Post-Quantum Public-key Authenticated Searchable Encryption with Forward Security: General Construction, Implementation, and Applications

Shiyuan Xu[1], Yibo Cao[2], Xue Chen[1,3], Siu-Ming Yiu[1⋆], and Yanmin Zhao[1]

[1] Department of Computer Science, The University of Hong Kong, Pok Fu Lam, Hong Kong
{syxu2, smyiu, ymzhao}@cs.hku.hk

[2] School of Cyberspace Security, Beijing University of Posts and Telecommunications, Beijing, China
a18613361692@163.com

[3] Department of Computing, The Hong Kong Polytechnic University, Hung Hom, Hong Kong
xue-serena.chen@connect.polyu.hk

Abstract. Public-key encryption with keyword search was first proposed by Boneh et al. (EUROCRYPT 2004), achieving the ability to search for ciphertext files. Nevertheless, this scheme is vulnerable to *inside keyword guessing attacks* (IKGA). Public-key authenticated encryption with keyword search (PAEKS), introduced by Huang et al. (Inf. Sci. 2017), on the other hand, is secure against IKGA. Nonetheless, it is susceptible to *quantum computing attacks*. Liu et al. and Cheng et al. addressed this problem by reducing to the lattice hardness (AsiaCCS 2022, ESORICS 2022). Furthermore, several scholars pointed out that the threat of secret key exposure delegates a severe and realistic concern, potentially leading to *privacy disclosure* (EUROCRYPT 2003, Compt. J. 2022). As a result, research focusing on mitigating key exposure and resisting quantum attacks for the PAEKS primitive is significant and far-reaching.

In this work, we present the first instantiation of post-quantum PAEKS primitive that is forward-secure and does not require trusted authorities, mitigating the secret key exposure while ensuring quantum-safe properties. We extended the scheme of Liu et al. (AsiaCCS 2022), and proposed a novel post-quantum PAEKS construction, namely FS-PAEKS. To begin with, we introduce the binary tree structure to represent the time periods, along with a lattice basis extension algorithm, and SamplePre algorithm to obtain the post-quantum one-way secret key evolution, allowing users to update their secret keys periodically. Furthermore, our scheme is proven to be IND-CKA, IND-IKGA, and IND-Multi-CKA in the quantum setting. In addition, we also compare the security of our primitive in terms of computational complexity and communication overhead with other top-tier schemes, and provide implementation details of the ciphertext generation and test algorithms. The proposed FS-PAEKS is more efficient than the FS-PEKS scheme (IEEE TDSC 2021). Lastly, we demonstrate three potential application scenarios of FS-PAEKS.

Keywords: Public-key authenticated encryption with keyword search · Post-quantum cryptography · Forward security · Multi-ciphertext indistinguishability · Trapdoor privacy · Generic construction.

⋆ Corresponding author

1 Introduction

Traditional public-key encryption with keyword search (PEKS) primitive contains three entities, that is, data owner, data user, and cloud server [1]. PEKS scheme realizes that encrypted data can easily be retrieved by the specific user through a specific trapdoor, which not only protects the data privacy but also realizes the searchability. A fundamental security criterion for PEKS is to against the chosen keywords attacks (CKA) [2]. Nevertheless, Byun et al. formalized the notation of trapdoor privacy (TP) for the PEKS scheme since if it only considers the CKA, the protocol may be threatened by the inside keyword guessing attacks (IKGA) [3]. To circumvent this problem, Huang et al. initialized a novel variant of PEKS, namely, public-key authenticated encryption with keyword search (PAEKS), combining the message authentication technique into the ciphertext generation algorithm. In this way, the trapdoor can merely be valid to the authenticated ciphertext for a specific sender. Numerous scholars commenced their research works on the PAEKS primitive due to its high security [4–9].

However, the above-mentioned PAEKS protocols are totally on the basis of the discrete logarithm assumption, which is vulnerable to quantum computing attacks. Liu et al. constructed a lattice-based PAEKS primitive that offers both CKA and IKGA security while also being resistant to quantum computing attacks [10].

Unfortunately, the security of ciphertext may be compromised if the secret key of a receiver is leaked due to inadequate storage or malicious actions by adversaries. To address this issue, Bellare et al. initially introduced the notation of forward security in digital signatures [11], which was later adapted by Canetti et al. for use in a forward secure public key encryption scheme [12, 13]. This protocol periodically updates the secret key, therefore even if it is compromised in one period, the security of other periods remains intact.

1.1 Motivation

As inappropriate storage of secret keys may lead to their compromise by malicious attackers, it is essential to update them within a certain period to ensure forward security. Zhang et al. have formalized the FS-PEKS scheme, achieving forward security, nevertheless, one disadvantage of this scheme is that a malicious attacker may acquire the keyword from the trapdoor [14]. In contrast, Jiang et al. presented a forward secure scheme for PAEKS, without considering the quantum computing attacks [15]. Among that, their constructions still need a trusted authority to calculate secret keys, which will result in additional storage overhead.

Huang et al. subsequently presented a PAEKS primitive, which was reduced to be secure under the discrete logarithm assumption [16]. However, with the advancement of quantum computers, Shor generalized a quantum algorithm, demonstrating the feasibility of solving classical cryptographic primitives in probabilistic polynomial times [17,18]. Consequently, classical PAEKS schemes are now vulnerable. Hence, several scholars transformed the traditional PAEKS primitive into the quantum-resistant PAEKS protocol and formalized the generic constructions based on lattice hardness [10,19]. Nevertheless, their schemes contain flaws due to the secret key leakage problem.

Therefore, with the aforementioned issues, it motivates the following question:

Can we construct and instantiate a generic post-quantum forward-secure PAEKS satisfied CI, TP, MCI security without trusted settings to mitigate the secret key leakage problem?

1.2 Our Contributions

The contributions of this work can be summarized as follows:

- We generalize the first PAEKS with forward security instantiation in the context of lattice without trusted authorities, mitigating the secret key exposure while enjoying quantum-safe. Our primitive extends Liu et al.'s scheme [10], and proposes a novel post-quantum PAEKS construction, namely FS-PAEKS. In addition, we formalize the CI, TP, and MCI security of the proposed FS-PAEKS primitive. Eventually, we give three potential application scenarios to demonstrate the utility of the proposed primitive.
- The proposed FS-PAEKS scheme enjoys quantum-safe forward security. We introduce a binary tree structure to update the receiver's secret key with different time periods. This property ensures that exposing the secret key corresponding to a specific time period does not enable an adversary to "crack" the primitive for any previous time period due to its one-way nature. Additionally, we further employ the minimal cover set technique to achieve secret key updating periodically for the receiver based on the key evolution mechanism. Finally, we utilize the lattice basis extension technique to maintain quantum-safe for updating secret keys.
- The proposed FS-PAEKS scheme can be proven secure in strong security models. Firstly, the FS-PAEKS does not need a trusted setup assumption in the initial phase. Furthermore, we guarantee that the trapdoor is valid from the receiver to prevent adversaries from adaptively accessing oracles to obtain the ciphertext for any keyword. This is achieved by ensuring that the ciphertext can only be obtained by a valid sender. Consequently, we introduce the pseudo-random smooth projective hash function (SPHF) to achieve the above property and forward-secure trapdoor privacy under IND-IKGA. In addition, the proposed scheme also achieves the forward-secure multi-ciphertext indistinguishability under IND-Multi-CKA due to its probability and IND-CKA of PAEKS.
- Ultimately, we give the security properties comparison with other PEKS and PAEKS primitives to show the superiority of our proposed primitive. Besides, we compare with Behnia et al.'s scheme [20], Zhang et al.'s scheme [14], and Liu et al.'s scheme [10] in terms of computational complexity and communication overhead. Both the theoretical and practical results illustrate that our proposed scheme outperforms others in terms of security (with little overheads than PAEKS but efficient than FS-PEKS scheme). To further demonstrate the versatility of our proposed primitive, we provide three potential application scenarios in which it can be effectively utilized.

1.3 Overview of Technique

Technical Roadmap. Informally speaking, constructing a forward-secure PAEKS primitive in the context of the lattice is a combination of PEKS scheme, public key encryption scheme, smooth projective hash functions (SPHF), binary tree structure, and lattice basis extension algorithm. More concretely, we begin by revisiting the post-quantum PAEKS primitive proposed by Liu et al. as the basic structure [10]. Next, we employ the SPHF technique to transform the encryption scheme into IND-CCA secure. We then take advantage of the hierarchical structure of the binary tree to represent time periods and utilized $\mathsf{node}(t)$ to represent the smallest minimal cover set for the secret key update periodically, following the approach outlined in Cash et al. [21]. To the best of our knowledge, it is the most efficient mechanism to realize key updates and it serves as a stepping stone toward our goal. Finally, we introduced the ExtBasis and SamplePre algorithms to facilitate the post-quantum one-way secret key evolution.

Smooth projective hash functions. Smooth projective hash functions (abb. SPHF), initially proposed by Cramer et al. [22], are utilized to transform one encryption primitive from IND-CPA to IND-CCA, which significantly enhanced the security level. Moreover, numerous scholars extended the SPHF tool to realize password-authenticated key exchange protocols [23–28]. We use a variant kind of SPHF, say "word-independent" SPHF, proposed by Katz et al. [29] for primitive construction. Generally speaking, the "word-independent" SPHF scheme includes five algorithms defined for the NP language $\mathcal{L}$ over one domain $\mathcal{X}$. The formal definition specifies below.
We define a language family ($\mathcal{L}_{Para_l,Trap_l}$) indexed by the language parameter $Para_l$ and language trapdoor $Trap_l$. Besides, we consider an NP language family ($\tilde{\mathcal{L}}_{Para_l}$) with witness relation $\tilde{\mathcal{K}}_{Para_l}$, s.t.

$$\tilde{\mathcal{L}}_{Para_l} := \{\chi \in \mathcal{X}_{Para_l} | \exists \omega, \tilde{\mathcal{K}}_{Para_l}(\chi, \omega) = 1\} \subseteq \mathcal{L}_{Para_l,Trap_l} \subseteq \mathcal{X}_{Para_l},$$

where $\mathcal{X}_{Para_l}$ is a family of sets. In addition, the membership in $\mathcal{X}_{Para_l}$ and $\tilde{\mathcal{K}}_{Para_l}$ can be checked in polynomial time with $Para_l$, and $\mathcal{L}_{Para_l,Trap_l}$ can be checked in polynomial time with $Para_l, Trap_l$. We describe the approximate "word-independent" SPHF scheme below.

- $\mathsf{Setup}(1^\lambda)$: Given λ as a security parameter, the PPT algorithm $\mathsf{Setup}(1^\lambda)$ will output a language parameter $Para_l$.
- $\mathsf{KeyGen}_{\mathsf{Hash}}(Para_l)$: Given a language parameter $Para_l$, the PPT algorithm $\mathsf{KeyGen}_{\mathsf{Hash}}(Para_l)$ outputs hk as the hashing key.
- $\mathsf{KeyGen}_{\mathsf{Proj}}(\mathsf{hk}, Para_l)$: Given hk and $Para_l$, the PPT algorithm $\mathsf{KeyGen}_{\mathsf{Proj}}(\mathsf{hk}, Para_l)$ outputs the projection key pk.
- $\mathsf{Hash}(\mathsf{hk}, Para_l, \chi)$: Given hk, $Para_l$ and a word $\chi \in \mathcal{X}_{Para_l}$, the deterministic algorithm $\mathsf{Hash}(\mathsf{hk}, Para_l, \chi)$ outputs $\mathsf{Hash} \in \{0,1\}^\delta$ as a hash value, where $\delta \in \mathbb{N}$.
- $\mathsf{ProjHash}(\mathsf{pk}, Para_l, \chi, \omega)$: Given pk, $Para_l$, $\chi \in \tilde{\mathcal{L}}_{Para_l}$ and a witness ω, the deterministic algorithm $\mathsf{ProjHash}(\mathsf{pk}, Para_l, \chi, \omega)$ outputs $\mathsf{ProjHash} \in \{0,1\}^\delta$ as a projected hash value, where $\delta \in \mathbb{N}$.

Informally speaking, an approximate "word-independent" SPHF protocol should satisfies the two attributes:

- ϵ-approximate correctness: Given one word $\chi \in \tilde{\mathcal{L}}_{Para_l}$ as well as the corresponding witness ω, the SPHF scheme is ϵ-approximate correct when:

$$\Pr[\mathsf{HD}(\mathsf{Hash}(\mathsf{hk}, Para_l, \chi), \mathsf{ProjHash}(\mathsf{pk}, Para_l, \chi, \omega)) > \epsilon \cdot \delta] \approx 0,$$

  where $\mathsf{HD}(a, b)$ means the hamming distance between two elements a and b.
- Pseudo-randomness: For some $\delta \in \mathbb{N}$, if one word $\chi \in \tilde{\mathcal{L}}_{Para_l}$, its hash value Hash is indistinguishable from a random element in $\{0,1\}^\delta$; otherwise, its hash value Hash is statistically indistinguishable from one random element chosen in $\{0,1\}^\delta$.

Binary tree for representing time periods. We use binary tree encryption primitive for enrolling time periods [13]. Informally, we define numerous time periods $t \in \{0, 1, \cdots, 2^d - 1\}$, where d is the depth of the binary from the root node to the deepest leaf. In this paper, the time period t will be described in binary expression $t = (t_1 t_2 \cdots t_d)$. For example, if the depth is three and the last leaf can be described as $t = (111)$.
On each time period, it only has one path from the root node to the current leaf node and we

define $\Theta^{(i)} = (\theta^{(1)}\theta^{(2)}\cdots\theta^{(i)})$, $i \in [1, d]$ as the path, where $\theta^{(i)} = 0$ if the i-th level node is the left leaf and $\theta^{(i)} = 1$ if the i-th level node is the right leaf. We also define $\mathsf{node}(t)$ to represent the smallest minimal cover set containing one ancestor of all leaves on the time period t and after the time period t, say including $\{t, t+1, \cdots, 2^d - 1\}$.
For simple understanding, we give one example in Fig.1, describing a $d = 4$ binary tree with 16 time periods in total. In this figure, we show the meaning of $\mathsf{node}(t)$ as: $\mathsf{node}(0000) = \{\mathsf{root}\}$, $\mathsf{node}(0001) = \{0001, 001, 01, 1\}$, $\mathsf{node}(0010) = \{001, 01, 1\}$, $\mathsf{node}(0011) = \{0011, 01, 1\}$, $\mathsf{node}(0100) = \{01, 1\}$, $\mathsf{node}(0101) = \{0101, 011, 1\}$, $\mathsf{node}(0110) = \{011, 1\}$, $\mathsf{node}(0111) = \{0111, 1\}$, $\mathsf{node}(1000) = \{1\}$, $\mathsf{node}(1001) = \{1001, 101, 11\}$, $\mathsf{node}(1010) = \{101, 11\}$, $\mathsf{node}(1011) = \{1011, 11\}$, $\mathsf{node}(1100) = \{11\}$, $\mathsf{node}(1101) = \{1101, 111\}$, $\mathsf{node}(1110) = \{111\}$, $\mathsf{node}(1111) = \{1111\}$.

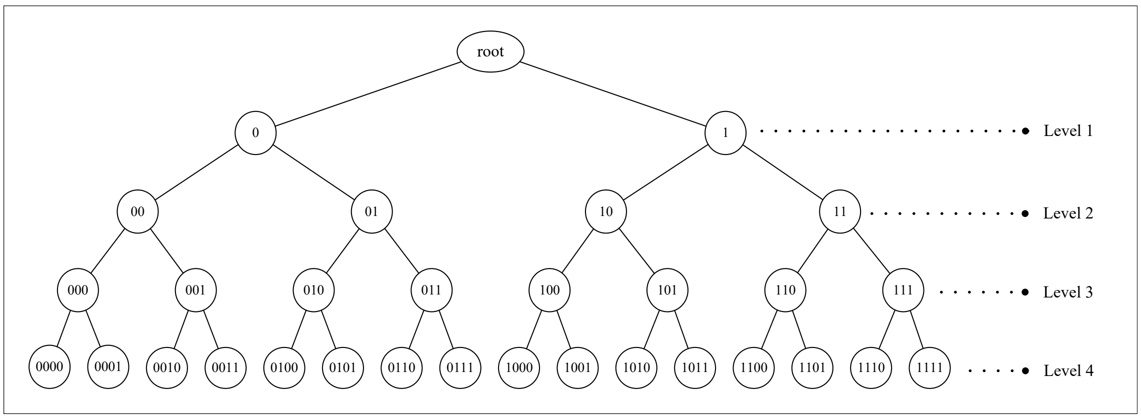

Fig. 1. Binary Tree of depth $d = 4$ with binary expression time period (node).

Lattice basis extension. We use the lattice basis extension primitive to realize the secret key one-way evolutionary mechanism (See Lemma 5 in Section 2.3). More concretely, we discretize the time period to 2^d segments, where d means the total depth of a binary tree. The matrix $\mathbf{M}_R$ is the public key for receiver and the matrix $\mathbf{S}_{\Theta^{(i)}}$ is the trapdoor, where $\Theta^{(i)} := (\theta_1, \theta_2, \cdots, \theta_j, \theta_{j+1}, \cdots, \theta_i)$. Consequently, the updated trapdoor can be calculated by its any ancestor's trapdoor, and root node is the trapdoor of the original ancestor.

We first define $F_{\Theta^{(i)}} := [\mathbf{M}_R \parallel A_1^{(\theta_1)} \parallel A_2^{(\theta_2)} \parallel \cdots \parallel A_i^{(\theta_i)}]$ as the corresponding matrix of $\Theta^{(i)}$. For any depth $j < i$, where $j, i \in [1, d]$, given the trapdoor $\mathbf{S}_{\Theta^{(j)}}$ on time j, we have:

$$\mathbf{S}_{\Theta^{(i)}} \leftarrow \mathsf{ExtBasis}(F_{\Theta^{(i)}}, \mathbf{S}_{\Theta^{(j)}})$$

After that, we specify the secret key update process as below.

$$sk_R(t) := (\mathbf{h}_R, \{\mathbf{r}_{R,1}\}, \{\mathbf{r}_{R,2}\}, \cdots, \{\mathbf{r}_{R,\kappa}\}, \mathbf{S}_{\Theta^{(i)}}),$$

where $\Theta^{(i)} \in \mathsf{node}(t)$ as the receiver's secret key on time t. Each node has the corresponding secret key in a binary tree. Receiver will update $sk_R(t)$ to $sk_R(t+1)$ through processing

$$sk_R(t+1) := (\mathbf{h}_R, \{\mathbf{r}_{R,1}\}, \{\mathbf{r}_{R,2}\}, \cdots, \{\mathbf{r}_{R,\kappa}\}, \mathbf{S}_{\Theta^{(i)}}),$$

where $\Theta^{(i)} \in \mathsf{node}(t+1)$.

1.4 Additional Related Works

Lattice-based PAEKS. Boneh et al. constructed the concept of public-key encryption with keyword search in 2004 [1]. Zhang et al. argued that the initial security model for keyword privacy is not complete and then defined a new security model to solve it [30]. However, the basic PEKS primitive cannot resist the IKGA since an inside adversary may deduce the keyword from a specific trapdoor. Huang et al. formalized the PAEKS protocol to solve this problem by combining keyword authentication with PEKS [16]. Nevertheless, Liu et al. and Cheng et al. introduced lattice-based PAEKS primitive to achieve quantum-safe [10, 31]. Many researchers utilized the PAEKS scheme to preserve privacy for the Internet of Things [7, 32, 33].

Forward Security. Forward security in the public-key cryptosystem was initialized by [12, 13]. Zeng et al. introduced the forward-security notation into the PEKS scheme for cloud computing [34]. Zhang et al. formalized the first lattice-based PEKS primitive with forward security [14]. After that, Yang et al. extended the FS-PEKS and then constructed a lattice-based forward secure identity-based encryption with PEKS, namely, FS-IBEKS [35]. Recently, Jiang et al. proposed a forward secure public-key authenticated encryption with conjunctive keyword search [15], but without considering the quantum computing attacks.

1.5 Outline of this paper

The rest of this paper is structured as follows. Section 2 covers the preliminary knowledge. In section 3, we present the syntax of forward-secure PAEKS primitive and its security models. Our proposed construction will be elaborated in Section 4, while the general instantiation of our FS-PAEKS scheme will be specified in Section 5. In Section 6, we give the parameters setting and the correctness proof. The security analysis and implementation with comparison are in Sections 6 and 7, respectively. Section 9 shows the applications of FS-PAEKS. Finally, we conclude this paper in Section 10.

2 Preliminaries

We hereby introduce the notations used in this paper and some fundamental knowledge of PEKS and lattice cryptographic primitives in this section.

2.1 Public-key Encryption with Keyword Search scheme

Public-key encryption with keyword search (abb. PEKS) was initially proposed by Boneh et al. [1]. A standard PEKS scheme consists of four algorithms:

- $(\mathsf{pk}_{\mathsf{PEKS}}, \mathsf{sk}_{\mathsf{PEKS}}) \leftarrow \mathsf{KeyGen}(\lambda)$: Given one security parameter λ, this probabilistic-polynomial time (PPT) algorithm outputs $\mathsf{pk}_{\mathsf{PEKS}}$ and $\mathsf{sk}_{\mathsf{PEKS}}$ as the public key and secret key for encryption and decryption, respectively.
- $\mathsf{ct}_{\mathsf{PEKS},kw} \leftarrow \mathsf{PEKS}(\mathsf{pk}_{\mathsf{PEKS}}, kw)$: After input the public key $\mathsf{pk}_{\mathsf{PEKS}}$ and the keyword kw, this PPT algorithm will output the ciphertext $\mathsf{ct}_{\mathsf{PEKS},kw}$.

- $\mathbf{Trap}_{\mathsf{PEKS},kw'} \leftarrow \mathsf{Trapdoor}(\mathsf{sk}_{\mathsf{PEKS}}, kw')$: Having input the secret key $\mathsf{sk}_{\mathsf{PEKS}}$ and the keyword kw', this PPT algorithm outputs the trapdoor $\mathbf{Trap}_{\mathsf{PEKS,kw'}}$.
- $(1 \text{ or } 0) \leftarrow \mathsf{Test}(\mathsf{ct}_{\mathsf{PEKS},kw}, \mathbf{Trap}_{\mathsf{PEKS},kw'})$: Having input the ciphertext $\mathsf{ct}_{\mathsf{PEKS},kw}$ and the trapdoor $\mathbf{Trap}_{\mathsf{PEKS},kw'}$, this deterministic algorithm outputs 1 if $kw = kw'$; Otherwise, this deterministic algorithm outputs 0.

Security Models. A secure PEKS scheme must satisfy the following properties:

- Correctness of PEKS: Given a security parameter *pp*, any valid public-secret key pairs $(\mathsf{pk}_{\mathsf{PEKS}}, \mathsf{sk}_{\mathsf{PEKS}})$, any keywords kw, kw', any ciphertexts generated by $\mathsf{PEKS}(\mathsf{pk}_{\mathsf{PEKS}}, kw)$, and any trapdoors generated by $\mathsf{Trapdoor}(\mathsf{sk}_{\mathsf{PEKS}}, kw')$, the PEKS scheme is correct if it satisfies:

$$\text{If } kw = kw\text{'}, \Pr[\mathsf{Test}(\mathsf{ct}, \mathbf{Trap}) = 1] \approx 1; \text{and}$$

$$\text{if } kw \neq kw\text{'}, \Pr[\mathsf{Test}(\mathsf{ct}, \mathbf{Trap}) = 0] \approx 1.$$

- Ciphertext Indistinguiability of PEKS: If it does not exist an adversary $\mathcal{A}$ can obtain any keyword information of the challenge ciphertext $\mathsf{ct}_{\mathsf{PEKS},kw}$, this PEKS scheme has indistinguishability against chosen keyword attacks (IND-CKA).

2.2 Labelled Public-key Encryption scheme

Labelled public-key encryption (abb. Labelled PKE) is one of the variants of public-key encryption [36]. In the remainder of this paper, we employ the Labelled PKE scheme for our construction and refer to it as PKE for brevity. A standard PKE scheme consists of three algorithms:

- $(\mathsf{pk}_{\mathsf{PKE}}, \mathsf{sk}_{\mathsf{PKE}}) \leftarrow \mathsf{KeyExt}(\lambda)$: Given one security parameter λ, this PPT algorithm outputs $\mathsf{pk}_{\mathsf{PKE}}$ and $\mathsf{sk}_{\mathsf{PKE}}$ as the public key and secret key for encryption and decryption, respectively.
- $\mathsf{ct}_{\mathsf{PKE}} \leftarrow \mathsf{Encrypt}(\mathsf{pk}_{\mathsf{PKE}}, \mathsf{label}, \mathsf{pt}_{\mathsf{PKE}}, \rho)$: After input the public key $\mathsf{pk}_{\mathsf{PKE}}$, one label label, the plaintext $\mathsf{pt}_{\mathsf{PKE}}$, and a randomness ρ, this PPT algorithm will output the ciphertext $\mathsf{ct}_{\mathsf{PKE}}$.
- $(\mathsf{pt}_{\mathsf{PKE}} \text{ or } \perp) \leftarrow \mathsf{Decrypt}(\mathsf{sk}_{\mathsf{PKE}}, \mathsf{label}, \mathsf{ct}_{\mathsf{PKE}})$: Having input the secret key $\mathsf{sk}_{\mathsf{PKE}}$, one label label, the ciphertext $\mathsf{ct}_{\mathsf{PKE}}$ and a randomness ρ, this deterministic algorithm outputs the plaintext $(\mathsf{pt}_{\mathsf{PKE}} \text{ or } \perp)$.

Security Models. A secure PKE scheme must satisfy the following security properties:

- Correctness of PKE: Given a security parameter λ, the public key and security key generated through $(\mathsf{pk}_{\mathsf{PKE}}, \mathsf{sk}_{\mathsf{PKE}}) \leftarrow \mathsf{KeyExt}(\lambda)$, one label label, the randomness ρ, one ciphertext generated by $\mathsf{ct}_{\mathsf{PKE}} \leftarrow \mathsf{Encrypt}(\mathsf{pk}_{\mathsf{PKE}}, \mathsf{label}, \mathsf{pt}_{\mathsf{PKE}}, \rho)$, the PKE scheme is correct if

$$\Pr[\mathsf{Decrypt}(\mathsf{sk}_{\mathsf{PKE}}, \mathsf{label}, \mathsf{ct}_{\mathsf{PKE}}) = \mathsf{pt}_{\mathsf{PKE}}] \approx 1.$$

- IND-CPA/IND-CCA security of PKE: One secure PKE protocol needs to satisfy the indistinguishability against chosen-plaintext attacks (IND-CPA) if there does not exist one adversary $\mathcal{A}$ can obtain any information of a challenge plaintext $\mathsf{pt}_{\mathsf{PKE}}$. In addition, we say one primitive realizes indistinguishability against chosen-ciphertext attacks (IND-CCA) if $\mathcal{A}$ is permitted to access the decryption query for any ciphertext $\mathsf{ct}_{\mathsf{PKE}}$ excepting for querying the challenge ciphertext.

2.3 Basic Knowledge of Lattice and Trapdoors

Definition 1 (Lattice). *[37] Suppose that $\mathbf{b_1}, \mathbf{b_2}, \cdots, \mathbf{b_n} \in \mathbb{R}^m$ are n linearly independent vectors. The m-dimensional lattice Λ is generated by a set of linear combinations, denoted as $\Lambda = \Lambda(\mathbf{B}) = \{x_1 \cdot \mathbf{b_1} + x_2 \cdot \mathbf{b_2} + \cdots + x_n \cdot \mathbf{b_n} | x_i \in \mathbb{Z}\}$, where $\mathbf{B} = \{\mathbf{b_1}, \mathbf{b_2}, \cdots, \mathbf{b_n}\} \in \mathbb{R}^{m\times n}$ is the basis of Λ.*

Definition 2 (q-ary Lattices). *[38] Given $n, m, q \in \mathbb{Z}$, and $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, we define the following q-ary Lattices and a coset:*

$$\Lambda_q(\mathbf{A}) := \{\mathbf{e} \in \mathbb{Z}^m | \exists \mathbf{s} \in \mathbb{Z}_q^n, \mathbf{A}^\top \mathbf{s} = \mathbf{e} \bmod q\}.$$

$$\Lambda_q^\perp(\mathbf{A}) := \{\mathbf{e} \in \mathbb{Z}^m | \mathbf{A}\mathbf{e} = 0 \bmod q\}.$$

$$\Lambda_q(\mathbf{A}^\mathbf{u}) := \{\mathbf{e} \in \mathbb{Z}^m | \mathbf{A}\mathbf{e} = \mathbf{u} \bmod q\}.$$

Definition 3 (Gaussian Distribution). *Given one positive parameter $\sigma \in \mathbb{R}^+$, one center $\mathbf{c} \in \mathbb{Z}^m$ and any $\mathbf{x} \in \mathbb{Z}^m$, we define $\mathcal{D}_{\sigma,c} = \frac{\rho_{\sigma,\mathbf{c}(x)}}{\rho_{\sigma,\mathbf{c}(\Lambda)}}$ for $\forall x \in \Lambda$ as the Discrete Gaussian Distribution over Λ with a center $\mathbf{c}$, where $\rho_{\sigma,\mathbf{c}(x)} = \exp(-\pi \frac{\|\mathbf{x}-\mathbf{c}\|^2}{\sigma^2})$ and $\rho_{\sigma,\mathbf{c}(\Lambda)} = \Sigma_{x\in\Lambda} \rho_{\sigma,\mathbf{c}(x)}$. Specially, we say $\mathcal{D}_{\sigma,0}$ abbreviated as $\mathcal{D}_\sigma$ when $\mathbf{c} = 0$.*

Definition 4. *[39] We define Ψ_α as the probability distribution over $\mathbb{Z}_q$ for the random variable $\lfloor qx \rceil$ by selecting $x \in \mathbb{R}$ from the normal distribution with mean 0 and the standard deviation $\frac{\alpha}{\sqrt{2\pi}}$.*

Lemma 1 (TrapGen(n, m, q)). *[40] Taking $n, m, q \in \mathbb{Z}$ as input, this PPT algorithm returns $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and $\mathbf{T_A} \in \mathbb{Z}_q^{m\times m}$, where $\mathbf{T_A}$ is a basis of $\Lambda_q^\perp(\mathbf{A})$ s.t. $\{\mathbf{A} : (\mathbf{A}, \mathbf{T_A}) \leftarrow \mathrm{TrapGen}(1^n, 1^m, q)\}$ is statistically close to $\{\mathbf{A} : \mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}\}$. In this way, we say $\mathbf{T_A}$ is a trapdoor of $\mathbf{A}$.*

Lemma 2 (SamplePre$(\mathbf{A}, \mathbf{T_A}, \mathbf{u}, \sigma)$). *[41] Given a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and its trapdoor $\mathbf{T_A} \in \mathbb{Z}_q^{m\times m}$, a vector $\mathbf{u} \in \mathbb{Z}_q^n$, and the parameter $\sigma \leq \|\tilde{\mathbf{T_A}}\| \cdot \omega(\sqrt{\log(m)})$, where $m \geq 2n\lceil \log q \rceil$, this* PPT *algorithm publishes one sample $\mathbf{e} \in \mathbb{Z}_q^m$ statistically distributed in $\mathcal{D}_{\Lambda_q^\mathbf{u}(\mathbf{A}),\sigma}$ s.t. $\mathbf{A}\mathbf{e} = \mathbf{u} \mod q$.*

Lemma 3 (NewBasisDel$(\mathbf{A}, \mathbf{R}, \mathbf{T_A}, \sigma)$). *[39] Taking a parameter $\sigma \in \mathbb{R}$, a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, one $\mathbb{Z}_q$-invertible matrix $\mathbf{R}$ sampled from the distribution $\mathcal{D}_{m\times m}$, and trapdoor $\mathbf{T_A}$ as input, this* PPT *algorithm will output one short lattice basis $\mathbf{T_B}$ of $\Lambda_q^\perp(\mathbf{B})$, where $\mathbf{B} = \mathbf{A}\mathbf{R}^{-1}$.*

Lemma 4 (SampleLeft$(\mathbf{A}, \mathbf{M}, \mathbf{T_A}, \mathbf{u}, \sigma)$). *[42] After input one matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and its corresponding trapdoor $\mathbf{T_A} \in \mathbb{Z}_q^{m\times m}$, one matrix $\mathbf{M} \in \mathbb{Z}_q^{n\times m_1}$, one vector $\mathbf{u} \in \mathbb{Z}_q^n$, and one parameter $\sigma \leq \|\tilde{\mathbf{T_A}}\| \cdot \omega(\sqrt{\log(m + m_1)})$, this* PPT *algorithm will output a sample $t \in \mathbb{Z}^{m+m_1}$ from the distribution statistically close to $\mathcal{D}_{\Lambda_q^\mathbf{u}([\mathbf{A}|\mathbf{M}]),\sigma}$ s.t. $[\mathbf{A}|\mathbf{M}] \cdot t = u \mod q$.*

Lemma 5 (ExtBasis$(\mathbf{A}'',\mathbf{S})$). *[21] For an input matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, one basis $\mathbf{S} \in \mathbb{Z}_q^{m\times m}$ of $\Lambda^\perp(\mathbf{A})$, and a matrix $\mathbf{A}' \in \mathbb{Z}_q^{n\times m'}$, this deterministic algorithm outputs a basis $\mathbf{S}''$ of $\Lambda^\perp(\mathbf{A}'') \subseteq \mathbb{Z}_q^{m\times m''}$ s.t. $\|\tilde{\mathbf{S}}\| = \|\tilde{\mathbf{S}''}\|$, and $\mathbf{A}'' = \mathbf{A}||\mathbf{A}'$, $m'' = m + m'$.*

3 Syntax of Forward-Secure PAEKS and Security Models

Public-key authenticated encryption with keyword search (PAEKS) was initially proposed by Huang et al. [16] as a variant of the traditional PEKS scheme [1], with the added benefit of satisfying the trapdoor privacy. Subsequently, Jiang et al. introduced the concept of forward security into the PAEKS primitive [15].

3.1 Syntax of FS-PAEKS scheme

We reviewed the FS-PAEKS scheme based on the discrete logarithm [15] and formalized the definition of forward-secure PAEKS primitive (including six PPT algorithms) as below.

- $\mathsf{Setup}(1^\lambda)$: After input one security parameter λ, this algorithm returns one system parameter pp.
- $\mathsf{KeyGen}_S(pp)$: Taking one system parameter pp as input, this algorithm publishes a public-secret key pair for a sender (pk_S, sk_S).
- $\mathsf{KeyGen}_R(pp)$: Taking one system parameter pp as input, this algorithm outputs a public-secret key pair for a receiver (pk_R, sk_R).
- $\mathsf{FS-PAEKS}(pk_R, sk_S, kw)$: Given a public key of one receiver pk_R, a secret key of one sender sk_R, and any keyword kw, this algorithm returns a ciphertext ct of kw with time t as output.
- $\mathsf{Trapdoor}(pk_S, sk_R, kw')$: Given a public key of one sender pk_S, a secret key of one receiver sk_R, and a keyword kw', this algorithm outputs a trapdoor $\mathbf{Trap}$ of kw' with time t'.
- $\mathsf{Test}(\mathsf{ct}, \mathbf{Trap})$: After input a ciphertext ct and a trapdoor $\mathbf{Trap}$, this algorithm returns 1 if the ct and $\mathbf{Trap}$ is related to one same keyword, that is, $kw = kw'$ holds; otherwise, it returns 0.

3.2 Core Technique

We leverage various lattice trapdoor algorithms to convert the DL-based public-secret keys and trapdoor generations to their lattice-based counterparts. To mitigate the risk of secret key leakage, we employ a binary tree architecture to represent the time period. Specifically, we use $\mathsf{node}(t)$ to represent the smallest minimal cover set for the secret key update periodically, and we utilize the lattice basis extension algorithm $\mathsf{ExtBasis}$ to realize the one-way key evolution mechanism.

3.3 Security Models

This chapter outlines the security models of our proposed primitive. Specifically, we establish security criteria that ensure that any probabilistic polynomial-time (PPT) adversary cannot obtain any keyword information from the ciphertext [1] and any (inside) PPT attacker cannot acquire any keyword information from the trapdoor [3,43]. We hereby define three key security parameters, namely ciphertext indistinguishability (CI) for forward-secure PAEKS under indistinguishability against chosen keywords attack (IND-CKA), the trapdoor privacy for forward-secure PAEKS under indistinguishability against inside keyword guessing attack (IND-IKGA), and the multi-ciphertext indistinguishability (MCI) for forward-secure PAEKS under indistinguishability against chosen multi-keywords attack (IND-Multi-CKA).

IND-CKA Game of FS-PAEKS

- **Setup**: After input one security parameter λ, the challenger $\mathcal{C}$ calls the Setup algorithm to obtain the public parameter pp. After that, $\mathcal{C}$ processes the KeyGen_S and KeyGen_R algorithms to compute the sender's and receiver's public-secret key pair, that is, (pk_S, sk_S) and (pk_R, sk_R). Ultimately, $\mathcal{C}$ sends pp, pk_S and pk_R to the adversary $\mathcal{A}$ and keeps the initial secret key sk_R secret.
- **Query 1**: In this query, $\mathcal{A}$ is permitted to adaptively access three oracles in some polynomial times.

  - **KeyUpdate Oracle** $\mathcal{O}_{KU}$: If the time period $t < T-1$, $\mathcal{C}$ will updates the time period from t to $t+1$; If the time period $t = T-1$, which means the current period is the last period, $\mathcal{C}$ will return an empty string sk_T.
  - **Ciphertext Oracle** $\mathcal{O}_C$: $\mathcal{A}$ requires that the time period t is larger than the target time period t^*. Given any keyword kw, $\mathcal{C}$ calls the $\mathsf{FS-PAEKS}(pp, pk_S, sk_S, pk_R, kw, t, d)$ algorithm to obtain the ciphertext ct at time period t and returns it to $\mathcal{A}$.
  - **Trapdoor Oracle** $\mathcal{O}_T$: $\mathcal{A}$ requires that the time period t is larger than the target time period t^*. Given any keyword kw, $\mathcal{C}$ calls the $\mathsf{Trapdoor}(pp, pk_S, pk_R, sk_R(t), kw')$ algorithm to obtain the trapdoor **Trap** in time period t and transmits it to $\mathcal{A}$.
- **Challenge**: In time period t^*, which has not been querried the $\mathcal{O}_T$, $\mathcal{A}$ selects two challenge keywords kw_0^* and kw_1^* and sends them to $\mathcal{C}$. This phase restricts that $\mathcal{A}$ never accesses the three oracles ($\mathcal{O}_{KU}, \mathcal{O}_C$ and $\mathcal{O}_T$) for the challenge keywords kw_0^* and kw_1^*. After that, $\mathcal{C}$ selects a bit $b \in \{0,1\}$ at random and calls the $\mathsf{FS-PAEKS}(pp, pk_S, sk_S, pk_R, kw_b^*, t^*, d)$ algorithm to calculate the challenge ciphertext ct^*. Finally, $\mathcal{C}$ sends ct^* to $\mathcal{A}$.
- **Query 2**: $\mathcal{A}$ has the ability to continue those queries as similar as **Query 1** with one limitation that $\mathcal{A}$ is not allowed to query the challenge keywords (kw_0^*, kw_1^*).
- **Guess**: After finished the above phases, $\mathcal{A}$ will output a guess bit $b' \in \{0,1\}$. Therefore, we say that $\mathcal{A}$ wins the game if and only if $b = b'$.

We hereby define the advantage of $\mathcal{A}$ wins the above game as

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

Definition 5 (IND-CKA secure of FS-PAEKS). *We say that an FS-PAEKS scheme satisfies forward-secure ciphertext indistinguishability (CI) under IND-CKA, if for any PPT adversary $\mathcal{A}$, the advantage $Adv_{\mathcal{A}}^{IND-CKA}(\lambda)$ is negligible.*

IND-IKGA Game of FS-PAEKS

- **Setup**: This process is same as the **IND-IKGA Game of FS-PAEKS**.
- **Query 1**: In this query, $\mathcal{A}$ is permitted to adaptively access three oracles ($\mathcal{O}_{KU}, \mathcal{O}_C$ and $\mathcal{O}_T$, are same as the **IND-IKGA Game of FS-PAEKS**) in some polynomial times.
- **Challenge**: In time period t^*, which has not been querried the $\mathcal{O}_T$, $\mathcal{A}$ selects two challenge keywords kw_0^* and kw_1^* and transmits them to $\mathcal{C}$. This phase restricts that $\mathcal{A}$ never accesses the three oracles ($\mathcal{O}_{KU}, \mathcal{O}_C$ and $\mathcal{O}_T$) for the challenge keywords kw_0^* and kw_1^*. After that, $\mathcal{C}$ selects a bit $b \in \{0,1\}$ at random and calls the $\mathsf{Trapdoor}(pp, pk_S, pk_R, sk_R(t^t), kw_b')$ algorithm to calculate the challenge trapdoor **Trap***. Finally, $\mathcal{C}$ sends **Trap*** to $\mathcal{A}$.
- **Query 2**: $\mathcal{A}$ has the ability to continue those queries as similar as **Query 1** with the limitation that $\mathcal{A}$ is not allowed to query the challenge keywords (kw_0^*, kw_1^*).
- **Guess**: After finished the above phases, $\mathcal{A}$ publishes a guess bit $b' \in \{0,1\}$. Thus, we say that $\mathcal{A}$ wins the game if and only if $b = b'$.

We define the advantage of $\mathcal{A}$ wins the above game as

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

Definition 6 (IND-IKGA secure of FS-PAEKS). *We say that an FS-PAEKS scheme satisfies forward-secure trapdoor privacy (TP) under IND-IKGA, if for any PPT adversary $\mathcal{A}$, the advantage $Adv_{\mathcal{A}}^{IND-IKGA}(\lambda)$ is negligible.*

IND-Multi-CKA Game of FS-PAEKS

- **Setup**: This process is same as the **IND-IKGA Game of FS-PAEKS**.
- **Query 1**: In this query, $\mathcal{A}$ is permitted to adaptively access three oracles ($\mathcal{O}_{KU}, \mathcal{O}_C$ and $\mathcal{O}_T$, same as the **IND-IKGA Game of FS-PAEKS**) in some polynomial times.
- **Challenge**: Given two tuples of challenge keywords $(kw^*_{0,1}, \cdots, kw^*_{0,n})$, $\mathcal{C}$ firstly selects a tuple $(kw^*_{0,i}, kw^*_{1,i})$ for some i $\mathsf{s.t.}$ $kw^*_{0,i} \neq kw^*_{1,i}$. After that, $\mathcal{C}$ selects a bit $b \in \{0,1\}$ randomly and calls the $\mathsf{FS-PAEKS}(pp, pk_S, sk_S, pk_R, kw^*_{b,i}, t^*, d)$ algorithm to calculate the challenge ciphertext ct^*. Moreover, $\mathcal{C}$ selects $n-1$ ciphertexts from the output space of $\mathsf{FS-PAEKS}$ algorithm, namely as, $(\mathsf{ct}_1, \mathsf{ct}_2, \cdots, \mathsf{ct}_{i-1}, \mathsf{ct}_{i+1}, \mathsf{ct}_{i+2}, \cdots, \mathsf{ct}_n)$.
- **Query 2**: $\mathcal{A}$ can continue the queries as in the **Query 1** with the restriction that $\mathcal{A}$ is not allowed to query the challenge keywords $kw^*_{i,j}$, where $i \in \{0,1\}$ and $j \in \{1, 2, \cdots, n\}$.
- **Guess**: After finished the above phases, $\mathcal{A}$ outputs a guess bit $b' \in \{0,1\}$ and $\mathcal{C}$ uses it as its output. We say that $\mathcal{A}$ wins the game if and only if $b = b'$.

Definition 7 (**IND-Multi-CKA secure of FS-PAEKS**). *We say that an FS-PAEKS scheme satisfies forward-secure multi-ciphertext under IND-Multi-CKA, if it satisfies CI under IND-CKA and it is a probabilistic algorithm.*

4 Our Proposed Construction

In this part, we illustrate the first generic construction of post-quantum FS-PAEKS based on the prototype of PEKS primitive, labelled PKE scheme, SPHF protocol, and binary tree architecture. To begin with, we define $\mathcal{KS}_{\mathsf{PEKS}}$ as the keyword space and one standard PEKS scheme icnludes four algorithms ($\mathsf{PEKS.KeyGen}$, $\mathsf{PEKS.PEKS}$, $\mathsf{PEKS.Trapdoor}$, $\mathsf{PEKS.Test}$). Moreover, we define $\mathcal{PKS}_{\mathsf{PKE}}$ and $\mathcal{PS}_{\mathsf{PKE}}$ as the public key space and plaintext space, respectively. A labelled PKE scheme consists of three algorithms ($\mathsf{PKE.KeyGen}$, $\mathsf{PKE.Encrypt}$, $\mathsf{PKE.Decrypt}$). We introduce the SPHF scheme, including four algorithms, that is, ($\mathsf{SPHF.KeyGen_{Hash}}, \mathsf{SPHF.KeyGen_{ProjHash}}, \mathsf{SPHF.Hash}, \mathsf{SPHF.ProjHash}$). Finally, we utilize the binary tree structure and the smallest minimal cover set to achieve the secret key update for the receiver. We also employ the $\mathsf{ExtBasis}$ algorithm to realize one-way secret key evolution, which ensures the security of the secret keys.
We first define the language of ciphertext as $(Para_l, Trap_l) = (\mathsf{pk_{PKE}}, \mathsf{sk_{PKE}})$, where $\mathsf{pk_{PKE}} \in \mathcal{PKS}_{\mathsf{PKE}}$, $\tilde{\mathcal{L}} := \{(\mathsf{label}, \mathsf{ct_{PKE}}, \mathsf{m_{PKE}}) | \exists \rho, \mathsf{ct_{PKE}} \leftarrow \mathsf{Encrypt}(\mathsf{pk_{PKE}}, \mathsf{label}, \mathsf{m_{PKE}}, \rho)\}$, and $\mathcal{L} := \{(\mathsf{label}, \mathsf{ct_{PKE}}, \mathsf{m_{PKE}}) | \mathsf{Decrypt}(\mathsf{sk_{PKE}}, \mathsf{label}, \mathsf{ct_{PKE}}) = \mathsf{m_{PKE}}\}$. Besides, we also define the witness relation $\tilde{\mathcal{K}}((\mathsf{label}, \mathsf{ct_{PKE}}, \mathsf{m_{PKE}}), \rho) = 1$ if and only if we have $\mathsf{ct_{PKE}} \leftarrow \mathsf{Encrypt}(\mathsf{pk_{PKE}}, \mathsf{label}, \mathsf{m_{PKE}}, \rho)\}$.

- $\mathsf{Setup}(1^\lambda, \mathsf{d})$: Given one security parameter λ, the Setup algorithm processes following operations as below:
  - Calculates $(\mathbf{pk}_{\mathsf{PKE}}, \mathbf{sk}_{\mathsf{PKE}}) \leftarrow \mathsf{PKE.KeyExt}(\lambda)$.
  - Selects one plaintext $\mathsf{m_{PKE}} \xleftarrow{\$} \mathcal{PKS}_{\mathsf{PKE}}$ and one label $\mathsf{label} \xleftarrow{\$} \{0,1\}^*$ randomly.
  - Selects two hash functions:
$$H_1 : \mathcal{PKS}_{\mathsf{PKE}} \times \mathcal{PS}_{\mathsf{PKE}} \times \{0,1\}^* \to \mathcal{PKS}_{\mathsf{PKE}};\ H_2 : \mathcal{KS}_{\mathsf{PEKS}} \times \{0,1\}^* \to \mathcal{KS}_{\mathsf{PEKS}}.$$
  - Selects $2\mathsf{d}$ matrices from $\mathbb{Z}_q^{n \times m}$ as $\mathsf{Matrices}$.
  - Outputs $\mathsf{pp} := (\lambda, \mathsf{mpk}, \mathsf{pk_{PKE}}, \mathsf{m_{PKE}}, \mathsf{label}, H_1, H_2, \mathsf{Matrices})$ as the public parameter.

- $\mathsf{KeyGen}_S(\mathsf{pp})$: Inputting one public parameter pp, the KeyGen_S algorithm processes these operations:
  - Calculates $\mathbf{h}_S \leftarrow \mathsf{SPHF.KeyGen_{Hash}}(\mathsf{mpk})$ and $\mathbf{p}_S \leftarrow \mathsf{SPHF.KeyGen_{Proj}}(\mathbf{h_S}, \mathsf{mpk})$.
  - Calculates $\mathsf{ct}_{\mathsf{PKE},S} \leftarrow \mathsf{PKE.Encrypt}(\mathsf{mpk}, \mathsf{label}, \mathsf{m}_{\mathsf{PKE}}, \rho_S)$, where ρ_S is a randomly selected witness s.t. $\tilde{\mathcal{K}}((\mathsf{label}, \mathsf{ct}_{\mathsf{PKE},S}, \mathsf{m}_{\mathsf{PKE}}), \rho_S) = 1$.
  - Outputs $\mathsf{pk}_S := (\mathbf{p}_S, \mathsf{ct}_{\mathsf{PKE},S})$ and $\mathsf{sk}_S := (\mathbf{h}_S, \rho_S)$ as the public key and secret key of the sender, respectively.
- $\mathsf{KeyGen}_R(\mathsf{pp})$: Given a public parameter pp, the KeyGen_R algorithm processes the following operations:
  - Calculates $\mathbf{h}_R \leftarrow \mathsf{SPHF.KeyGen_{Hash}}(\mathsf{mpk})$ and $\mathbf{p}_R \leftarrow \mathsf{SPHF.KeyGen_{Proj}}(\mathbf{h_R}, \mathsf{mpk})$.
  - Calculates $\mathsf{ct}_{\mathsf{PKE},R} \leftarrow \mathsf{PKE.Encrypt}(\mathsf{mpk}, \mathsf{label}, \mathsf{m}_{\mathsf{PKE}}, \rho_R)$, where ρ_R is a randomly selected witness s.t. $\tilde{\mathcal{K}}((\mathsf{label}, \mathsf{ct}_{\mathsf{PKE},R}, \mathsf{m}_{\mathsf{PKE}}), \rho_R) = 1$.
  - Calculates $(\mathsf{pk}_{\mathsf{PEKS}}, \mathsf{sk}_{\mathsf{PEKS}}) \leftarrow \mathsf{PEKS.KeyGen}(1^\lambda)$.
  - Outputs $\mathsf{pk}_R := (\mathbf{p}_R, \mathsf{ct}_{\mathsf{PKE},R}, \mathsf{pk}_{\mathsf{PEKS}})$ and $\mathsf{sk}_R := (\mathbf{h}_R, \rho_R, \mathsf{sk}_{\mathsf{PEKS}})$ as the public key and secret key of the receiver, respectively.
- $\mathsf{KeyUpdate}(\mathsf{pp}, \mathsf{pk}_R, \mathsf{sk}_R, t, \mathsf{d})$: Given a public parameter pp, a public key pk_R and a secret key sk_R of the initial receiver, a time period t, and a depth d, the $\mathsf{KeyUpdate}$ algorithm processes the following operations:
  - Defines $F_{\Theta^{(i)}}$ as the corresponding matrix of $\Theta^{(i)}$.
  - For any $j < i$ where $j, i \in [1, d]$, calculates $\mathbf{S}_{\Theta^{(i)}} \leftarrow \mathsf{ExtBasis}(F_{\Theta^{(i)}}, \mathbf{S}_{\Theta^{(j)}})$, where $\mathbf{S}_{\Theta^{(j)}}$ is the trapdoor on time period j.
  - Defines $\mathsf{sk}_R(t) := (sk_R, \mathbf{S}_{\Theta^{(i)}})$, where $\Theta^{(i)} \in \mathsf{node}(t)$.
  - Defines and outputs $\mathsf{sk}_R(t+1) := (sk_R, \mathbf{S}_{\Theta^{(i)}})$, where $\Theta^{(i)} \in \mathsf{node}(t+1)$.
- $\mathsf{FS-PAEKS}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{sk}_S, \mathsf{pk}_R, kw, t, \mathsf{d})$: Given a public parameter pp, a public key pk_S and a secret key sk_S of one sender, a public key pk_R of one receiver, one keyword $kw \in \mathcal{KS}_{\mathsf{FS-PAEKS}}$ the time period t, and the depth d, the $\mathsf{FS-PAEKS}$ algorithm processes the following operations:
  - Calculates $\mathsf{Hash}_S \leftarrow \mathsf{SPHF.Hash}(\mathsf{h}_S, \mathsf{mpk}, (\mathsf{ct}_{\mathsf{PKE},R}, \mathsf{m}_{\mathsf{PKE}}))$.
  - Calculates $\mathsf{ProjHash}_S \leftarrow \mathsf{SPHF.ProjHash}(\mathsf{p}_R, \mathsf{mpk}, (\mathsf{ct}_{\mathsf{PKE},S}, \mathsf{m}_{\mathsf{PKE}}), \rho_S)$.
  - Calculates $kw_S \leftarrow H_2(kw, \mathsf{Hash}_S \oplus \mathsf{ProjHash}_S)$
  - Calculates and outputs $\mathsf{ct} \leftarrow \mathsf{PEKS.PEKS}(\mathsf{pk}_{\mathsf{PEKS}}, kw_S)$.
- $\mathsf{Trapdoor}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{pk}_R, \mathsf{sk}_R(t), kw')$: After input a public parameter pp, a public key pk_S of one sender, a public key pk_R and a secret key $\mathsf{sk}_R(t)$ of one receiver, one keyword $kw' \in \mathcal{KS}_{\mathsf{FS-PAEKS}}$, the $\mathsf{Trapdoor}$ algorithm processes the following operations:
  - Calculates $\mathsf{Hash}_R \leftarrow \mathsf{SPHF.Hash}(\mathsf{h}_R, \mathsf{mpk}, (\mathsf{ct}_{\mathsf{PKE},S}, \mathsf{m}_{\mathsf{PKE}}))$.
  - Calculates $\mathsf{ProjHash}_R \leftarrow \mathsf{SPHF.ProjHash}(\mathsf{p}_R, \mathsf{mpk}, (\mathsf{ct}_{\mathsf{PKE},R}, \mathsf{m}_{\mathsf{PKE}}), \rho_R)$.
  - Calculates $kw'_R \leftarrow H_2(kw', \mathsf{Hash}_R \oplus \mathsf{ProjHash}_R)$.
  - Calculates $\mathbf{Trap_1} \leftarrow \mathsf{PEKS.Trapdoor}(\mathsf{sk}_{\mathsf{PEKS}}, kw'_R)$.
  - Calculates $\mathbf{Trap_2} \leftarrow \mathsf{SamplePre}(\mathbf{S}_{\Theta^{(t)}}, H_3(kw'), \sigma_3)$.
  - Defines and outputs $\mathbf{Trap} := (\mathbf{Trap_1}, \mathbf{Trap_2})$.
- $\mathsf{Test}(\mathsf{pp}, \mathsf{ct}, \mathbf{Trap})$: Taking one public parameter pp, one ciphertext ct, and the trapdoor $\mathbf{Trap}$ as input, the Test algorithm outputs $\mathsf{PEKS.Test}(\mathsf{ct}, \mathbf{Trap})$.

5 Instantiation of Our Construction

In this section, we construct the first post-quantum PAEKS with forward security instantiation based on the lattice hardness, FS-PAEKS, including seven algorithms.

- $\mathsf{Setup}(1^\lambda, d)$: After inputs a security parameter λ, the depth d of one binary tree, system parameters $q, n, m, \sigma_1, \sigma_2, \alpha, \sigma_3, T$, where q is a prime, σ_1, σ_2 and σ_3 is the preimage sample parameter, α is the gaussian distribution parameter, and $T = 2^d$ as the total number of time periods, this algorithm executes the following operations.
  - Calls $\kappa, \rho, \ell \leftarrow \mathsf{poly}(n)$ and selects $\mathbf{m} = m_1 m_2 \cdots m_\kappa \xleftarrow{\$} \{0,1\}^\kappa$ randomly.
  - Selects matrices $A_1^{(0)}, A_1^{(1)}, A_2^{(0)}, A_2^{(1)}, \cdots, A_d^{(0)}, A_d^{(1)} \in \mathbb{Z}_q^{n\times m}$.
  - Calls $\mathsf{TrapGen}(n, m, q)$ algorithm to generate a matrix $\mathbf{A_0}$ and the basis $\mathbf{T_{A_0}}$ of $\Lambda^\perp(\mathbf{A_0})$.
  - Sets $\mathbf{A_0}$ as a public key of PKE and $\mathbf{T_{A_0}}$ as a secret key of PKE.
  - Selects one element $u \xleftarrow{\$} \mathcal{U}$ randomly as the label of PKE.
  - Selects three Hash functions
$$H_1 : \mathbb{Z}^{n\times m} \times \{0,1\}^\kappa \times \mathcal{U} \rightarrow \mathbb{Z}_q^{n\times m};$$
$$H_2 : \{1,-1\}^\ell \times \{0,1\}^\kappa \rightarrow \{1,-1\}^\ell;$$
$$H_3 : \{1,-1\}^\ell \rightarrow \mathbb{Z}_q^n.$$
    Selects one Injective function $H_4 : \mathcal{R} \rightarrow \mathbb{Z}_q^{n\times n}$.
  - Calculates
$$\mathbf{A} \leftarrow H_1(\mathbf{T_{A_0}}, \mathbf{m}, u) \in \mathbb{Z}^{n\times m} \tag{1}$$
    as the master public key of PKE.
  - Ultimately, this algorithm returns a public parameter as
$$pp := (\lambda, q, n, m, \sigma_1, \sigma_2, \sigma_3, \kappa, \rho, \ell, \mathbf{T_{A_0}}, A_1^{(0)}, A_1^{(1)}, A_2^{(0)}, A_2^{(1)}, \cdots, A_d^{(0)}, A_d^{(1)}, \mathbf{A}, \mathbf{m}, u, H_1, H_2, H_3, H_4).$$
- $\mathsf{KeyGen}_S(pp)$: Taking a public parameter pp as input, this algorithm will execute the following steps to generate the public key and secret key of the sender.
  - Sets gadget matrix $\mathbf{G} := \mathbf{I}_n \otimes \mathbf{g}^\top$, $\mathbf{g}^\top = [1, 2, \cdots, 2^k], k = \lceil \mathsf{log} q \rceil - 1$.
  - Calculates
$$\mathbf{A}_{\mathsf{label}} = \mathbf{A} + \begin{bmatrix} 0 \\ \mathbf{G}H_4(u) \end{bmatrix} = \mathbf{A} + \begin{bmatrix} 0 \\ (\mathbf{I}_n \otimes \mathbf{g}^\top) H_4(u) \end{bmatrix}. \tag{2}$$
  - Selects one matrix $\mathbf{h}_S \xleftarrow{\$} D^m_{\mathbb{Z},s}$ at random.
  - Calculates the matrix $\mathbf{p}_S = \mathbf{A}_{\mathsf{label}}^\top \cdot \mathbf{h}_S \in \mathbb{Z}_q^n$.
  - For $i = 1, 2, \cdots, \kappa$, selects vectors $\mathbf{s}_i \xleftarrow{\$} \mathbb{Z}_q$ and vectors $\mathbf{e}_{S,i} \xleftarrow{\$} D^m_{\mathbb{Z},t}$ randomly s.t. $\|\mathbf{e}_{S,i}\| \leq 2t\sqrt{m}$ and then calculates
$$\mathbf{c}_{S,i} = \mathbf{A}_{\mathsf{label}}^\top \cdot \mathbf{s}_i + \mathbf{e}_{S,i} + m_i [0, 0, \cdots, 0, \lceil \frac{q}{2} \rceil]^\top \bmod q. \tag{3}$$
  - Outputs $pk_S := (\mathbf{p}_S, \{\mathbf{c}_{S,1}\}, \{\mathbf{c}_{S,2}\}, \cdots, \{\mathbf{c}_{S,\kappa}\})$ and $sk_S := (\mathbf{h}_S, \{\mathbf{s}_1\}, \{\mathbf{s}_2\}, \cdots, \{\mathbf{s}_\kappa\})$as a public key and a secret key of one sender, respectively.
- $\mathsf{KeyGen}_R(pp)$: Taking a public parameter pp as input, it executes the following steps to compute the initial public key and initial secret key for one receiver.
  - Calls $\mathsf{TrapGen}(n, m, q)$ algorithm to generate a matrix $\mathbf{M}_R$ and the basis $\mathbf{S}_R$ of $\Lambda^\perp(\mathbf{M}_R)$.
  - For $i = 1, 2, \cdots, \ell$, selects matrices $\mathbf{M}_{R,i} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$ randomly.

- Selects a matrix $\mathbf{C}_R \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$ and a vector $\mathbf{r}_R \xleftarrow{\$} \mathbb{Z}_q^n$ at random.
- Sets gadget matrix $\mathbf{G} := \mathbf{I}_n \otimes \mathbf{g}^\top$, $\mathbf{g}^\top = [1, 2, \cdots, 2^k], k = \lceil \log q \rceil - 1$.
- Calculates

$$\mathbf{A}_{\mathsf{label}} = \mathbf{A} + \begin{bmatrix} 0 \\ \mathbf{G}H_4(u) \end{bmatrix} = \mathbf{A} + \begin{bmatrix} 0 \\ (\mathbf{I}_n \otimes \mathbf{g}^\top)H_4(u) \end{bmatrix}. \quad (4)$$

- Selects one matrix $\mathbf{h}_R \xleftarrow{\$} D^m_{\mathbb{Z},s}$ at random.
- Calculates the matrix $\mathbf{p}_R = \mathbf{A}_{\mathsf{label}}^\top \cdot \mathbf{h}_R \in \mathbb{Z}_q^n$.
- For $i = 1, 2, \cdots, \kappa$, selects vectors $\mathbf{r}_i \xleftarrow{\$} \mathbb{Z}_q$ and vectors $\mathbf{e}_{R,i} \xleftarrow{\$} D^m_{\mathbb{Z},t}$ randomly s.t. $\|\mathbf{e}_{R,i}\| \leq 2t\sqrt{m}$ and then calculates

$$\mathbf{c}_{R,i} = \mathbf{A}_{\mathsf{label}}^\top \cdot \mathbf{r}_i + \mathbf{e}_{R,i} + m_i[0, 0, \cdots, 0, \lceil \frac{q}{2} \rceil]^\top \bmod q. \quad (5)$$

- Outputs $pk_R := (\mathbf{p}_R, \{\mathbf{c}_{R,1}\}, \{\mathbf{c}_{R,2}\}, \cdots, \{\mathbf{c}_{R,\kappa}\}, \mathbf{M}_R, \mathbf{M}_{R,1}, \mathbf{M}_{R,2}, \cdots, \mathbf{M}_{R,\ell}, \mathbf{C}_R, \mathbf{r}_R)$ and $sk_R := (\mathbf{h}_R, \{\mathbf{r}_1\}, \{\mathbf{r}_2\}, \cdots, \{\mathbf{r}_\kappa\})$ as the initial (root node) public key and secret key of the receiver, respectively.

– $\mathsf{KeyUpdate}(pp, pk_R, sk_R, t, d)$: Having input a public parameter pp, time t, initial public key pk_R, and initial secret key sk_R, this algorithm will process the following steps.
  - Defines $t := (t_1 t_2 \cdots t_i)$, where t means the binary representation of time and $i \in [1, d]$, $t_i \in \{0, 1\}$, d is the depth of the binary tree.
  - Defines $\Theta^{(i)} := (\theta_1, \theta_2, \cdots, \theta_i) \in \mathsf{node}(t)$, where $i \in [1, d], \theta_i \in \{0, 1\}$ as the path from the root to the current node.
  - Defines $F_{\Theta^{(i)}} := [\mathbf{M}_R \parallel A_1^{(\theta_1)} \parallel A_2^{(\theta_2)} \parallel \cdots \parallel A_i^{(\theta_i)}]$ as the corresponding matrix of $\Theta^{(i)}$. For example, $F_{0100} = [\mathbf{M}_R \parallel A_1^0 \parallel A_2^1 \parallel A_3^0 \parallel A_4^0]$, $F_{101} = [\mathbf{M}_R \parallel A_1^1 \parallel A_2^0 \parallel A_3^1]$.
  - For any $j < i$, where $j, i \in [1, d]$, given the trapdoor $\mathbf{S}_{\Theta^{(j)}}$ on time j, calls $\mathsf{ExtBasis}(F_{\Theta^{(i)}}, \mathbf{S}_{\Theta^{(j)}})$ to generate $\mathbf{S}_{\Theta^{(i)}}$, where $\Theta^{(i)} := (\theta_1, \theta_2, \cdots, \theta_j, \theta_{j+1}, \cdots, \theta_i)$. Thus, the updated trapdoor can be calculated by its any ancestor's trapdoor.
  - Define $sk_R(t) := (\mathbf{h}_R, \{\mathbf{r}_{R,1}\}, \{\mathbf{r}_{R,2}\}, \cdots, \{\mathbf{r}_{R,\kappa}\}, \mathbf{S}_{\Theta^{(i)}})$, where $\Theta^{(i)} \in \mathsf{node}(t)$ as the receiver's secret key on time t. Each node has the corresponding secret key in a binary tree.
  - Receiver updates $sk_R(t)$ to $sk_R(t+1)$ through calculating $sk_R(t+1) := (\mathbf{h}_R, \{\mathbf{r}_{R,1}\}, \{\mathbf{r}_{R,2}\}, \cdots, \{\mathbf{r}_{R,\kappa}\}, \mathbf{S}_{\Theta^{(i)}})$, where $\Theta^{(i)} \in \mathsf{node}(t+1)$. We show one example here, supposing that receiver updates $sk_R(1010)$ to $sk_R(1011)$. Given $sk_R(1010) = (\mathbf{h}_R, \{\mathbf{r}_{R,1}\}, \{\mathbf{r}_{R,2}\}, \cdots, \{\mathbf{r}_{R,\kappa}\}, \mathbf{S}_{101}, \mathbf{S}_{11})$, the updated secret key is $sk_R(1011) = (\mathbf{h}_R, \{\mathbf{r}_{R,1}\}, \{\mathbf{r}_{R,2}\}, \cdots, \{\mathbf{r}_{R,\kappa}\}, \mathbf{S}_{1011}, \mathbf{S}_{11})$.

– $\mathsf{FS-PAEKS}(pp, pk_S, sk_S, pk_R, kw, t, d)$: Given the public parameter pp, the sender's public key and secret key pk_S, sk_S, the receiver's public key pk_R, one keyword $kw \in \{1, -1\}^\ell$, the time period t, and the depth of the binary tree d, this algorithm executes the following procedures.
  - For $i = 1, 2, \cdots, \kappa$, calculates

$$h_{S,i} \leftarrow \lfloor \frac{2(\mathbf{c}_{R,i}^\top \cdot \mathbf{h}_S (\bmod q))}{q} \rceil, \quad (6)$$

$$p_{S,i} \leftarrow \lfloor \frac{2(\mathbf{s}_i^\top \cdot \mathbf{p}_R (\bmod q))}{q} \rceil, \quad (7)$$

  and defines $y_{S,i} = h_{S,i} \cdot p_{S,i}$.

- Defines and calculates
$$\mathbf{y}_S = y_{S,1}y_{S,2}\cdots y_{S,\kappa} \in \{0,1\}^{\kappa}. \tag{8}$$
- Calculates
$$\mathbf{dk}_S = dk_{S,1}dk_{S,2}\cdots dk_{S,\ell} \leftarrow H_2(kw, \mathbf{y}_S) \in \{1,-1\}^{\ell}. \tag{9}$$
- Defines and calculates
$$\mathbf{M}_{dk} = \mathbf{C}_R + \sum_{i=1}^{\ell} dk_{S,i}\mathbf{M}_{R,i}. \tag{10}$$
- Calculates
$$\mathbf{F}_{dk} = [\mathbf{M}_R \parallel \mathbf{M}_{dk}] = [\mathbf{M}_R \parallel \mathbf{C}_R + \sum_{i=1}^{\ell} dk_{S,i}\mathbf{M}_{R,i}]. \tag{11}$$
- Defines $\mathbf{F}_t := [\mathbf{M}_R \parallel A_1^{t_1} \parallel A_2^{t_2} \parallel \cdots \parallel A_d^{t_d}]$.
- For $j = 1, 2, \cdots, \rho$, processes the following operations as below:
  * Selects $b_j \xleftarrow{\$} \{0,1\}$ and $\mathbf{s}_j \xleftarrow{\$} \mathbb{Z}_q^n$ randomly;
  * For $i = 1, 2, \cdots, \ell$, selects $\mathbf{R}_{i_j} \xleftarrow{\$} \{1,-1\}^{\frac{(d+3)m}{2} \times \frac{(d+3)m}{2}}$;
  * Defines and calculates
$$\bar{\mathbf{R}}_j = \sum_{i=1}^{\ell} dk_{S,i}\mathbf{R}_{i_j} \in \{-\ell, -\ell+1, \cdots, \ell\}^{\frac{(d+3)m}{2} \times \frac{(d+3)m}{2}}; \tag{12}$$
  * Selects $x_j \leftarrow \mathbf{\Psi}_\alpha \in \mathbb{Z}_q$ and $\mathbf{y}_j \leftarrow \mathbf{\Psi}_\alpha^{\frac{(d+3)m}{2}} \in \mathbb{Z}_q^{\frac{(d+3)m}{2}}$ as noise vectors;
  * Calculates
$$\mathbf{z}_j \leftarrow \bar{\mathbf{R}}_j^\top \mathbf{y}_j \in \mathbb{Z}_q^{\frac{(d+3)m}{2}}; \tag{13}$$
$$c_{0_j} = (\mathbf{r}_R^\top + H_3(kw)^\top)\mathbf{s}_j + x_j + b_j\lfloor\frac{q}{2}\rfloor \in \mathbb{Z}_q; \tag{14}$$
$$\mathbf{c}_{\mathbf{1}_j} = (\mathbf{F}_{dk} \parallel \mathbf{F}_t)^\top \mathbf{s}_j + \begin{bmatrix}\mathbf{y}_j \\ \mathbf{z}_j\end{bmatrix} \in \mathbb{Z}_q^{(d+3)m}. \tag{15}$$
- Outputs the forward-secure searchable ciphertext $\mathsf{ct} := (\{c_{0_j}, \mathbf{c}_{\mathbf{1}_j}, b_j\}_{j=1}^{\rho})$.

– $\mathsf{Trapdoor}(pp, pk_S, pk_R, sk_R(t), kw')$: After input the public parameter pp, the public key of the sender pk_S, the public key of the receiver pk_R, the secret key of the receiver $sk_R(t)$ with time t and one keyword $kw' \in \{1,-1\}^{\ell}$, this algorithm will process the following steps.
- For $i = 1, 2, \cdots, \kappa$, calculates
$$h_{R,i} \leftarrow \lfloor\frac{2(\mathbf{c}_{S,i}^\top \cdot \mathbf{h}_R (\mathrm{mod} q))}{q}\rceil, \tag{16}$$
$$p_{R,i} \leftarrow \lfloor\frac{2(\mathbf{s}_{R,i}^\top \cdot \mathbf{p}_S (\mathrm{mod} q))}{q}\rceil, \tag{17}$$
and defines $y_{R,i} = h_{R,i} \cdot p_{R,i}$.
- Defines $\mathbf{y}_R = y_{R,1}y_{R,2}\cdots y_{R,\kappa} \in \{0,1\}^{\kappa}$.

  - Calculates
$$\mathbf{dk}_R = dk_{R,1}dk_{R,2}\cdots dk_{R,\ell} \leftarrow H_2(kw', \mathbf{y}_R) \tag{18}$$
  - Defines and calculates
$$\mathbf{M}_{dk} = \mathbf{C}_R + \sum_{i=1}^{\ell} dk_{R,i}\mathbf{M}_{R,i} \tag{19}$$
  - Calls $\mathsf{SampleLeft}(\mathbf{M}_R, \mathbf{M}_{dk}, \mathbf{S}_R, \mathbf{r}_R, \sigma_2)$ algorithm to generate $\mathbf{Trap_1} \in \mathbb{Z}_q^{2m}$.
  - If $sk_R(t)$ includes the basis $\mathbf{S}_{\Theta^{(t)}}$, this algorithm will continue the remainder procedures; If $sk_R(t)$ does not include the basis $\mathbf{S}_{\Theta^{(t)}}$, this algorithm will call $\mathsf{ExtBasis}(F_{\Theta^{(t)}}, \mathbf{S}_{\Theta^{(i)}})$ to generate it and then continue the remainder procedures.
  - Calls $\mathsf{SamplePre}(\mathbf{S}_{\Theta^{(t)}}, H_3(kw'), \sigma_3)$ algorithm to generate $\mathbf{Trap_2} \in \mathbb{Z}_q^{(d+1)m}$.
  - Outputs $\mathbf{Trap} := (\mathbf{Trap_1}, \mathbf{Trap_2})$.
- $\mathsf{Test}(pp, \mathsf{ct}, \mathbf{Trap})$:
  - For $j = 1, 2, \cdots, \rho$, calculates
$$v_j = c_{0_j} - \begin{pmatrix}\mathbf{Trap_1}\\ \mathbf{Trap_2}\end{pmatrix}^{\top} \mathbf{c}_{1_j}. \tag{20}$$
  - Checks whether it satisfies $\lfloor v_j - \lfloor \frac{q}{2} \rfloor \rfloor$:
If the inequality holds, sets $v_j = 1$;
Otherwise, sets $v_j = 0$.
  - This algorithm outputs 1 if and only if for $\forall j = 1, 2, \cdots, \rho$, it satisfies $v_j = b_j$, which implies the $\mathsf{Test}(pp, \mathsf{ct}, \mathbf{Trap})$ algorithm succeeds;
Otherwise, it outputs 0, which implies the $\mathsf{Test}(pp, \mathsf{ct}, \mathbf{Trap})$ algorithm fails.

6 Parameters and Correctness

6.1 Parameters Setting

Here, we illustrate the following restrictions of parameters choosing to guarantee the rationality and correctness of our scheme [40], [42], [44], [45].

1. $m \geq 6n \log q$ to make $\mathsf{TrapGen}(n, m, q)$ algorithm process properly.
2. $s \geq \eta_\epsilon(\Lambda^\perp(\mathbf{A}_{\mathsf{label}}))$ for some $\epsilon = \mathsf{negl}(n)$ and $t = \sigma_1\sqrt{m} \cdot (\sqrt{\log n})$ to make $\mathsf{KeyGen}_S(pp)$ and $\mathsf{KeyGen}_R(pp)$ run properly.
3. $\sigma_1 = 2\sqrt{n}$ and $q > \frac{2\sqrt{n}}{\alpha}$ to make the lattice reduction algorithm is correct.
4. $\sigma_2 > \ell \cdot m \cdot \omega(\sqrt{\log n})$ to make $\mathsf{SampleLeft}(\mathbf{A}, \mathbf{M}, \mathbf{T_A}, \mathbf{u}, \sigma)$ algorithm execute properly.
5. $m \geq 2n\lceil \log q \rceil$ and $\sigma_3 \geq \parallel \tilde{\mathbf{B}} \parallel \cdot \omega(\sqrt{\log n})$ to make $\mathsf{SamplePre}(\mathbf{A}, \mathbf{T_A}, \mathbf{u}, \sigma)$ algorithm operate properly.
6. $\frac{(d+3)m}{2}$ is an integer to make $\mathsf{FS-PAEKS}(pp, pk_S, sk_S, pk_R, kw, t, d)$ algorithm work properly.
7. $q > \sigma_1 m^{\frac{3}{2}} \omega(\sqrt{\log n})$ to make first error term is bounded legitimately and $\mathbf{y}_S = \mathbf{y}_R$.
8. $\alpha < [\sigma_2 \ell m \omega(\sqrt{\log n})]^{-1}$ and $q = \Omega(\sigma_2 m^{\frac{3}{2}})$ to make second error term is bounded legitimately.

6.2 Correctness Proof

Our cryptographic primitive comprises two error terms, and we demonstrate that if both of these terms are bounded legitimately, the entire scheme is correct. The correctness proof is presented via the following two theorems.

Theorem 1. *If the keywords holds $kw = kw'$ and the* first error term $(\mathbf{r}_{R,i}^{\top} \cdot \mathbf{h}_{S,i}$ *and* $\mathbf{e}_{S,i}^{\top} \cdot \mathbf{h}_{R,i})$ *is less than* $\frac{\epsilon \cdot q}{8}$*, then we obtain the equality* $\mathbf{dk}_S = \mathbf{dk}_R$.

Proof. For $i = 1, 2, \cdots, \kappa$, calculates:

$$\begin{aligned} h_{S,i} &= \lfloor \frac{2(\mathbf{c}_{R,i}^{\top} \cdot \mathbf{h}_S (\mathrm{mod} q))}{q} \rceil \\ &= \lfloor \frac{2(\mathbf{r}_i^{\top} \cdot \mathbf{A}_{\mathsf{label}}) \cdot \mathbf{h}_S (\mathrm{mod} q)}{q} + \underbrace{\frac{2\mathbf{r}_{R,i}^{\top} \cdot \mathbf{h}_S (\mathrm{mod} q)}{q}}_{\text{first error term}} \rceil \\ &= \lfloor \frac{2((\mathbf{r}_i^{\top} \cdot \mathbf{A}_{\mathsf{label}}) \cdot \mathbf{h}_S (\mathrm{mod} q))}{q} \rceil \\ &= p_{R,i}; \end{aligned} \tag{21}$$

For $i = 1, 2, \cdots, \kappa$, calculates:

$$\begin{aligned} h_{R,i} &= \lfloor \frac{2(\mathbf{c}_{S,i}^{\top} \cdot \mathbf{h}_R (\mathrm{mod} q))}{q} \rceil \\ &= \lfloor \frac{2(\mathbf{s}_i^{\top} \cdot \mathbf{A}_{\mathsf{label}}) \cdot \mathbf{h}_R (\mathrm{mod} q)}{q} + \underbrace{\frac{2\mathbf{r}_{S,i}^{\top} \cdot \mathbf{h}_R (\mathrm{mod} q)}{q}}_{\text{first error term}} \rceil \\ &= \lfloor \frac{2((\mathbf{r}_i^{\top} \cdot \mathbf{A}_{\mathsf{label}}) \cdot \mathbf{h}_R (\mathrm{mod} q))}{q} \rceil \\ &= p_{S,i} \end{aligned} \tag{22}$$

For $i = 1, 2, \cdots, \kappa$, we have the following equalities: $y_{S,i} = h_{S,i} \cdot p_{S,i} = p_{R,i} \cdot p_{S,i} = p_{S,i} \cdot p_{R,i} = h_{R,i} \cdot p_{R,i} = y_{R,i}$. Therefore, we can say that $\mathbf{y}_S = \mathbf{y}_R$. In addition, because of $kw = kw'$, we obtain that $\mathbf{dk}_S = H_2(kw, \mathbf{y}_S) = H_2(kw', \mathbf{y}_S) = H_2(kw', \mathbf{y}_R) = \mathbf{dk}_R$.

Theorem 2. *If the* second error term $(x_j - \begin{pmatrix} \mathbf{Trap_1} \\ \mathbf{Trap_2} \end{pmatrix}^{\top} \begin{bmatrix} \mathbf{y}_j \\ \mathbf{z}_j \end{bmatrix})$ *has been bounded by* $((q \cdot \sigma_2 \cdot \ell \cdot m \cdot \alpha \cdot \omega(\sqrt{\log m}) + \mathcal{O}(\ell \sigma_2 m^{\frac{3}{2}})) \leq \frac{q}{5})$*, then the* $\mathsf{Test}(pp, \mathsf{ct}, \mathbf{Trap})$ *algorithm outputs the correct result, that is,* b_j *is correct.*

Proof.

$$
\begin{aligned}
v_j &= c_{0_j} - \begin{pmatrix} \mathbf{Trap_1} \\ \mathbf{Trap_2} \end{pmatrix}^\top \mathbf{c_1}_j \\
&= (\mathbf{r}_R^\top + H_3(kw)^\top)\mathbf{s}_j + x_j + b_j \lfloor \frac{q}{2} \rfloor - \begin{pmatrix} \mathbf{Trap_1} \\ \mathbf{Trap_2} \end{pmatrix}^\top \mathbf{c_1}_j \\
&= \mathbf{r}_R^\top \mathbf{s}_j + x_j + b_j \lfloor \frac{q}{2} \rfloor + H_3(kw)^\top \mathbf{s}_j - \begin{pmatrix} \mathbf{Trap_1} \\ \mathbf{Trap_2} \end{pmatrix}^\top \mathbf{c_1}_j \\
&= \mathbf{r}_R^\top \mathbf{s}_j + x_j + b_j \lfloor \frac{q}{2} \rfloor + H_3(kw)^\top \mathbf{s}_j - \begin{pmatrix} \mathbf{Trap_1} \\ \mathbf{Trap_2} \end{pmatrix}^\top [(\mathbf{F}_{dk} \parallel \mathbf{F}_t)^\top \mathbf{s}_j + \begin{bmatrix} \mathbf{y}_j \\ \mathbf{z}_j \end{bmatrix}] \\
&= \mathbf{r}_R^\top \mathbf{s}_j + x_j + b_j \lfloor \frac{q}{2} \rfloor + H_3(kw)^\top \mathbf{s}_j - (\mathbf{Trap_1}\mathbf{F}_{dk} + \mathbf{Trap_1}\mathbf{F}_t)\mathbf{s}_j - \begin{pmatrix} \mathbf{Trap_1} \\ \mathbf{Trap_2} \end{pmatrix}^\top \begin{bmatrix} \mathbf{y}_j \\ \mathbf{z}_j \end{bmatrix} \\
&= b_j \lfloor \frac{q}{2} \rfloor + \underbrace{x_j - \begin{pmatrix} \mathbf{Trap_1} \\ \mathbf{Trap_2} \end{pmatrix}^\top \begin{bmatrix} \mathbf{y}_j \\ \mathbf{z}_j \end{bmatrix}}_{\text{second error term}}
\end{aligned} \tag{23}
$$

Therefore, as mentioned in Lemma 22 of reference [42], for $j = 1, 2, \cdots, \rho$, if the given keywords are absolutely identical, we can conclude that $v_j = b_j$.

7 Security Analysis

This section shows two theorems and one corollary to show that the proposed FS-PAEKS primitive satisfies CI under IND-CKA, TP under IND-IKGA, and MCI under IND-Multi-CKA. We illustrate the proofs of the two theorems through the sequence-of-games tool, proposed by Shoup [46] and give the analysis of the corollary.

Theorem 3. *The proposed* $\mathsf{FS-PAEKS}$ *scheme satisfies* CI *under* $\mathsf{IND-CKA}$ *if the* SPHF *protocol satisfies pseudo-randomness and the hash function* H_2 *is a random oracle.*

Proof. We finished the security analysis through four games as below.
$\widehat{\textbf{Game 0}}$: We simulate a real security game for the adversary $\mathcal{A}$ and define $Adv_{\mathcal{A}}^{\widehat{\textbf{Game 0}}}(\lambda) := \epsilon$. $\mathcal{A}$ has the ability to perform three oracle queries and the challenger $\mathcal{C}$ will reply to the following responses after receiving some keyword kw from $\mathcal{A}$.

- $\mathcal{O}_{\mathcal{KU}}$: If the time period $t < T-1$, $\mathcal{C}$ updates $sk_R(t+1) \leftarrow \mathsf{KeyUpdate}(pp, pk_R, sk_R, t, d)$ and returns $sk_R(t+1)$ to $\mathcal{A}$; If the time period $t = T-1$, $\mathcal{C}$ returns an empty string sk_T to $\mathcal{A}$.
- $\mathcal{O}_{\mathcal{C}}$: Given keyword kw, $\mathcal{C}$ calculates $\mathsf{ct} \leftarrow \mathsf{FS-PAEKS}(pp, pk_S, sk_S, pk_R, kw, t, d)$ and returns ct to $\mathcal{A}$.
- $\mathcal{O}_{\mathcal{T}}$: Given keyword kw, $\mathcal{C}$ calculates $\mathbf{Trap} \leftarrow \mathsf{Trapdoor}(pp, pk_S, pk_R, sk_R(t), kw)$ and returns $\mathbf{Trap}$ to $\mathcal{A}$.

Oracle $\mathcal{O}_{\mathcal{KU}}$

if $t < T-1$
updates $sk_R(t)$ to $sk_R(t+1)$
returns $sk_R(t+1)$ to $\mathcal{A}$
else if $t = T-1$
returns empty string sk_T to $\mathcal{A}$

Oracle $\mathcal{O}_{\mathcal{C}}$

given keyword kw
calculates ct
returns ct to $\mathcal{A}$

Oracle $\mathcal{O}_{\mathcal{T}}$

given keyword kw
calculates **Trap**
returns **Trap** to $\mathcal{A}$

$\widehat{\textbf{Game 1}}$: This game is identical to $\widehat{\textbf{Game 0}}$, except changing the calculation method of ct^* in the **Challenge** query. To be more specific, $\mathcal{C}$ selects $h_{S,i} \xleftarrow{\$} \mathcal{OS}_{h_{S,i}}$ randomly ($\mathcal{OS}_{h_{S,i}}$ is the output space of $h_{S,i}$) instead of calculating $h_{S,i} \leftarrow \lfloor \frac{2(\mathbf{c}_{R,i}^\top \cdot \mathbf{h}_S (\bmod q))}{q} \rceil$, where $i = 1, 2, \cdots, \kappa$. For the view of $\mathcal{A}$, $\widehat{\textbf{Game 1}}$ and $\widehat{\textbf{Game 0}}$ are statistically indistinguishable due to the fact that the output of $h_{S,i}$ satisfies pseudo-randomness. Hence, we acquire:

$$|Adv_{\mathcal{A}}^{\widehat{\textbf{Game 1}}}(\lambda) - Adv_{\mathcal{A}}^{\widehat{\textbf{Game 0}}}(\lambda)| \leq \mathbf{negl}(\lambda).$$

$\underline{\widehat{\textbf{Game 1}}:}$
$\mathcal{C}$ randomly selects $h_{S,i} \xleftarrow{\$} \mathcal{OS}_{h_{S,i}}$
Game 1 and **Game 0** are statistically indistinguishable

$\widehat{\textbf{Game 2}}$: This game is identical to $\widehat{\textbf{Game 1}}$, except changing one more time of the calculation method for ct^* in the **Challenge** query. In detail, $\mathcal{A}$ sends kw_0^* and kw_1^* to $\mathcal{C}$, $\mathcal{C}$ then selects a bit $b \in \{0, 1\}$ randomly and samples $\mathbf{dk}_S \xleftarrow{\$} \mathcal{KS}_{\mathsf{FS-PAEKS}}$ randomly ($\mathcal{KS}_{\mathsf{FS-PAEKS}}$ is the keyword space of $\mathsf{FS-PAEKS}(pp, pk_S, sk_S, pk_R, kw, t, d)$ algorithm), instead of calculating $\mathbf{dk}_S \leftarrow H_2(kw_b^*, \mathbf{y}_S)$, where $\mathbf{y}_S = y_{S,1} y_{S,1} \cdots y_{S,\kappa}$. In this way, the output of $H_2(kw_b^*, \mathbf{y}_S)$ is random since $h_{S,i}$ satisfies pseudo-randomness and H_2 is also a random oracle. Accordingly, in $\mathcal{A}$'s view, $\widehat{\textbf{Game 2}}$ and $\widehat{\textbf{Game 1}}$ are statistically indistinguishable. Thus, we can say:

$$|Adv_{\mathcal{A}}^{\widehat{\textbf{Game 2}}}(\lambda) - Adv_{\mathcal{A}}^{\widehat{\textbf{Game 1}}}(\lambda)| \leq \mathbf{negl}(\lambda).$$

$\underline{\widehat{\textbf{Game 2}}:}$
$\mathcal{A}$ sends kw_0^* and kw_1^* to $\mathcal{C}$
$\mathcal{C}$ randomly selects $b \in \{0, 1\}$
$\mathcal{C}$ samples $\mathbf{dk}_S \xleftarrow{\$} \mathcal{KS}_{\mathsf{FS-PAEKS}}$
Game 2 and **Game 1** are statistically indistinguishable

$\widehat{\textbf{Game 3}}$: Till now, the keyword is generated by $\mathbf{dk}_S \xleftarrow{\$} \mathcal{KS}_{\mathsf{FS-PAEKS}}$ at random, the challenge ciphertext $\mathsf{ct}^* = (\{c_{0_j}^*, \mathbf{c}_{\mathbf{1}_j}^*, b_j^*\}_{j=1}^{\rho})$ is generated through calling $\mathsf{FS-PAEKS}(pp, pk_S, sk_S, pk_R, kw_b^*, t^*, d)$ algorithm, where $c_{0_j}^* = (\mathbf{r}_R^\top + H_3(kw_b^*)^\top)\mathbf{s}_j + x_j + b_j \lfloor \frac{q}{2} \rfloor$, $\mathbf{c}_{\mathbf{1}_j}^* = (\mathbf{F}_{dk} \parallel \mathbf{F}_{t^*})^\top \mathbf{s}_j + \begin{bmatrix} \mathbf{y}_j \\ \mathbf{z}_j \end{bmatrix}$, and $b_j^* \xleftarrow{\$} \{0, 1\}$ randomly. Therefore, ct^* does not divulge any information regarding to the challenge keywords (kw_0^*, kw_1^*). As for $\mathcal{A}$, the only way to acquire the keyword is by guessing absolutely. Consequently, we obtain the following:

$$|Adv_{\mathcal{A}}^{\widehat{\textbf{Game 3}}}(\lambda)| = 0.$$

$\underline{\widehat{\textbf{Game 3}}:}$
$\mathbf{dk}_S \xleftarrow{\$} \mathcal{KS}_{\mathsf{FS-PAEKS}}$ randomly
$\mathsf{ct}^* = (\{c_{0_j}^*, \mathbf{c}_{\mathbf{1}_j}^*, b_j^*\}_{j=1}^{\rho}) \leftarrow \mathsf{FS-PAEKS}(pp, pk_S, sk_S, pk_R, kw_b^*, t^*, d)$
$c_{0_j}^* = (\mathbf{r}_R^\top + H_3(kw_b^*)^\top)\mathbf{s}_j + x_j + b_j\lfloor\frac{q}{2}\rfloor$
$\mathbf{c}_{\mathbf{1}_j}^* = (\mathbf{F}_{dk} \parallel \mathbf{F}_{t^*})^\top\mathbf{s}_j + \begin{bmatrix} \mathbf{y}_j \\ \mathbf{z}_j \end{bmatrix}$
$b_j^* \xleftarrow{\$} \{0,1\}$

Theorem 4. *The proposed* $\mathsf{FS-PAEKS}$ *scheme satisfies* TP *under* $\mathsf{IND-IKGA}$ *if the* SPHF *protocol satisfies pseudo-randomness and the hash function* H_2 *is a random oracle.*

Proof. We finished the security analysis through four games as below.
$\widehat{\textbf{Game 0}}$: We simulate a real security game for the adversary $\mathcal{A}$ and define $Adv_{\mathcal{A}}^{\widehat{\textbf{Game 0}}}(\lambda) := \epsilon$. $\mathcal{A}$ has the ability to perform three oracle queries and the challenger $\mathcal{C}$ will reply to the responses (same as the proof of the former theorem) after receiving some keyword kw from $\mathcal{A}$.

$\underline{\widehat{\textbf{Game 0}}:}$
defines $Adv_{\mathcal{A}}^{\widehat{\textbf{Game 0}}}(\lambda) := \epsilon$
$\mathcal{C}$ reply to the responses after receiving keyword kw from $\mathcal{A}$

$\widehat{\textbf{Game 1}}$: This game is identical to $\widehat{\textbf{Game 0}}$, except changing the calculation method of $\textbf{Trap}^*$ in the **Challenge** query. To be more specific, $\mathcal{C}$ selects $h_{R,i} \xleftarrow{\$} \mathcal{OS}_{h_{R,i}}$ randomly ($\mathcal{OS}_{h_{R,i}}$ is the output space of $h_{R,i}$) instead of calculating $h_{R,i} \leftarrow \lfloor\frac{2(\mathbf{c}_{S,i}^\top \cdot \mathbf{h}_R(\bmod q))}{q}\rceil$, where $i = 1, 2, \cdots, \kappa$. For the view of $\mathcal{A}$, $\widehat{\textbf{Game 1}}$ and $\widehat{\textbf{Game 0}}$ are statistically indistinguishable due to the fact that the output of $h_{R,i}$ satisfies pseudo-randomness. Hence, we acquire:

$$|Adv_{\mathcal{A}}^{\widehat{\textbf{Game 1}}}(\lambda) - Adv_{\mathcal{A}}^{\widehat{\textbf{Game 0}}}(\lambda)| \leq \mathbf{negl}(\lambda).$$

$\underline{\widehat{\textbf{Game 1}}:}$
$\mathcal{C}$ randomly selects $h_{R,i} \xleftarrow{\$} \mathcal{OS}_{h_{R,i}}$
Game 1 and **Game 0** are statistically indistinguishable

$\widehat{\textbf{Game 2}}$: This game is identical to $\widehat{\textbf{Game 1}}$, except changing one more time of the calculation method for $\textbf{Trap}^*$ in the **Challenge** query. In detail, $\mathcal{A}$ sends kw_0^* and kw_1^* to $\mathcal{C}$, $\mathcal{C}$ then selects a bit $b \in \{0,1\}$ randomly and samples $\mathbf{dk}_R \xleftarrow{\$} \mathcal{KS}_{\mathsf{FS-PAEKS}}$ randomly, instead of calculating $\mathbf{dk}_R \leftarrow$

$H_2(kw_b^*, \mathbf{y}_R)$, where $\mathbf{y}_R = y_{R,1}y_{R,1} \cdots y_{R,\kappa}$. In this way, the output of $H_2(kw_b^*, \mathbf{y}_R)$ is random since $h_{R,i}$ satisfies pseudo-randomness and H_2 is also a random oracle. Accordingly, in $\mathcal{A}$'s view, $\widehat{\textbf{Game}}$ **2** and $\widehat{\textbf{Game}}$ **1** are statistically indistinguishable. Thus, we can say:

$$|Adv_{\mathcal{A}}^{\widehat{\textbf{Game}}\ \textbf{2}}(\lambda) - Adv_{\mathcal{A}}^{\widehat{\textbf{Game}}\ \textbf{1}}(\lambda)| \leq \textbf{negl}(\lambda).$$

$\underline{\widehat{\textbf{Game}}\ \textbf{2}:}$
$\mathcal{A}$ sends kw_0^* and kw_1^* to $\mathcal{C}$
$\mathcal{C}$ selects $b \in \{0, 1\}$ randomly
$\mathcal{C}$ samples $\mathbf{dk}_R \xleftarrow{\$} \mathcal{KS}_{\mathsf{FS-PAEKS}}$ randomly
Game 2 and **Game 1** are statistically indistinguishable

$\widehat{\textbf{Game}}$ **3**: Till now, the keyword is generated by $\mathbf{dk}_R \xleftarrow{\$} \mathcal{KS}_{\mathsf{FS-PAEKS}}$ at random, the challenge trapdoor $\textbf{Trap}^* = (\textbf{Trap}_\textbf{1}{}^*, \textbf{Trap}_\textbf{2}{}^*)$ is generated through calling $\mathsf{Trapdoor}(pp, pk_S, pk_R, sk_R(t^*), kw_b^*)$ algorithm. Therefore, $\textbf{Trap}^*$ does not divulge any information regarding to the challenge keywords (kw_0^*, kw_1^*). As for $\mathcal{A}$, the only way to acquire the keyword is by guessing absolutely. Consequently, we obtain:

$$|Adv_{\mathcal{A}}^{\widehat{\textbf{Game}}\ \textbf{3}}(\lambda)| = 0.$$

$\underline{\widehat{\textbf{Game}}\ \textbf{3}:}$
$\mathbf{dk}_R \xleftarrow{\$} \mathcal{KS}_{\mathsf{FS-PAEKS}}$ randomly
$\textbf{Trap}^* = (\textbf{Trap}_\textbf{1}{}^*, \textbf{Trap}_\textbf{2}{}^*) \leftarrow \mathsf{Trapdoor}(pp, pk_S, pk_R, sk_R(t^*), kw_b^*)$

Corollary 1. *The proposed* FS − PAEKS *scheme satisfies* MCI *under* IND − Multi − CKA *if it satisfies* CI *under* IND − CKA *and the* PEKS.PEKS *algorithm in our* FS − PAEKS *algorithm is probabilistic.*

Analysis. Our FS − PAEKS algorithm involved PEKS.PEKS algorithm. To the best of our knowledge, the existing PEKS.PEKS algorithm satisfies probabilistic [1,20]. Thus, our FS − PAEKS scheme is also probabilistic. In addition, we have proved that our scheme satisfies CI under IND − CKA. Consequently, the proposed FS − PAEKS scheme satisfies MCI under IND − Multi − CKA.

8 Implementation and Comparison

In this section, we present a comprehensive performance evaluation and give a comparison with other existing top-tier PEKS and PAEKS schemes (including Boneh et al. [1], Huang et al. [16], Behnia et al. [20], Zhang et al. [47], Zhang et al. [14], Liu et al. [10], Emura [48], Cheng et al. [31]) with regards to the security properties, computation overhead, communication overhead, and computational complexity, respectively.

We reviewed and cryptanalyzed eight PEKS and PAEKS schemes and illustrate the six security properties comparison in terms of FS, CI, MCI, TP, PQ, and WTA in Table. 1. To begin with, there is only one scheme [14] that satisfies the FS property. Moreover, although several schemes satisfy CI and MCI properties, their primitives may be tampered with by generating ciphertexts to guess keywords and performing tests adaptively, which significantly reduced the feasibility and security ([1], [16], [20], [47], [14]). In addition, some schemes have not taken post-quantum or without trusted settings into consideration ([1], [16], [47]). In a nutshell, to achieve a higher security level and a practical utility, we not only consider the fundamental security properties (CI, MCI, and TP), but also take post-quantum, forward security, and without trusted authority into account to show the security superiority of our proposed primitive.

Table 1. Security properties comparison with other existing PEKS and PAEKS schemes

Schemes	**FS**	**CI**	**MCI**	**TP**	**PQ**	**WTA**
Boneh et al. [1]	×	✓	✓	×	×	✓
Huang et al. [16]	×	×	×	×	×	✓
Behnia et al. [20]	×	✓	✓	×	✓	✓
Zhang et al. [47]	×	✓	✓	×	✓	×
Zhang et al. [14]	✓	✓	✓	×	✓	✓
Liu et al. [10]	×	✓	✓	✓	✓	✓
Emura [48]	×	✓	✓	✓	✓	✓
Cheng et al. [31]	×	✓	✓	✓	✓	✓
Our scheme	✓	✓	✓	✓	✓	✓

Notes. ✓: This scheme satisfies the corresponding property. ×: This scheme does not satisfy the corresponding property. **FS**: Forward security. **CI**: Ciphertext indistinguishability. **MCI** : Multi-ciphertext indistinguishability. **TP**: Trapdoor privacy. **PQ**: Post-quantum. **WTA**: Without trusted authority.

Besides, we subsequently compared the computational complexity and communication overhead through theoretical analysis with other recent post-quantum PEKS and PAEKS schemes ([20], [14], [10]) in Table. 2, and Table. 3. As for the Table. 2, we just consider the most time-consuming computational operations of each scheme, that is, multiplication (T_{Mul}), hash function (T_{HF}), SampleLeft(T_{SL}) algorithm, SamplePre(T_{SP}) algorithm, and BasisDel(T_{BD}) algorithm. We describe the ciphertext generation, trapdoor generation, and test generation, respectively. With regard to the Table. 3, we analyze the communication overhead in terms of ciphertext size and trapdoor size of each scheme.

Table 2. Computational complexity comparison

Schemes	**Ciphertext Generation**	**Trapdoor Generation**	**Test Generation**
Behnia et al. [20]	$\rho(m^2+2nm+n+\ell+1)T_{Mul}$	$\ell T_{Mul}+T_{SL}$	$2\rho m T_{Mul}$
Zhang et al. [14]	$T_{HF}+(\rho n+nm^2+\rho)T_{Mul}+T_{SP}$	$T_{HF}+nm^2TMul$ $+\mathrm{T}_{BD}+T_{SP}$	$T_{HF}+(\ell m+nm)T_M$
Liu et al. [10]	$T_{HF}+(\kappa(m+n+1)$ $+\rho(m^2+2nm+n+\ell+1))T_{Mul}$	$T_{HF}+(\kappa(m+n+1)$ $+\ell)T_{Mul}+T_{SL}$	$2\rho m T_M$
Our scheme	$(\rho+1)T_{HF}+(\kappa(m+n+1)$ $+\rho(\frac{(d+3)^2m^2}{4}+(d+3)nm$ $+2\mathrm{n}+\ell+1))T_{Mul}$	$2T_{HF}+(\kappa(m+n+1)$ $+\ell)T_{Mul}+T_{SL}+T_{SP}$	$(d+3)\rho m T_M$

Notes. κ : This parameter is related to the security parameter λ. ρ : This parameter is related to the security parameter. m : This parameter means the dimension. q : This parameter means modules. ℓ : This parameter means the length of the keyword. d : This parameter means the depth of the binary tree.

Table 3. Communication overhead comparison

Schemes	**Ciphertext Size**	**Trapdoor Size**
Behnia et al. [20]	$\kappa(\vert q\vert + 2m\vert q\vert + 1)$	$2m\vert q\vert$
Zhang et al. [14]	$(\ell + m\ell + m)\vert q\vert$	$m\vert q\vert$
Liu et al. [10]	$\rho(\vert q\vert + 2m\vert q\vert + 1)$	$2m\vert q\vert$
Our scheme	$\rho(\vert q\vert + (d+3)m\vert q\vert + 1)$	$(d+3)m\vert q\vert$

Notes. κ : This parameter is related to the security parameter. ρ : This parameter is related to the security parameter. m : This parameter means the dimension. q : This parameter means modules. ℓ : This parameter means the length of the keyword. d : This parameter means the depth of the binary tree.

Furthermore, we also implemented the computational overheads in terms of ciphertext generation and test algorithm (Fig. 2 and Fig. 3, respectively) through C++ language on Windows 10, AMD Ryzen 7 5800H CPU with Radeon Graphics 3.20 GHz and 16 GB memory. We set the parameters as $d = 3, m = 9753, n = 256, q = 4096, \ell = 10, \rho = 10, \kappa = 10, \sigma_1 = 8, \sigma_2 = 8$ to realize the 80-bit security level, where d is the depth of the binary tree, ℓ is the length of the keyword kw, ρ, κ are related to the security parameter. The SHA256 hash function was simulated by adopting OpenSSL (https://www.openssl.org/source/).

9 Potential Applications of FS-PAEKS

- **Combining with Electronic Medical Records (EMRs).** Numerous scholars have utilized PEKS primitive to search the EMRs and protect the EMRs' privacy for patients [15, 49, 50]. However, a malicious attacker may recover the keyword kw from the previous search trapdoor **Trap** through keyword guessing attacks. Besides, if the secret keys of patients have been compromised, their sensitive medical data may be disclosed by adversaries. Compared with the existing schemes, our $\mathsf{FS-PAEKS}$ protocol completely avoids those problems and provides better security.
- **Combining with blockchain networks.** Encrypting data for confidentiality purposes before storing it on a blockchain is a widely adopted approach among researchers [51–53]. However, conducting keyword searches on the blockchain has become increasingly challenging. To address this issue, we presented the $\mathsf{FS-PAEKS}$ primitive, which effectively encrypts data while simultaneously ensuring their privacy is maintained.
- **Combining with Industrial Internet of Things (IIoTs).** The PAKES protocol has been employed by several scholars to safeguard the privacy of IIoTs while simultaneously achieving CI and TP security [33]. However, they have failed to account for the potential risks of quantum computing attacks and the likelihood of secret key leakage during communication. Our $\mathsf{FS-PAEKS}$ primitive not only satisfies the requirements outlined by these scholars but also offers enhanced security features such as resistance to quantum attacks and elimination of potential secret key leakage risks. Furthermore, our scheme addresses a previously unresolved issue in its work by satisfying the MCI security requirement.

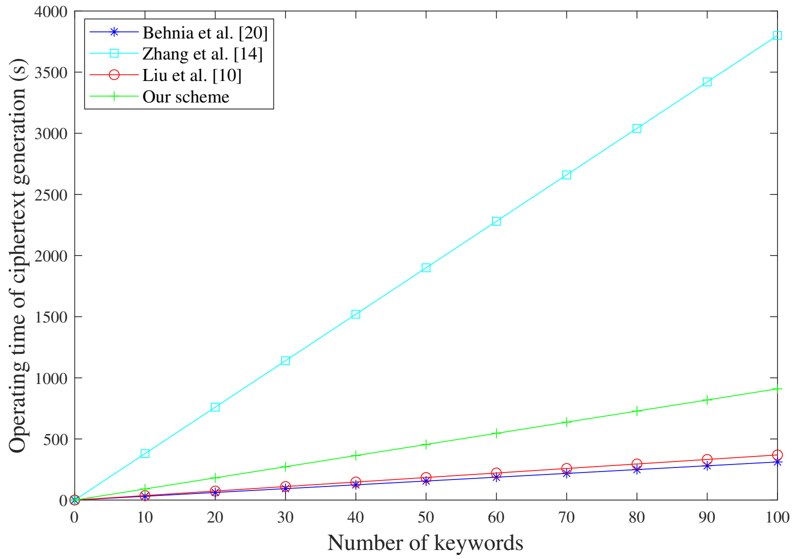

Fig. 2. Ciphertext generation comparison

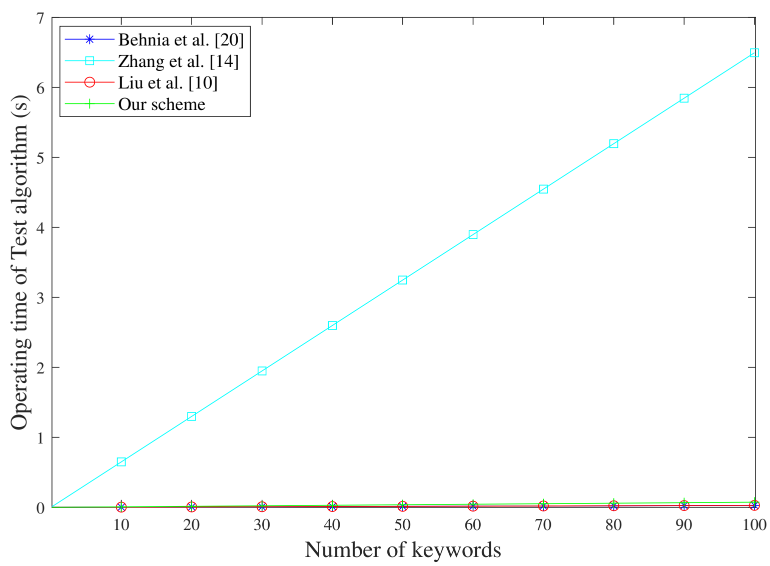

Fig. 3. Test algorithm comparison

10 Conclusions

In this paper, we generalized the first post-quantum public-key authenticated searchable encryption with forward security primitive, namely FS-PAEKS. Our proposed primitive addresses the challenge of secret key exposure while enjoying quantum-safe security without the need for trusted authorities. Technically speaking, we introduced the binary tree structure, the minimal cover set, and ExtBasis and SamplePre algorithms to achieve the post-quantum one-way secret key evolution. Moreover, we demonstrate the proposed scheme satisfies IND-CKA, IND-IKGA, and IND-Multi-CKA in the quantum setting. Besides, we elaborate on the security comparisons with other primitives and also implemented the ciphertext generation and test algorithms. Our proposed primitive offers enhanced efficiency compared to the FS-PEKS protocol. Ultimately, we show three practical applications for FS-PAEKS to illustrate its feasibility and practicability.

We hereby address two open problems, that is, how to construct a post-quantum FS-PAEKS scheme without a random oracle model and construct a post-quantum FS-PAEKS supporting searching multi-keywords.

References

1. Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *Advances in Cryptology-EUROCRYPT 2004: International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004. Proceedings 23*, pages 506–522. Springer, 2004.
2. Joonsang Baek, Reihaneh Safavi-Naini, and Willy Susilo. Public key encryption with keyword search revisited. In *Computational Science and Its Applications–ICCSA 2008: International Conference, Perugia, Italy, June 30–July 3, 2008, Proceedings, Part I 8*, pages 1249–1259. Springer, 2008.
3. 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 *Secure Data Management: Third VLDB Workshop, SDM 2006, Seoul, Korea, September 10-11, 2006. Proceedings 3*, pages 75–83. Springer, 2006.
4. Baodong Qin, Yu Chen, Qiong Huang, Ximeng Liu, and Dong Zheng. Public-key authenticated encryption with keyword search revisited: Security model and constructions. *Information Sciences*, 516:515–528, 2020.
5. Mahnaz Noroozi and Ziba Eslami. Public key authenticated encryption with keyword search: revisited. *IET Information Security*, 13(4):336–342, 2019.
6. Baodong Qin, Hui Cui, Xiaokun Zheng, and Dong Zheng. Improved security model for public-key authenticated encryption with keyword search. In *Provable and Practical Security: 15th International Conference, ProvSec 2021, Guangzhou, China, November 5–8, 2021, Proceedings 15*, pages 19–38. Springer, 2021.
7. Yang Lu and Jiguo Li. Lightweight public key authenticated encryption with keyword search against adaptively-chosen-targets adversaries for mobile devices. *IEEE Transactions on Mobile Computing*, 21(12):4397–4409, 2021.
8. Xiangyu Pan and Fagen Li. Public-key authenticated encryption with keyword search achieving both multi-ciphertext and multi-trapdoor indistinguishability. *Journal of Systems Architecture*, 115:102075, 2021.
9. Qiong Huang, Peisen Huang, Hongbo Li, Jianye Huang, and Hongyuan Lin. A more efficient public-key authenticated encryption scheme with keyword search. *Journal of Systems Architecture*, 137:102839, 2023.

10. Zi-Yuan Liu, Yi-Fan Tseng, Raylin Tso, Masahiro Mambo, and Yu-Chi Chen. Public-key authenticated encryption with keyword search: Cryptanalysis, enhanced security, and quantum-resistant instantiation. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, pages 423–436, 2022.
11. Mihir Bellare and Sara K Miner. A forward-secure digital signature scheme. In *Advances in Cryptology—CRYPTO'99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings 19*, pages 431–448. Springer, 1999.
12. Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In *Advances in Cryptology—EUROCRYPT 2003: International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4–8, 2003 Proceedings 22*, pages 255–271. Springer, 2003.
13. Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. *Journal of Cryptology*, 20:265–294, 2007.
14. Xiaojun Zhang, Chunxiang Xu, Huaxiong Wang, Yuan Zhang, and Shixiong Wang. Fs-peks: Lattice-based forward secure public-key encryption with keyword search for cloud-assisted industrial internet of things. *IEEE Transactions on dependable and secure computing*, 18(3):1019–1032, 2021.
15. Zhe Jiang, Kai Zhang, Liangliang Wang, and Jianting Ning. Forward secure public-key authenticated encryption with conjunctive keyword search. *The Computer Journal*, 2022.
16. Qiong Huang and Hongbo Li. An efficient public-key searchable encryption scheme secure against inside keyword guessing attacks. *Information Sciences*, 403:1–14, 2017.
17. Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.
18. Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.
19. Zi-Yuan Liu, Yi-Fan Tseng, Raylin Tso, Masahiro Mambo, and Yu-Chi Chen. Public-key authenticated encryption with keyword search: A generic construction and its quantum-resistant instantiation. *The Computer Journal*, 65(10):2828–2844, 2022.
20. Rouzbeh Behnia, Muslum Ozgur Ozmen, and Attila Altay Yavuz. Lattice-based public key searchable encryption from experimental perspectives. *IEEE Transactions on Dependable and Secure Computing*, 17(6):1269–1282, 2020.
21. David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. *Journal of cryptology*, 25:601–639, 2012.
22. Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In *Advances in Cryptology—EUROCRYPT 2002: International Conference on the Theory and Applications of Cryptographic Techniques Amsterdam, The Netherlands, April 28–May 2, 2002 Proceedings 21*, pages 45–64. Springer, 2002.
23. Jonathan Katz and Vinod Vaikuntanathan. Smooth projective hashing and password-based authenticated key exchange from lattices. In *Asiacrypt*, volume 5912, pages 636–652. Springer, 2009.
24. Ran Canetti, Dana Dachman-Soled, Vinod Vaikuntanathan, and Hoeteck Wee. Efficient password authenticated key exchange via oblivious transfer. In *Public Key Cryptography–PKC 2012: 15th International Conference on Practice and Theory in Public Key Cryptography, Darmstadt, Germany, May 21-23, 2012. Proceedings 15*, pages 449–466. Springer, 2012.
25. Michel Abdalla, Fabrice Benhamouda, and Philip MacKenzie. Security of the j-pake password-authenticated key exchange protocol. In *2015 IEEE Symposium on Security and Privacy*, pages 571–587. IEEE, 2015.
26. Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. Opaque: an asymmetric pake protocol secure against pre-computation attacks. In *Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part III 37*, pages 456–486. Springer, 2018.
27. Andreas Erwig, Julia Hesse, Maximilian Orlt, and Siavash Riahi. Fuzzy asymmetric password-authenticated key exchange. In *Advances in Cryptology–ASIACRYPT 2020: 26th International Con-*

ference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part II 26, pages 761–784. Springer, 2020.
28. Michel Abdalla, Thorsten Eisenhofer, Eike Kiltz, Sabrina Kunzweiler, and Doreen Riepel. Password-authenticated key exchange from group actions. In *Advances in Cryptology–CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceedings, Part II*, pages 699–728. Springer, 2022.
29. Jonathan Katz and Vinod Vaikuntanathan. Round-optimal password-based authenticated key exchange. *Journal of Cryptology*, 26:714–743, 2013.
30. Rui Zhang and Hideki Imai. Generic combination of public key encryption with keyword search and public key encryption. In *Cryptology and Network Security: 6th International Conference, CANS 2007, Singapore, December 8-10, 2007. Proceedings 6*, pages 159–174. Springer, 2007.
31. Leixiao Cheng and Fei Meng. Public key authenticated encryption with keyword search from lwe. In *Computer Security–ESORICS 2022: 27th European Symposium on Research in Computer Security, Copenhagen, Denmark, September 26–30, 2022, Proceedings, Part I*, pages 303–324. Springer, 2022.
32. Lisha Yao, Jian Weng, Anjia Yang, Xiaojian Liang, Zhenghao Wu, Zike Jiang, and Lin Hou. Scalable cca-secure public-key authenticated encryption with keyword search from ideal lattices in cloud computing. *Information Sciences*, 624:777–795, 2023.
33. Lang Pu, Chao Lin, Biwen Chen, and Debiao He. User-friendly public-key authenticated encryption with keyword search for industrial internet of things. *IEEE Internet of Things Journal*, 2023.
34. Ming Zeng, Haifeng Qian, Jie Chen, and Kai Zhang. Forward secure public key encryption with keyword search for outsourced cloud storage. *IEEE transactions on cloud computing*, 10(1):426–438, 2019.
35. Xinmin Yang, Xinjian Chen, Jianye Huang, Hongbo Li, and Qiong Huang. Fs-ibeks: Forward secure identity-based encryption with keyword search from lattice. *Computer Standards & Interfaces*, 86:103732, 2023.
36. Michel Abdalla, Fabrice Benhamouda, and David Pointcheval. Public-key encryption indistinguishable under plaintext-checkable attacks. *IET Information Security*, 10(6):288–303, 2016.
37. Miklós Ajtai. Generating hard instances of lattice problems. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 99–108, 1996.
38. Chris Peikert. An efficient and parallel gaussian sampler for lattices. In *Advances in Cryptology–CRYPTO 2010: 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings 30*, pages 80–97. Springer, 2010.
39. Shweta Agrawal, Dan Boneh, and Xavier Boyen. Lattice basis delegation in fixed dimension and shorter-ciphertext hierarchical ibe. In *Advances in Cryptology–CRYPTO 2010: 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings 30*, pages 98–115. Springer, 2010.
40. Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *Eurocrypt*, volume 7237, pages 700–718. Springer, 2012.
41. Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 197–206, 2008.
42. Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (h) ibe in the standard model. In *Eurocrypt*, volume 6110, pages 553–572. Springer, 2010.
43. Hyun Sook Rhee, Jong Hwan Park, Willy Susilo, and Dong Hoon Lee. Trapdoor security in a searchable public-key encryption scheme with a designated tester. *Journal of Systems and Software*, 83(5):763–771, 2010.
44. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):1–40, 2009.
45. Zengpeng Li and Ding Wang. Achieving one-round password-based authenticated key exchange over lattices. *IEEE transactions on services computing*, 15(1):308–321, 2019.
46. Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. *cryptology eprint archive*, 2004.

47. Xiaojun Zhang, Yao Tang, Huaxiong Wang, Chunxiang Xu, Yinbin Miao, and Hang Cheng. Lattice-based proxy-oriented identity-based encryption with keyword search for cloud storage. *Information Sciences*, 494:193–207, 2019.
48. Keita Emura. Generic construction of public-key authenticated encryption with keyword search revisited: stronger security and efficient construction. In *Proceedings of the 9th ACM on ASIA Public-Key Cryptography Workshop*, pages 39–49, 2022.
49. Gang Xu, Shiyuan Xu, Yibo Cao, Ke Xiao, Xiu-Bo Chen, Mianxiong Dong, and Shui Yu. Aaq-peks: An attribute-based anti-quantum public-key encryption scheme with keyword search for e-healthcare scenarios. *Cryptology ePrint Archive*, 2023.
50. Hongbo Li, Qiong Huang, Jianye Huang, and Willy Susilo. Public-key authenticated encryption with keyword search supporting constant trapdoor generation and fast search. *IEEE Transactions on Information Forensics and Security*, 18:396–410, 2022.
51. Lanxiang Chen, Wai-Kong Lee, Chin-Chen Chang, Kim-Kwang Raymond Choo, and Nan Zhang. Blockchain based searchable encryption for electronic health record sharing. *Future generation computer systems*, 95:420–429, 2019.
52. Gang Xu, Shiyuan Xu, Yibo Cao, Fan Yun, Yu Cui, Yiying Yu, and Ke Xiao. Ppseb: a postquantum public-key searchable encryption scheme on blockchain for e-healthcare scenarios. *Security and Communication Networks*, 2022, 2022.
53. Mingyue Li, Chunfu Jia, Ruizhong Du, Wei Shao, and Guanxiong Ha. Dse-rb: A privacy-preserving dynamic searchable encryption framework on redactable blockchain. *IEEE Transactions on Cloud Computing*, 2022.