma > 0 \) and \( c \in \mathbb{Z}^N \).

## 3 Building Blocks

In this section, we recall three crucial cryptographic primitives, namely identity-independent 2-tier IBKEM, IBE, and PRG, which are used as the building blocks in our generic construction.
Algorithm 1 Basis_Generation [DLP14]

**Input:** $N, q$

**Output:** $h \in \mathcal{R}_q, B_{f, g} \in \mathbb{Z}_q^{2N \times 2N}$.

*Initialization: $\sigma_f = 1.17 \sqrt{\frac{q}{2N}}$.*

1. $f, g \leftarrow D_{N, \sigma_f}$.
2. Norm $\leftarrow \max \left( \|g - f\|, \left\| \frac{gf}{f + g} \right\|, \left\| \frac{gf}{f + g} \right\| \right)$.
3. if (Norm $> 1.17\sqrt{q}$) then
   4. Go to Step 1.
5. end if
6. Using extended Euclidean algorithm, compute $d_5, d_6 \in \mathbb{R}$ and $'_5, '_6 \in \mathbb{Z}$ such that
   - $-d_5 \cdot 5 = '_5$ and $-d_6 \cdot 6 = '_6$.
7. if (GCD($'_5, '_6$) $\neq 1$ or GCD($'_5, f$) $\neq 1$) then
   8. Go to Step 1.
9. end if
10. Using extended Euclidean algorithm, compute $D, E \in \mathbb{Z}$ such that
    - $D \cdot '_'5 + E \cdot '_'6 = 1$, and
    - $v_5 \leftarrow d_5 G', v_6 \leftarrow d_6 G'$.
11. Compute $h = g \cdot f^{-1} \mod q$ and $B_{f, g} = \begin{pmatrix} \mathcal{A}_N(g) & -\mathcal{A}_N(f) \\ \mathcal{A}_N(G) & -\mathcal{A}(F) \end{pmatrix}$.
12. return $h$ and $B_{f, g}$.

Algorithm 2 Gaussian_Sampler [DLP14]

**Input:** $B_{f, g}, \sigma, c$

**Output:** $v \in \mathbb{Z}_q^{2N}$.

1. $v_{2N} \leftarrow 0$.
2. $c_{2N} \leftarrow c$.
3. for $i \leftarrow 2N, \cdots , 1$ do
   4. $c'_i \leftarrow \langle c_i, b_j \rangle / \|b_i\|^2$.
   5. $\sigma'_i \leftarrow \sigma / \|b_i\|^2$.
   6. $z_i \leftarrow \text{SampleZ}(\sigma'_i, c'_i)$.
   7. $c_{i-1} \leftarrow c_i - z_i b_i, v_{i-1} \leftarrow v_i + z_i b_i$.
4. end for
5. return $v_0$. 
3.1 Identity-independent 2-tier IBKEM

An identity-independent 2-tier IBKEM $\Omega$ comprises the five algorithms: $(\text{Setup}, \text{Extract}, \text{Enc}_1, \text{Enc}_2, \text{Dec})$ along with an identity space $\text{IDS}$, ciphertext space $\text{CS}$, and symmetric key space $\text{KS}$. These algorithms are described as follows.

- **Setup$(1^\lambda)$** $\rightarrow$ $(\text{msk}, \text{mpk})$: This is the *setup* algorithm that takes the security parameter $\lambda$ as its input and outputs a master private key $\text{msk}$ and a master public key $\text{mpk}$.

- **Extract$(\text{msk}, \text{id})$** $\rightarrow$ $\text{sk}_\text{id}$: This is the *extraction* algorithm that takes the two inputs of a master private key $\text{msk}$ and identity $\text{id} \in \text{IDS}$ and outputs a private key $\text{sk}_\text{id}$ for the identity.

- **Enc$_1$(mpk)** $\rightarrow$ $(\text{ct}, r)$: This is the *first encapsulation* algorithm that takes the input of a master public key $\text{mpk}$ and outputs a ciphertext $\text{ct} \in \text{CS}$ and a randomness $r$.

- **Enc$_2$(mpk, id, r)** $\rightarrow$ $k/\bot$: This is the *second encapsulation* algorithm that takes the three inputs of a master public key $\text{mpk}$, identity $\text{id}$, and randomness $r$ and outputs either a symmetric key $k \in \text{KS}$ or the reject symbol $\bot$.

- **Dec($\text{sk}_\text{id}$, id, ct)** $\rightarrow$ $k/\bot$: This is the *decryption* algorithm that takes the three inputs of a private key $\text{sk}_\text{id}$, identity $\text{id}$, and ciphertext $\text{ct}$ and outputs either symmetric key $k \in \text{KS}$ or a reject symbol $\bot$.

**Definition 5** (Correctness). An identity-independent 2-tier IBKEM $\Omega$ is correct if for all security parameters $\lambda$, all master key pairs $(\text{msk}, \text{mpk})$ output by $\text{Setup}(1^\lambda)$, all private keys $\text{sk}_\text{id}$ for identity $\text{id}$ output by $\text{Extract}(\text{msk}, \text{id})$, all $(\text{ct}, r)$ pairs output by $\text{Enc}_1(\text{mpk})$, and all $k$ values output by $\text{Enc}_2(\text{mpk}, \text{id}, r)$, the following equation holds:

$$\Pr[\text{Dec}(\text{sk}_\text{id}, \text{id}, \text{ct}) = k] = 1 - \text{negl}(\lambda).$$

The basis security requirement of identity-independent 2-tier IBKEM is to meet the indistinguishability under adaptively-identity chosen-plaintext attacks (IND-ID-CPA), which ensures that no PPT adversary can distinguish whether the challenge ciphertext is generated from the $\text{Enc}_1$ and $\text{Enc}_2$ algorithm or is randomly chosen from the ciphertext space $\text{CS}$. This security requirement can be modeled by the following security game played between an adversary $\mathcal{A}$ and a challenger $\mathcal{B}$.

**Game - IND-ID-CPA:**

- **Initialization.** The challenger $\mathcal{B}$ first runs $(\text{msk}, \text{mpk}) \leftarrow \text{Setup}(1^\lambda)$. $\mathcal{B}$ then sends the master public key $\text{mpk}$ to $\mathcal{A}$ and keeps the master private key $\text{msk}$ secret.

- **Phase 1.** The adversary $\mathcal{A}$ is given access to query the extract oracle with any identity $\text{id}$, and $\mathcal{B}$ returns a valid private key $\text{sk}_\text{id}$ for identity $\text{id}$ by using $\text{Extract}$ algorithm.

- **Challenge.** $\mathcal{A}$ submits $\mathcal{B}$ an identity $\text{id}^*$ that has not been queried to extract oracle in Phase 1. $\mathcal{B}$ randomly selects a bit $b \in \{0, 1\}$. If $b = 0$, $\mathcal{B}$ generate a true ciphertext by using $\text{Enc}_1$ and $\text{Enc}_2$. Otherwise, $\mathcal{B}$ randomly selects a ciphertext from the ciphertext space. $\mathcal{B}$ then returns the ciphertext as a challenge to $\mathcal{A}$.

- **Phase 2.** $\mathcal{A}$ can continue querying the extract oracle as Phase 1. The only restriction is that $\mathcal{A}$ cannot query the extract oracle with the identity $\text{id}^*$.

- **Guess.** $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$.

The advantage of $\mathcal{A}$ is defined as

$$\text{Adv}^{\text{IND-ID-CPA}}_{\Omega, \mathcal{A}}(\lambda) := | \Pr[b = b'] - \frac{1}{2} |.$$

**Definition 6** (IND-ID-CPA Security of IBKEM). An identity-independent 2-tier IBKEM scheme $\Omega$ is IND-ID-CPA secure if for all PPT adversaries $\mathcal{A}$, $\text{Adv}^{\text{IND-ID-CPA}}_{\Omega, \mathcal{A}}(\lambda)$ is negligible.
3.2 IBE

An IBE scheme $\Psi$ comprises four algorithms (Setup, Extract, Enc, Dec) along with an identity space $IDS$, ciphertext space $CS$, and plaintext space $PS$, described as follows.

- **Setup($1^\lambda$) → (msk, mpk):** This is the setup algorithm that takes the security parameter $\lambda$ as its input and outputs a master private key $msk$ and master public key $mpk$.

- **Extract(msk, id) → sk_{id}:** This is the extraction algorithm that takes the two inputs of a master private key $msk$ and identity $id \in IDS$ and outputs a private key $sk_{id}$ for the identity.

- **Enc(mpk, id, m) → ct_{id}:** This is the encryption algorithm that takes the three inputs of a master public key $mpk$, identity $id$, and plaintext $m \in PS$ and outputs a ciphertext $ct_{id} \in CS$.

- **Dec(sk_{id}, ct_{id}) → m:** This is the decryption algorithm that takes the two inputs of a private key $sk_{id}$ (for identity $id$) and ciphertext $ct_{id}$ and outputs a plaintext $m \in PS$.

**Definition 7 (Correctness of IBE).** An IBE $\Psi$ is correct if, for all security parameters $\lambda$, all master key pairs $(msk, mpk)$ output by Setup($1^\lambda$), all private keys $sk_{id}$ for identity id output by Extract(msk, id), and all ciphertexts $(ct_{id})$ output by Enc(mpk, id, m), the following equation holds:

$$\Pr[\text{Dec}(sk_{id}, ct_{id}) = m] = 1 - \negl(\lambda).$$

The basic requirement of IBE is to meet IND-ID-CPA which is similar to the IND-ID-CPA game of identity-independent 2-tier IBKEM in Section 3.1. The difference is as follows: In Challenge phase, $A$ sends ($id^*, m_0^*, m_1^*$) to $B$ instead of only a challenge identity $id^*$, where $m_0, m_1$ are two messages with the same length. Then, according to the bit $b$, $B$ returns $ct^* \leftarrow \text{Enc}(mpk, id^*, m_b^*)$.

Our instantiation requires a stronger security requirement called indistinguishability and anonymity against chosen plaintext and chosen identity attacks (IND-ANON-ID-CPA), which ensures that no PPT adversary can retrieve any information pertaining to the identity and the message from a challenge ciphertext, as modeled by the following game.

**Game - IND-ANON-ID-CPA:**

- **Initialization.** The challenger $B$ first runs $(msk, mpk) \leftarrow \text{Setup}(1^\lambda)$ and then sends the master public key $mpk$ to $A$ and keeps master private key $msk$ secret.

- **Phase 1.** The adversary $A$ is given access to query the extract oracle with any identity $id$, and $B$ returns a valid private key $sk_{id}$ for identity $id$ by using the Extract algorithm.

- **Challenge.** $A$ submits $B$ two messages $m_0^*, m_1^*$ and two identities $id_0^*, id_1^*$ that have not been queried to extract the oracle. $B$ randomly chooses a bit $b \in \{0, 1\}$ and then computes $ct^* \leftarrow \text{Enc}(mpk, id_b^*, m_b^*)$. Finally, $B$ returns the challenge ciphertext $ct^*$ to $A$.

- **Phase 2.** $A$ can continue querying the oracle as Phase 1. The only restriction is that $A$ cannot query the extract oracle with $id_0^*$ and $id_1^*$.

- **Guess.** $A$ outputs a bit $b' \in \{0, 1\}$.

The advantage of $A$ is defined as

$$\text{Adv}_{\Psi, A}^{\text{IND-ANON-ID-CPA}}(\lambda) := |Pr[b = b'] - \frac{1}{2}|.$$

**Definition 8 (IND-ANON-ID-CPA Security of IBE).** An IBE scheme $\Psi$ is IND-ANON-ID-CPA secure if $\text{Adv}_{\Psi, A}^{\text{IND-ANON-ID-CPA}}(\lambda)$ is negligible for all PPT adversaries $A$.

For analytical convenience, in this work, we consider an IBE to be anonymous if the IBE is IND-ANON-ID-CPA secure.
3.3 Pseudorandom Generator (PRG)

Informally, suppose that a distribution $\mathcal{D}$ is pseudorandom if no PPT distinguisher that can distinguish a string $s$ is either selected from the distribution $\mathcal{D}$ or randomly selected from a uniform distribution. We provide the following definition of the pseudorandom generator in [KL20].

**Definition 9** (Pseudorandom Generator). Let $F : \{0, 1\}^n \rightarrow \{0, 1\}^m$ be a deterministic PPT algorithm, where $m = \text{poly}(n)$ and $m > n$. We say that $F$ is a pseudorandom generator if the following two conditions are satisfied:

- **Expansion:** For every $n$, it holds that $m > n$.
- **Pseudorandomness:** For all PPT distinguishers $\mathcal{D}$,

$$\left| \Pr[\mathcal{D}(r) = 1] - \Pr[\mathcal{D}(F(s)) = 1] \right| \leq \text{negl}(n),$$

where $r \leftarrow \{0, 1\}^m$ and seed $s \leftarrow \{0, 1\}^n$.

4 PAEKS

In this section, we introduce the system model and the security requirements of PAEKS.

4.1 System Model

The system model introduced here is a “variant” system model of PAEKS. Compared with the system model in previous PAEKS schemes [HL17, QCH+20], this system model also requires a certificate authority for issuing certificates for public keys but requires an additional trusted authority. In more detail, in addition to the certificate authority, there are four entities: a trusted authority, data sender, data receiver, and cloud server (FIGURE 2). The data sender and data receiver need to interact with the trusted authority to obtain their full private keys.

A PAEKS scheme $\Pi$ comprises six algorithms: (Setup, KeyGenS, KeyGenR, PAEKS, Trapdoor, Test) together with a keyword space $W$, which are detailed as follows.

- **Setup($1^\lambda$) $\rightarrow$ (pp, msk):** This is the setup algorithm that takes the security parameter $\lambda$ as input, and outputs a system parameter pp and a master private key msk. Note that the master private key is held by trusted authority.
• KeyGen\(S\)(pp, msk) \rightarrow (pk_S, sk_S): This is the \textit{data sender key generation} algorithm interacted between data sender and trusted authority. It takes a system parameter pp and master private key msk, and outputs data sender’s public key pk\(S\) and private key sk\(S\).
• KeyGen\(R\)(pp, msk) \rightarrow (pk_R, sk_R): This is the \textit{data receiver key generation} algorithm interacted between data receiver and trusted authority. It takes a system parameter pp and master private key msk, and outputs data receiver’s public key pk\(R\) and private key sk\(R\).
• PAEKS(pp, pk_S, sk_S, pk_R, kw) \rightarrow ct: This is the \textit{authenticated encryption} algorithm that takes a system parameter pp, data sender’s public key pk\(S\) and private key sk\(S\), data receiver’s public key pk\(R\), and a keyword kw \(\in W\), and outputs a searchable ciphertext ct.
• Trapdoor(pp, pk_R, sk_R, pk_S, kw) \rightarrow td: This is the \textit{trapdoor} algorithm that takes a system parameter pp, data receiver’s public key pk\(R\) and private key sk\(R\), data sender’s public key pk\(S\), and a keyword kw \(\in W\), and outputs a trapdoor td.
• Test(pp, ct, td) \rightarrow 1/0: This is the test algorithm that takes a system parameter pp, searchable ciphertext ct, and a trapdoor td, and outputs 1 if ct and td correspond to the same keyword; outputs 0, otherwise.

\textbf{Definition 10} (Correctness of PAEKS). A PAEKS scheme \(\Pi\) is correct if, for all security parameters \(\lambda\), all system parameter/master private key pairs (pp, msk) output by Setup\((\lambda^3)\), all data sender’s key pairs (pk\(_S\), sk\(_S\)) output by KeyGen\(S\)(pp, msk), all data receiver’s key pairs (pk\(_R\), sk\(_R\)) output by KeyGen\(R\)(pp, msk), all searchable ciphertexts ct output by PAEKS(pp, pk\(_S\), sk\(_S\), pk\(_R\), kw), and all trapdoors td output by Trapdoor(pp, pk\(_R\), sk\(_R\), pk\(_S\), kw), the following equation holds with overwhelming probability:

\[
\text{Test}(pp, ct, td) = \begin{cases} 1, & \text{if ct, td contains the same kw;} \\ 0, & \text{otherwise.} \end{cases}
\]

\subsection{4.2 Security Requirements}

The security requirements of our variant PAEKS follow the security requirements in the original PAEKS scheme: IND-cka and IND-IKGA. Specifically, IND-cka and IND-IKGA securities ensure that no PPT adversary can obtain any information regarding the keyword from the searchable ciphertext and trapdoor, respectively. We follow the method of [NE19] to model the aforementioned two security requirements in the multi-user context by using two security games featuring interaction between an adversary \(A\) and a challenger \(B\). Because the malicious insider has more power than the malicious outsider has, we only consider the IND-IKGA in this work. Here we note that to capture multi-user context, we use pk\(_U\) and sk\(_U\) to denote some user \(U\)’s public key and private key, respectively. In addition, oracle \(O_{\text{PAEKS}}(kw, pk_U)\) means that \(A\) wants to obtain a ciphertext that is authenticated by the data sender \(S\) and can be tested by the user \(U\)’s trapdoors; oracle \(O_{\text{Trapdoor}}(kw, pk_U)\) means that \(A\) wants to obtain a trapdoor generated by the data receiver \(R\) where this trapdoor can test a ciphertext authenticated by the user \(U\) and encrypted for the data receiver \(R\).

**Game - IND-CKA:**

- \textbf{Initialization.} The challenger \(B\) first runs \((pp, msk) \leftarrow \text{Setup}(\lambda^3)\). The algorithm then runs \((pk_S, sk_S) \leftarrow \text{KeyGen}_S(pp, msk)\) and \((pk_R, sk_R) \leftarrow \text{KeyGen}_R(pp, msk)\). Finally, \(B\) sends the system parameter pp, data sender’s public key pk\(S\), and data receiver’s public key pk\(R\) to \(A\) while keeping secret the master private key msk, data sender’s private key sk\(S\), and data receiver’s private key sk\(R\).
- \textbf{Phase 1.} \(A\) can make polynomially many queries to oracles \(O_{\text{PKGen}}\), \(O_{\text{PKGen}}\), \(O_{\text{PAEKS}}\), and \(O_{\text{Trapdoor}}\). \(B\) then responds as follows.
  - \(O_{\text{PKGen}_S}(U)\): For \(U \notin \{S, R\}\), \(B\) runs \((pk_U, sk_U) \leftarrow \text{KeyGen}_S(pp, msk)\). Then, \(B\) returns pk\(_U\) to \(A\), and keeps sk\(_U\) secret.
  - \(O_{\text{PKGen}_R}(U)\): For \(U \notin \{S, R\}\), \(B\) runs \((pk_U, sk_U) \leftarrow \text{KeyGen}_R(pp, msk)\). Then, \(B\) returns pk\(_U\) to \(A\), and keeps sk\(_U\) secret.
  - \(O_{\text{PAEKS}}(kw, pk_U)\): \(B\) computes ct \(\leftarrow \text{PAEKS}(pp, pk_S, sk_S, pk_R, kw)\) and returns ct to \(A\).
  - \(O_{\text{Trapdoor}}(kw, pk_U)\): \(B\) computes td \(\leftarrow \text{Trapdoor}(pp, pk_R, sk_R, pk_U, kw)\) and returns td to \(A\).
• **Challenge.** After the end of Phase 1, $\mathcal{A}$ outputs two keywords $kw^*_0, kw^*_1 \in W$ with the following restriction: for $i = 0, 1$, $(kw^*_i, pk_S)$ have not been queried to oracle $O_{\text{Trapdoor}}$ in Phase 1. $B$ then chooses a random bit $b \in \{0, 1\}$ and returns $ct^* = (\Psi, ct^*, h) \leftarrow \text{PAEKS}(\text{pp}, pk_S, sk_R, kw^*_0)$ to $\mathcal{A}$.

• **Phase 2.** $\mathcal{A}$ can continue to make queries, as was the case in Phase 1. The only restriction is that $\mathcal{A}$ cannot make any query to $O_{\text{Trapdoor}}$ on $(kw^*_i, pk_S)$ for $i = 0, 1$.

• **Guess.** $\mathcal{A}$ outputs its guess $b' \in \{0, 1\}$.

The advantage of $\mathcal{A}$ is defined as

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

**Definition 11** (IND-CKA security of PAEKS). A PAEKS scheme $\Pi$ is IND-CKA secure if for all PPT adversaries $\mathcal{A}$, $\text{Adv}^{\text{IND-CKA}}_{\Pi, \mathcal{A}}(\lambda)$ is negligible.

**Game - IND-IKGA:**

• **Initialization.** The challenger $B$ first runs $(\text{pp}, msk) \leftarrow \text{Setup}(\lambda)$ and then runs $(pk_S, sk_S) \leftarrow \text{KeyGen}_S(\text{pp}, msk)$ and then runs $(pk_R, sk_R) \leftarrow \text{KeyGen}_R(\text{pp}, msk)$. Finally, $B$ sends the system parameter pp, data sender’s public key $pk_S$, and data receiver’s public key $pk_R$ to $\mathcal{A}$ while keeping secret the master private key msk, data sender’s private key $sk_S$, and data sender’s private key $sk_R$.

• **Phase 1.** $\mathcal{A}$ can make polynomially many queries to oracles $O_{\text{PKGen}}, O_{\text{PKGen}}, O_{\text{PAEKS}},$ and $O_{\text{Trapdoor}}$. $B$ then responds as follows.

  - $O_{\text{PKGen}}(U)$: For $U \notin \{S, R\}$, $B$ runs $(pk_U, sk_U) \leftarrow \text{KeyGen}_S(\text{pp}, msk)$. Then, $B$ returns $pk_U$ to $\mathcal{A}$, and keeps $sk_U$ secret.

  - $O_{\text{PKGen}}(U)$: For $U \notin \{S, R\}$, $B$ runs $(pk_U, sk_U) \leftarrow \text{KeyGen}_R(\text{pp}, msk)$. Then, $B$ returns $pk_U$ to $\mathcal{A}$, and keeps $sk_U$ secret.

  - $O_{\text{PAEKS}}(kw, pk_U)$: $B$ computes $ct \leftarrow \text{PAEKS}(\text{pp}, pk_S, sk_S, pk_U, kw)$ and returns $ct$ to $\mathcal{A}$.

  - $O_{\text{Trapdoor}}(kw, pk_U)$: $B$ computes $td \leftarrow \text{Trapdoor}(\text{pp}, pk_R, sk_R, pk_U, kw)$ and returns $td$ to $\mathcal{A}$.

• **Challenge.** After the end of Phase 1, $\mathcal{A}$ outputs two keywords $kw^*_0, kw^*_1 \in W$ with the following restriction: for $i = 0, 1$, $(kw^*_i, pk_R)$ have not been queried to oracle $O_{\text{PAEKS}}$ in Phase 1. $B$ then selects a random bit $b \in \{0, 1\}$ and returns $td^* \leftarrow \text{Trapdoor}(\text{pp}, pk_R, sk_R, pk_S, kw^*_b)$ to $\mathcal{A}$.

• **Phase 2.** $\mathcal{A}$ can continue to make queries, as was the case in Phase 1. The only restriction is that $\mathcal{A}$ cannot make any query to $O_{\text{PAEKS}}$ on $(kw^*_i, pk_R)$ for $i = 0, 1$.

• **Guess.** $\mathcal{A}$ outputs its guess $b' \in \{0, 1\}$.

The advantage of $\mathcal{A}$ is defined as

$$\text{Adv}^{\text{IND-IKGA}}_{\Pi, \mathcal{A}}(\lambda) := |\Pr[b = b'] - \frac{1}{2}|.$$

**Definition 12** (IND-IKGA security of PAEKS). A PAEKS scheme $\Pi$ is IND-IKGA secure if for all PPT adversaries $\mathcal{A}$, $\text{Adv}^{\text{IND-IKGA}}_{\Pi, \mathcal{A}}(\lambda)$ is negligible.

### 5 Generic PAEKS Construction

Abdalla et al. [ABC*08] proposed a generic construction that allows any anonymous IBE to be converted to a PEKS scheme. In their construction, they took each keyword as an identity and use “identity” to generate ciphertext; while taking the trapdoor as the identity’s private key. If the cloud server can use the trapdoor to “decrypt” the ciphertext, that means that the trapdoor and the ciphertext are associated with the same keyword. Unfortunately, schemes constructed in this way cannot withstand IKGA because the malicious cloud server can adaptively generate ciphertext with any keyword. Inspired by [ABC*08], we construct a generic PAEKS to further against IKGA. Specifically, we demonstrate how a PAEKS scheme can be constructed by combing an anonymous IBE, FG, and identity-independent 2-tier IBKEM.

The core conception of our construction to resist IKGA is that we use identity-independent 2-tier IBKEM, allowing data sender and data receiver can obtain a shared key without interaction. If they can obtain a shared
key, we say that they authenticate each other. The data sender and data receiver then use this shared key to extend the keyword to a high-entropy randomness by using PRG. Rather than using the original keyword, they use the extended keyword to generate a ciphertext and trapdoor, respectively. Following the idea of [ABC’08], the data sender takes the randomness as an “identity” to generate a ciphertext for the data receiver. The data receiver can extract a private key for this identity and take this private key as the corresponding trapdoor. After receiving the trapdoor uploaded by the data receiver, the cloud server can search for the ciphertext containing keywords which is the same as in the trapdoor.

Since the malicious cloud server cannot obtain any information of the shared key, he/she cannot adaptively generate valid ciphertexts for IKGA. In addition, because the keywords in the ciphertext and the trapdoor are the output of PRG with the shared key as input, and because the IBE is anonymous, the malicious cloud server cannot obtain any information regarding the key from the ciphertext and trapdoor.

To construct a PAEKS scheme $\Pi = \langle \text{Setup}, \text{KeyGen}, \text{KeyGen}_r, \text{PAEKS}, \text{Trapdoor}, \text{Test} \rangle$ with the keyword space $W$, we use the following cryptosystems as the building block. Let $\Psi = \langle \text{Setup}, \text{Extract}, \text{Enc}, \text{Dec} \rangle$ be an anonymous IBE scheme with the identity space $\mathcal{Y}$. IDS, ciphertext space $\mathcal{Y}.CS$, and plaintext space $\mathcal{Y}.PS$. Let $\Omega = \langle \text{Setup}, \text{Extract}, \text{Enc}_1, \text{Enc}_2, \text{Dec} \rangle$ be an identity-independent 2-tier IBKEM scheme with the identity space $\Omega.IDS$, ciphertext space $\Omega.CS$, and symmetric key space $\Omega.KS$. In addition, let $F : X \mapsto Y$ be a PRG that maps $X$ to $Y$, where $X = \{KW||shk \mid kw \in W \wedge shk \in \Omega.KS\}$ and $Y = \Psi.IDS$. The generic construction is detailed in the subsequent paragraphs. Note that although our construction is based on identity-based cryptosystems, the entire construction remains in the public key setting.

- Setup($1^\lambda$) $\rightarrow$ (pp, msk): Given a security parameter $\lambda$, this algorithm runs as follows.
  1. Choose a proper PRG $F : X \rightarrow Y$.
  2. Choose a secure hash function $H : \{0, 1\}^α \rightarrow \{0, 1\}^β$, where $α, β \in \mathbb{Z}^+$.
  3. Generate $(\Omega.\text{msk}, \Omega.\text{mpk}) \leftarrow \Omega.\text{Setup}(1^\lambda)$.
  4. Output system parameter $pp := (\lambda, \Omega.\text{mpk}, H, F)$ and master private key $msk := \Omega.\text{msk}$. Note that $msk$ is kept secret by the trusted authority.

- KeyGen$(\ast, msk)$ $\rightarrow$ (pk$_\ast$, sk$_\ast$): Given a system parameter $pp = (\lambda, \Omega.\text{mpk}, H, F)$ and a master private key $msk = \Omega.\text{msk}$, data receiver and trusted authority interact as follows.
  1. The data sender first computes $(\Omega.ct_s, \Omega.rs) \leftarrow \Omega.\text{Enc}_1(\Omega.\text{mpk})$, randomly chooses $ran_s \leftarrow \Omega.IDS$, and submits $ran_s$ to the trusted authority.
  2. The trusted authority then returns $\Omega.sk_{ran_s} \leftarrow \Omega.\text{Extract}(\Omega.\text{msk}, ran_s)$ to the data sender via private channel.
  3. Data sender outputs his/her public key $pk_s := (ran_s, \Omega.ct_s)$ and private key $sk_s := (\Omega.sk_{ran_s}, \Omega.rs)$.

- KeyGen$(\ast, msk)$ $\rightarrow$ (pk$_\ast$, sk$_\ast$): Given a system parameter $pp = (\lambda, \Omega.\text{mpk}, H, F)$ and a master private key $msk = \Omega.\text{msk}$, data receiver and trusted authority interact as follows.
  1. The data receiver first computes $(\Omega.ct_r, \Omega.rs) \leftarrow \Omega.\text{Enc}_1(\Omega.\text{mpk})$, randomly chooses $ran_r \leftarrow \Omega.IDS$, and submits $ran_r$ to the trusted authority.
  2. The trusted authority then returns $\Omega.sk_{ran_r} \leftarrow \Omega.\text{Extract}(\Omega.\text{msk}, ran_r)$ to the data receiver via private channel.
  3. Data receiver computes $(\Psi.\text{mpk}, \Psi.\text{msk}) \leftarrow \Psi.\text{Setup}(1^\lambda)$.
  4. Finally, data receiver outputs data receiver’s public key $pk_r := (ran_r, \Omega.ct_r, \Psi.\text{mpk})$ and private key $sk_r := (\Omega.sk_{ran_r}, \Omega.rs, \Psi.\text{msk})$.

- PAEKS$(pp, pk_s, sk_s, pk_r, kw) \rightarrow ct$: Given a system parameter $pp = (\lambda, \Omega.\text{mpk}, H, F)$, a data sender’s public key $pk_s = (ran_s, \Omega.ct_s)$ and private key $sk_s = (\Omega.sk_{ran_s}, \Omega.rs)$, a data receiver’s public key $pk_r = (\Omega.ran_r, \Omega.ct_r, \Psi.\text{mpk})$, and a keyword $kw \in W$, data sender works as follows.
  1. Compute $k_{S,R,1} \leftarrow \Omega.\text{Dec}(\Omega.sk_{ran_s}, ran_s, \Omega.ct_r)$.
  2. Compute $k_{S,R,2} \leftarrow \Omega.\text{Enc}_2(\Omega.\text{mpk}, ran_r, \Omega.rs)$.
  3. Compute $shk_s \leftarrow k_{S,R,1} \oplus k_{S,R,2}$, where $\oplus$ is an operation compatible with the key space.
  4. Compute $f_s \leftarrow F(kw||shk_s)$.
The proposed PAEKS scheme is IND-IKGA secure under the standard model. The following provides two security proofs to show that our generic construction is IND-CKA secure and identity-independent 2-tier IBKEM are correct.

**Correctness.** Notably, the data sender and data receiver rely on the underlying identity-independent 2-tier IBKEM to exchange an extended keyword and the extended keyword acts as an identity in the underlying IBE scheme. Therefore, the proposed construction is correct if and only if the underlying anonymous IBE and identity-independent 2-tier IBKEM are correct.

### 6 Security Proofs

The following provides two security proofs to show that our generic construction is IND-CKA secure and IND-IKGA secure under the standard model.

**Theorem 1.** The proposed PAEKS scheme $\Pi$ is IND-CKA secure if the underlying IBE scheme $\Psi$ is IND-ANON-ID-CPA secure.

**Proof of Theorem 1.** If adversary $\mathcal{A}$ can win the IND-CKA game with a non-negligible advantage, then challenger $\mathcal{B}$ can win the IND-ANON-ID-CPA game of the underlying IBE scheme $\Psi$ with a non-negligible advantage. Their interaction is described as follows.

- **Initialization.** Given the security parameter $\lambda$, $\mathcal{B}$ first chooses a proper secure hash function $H$ and pseudorandom generator $F$ and invokes the IND-ANON-ID-CPA game of $\Psi$ to obtain $\Psi.mpk$. Next, $\mathcal{B}$ executes the following steps.
  - Compute $(\Psi.mpk, \lambda)$.
  - Compute $(\lambda, \Omega.cts, \Omega.\tau)$ and $(\lambda, \Omega.\tau)$.
  - Randomly choose $\lambda$ and $\alpha$ from $\Omega.IIDS$.
  - Compute $\lambda.\sk_{\tau}$ and $\lambda.\sk_{\tau}$ from $\Omega.IIDS$.

Finally, $\mathcal{B}$ sends the data sender’s public key $\sk := (\lambda, \Omega.cts)$, data receiver’s public key $\sk := (\lambda, \Omega.cts)$, and system parameter $pp := (\lambda, \Omega.mpk, H, F)$ to $\mathcal{A}$, and keeps $(\Omega.mpk, \lambda.\sk_{\tau}$) secret.

- **Phase 1.** $\mathcal{A}$ can make polynomially many queries to oracles $\mathcal{O}_{\text{PKGen}}(U), \mathcal{O}_{\text{PKGen}}(U), \mathcal{O}_{\text{PAEKS}}(kw, pk_U)$, and $\mathcal{O}_{\text{trapdoor}}(kw, pk_U)$. $\mathcal{B}$ then responds as follows.
  - $\mathcal{O}_{\text{PKGen}}(U)$: For $U \notin \{S, R\}$, $\mathcal{B}$ first computes $(\Omega.\ct_U, \Omega.\tau_U)$, and chooses $\Omega.\sk_{\tau_U}$ from $\Omega.IIDS$, and computes $\Omega.\sk_{\tau_U}$ from $\Omega.IIDS$. $\mathcal{B}$ then returns $\sk_U := (\Omega.\sk_{\tau_U})$ to $\mathcal{A}$ and keeps $\sk_U := (\Omega.\sk_{\tau_U})$ secret.
Theorem 2. The proposed PAEKS scheme \( \Pi \) is IND-IKGA secure if the underlying pseudorandom generator \( F \) satisfies pseudorandomness and identity-independent 2-tier IBKEM is IND-ID-CPA secure.

Proof of Theorem 2. Let \( \mathcal{A} \) be a PPT adversary that attacks the IND-IKGA security of the PAEKS scheme \( \Pi \) with advantage \( \text{Adv}_{\Pi,\text{IND-IKGA}}(\lambda) \). We prove Theorem 2 through the following three games, where we define \( E_i \) to be the event that \( \mathcal{A} \) wins Game_1.

Game_0: This is the original IND-IKGA game, defined in Section 4. By the definition, we have

\[
\text{Adv}_{\Pi,\text{IND-IKGA}}(\lambda) = \left| \Pr[E_0] - \frac{1}{2} \right|
\]

Game_1: This game is identical to Game_0, except that \( k_{R,5,2} \) is randomly chosen from the output range of \( \Omega.\text{Enc}_2 \).

Lemma 1. For all PPT algorithms, \( \mathcal{A}_0 \), \( \left| \Pr[E_0] - \Pr[E_1] \right| \) is negligible if the underlying identity-independent 2-tier IBKEM scheme \( \Omega \) is IND-ID-CPA secure.

Proof of Lemma 1. Suppose there exists an adversary \( \mathcal{A}_0 \) such that \( \left| \Pr[E_0] - \Pr[E_1] \right| \) is non-negligible, then there exists another challenger \( \mathcal{B}_0 \) that can win the IND-ID-CPA game of the underlying identity-independent 2-tier IBKEM with non-negligible advantage.
• **Initialization.** Given a master private key $\Omega . msk$ of the underlying identity-independent 2-tier IBKEM, $B_0$ first chooses two randomness $ran_5$ and $ran_8$ from $\Omega . I D S$, a proper secure hash function $H$, and a pseudorandom generator $F$. $B_0$ runs $(\Psi . mpk, \Psi . msk) \leftarrow \Psi . \text{Setup}(\lambda^4)$. Then, $B_0$ invokes the IND-ID-CPA game of $\Omega$ with $ran_8$ to obtain $(\Omega . mpk, C^*, K^*)$. $B_0$ then computes $(\Omega . ct_{5}, \Omega . r_5) \leftarrow \Omega . \text{Enc}_1(\Omega . mpk)$. Additionally, $B$ invokes $\Omega . \text{Extract}$ oracle of the IND-ID-CPA game on $ran_5$, and is given $\Omega . sk_{ran_5}$. Finally, $B_0$ sends the data sender’s public key $pk_2 \leftarrow (ran_5, \Omega . ct_{5})$, data receiver’s public key $pk_8 \leftarrow (ran_8, \Omega . ct_{8} = C^*, \Psi . mpk)$, and system parameter $pp \leftarrow (\lambda, \Omega . mpk, H, F)$ to $A_{01}$, and keeps $(\Psi . msk, \Omega . r_5, K^*)$ secret.

• **Phase 1.** $A_{01}$ can make polynomially many queries to oracles as was the case in a previous game, $B_0$ responds as follows.

  - $O_{\text{PKGen}}(U)$: For $U \not\in \{S, R\}$, $B_0$ first randomly chooses $ran_U$ from $\Omega . I D S$ and runs $(\Omega . ct_U, \Omega . r_U) \leftarrow \Omega . \text{Enc}_1(\Omega . mpk)$. $B_0$ then invokes $\Omega . \text{Extract}$ oracle of the IND-ID-CPA game on $ran_U$, and is given $\Omega . sk_{ran_U}$. Finally, $B_0$ returns $pk_U \leftarrow (ran_U, \Omega . ct_U)$ to $A_{01}$ and keeps $sk_U \leftarrow (\Omega . sk_{ran_U}, \Omega . r_U)$ secret.

  - $O_{\text{PKGen}}(U)$: For $U \not\in \{S, R\}$, $B_0$ first randomly chooses $ran_U$ from $\Omega . I D S$ and runs $(\Omega . ct_U, \Omega . r_U) \leftarrow \Omega . \text{Enc}_1(\Omega . mpk)$. Then $B_0$ invokes $\Omega . \text{Extract}$ oracle of the IND-ID-CPA game on $ran_U$, and is given $\Omega . sk_{ran_U}$. $B_0$ also computes $(\Psi . mpk_U, \Psi . msk_U) \leftarrow \Psi . \text{Setup}(\lambda^4)$. Finally, $B_0$ returns $pk_U \leftarrow (ran_U, \Omega . ct_U, \Psi . mpk_U)$ to $A_{01}$ and keeps $sk_U \leftarrow (\Omega . sk_{ran_U}, \Omega . r_U, \Psi . msk_U)$ secret.

  - $O_{\text{PAEKS}}(kw, pk_U)$: $B_0$ first computes $k_{S,U,1} \leftarrow \Omega . \text{Enc}_2(\Omega . mpk, ran_S, \Omega . r_U)$ and $k_{S,U,2} \leftarrow \Omega . \text{Dec}(\Omega . sk_{ran_U}, ran_S, \Omega . ct_U)$. Then, $B_0$ computes $shk \leftarrow k_{S,U,1} \oplus k_{S,U,2}$ and computes $f_S \leftarrow F(kw||shk_S)$. Next, $B_0$ randomly chooses $\xi \leftarrow \Psi . FS$, computes $\Psi . ct \leftarrow \Psi . \text{Enc}(\Psi . mpk_U, f_S, \xi)$, and computes $h = H(\Psi . ct, \xi)$. Finally, $B_0$ returns $ct \leftarrow (\Psi . ct, h)$ to $A_{01}$.

  - $O_{\text{Trapdoor}}(kw, pk_U)$: $B_0$ first computes $k_{R,U,1} \leftarrow \Omega . \text{Enc}_2(\Omega . mpk, ran_R, \Omega . r_U)$ and $k_{R,U,2} \leftarrow \Omega . \text{Dec}(\Omega . sk_{ran_U}, ran_R, \Omega . ct_U)$. Then, $B_0$ computes $shk \leftarrow k_{R,U,1} \oplus k_{R,U,2}$ and computes $f_S \leftarrow F(kw||shk_R)$. Next, $B_0$ computes $\Psi . sk \leftarrow \Psi . \text{Extract}(\Psi . msk, f_S)$. Finally, $B_0$ returns a trapdoor $td \leftarrow \Psi . sk$ to $A_{01}$.

• **Challenge.** After the end of **Phase 1**, $A_{01}$ outputs two keywords $kw^*_0, kw^*_1 \in W$ with the following restriction: for $i = 0, 1$, $(kw^*_i, pk_R)$ have not been queried to oracle $O_{\text{PAEKS}}$ in **Phase 1**. $B_0$ then runs the following steps:

  1. Random choose a bit $\beta \in \{0, 1\}$.
  2. Compute $k_{R,S,1} = \Omega . \text{Enc}_2(\Omega . mpk, ran_S, \Omega . r_S)$.
  3. Set $k_{R,S,2} \leftarrow K^*$.
  4. Compute $shk_R \leftarrow k_{R,S,1} \oplus k_{R,S,2}$, where $\oplus$ is an operation compatible with the key space.
  5. Compute $f_R \leftarrow F(kw^*_i||shk_R)$.
  6. Return a challenge trapdoor $td^* \leftarrow \Psi . \text{Extract}(\Psi . msk, f_R)$ to $A_{01}$.

• **Phase 2.** $A_{01}$ can continue to make queries, same as in **Phase 1.** The only restriction is that $A_{01}$ cannot make any query to $O_{\text{PAEKS}}$ on $(kw^*_i, pk_R)$, for $i = 0, 1$.

• **Guess.** $A_{01}$ outputs its guess $b^*$.

If $k_{R,S,2} = K^*$ is generated from $\Omega . \text{Enc}_2(\Omega . mpk, ran_S, \Omega . r_S)$, $B_0$ provides the view of Game 0 to $A_{01}$; if $k_{R,S,2} = K^*$ is a random string sampled from the output range of $\Omega . \text{Enc}_2$ algorithm, then $B_0$ provides the view of Game 1 to $A_{01}$. Hence, if $| Pr[E_0] - Pr[E_1] |$ is non-negligible, $B_0$ has a non-negligible advantage against the IND-ID-CPA game of the underlying identity-independent 2-tier IBKEM scheme. Therefore, the advantage of $A_{01}$ is

$$| Pr[E_0] - Pr[E_1] | \leq \text{Adv}^{\text{IND-ID-CPA}}_{\Omega, B_{01}}(\lambda).$$

\[ \square \]

Game 2: In this game, we make the following minor conceptual change to the aforementioned game. In the challenge phase, the challenger $B$ substitutes the value $td^* \leftarrow \Psi . \text{Enc}(pk_R, f_R, \xi)$ with $td^* \leftarrow \Psi . \text{Enc}(pk_R, f'_R, \xi)$, where $f'_R$ is randomly selected from the output space $Y$ of the underlying pseudorandom generator $F$.

**Lemma 2.** For all PPT algorithms $A_{12}$, $| Pr[E_1] - Pr[E_2] |$ is negligible if the underlying pseudorandom generator $F$ satisfies pseudorandomness.
Proof of Lemma 2. If \( \mathcal{A}_{12} \) can win the IND-IKGA game with non-negligible advantage, then there exists a challenger \( \mathcal{B}_{12} \) that can win the pseudorandom game of the underlying pseudorandom generator with non-negligible advantage. \( \mathcal{B}_{12} \) constructs a hybrid game, interacting with \( \mathcal{A}_{12} \) as follows. Given a challenge string \( T \in \mathcal{Y} \) and the description of a pseudorandom generator \( F' \), \( \mathcal{B}_{12} \) constructs a hybrid game, interacting with \( \mathcal{A}_{12} \) as follows.

- **Initialization.** \( \mathcal{B}_{12} \) chooses the public parameter following the proposed construction, with the following exception: rather than selecting a proper pseudorandom generator from the pseudorandom generator family, \( \mathcal{B}_{12} \) sets \( F' \) as the underlying pseudorandom generator. \( \mathcal{B}_{12} \) then follows the previous game to generate the system parameter \( pp \), data sender’s key pair \((pk_s, sk_s)\), and data receiver’s key pair \((pk_r, sk_r)\). Finally, \( \mathcal{B}_{12} \) sends \((pp, pk_s, pk_r)\) to \( \mathcal{A}_{12} \) and keeps \((msk, sk_s, sk_r)\) secret.

- **Phase 1.** \( \mathcal{A}_{12} \) can make polynomially many queries to oracles as was the case in Game 1.

- **Challenge.** After the end of Phase 1, \( \mathcal{A}_{12} \) outputs two keywords \( kw^*_0, kw^*_1 \in W \) with the following restriction: for \( i = 0, 1 \), \((kw^*_i, pk_r)\) have not been queried to oracle \( O_{PAEKS} \) in Phase 1. \( \mathcal{B}_{12} \) then runs the subsequent steps.

  1. Set \( f^* = T \).
  2. Compute \( td^* = \Psi.\text{Extract}(\Psi.msk, f^*) \).
  3. Return \( td^* \) to \( \mathcal{A}_{12} \).

- **Phase 2.** \( \mathcal{A}_{12} \) can continue to make queries, same as in Phase 1. The only restriction is that \( \mathcal{A}_{12} \) cannot make any query to \( O_{PAEKS} \) on \((kw^*_i, pk_r)\) for \( i = 0, 1 \).

- **Guess.** \( \mathcal{A}_{12} \) outputs its guess \( b' \).

If \( T \) is generated from \( F' \), \( \mathcal{B}_{12} \) provides the view of Game 1 to \( \mathcal{A}_{12} \); if \( T \) is a random string sampled from \( \mathcal{Y} \), then \( \mathcal{B}_{12} \) provides the view of Game 2 to \( \mathcal{A}_{12} \). Hence, if \( |\Pr[E_1] - \Pr[E_2]| \) is non-negligible, \( \mathcal{B}_{12} \) has a non-negligible advantage against the pseudorandom generator security game. Therefore, the advantage of \( \mathcal{A}_{12} \) is

\[
|\Pr[E_1] - \Pr[E_2]| \leq \text{Adv}^{\text{PRG}}_{F, \mathcal{B}_{12}}(\lambda).
\]

Lemma 3. \( \Pr[E_2] = \frac{1}{2} \).

Proof of Lemma 3. The proof of this lemma is intuitive. Because the trapdoor \( td^* \) contains no information regarding the keyword, the adversary can only return \( b' \) by guessing. □

Combining Lemmas 1, 2, and 3, we can conclude that the advantage of \( \mathcal{A} \) in winning the IND-IKGA game is

\[
\text{Adv}^{\text{IND-IKGA}}_{\mathcal{A}}(\lambda) = \left| \Pr[E_0] - \frac{1}{2} \right| = \left| \Pr[E_0] - \Pr[E_1] + \Pr[E_1] - \Pr[E_2] + \Pr[E_2] - \frac{1}{2} \right| 
\leq \text{Adv}^{\text{IND-ID-CPA}}_{\mathcal{A}, \mathcal{B}_{11}}(\lambda) + \text{Adv}^{\text{PRG}}_{F, \mathcal{B}_{12}}(\lambda).
\]

This completes the proof. □
Table 2: Comparison of Security Properties with Other PAEKS Schemes

<table>
<thead>
<tr>
<th>Schemes</th>
<th>IKGA</th>
<th>QR</th>
<th>NTA</th>
<th>Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>HL17 [HL17]</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>ROM</td>
</tr>
<tr>
<td>HMZKL17 [HMZ*18]</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>ROM</td>
</tr>
<tr>
<td>NE18 [NE19]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>ROM</td>
</tr>
<tr>
<td>LHSYS19 [LHS*19]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>ROM</td>
</tr>
<tr>
<td>WZMKH19 [WZM*19]</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>ROM</td>
</tr>
<tr>
<td>LLYSTH19 [LLY*19]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>ROM</td>
</tr>
<tr>
<td>QCHLZ20 [QCH*20]</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>ROM</td>
</tr>
<tr>
<td>PSE20 [PSE20]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>ROM</td>
</tr>
<tr>
<td>Ours</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>SM*</td>
</tr>
</tbody>
</table>

✓: the scheme supports the corresponding feature.
✗: the scheme fails in supporting the corresponding feature.
ROM: random oracle model.
SM: standard model.
QR: Quantum-resistance.
NTA: No trusted authority.
*Our generic construction supports standard model, while our instantiation only supports ROM since the underlying scheme [DLP14] is only proven secure under ROM.

Table 3: Notations of Operations and Their Running Time (ms)

<table>
<thead>
<tr>
<th>Notations</th>
<th>Operations</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_H$</td>
<td>Hash-to-point</td>
<td>47.312</td>
</tr>
<tr>
<td>$T_B$</td>
<td>Bilinear pairing</td>
<td>30.829</td>
</tr>
<tr>
<td>$T_{GM}$</td>
<td>General multiplication over point</td>
<td>0.098</td>
</tr>
<tr>
<td>$T_{EX}$</td>
<td>Modular exponentiation</td>
<td>20.352</td>
</tr>
<tr>
<td>$T_{PA}$</td>
<td>Addition over point</td>
<td>0.006</td>
</tr>
<tr>
<td>$T_HA$</td>
<td>General hash function</td>
<td>0.072</td>
</tr>
<tr>
<td>$T_{PRG}$</td>
<td>Pseudorandom generation</td>
<td>0.047</td>
</tr>
<tr>
<td>$T_{RM}$</td>
<td>Multiplication over polynomial ring</td>
<td>0.309</td>
</tr>
<tr>
<td>$T_{PRA}$</td>
<td>Addition over polynomial ring</td>
<td>0.027</td>
</tr>
<tr>
<td>$T_{SAM}$</td>
<td>Gaussian_Sampler function</td>
<td>2.847</td>
</tr>
</tbody>
</table>

Table 4: Experimentation Platform Information

<table>
<thead>
<tr>
<th>Description</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>ARMv7 Processor rev 4 1.2GHz</td>
</tr>
<tr>
<td>CPU processor number</td>
<td>4</td>
</tr>
<tr>
<td>Operation system</td>
<td>Raspbian GNU/Linux 8</td>
</tr>
<tr>
<td>Linux kernel version</td>
<td>Raspberry 4.4.34-v7+</td>
</tr>
<tr>
<td>Random access memory</td>
<td>1GB</td>
</tr>
<tr>
<td>Solid state disk</td>
<td>16GB</td>
</tr>
</tbody>
</table>
Table 5: Comparison of Needing Operations with Other PAEKS Schemes

<table>
<thead>
<tr>
<th>Schemes</th>
<th>Ciphertext generation</th>
<th>Trapdoor generation</th>
<th>Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>[HL17]</td>
<td>$T_H + 3T_{EX} + T_{GM}$</td>
<td>$T_H + T_{BP} + T_{EX}$</td>
<td>$2T_{BP} + T_{GM}$</td>
</tr>
<tr>
<td>[HMZK17]</td>
<td>$T_H + 3T_{BP} + 5T_{EX} + 2T_{PA} + 2T_{HA}$</td>
<td>$T_H + T_{BP} + 3T_{EX} + 2T_{PA} + 2T_{HA}$</td>
<td>$2T_{BP} + 2T_{EX} + T_{GM} + 2T_{PA} + 2T_{HA}$</td>
</tr>
<tr>
<td>[NE18]</td>
<td>$T_H + 3T_{EX} + T_{GM}$</td>
<td>$T_H + T_{BP} + T_{EX}$</td>
<td>$2T_{BP} + T_{GM}$</td>
</tr>
<tr>
<td>[LHSYS19]</td>
<td>$2T_H + 2T_{BP} + 3T_{EX}$</td>
<td>$4T_H + T_{BP} + T_{GM}$</td>
<td>$2T_{BP} + T_{GM} + 2T_{EX}$</td>
</tr>
<tr>
<td>[WZMKH19]</td>
<td>$T_H + 6T_{EX} + 2T_{PA} + 2T_{HA}$</td>
<td>$T_H + T_{BP} + 9T_{EX} + 4T_{PA} + T_{HA}$</td>
<td>$2T_{BP} + 5T_{EX} + 2T_{GM} + 2T_{PA} + 2T_{HA}$</td>
</tr>
<tr>
<td>[LLYSTH19]</td>
<td>$T_H + 3T_{EX} + T_{PA}$</td>
<td>$T_H + T_{BP} + 4T_{EX} + 2T_{PA}$</td>
<td>$2T_{BP} + 2T_{EX} + T_{GM} + 2T_{PA}$</td>
</tr>
<tr>
<td>[QCHLZ20]</td>
<td>$3T_H + 2T_{BP} + 3T_{EX} + T_{HA}$</td>
<td>$3T_H + T_{BP} + 2T_{EX}$</td>
<td>$T_H + T_{BP}$</td>
</tr>
<tr>
<td>[PSE20]</td>
<td>$2T_{HA} + T_{PRG} + 4T_{PRM} + 5T_{PRA}$</td>
<td>$T_{PRG} + 2T_{PRM} + 2T_{PRA} + T_{SAM}$</td>
<td>$T_{HA} + T_{PRA} + T_{PRM}$</td>
</tr>
</tbody>
</table>

Figure 3: Comparison of computational costs with other PAEKS schemes.
et al. scheme with Ducas et al.’s anonymous IBE [DLP14] to instantiate a quantum-resistant PAEKS scheme. More precisely, following the idea in [BC18], we tweak [DLP14] to obtain an NTRU-based identity-independent 2-tier IBKEM. Then, we combine this IBKEM scheme with Ducas et al.’s anonymous IBE [DLP14] to instantiate a quantum-resistant PAEKS scheme.

7 Concrete Instantiation

In this section, we give a concrete instantiation by adopting Ducas et al.’s IBE [DLP14], which is secure under the NTRU assumption and has proved anonymous by [BOY20]. More precisely, following the idea in [BC18], we tweak [DLP14] to obtain an NTRU-based identity-independent 2-tier IBKEM. Then, we combine this IBKEM scheme with Ducas et al.’s anonymous IBE [DLP14] to instantiate a quantum-resistant PAEKS scheme.

- Setup(1^4): Given a security parameter λ, this algorithm runs as follows.
  1. Select N = poly(λ), and a large prime q.
  2. Compute (h, B) ← Basis_Generation(N, q).
  3. Choose a proper PRG F and two secure hash functions \( H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_q^N \) and \( H_2 : \{0,1\}^* \rightarrow \{0,1\}^N \).
  4. Outputs pp := (N, q, h, F, H_1, H_2) and master private key msk := B. Note that msk is kept secret by the trusted authority.

- KeyGen_S(pp, msk): Given a system parameter pp and a master private key msk, data sender and trusted authority interact as follows.
  1. Data sender first chooses \( r_S, e_S \leftarrow \{-1,0,1\}^N \), \( v_S \leftarrow \mathcal{R}_q \), computes \( u_S \leftarrow r_S + h + e_S \in \mathcal{R}_q \), and randomly choose \( t_S \leftarrow \mathbb{Z}_q^N \). Then, he/she submits \((t_S, u_S, v_S)\) to trusted authority. The trusted authority computes the following steps.
    (a) Compute \( (s_{S,1}, s_{S,2}) \leftarrow \text{Gaussian Sampler}(B, \sigma, (t_S,0)) \), such that \( s_{S,1} + s_{S,2} + h = t_S \).
    (b) Return \((s_{S,1}, s_{S,2})\) to data sender.
  2. Data sender outputs his/her public key \( pk_S := (t_S, u_S, v_S) \) and keeps private key \( sk_S := (s_{S,1}, s_{S,2}, r_S) \) secret.

- KeyGen_R(pp, msk): Given a system parameter pp and a master private key msk, data receiver and trusted authority interact as follows.
  1. Data receiver first chooses \( r_R, e_R \leftarrow \{-1,0,1\}^N \), \( v_R \leftarrow \mathcal{R}_q \), computes \( u_R \leftarrow r_R + h + e_R \in \mathcal{R}_q \), and randomly chooses \( t_R \leftarrow \mathbb{Z}_q^N \). Then, he/she submits \((t_R, u_R, v_R)\) to trusted authority. The trusted authority computes the following steps.
    (a) Compute \( (s_{R,1}, s_{R,2}) \leftarrow \text{Gaussian Sampler}(B, \sigma, (t_R,0)) \), such that \( s_{R,1} + s_{R,2} + h = t_R \).
    (b) Return \((s_{R,1}, s_{R,2})\) to data receiver.
  2. Data receiver then computes \((h_R, B_R) \leftarrow \text{Basis Generation}(N, q)\).
  3. Data receiver outputs his/her public key \( pk_R = (t_R, u_R, v_R, h_R) \) and keeps private key \( sk_R = (s_{R,1}, s_{R,2}, r_R, B_R) \) secret.

<table>
<thead>
<tr>
<th>Schemes</th>
<th>Ciphertext</th>
<th>Trapdoor</th>
</tr>
</thead>
<tbody>
<tr>
<td>HL17 [HL17]</td>
<td>2</td>
<td>G_1</td>
</tr>
<tr>
<td>HMZKL17 [HMZ'18]</td>
<td></td>
<td>G_1</td>
</tr>
<tr>
<td>NE18 [NE19]</td>
<td></td>
<td>G_1</td>
</tr>
<tr>
<td>LHSYS19 [LHS'19]</td>
<td>2</td>
<td>G_1</td>
</tr>
<tr>
<td>WZMKH19 [WZM'19]</td>
<td>2</td>
<td>G_1</td>
</tr>
<tr>
<td>LLYSTH19 [LLY'19]</td>
<td></td>
<td>G_1</td>
</tr>
<tr>
<td>QCHLZ20 [QCH'20]</td>
<td></td>
<td>G_1</td>
</tr>
<tr>
<td>PSE20 [PSE20]</td>
<td>2</td>
<td>G_1</td>
</tr>
<tr>
<td>Ours</td>
<td>2N</td>
<td>q</td>
</tr>
</tbody>
</table>

- N: lattice dimension.
- G_1, G_T: cyclic group.
- r: group order.
- q: module.

Table 6: Comparison of Communication Costs with Other PAEKS Schemes
• PAEKS(pp, pkS, skS, pkR, kw): Given a system parameter pp, data sender’s public key pkS and private key skS, data receiver’s public key pkR, and a keyword kw ∈ \{0, 1\}*, data sender runs the following steps.

1. \(k_{SR,1} = [(2/q) \cdot (uR - uR \cdot s_{S,2})]\).
2. \(k_{SR,2} = [(2/q) \cdot (uS - rS \cdot fR)]\).
3. \(shkS \leftarrow k_{SR,1} \oplus k_{SR,2}\).
4. Compute \(fS \leftarrow F(kw||shkS)\).
5. Choose a random \(\xi \leftarrow \{0, 1\}^N\), and randoms \(r, e_1, e_2 \leftarrow \{-1, 0, 1\}^N\).
6. Compute \(u_{kw} \leftarrow r \cdot hR + e_1 \in R_q\).
7. Compute \(v_{kw} \leftarrow r \cdot H_1(fS) + e_2 + \lfloor q/2 \rfloor \cdot \xi \in R_q\).
8. Compute \(h = H_2(u_{kw}, v_{kw}, \xi)\).
9. Output a searchable ciphertext \(ct := (u_{kw}, v_{kw}, h)\).

• Trapdoor(pp, pkR, skR, pkS, kw): Given a system parameter pp, data receiver’s public key pkR and private key skR, data sender’s public key pkS, and a keyword kw ∈ \{0, 1\}*, data receiver runs the following steps.

1. \(k_{RS,1} = [(2/q) \cdot (uS - uS \cdot s_{R,2})]\).
2. \(k_{RS,2} = [(2/q) \cdot (uR - rR \cdot fS)]\).
3. \(shkR \leftarrow k_{RS,1} \oplus k_{RS,2}\).
4. Compute \(fR \leftarrow F(kw||shkR)\).
5. Compute \((s_{kw,1}, s_{kw,2}) \leftarrow (H_1(fR), 0) - \text{Gaussian Sampler}(B_0, \sigma, (H_1(fR), 0))\).
6. Output a trapdoor \(td := s_{kw,2}\).

• Test(pp, ct, td): Given a system parameter pp, a searchable ciphertext \(ct = (u_{kw}, v_{kw}, h)\), and a trapdoor \(td = s_{kw,2}\), cloud server works as follows.

1. Compute \(\bar{\xi} = [(2/q) \cdot (v_{kw} - u_{kw} \cdot s_{kw,2})]\).
2. If \(H_2(u_{kw}, v_{kw}, \bar{\xi}) = h\), output 1; otherwise, output 0.

Lemma 4. Our concrete instantiation is correct if the parameter \(q\) is large enough to remove the noise items.

Proof. First of all, we show that \(k_{SR,1} = k_{RS,2}\) and \(k_{SR,2} = k_{RS,1}\):
Then, if the keywords kw in ct and td are the same, since Eq. (1) and Eq. (2) are holds, we have:

\[
\begin{align*}
f_S &= F(kw||shk_S) = F(kw|(ks_{S,R_1} \oplus ks_{R,R_2})) \\
&= F(kw|(k_{R,S_1} \oplus k_{R,S_2})) = F(kw||shk_R) \\
&= f_R
\end{align*}
\]

With the result of Eq. (3), we have:

\[
\xi' = [\left(\frac{2}{q}\right) \cdot (v_{kw} - u_{kw} \ast s_{kw,2})] \\
= [\left(\frac{2}{q}\right) \cdot (r \ast H_1(f_2) + e_2 + \lfloor q/2 \rfloor \cdot \xi - r \ast h_{R} \ast s_{kw,2} - e_1 \ast s_{kw,2})] \\
= [\left(\frac{2}{q}\right) \cdot (r \ast H_1(f_2) + e_2 + \lfloor q/2 \rfloor \cdot \xi - r \ast (H_1(f_R) - s_{kw,1}) - e_1 \ast s_{kw,2})] \\
= [\left(\frac{2}{q}\right) \cdot (r \ast H_1(f_2) - r \ast H_1(f_R) + \lfloor q/2 \rfloor \cdot \xi + e_2 - r \ast s_{kw,1} - e_1 \ast s_{kw,2})] \\
= [\left(\frac{2}{q}\right) \cdot \left(\lfloor q/2 \rfloor \cdot \xi + e_2 - r \ast s_{kw,1} - e_1 \ast s_{kw,2}\right)]
\]

Therefore, \(H_2(u_{kw}, v_{kw}, \xi') = \xi\) holds.

\[\square\]

8 Comparison and Analysis

To the best of our knowledge, there is no quantum-resistant PAEKS currently. Although existing PAEKS schemes [HMZ*18, LLY*19, NE19, PSE20, QCH*20, LHS*19, WZM*19] can defend against IKGA, these schemes cannot defend against quantum attacks because the security of these schemes is based on the discrete logarithm assumption. In this section, we first compare our proposed instantiation with these existing schemes with respect to their security properties. We then compared these schemes with respect to their computational and communication complexities.

Table 2 lists the results of our comparison between our instantiation and its counterpart PAEKS schemes with respect to their security properties. Because our instantiation inherits the security of [DLP14], it can be considered to be based on the lattice hard assumption. In other words, only our instantiation has the ability to resist quantum attacks and IKGA simultaneously. However, our current construction requires a trusted authority to generate the user’s private key, which other PAEKS schemes do not require it.

We subsequently conducted such a comparison with respect to computational complexity when generating searchable ciphertexts and trapdoors and testing. For simplicity, we only considered the time-consuming operations listed in Table 3. Experiments simulating these operations were performed on an IoT device (Raspberry Pi 3 Model B) where the specification of the device is detailed in Table 4. In particular, the operations of \(T_H, T_{BP}, T_{GM}, T_{EX},\) and \(T_{PA}\) were obtained by using a pairing-based cryptography library (PBC)—under Type-A pairing with a 160-bit group order, 512-bit base field, and 1024-bit group element for \(G_1\) and \(G_T\) [Lyn07]. As for \(T_{SAM}, T_{PRM}, T_{PR},\) we simulated it by using SAFECrypto project\(^1\) [OOM*16] that implementing [DLP14] with its suggested parameters, i.e., \(N = 512, q = 4206593, l = 18,\) and \(N^{th}\) root of unity = 990. Moreover, \(T_{PQC}\) was obtained using the AES-256 algorithm\(^2\), and \(T_{SHA}\) was simulated using the SHA3-256 algorithm\(^3\). The computational costs for the methods are compared in Table 5. The results indicate that our instantiation took the least time to generate the ciphertext and trapdoor as well as to perform tests (only take 1.562, 3.566, 0.408 (ms), respectively); such speed was due to our method not requiring any time-consuming operations, such as hash-to-point, bilinear pairing, and modular exponentiation.

Additionally, we also conducted such a comparison with respect to communication complexity (which was indicated by the size of the ciphertext and trapdoor). The comparison results are detailed in Table 6. For the pairing-based schemes, the pairing operation is represented by \(e : G_1 \times G_1 \rightarrow G_T,\) where \(G_1\) and \(G_T\) are

---

\(^1\)https://github.com/safecrypto/libsafercrypto  
\(^2\)https://github.com/kokke/tiny-AES-c  
\(^3\)https://github.com/brainhub/SHA3IUF
1024-bit elements. Moreover, because the group order of the pairing $r$ is 160 bit; therefore, $|r| = 160$. For our instantiation, $N = 512$ and $|q| = 23$. To ensure security, our instantiation must be set in high dimensions. Therefore, in contrast to its counterpart schemes, our instantiation yielded larger ciphertext and trapdoor sizes, which are $2N|q| + N = 24064$ bits and $N|q| = 11776$ bits, respectively.

9 Conclusion and Future Work

In this work, we introduced a new method for constructing a generic PAEKS scheme, which is secure against IND-CKA and IND-IKGA under multi-user context in standard model, if the underlying building blocks are secure under the standard model. In addition, we provided a concrete instantiation based on the lattice hard assumption which is secure under ROM. Compared with current PAEKS schemes, our instantiation is not only the first PAEKS scheme that is quantum-resistant but also the most efficient scheme with respect to computational cost. Because the current construction is modeled by a variant public-key model that requires an additional trusted authority. We will continue to investigate how to remove this requirement to reduce communication costs and increase security.

Acknowledgment

The authors thank the anonymous reviewers 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


[BRPL06] Jin Wook Byun, Hyun Suk Rhee, Hyun-A Park, and Dong Hoon Lee. Off-line keyword guessing attacks on recent keyword search schemes over encrypted data. In Willem Jonker and Milan Petkovic, editors, Proceedings of the Third VLDB Workshop on Secure Data Management, SDM 06,


