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

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

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

[2] School of Cyberspace Security, Beijing University of Posts and Telecommunications, Beijing, China

[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 (PEKS) was first proposed by Boneh et al. (EUROCRYPT 2004), achieving the ability to search for ciphertext files. Nevertheless, it 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 far-reaching.

In this work, we present the *first* generic construction and instantiation of forward-secure PAEKS primitive based on lattice hardness without trusted authorities, mitigating the secret key exposure while ensuring quantum-safe properties. We extend the scheme of Liu et al. (AsiaCCS 2022), and formalize 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 and IND-IKGA secure in a 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. Ultimately, we demonstrate two potential applications of FS-PAEKS.

C. Ge and M. Yung (Eds.): Inscrypt 2023, LNCS 14526, pp. 274–298, 2024.
https://doi.org/10.1007/978-981-97-0942-7_14

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

1 Introduction

Traditional 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 [2]. A fundamental security criterion for PEKS is to against the chosen keyword attacks (CKA) [3]. 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) [4]. 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 a ciphertext generation algorithm [5]. 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 [6–11].

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 [12]. 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, several scholars introduced the notation of forward security in digital signatures [13–15], which was later adapted by Canetti et al. for use in a forward secure public key encryption scheme [16]. 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 [17,18], it is essential to update them within a certain period to ensure forward security. Zhang et al. 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 [19]. In contrast, Jiang et al. presented a forward secure scheme for PAEKS, without considering quantum computing attacks [20]. 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 [5]. However, with the advancement of quantum computers, Shor generalized a quantum algorithm, demonstrating the feasibility of solving classical cryptographic primitives in probabilistic polynomial times [21,22]. 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 [12,23]. Nevertheless, their schemes contain flaws due to the secret key leakage problem.

Therefore, the aforementioned issues motivate 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

We resolve the above question affirmatively and summarize our contributions as follows.

– We generalize the first PAEKS with forward security instantiation in lattice without trusted authorities, mitigating the secret key exposure while enjoying quantum safety. Our primitive extends Liu et al.'s scheme [12], and proposes a novel post-quantum forward secure PAEKS construction, namely FS-PAEKS. In addition, we formalize the CI, TP, and MCI security of the proposed FS-PAEKS 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. It 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 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 initial phase does not need a trusted setup assumption and the ciphertext can only be obtained by a valid sender. In this way, the trapdoor is valid from a receiver, which avoids adversaries adaptively accessing oracles to obtain the ciphertext for any keyword. Consequently, we introduce a pseudo-random smooth projective hash function to achieve the above property and forward-secure trapdoor privacy under IND-IKGA. In addition, our scheme has also proven to be IND-CKA and IND-Multi-CKA secure in a quantum setting.
– Eventually, we give a security properties comparison with the other eight PEKS and PAEKS primitives. Besides, we compare with Behnia et al.'s scheme [24], Zhang et al.'s scheme [19], and Liu et al.'s scheme [12] in terms of computational complexity and communication overhead theoretically.

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, public key encryption, smooth projective hash functions (SPHF), binary tree structure, and lattice basis extension algorithm. More concretely, we begin by revisiting the post-quantum PAEKS primitive as the basic structure [12]. Next, we employ the SPHF technique to transform the primitive into IND-CCA secure. We then take advantage of the hierarchical structure of the binary tree to represent time periods and utilize $\mathsf{node}(t)$ to represent the smallest minimal cover set for secret key update periodically, following the approach outlined in Cash et al. [25]. 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 introduce the ExtBasis and SamplePre algorithms to facilitate the post-quantum one-way secret key evolution.

Smooth Projective Hash Functions. Smooth projective hash functions, initially proposed by Cramer et al. [26], are utilized to transform one encryption primitive from IND-CPA to IND-CCA. Moreover, numerous scholars extended the SPHF tool to realize password-authenticated key exchange protocols [27–32]. We use a variant kind of SPHF, say "word-independent" SPHF, proposed by Katz et al. [33] for primitive construction. Generally speaking, the "word-independent" SPHF scheme includes five algorithms defined for the NP language $\mathcal{L}$ over a domain $\mathcal{X}$.

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.

- $\mathsf{Setup}(\lambda)$: Given a security parameter λ, this PPT algorithm outputs a language parameter $Para_l$.
- $\mathsf{KeyGen_{Hash}}(Para_l)$: Given $Para_l$, this PPT algorithm outputs outputs hk as the hashing key.
- $\mathsf{KeyGen_{Proj}}(\mathsf{hk}, Para_l)$: Given hk and $Para_l$, this PPT algorithm outputs outputs the projection key pk.
- $\mathsf{Hash}(\mathsf{hk}, Para_l, \chi)$: Given hk, $Para_l$ and a word $\chi \in \mathcal{X}_{Para_l}$, this deterministic algorithm 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, this deterministic algorithm 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 satisfies two attributes:

(1) ϵ-approximate correctness: Given a word $\chi \in \tilde{\mathcal{L}}_{Para_l}$, and 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.
(2) Pseudo-randomness: For some $\delta \in \mathbb{N}$, if a word $\chi \in \tilde{\mathcal{L}}_{Para_l}$, its hash value Hash is indistinguishable from a random element in $\{0,1\}^{\delta}$; Otherwise, Hash is statistically indistinguishable from a random element chosen in $\{0,1\}^{\delta}$.

Binary Tree for Representing Time Periods. We use binary tree encryption primitive for enrolling time periods [16]. 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 four and the last leaf can be described as $t = (1111)$. 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 an 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: $\mathsf{node}(0000) = \{\mathsf{root}\}, \mathsf{node}(0001) = \{0001, 001, 01, 1\}, \mathsf{node}(0010) = \{001, 01, 1\}, \mathsf{node}(0011) = \{0011, 01, 1\}$, $\mathsf{node}(0100) = \{01, 1\}$, $\mathsf{node}\ (0101) = \{0101, 011, 1\}$, $\mathsf{node}(0110) = \{011, 1\}$, $\mathsf{node}(0111) = \{0111, 1\}, \mathsf{node}(1000) = \{1\}, \mathsf{node}(1001) = \{1001, 101, 11\}, \mathsf{node}(1010) = \{101, 11\}, \mathsf{node}(1011) = \{1011, 11\}$, $\mathsf{node}(1100) = \{11\}, \mathsf{node}(1101) = \{1101, 111\}, \mathsf{node}(1110) = \{111\}, \mathsf{node}(1111) = \{1111\}$.

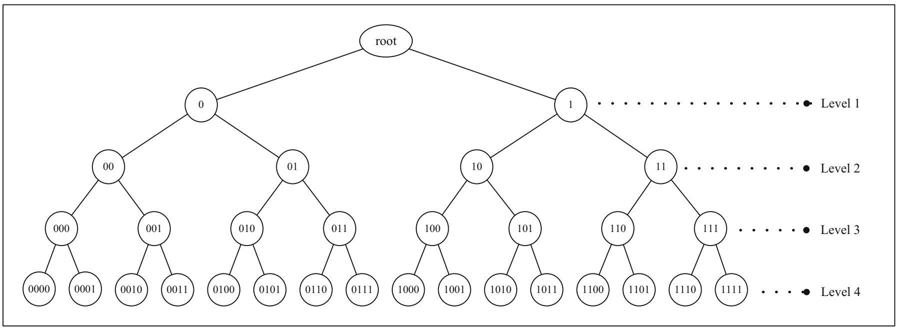

Fig. 1. Binary tree of depth $d = 4$ with binary expression time period (node).

Lattice Basis Extension. We use the lattice basis extension algorithm to construct a secret key one-way evolutionary mechanism (See Lemma 5 in Sect. 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 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

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

1.4 Related Works

Lattice-Based PAEKS. Boneh et al. constructed the concept of PEKS in 2004 [1]. Zhang et al. argued that its security model for keyword privacy is not complete and then defined a new security model [34]. 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 a PAEKS protocol to solve this problem by combining keyword authentication with PEKS [5]. Nevertheless, Liu et al. and Cheng et al. introduced lattice-based PAEKS primitive to achieve quantum resistance [12,35]. Many researchers utilized the PAEKS scheme to preserve privacy for the Internet of Things [9,36,37].

Forward Security. Forward security (FS) in the public-key cryptosystem was initialized by [16]. Zeng et al. introduced the FS notation into the PEKS scheme for cloud computing [38]. Zhang et al. formalized the first lattice-based FS-PEKS primitive [19]. After that, Yang et al. extended the FS-PEKS and constructed a lattice-based FS identity-based encryption with PEKS, namely, FS-IBEKS [39]. Recently, Jiang et al. presented a forward secure public-key authenticated encryption with conjunctive keyword search [20], but without considering the quantum attacks.

1.5 Outline

The rest of this paper is structured as follows. Section 2 covers the preliminary knowledge. In Sect. 3, we present the syntax of forward-secure PAEKS primitive and its security models. The generic construction will be elaborated in Sect. 4, while the security analysis will be specified in Sect. 5. In Sect. 6, we give the lattice-based instantiation. The parameters setting with correctness and theoretical comparison are illustrated in Sects. 7 and 8, respectively. Section 9 shows two applications of FS-PAEKS. Finally, we conclude this paper in Sect. 10.

2 Preliminaries

2.1 Public-Key Encryption with Keyword Search Scheme

Public-key encryption with keyword search (abbr. 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 a security parameter λ, this probabilistic-polynomial time (PPT) algorithm outputs $\mathsf{pk}_{\mathsf{PEKS}}$ and $\mathsf{sk}_{\mathsf{PEKS}}$ as a public key and secret key, respectively.
- $\mathsf{ct}_{\mathsf{PEKS},kw} \leftarrow \mathsf{PEKS}(\mathsf{pk}_{\mathsf{PEKS}}, kw)$: After inputting a public key $\mathsf{pk}_{\mathsf{PEKS}}$ and a keyword kw, this PPT algorithm will output a ciphertext $\mathsf{ct}_{\mathsf{PEKS},kw}$.
- $\mathbf{Trap}_{\mathsf{PEKS},kw'} \leftarrow \mathsf{Trapdoor}(\mathsf{sk}_{\mathsf{PEKS}}, kw')$: Given a secret key $\mathsf{sk}_{\mathsf{PEKS}}$ and a keyword kw', this PPT algorithm outputs a trapdoor $\mathbf{Trap}_{\mathsf{PEKS},kw'}$.
- $(\mathsf{1}$ or $\mathsf{0}) \leftarrow \mathsf{Test}(\mathsf{ct}_{\mathsf{PEKS},kw}, \mathbf{Trap}_{\mathsf{PEKS},kw'})$: After input a ciphertext $\mathsf{ct}_{\mathsf{PEKS},kw}$ and a trapdoor $\mathbf{Trap}_{\mathsf{PEKS},kw'}$, this deterministic algorithm outputs $\mathsf{1}$ if $kw = kw'$; Otherwise, it outputs $\mathsf{0}$.

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

(1) Correctness: Given a security parameter λ, 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:

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

(2) Ciphertext Indistinguiability: 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 ciphertext indistinguishability against chosen keyword attacks (IND-CKA).

2.2 Labelled Public-Key Encryption Scheme

Labelled public-key encryption (abbr. Labelled PKE) is one of the variants of public-key encryption [40]. 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 a 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.

– $\mathsf{ct_{PKE}} \leftarrow \mathsf{Encrypt}(\mathsf{pk_{PKE}}, \mathsf{label}, \mathsf{pt_{PKE}}, \rho)$: Given a public key $\mathsf{pk_{PKE}}$, a label label, a plaintext $\mathsf{pt_{PKE}}$, and a randomness ρ, this PPT algorithm outputs the ciphertext $\mathsf{ct_{PKE}}$.
– ($\mathsf{pt_{PKE}}$ or $\perp$) $\leftarrow \mathsf{Decrypt}(\mathsf{sk_{PKE}}, \mathsf{label}, \mathsf{ct_{PKE}})$: Given a secret key $\mathsf{sk_{PKE}}$, a label label, a ciphertext $\mathsf{ct_{PKE}}$ and a randomness ρ, this deterministic algorithm outputs the plaintext ($\mathsf{pt_{PKE}}$ or $\perp$).

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

(1) Correctness: Given a security parameter λ, a public key and secret key generated by ($\mathsf{pk_{PKE}}$, $\mathsf{sk_{PKE}}$) $\leftarrow \mathsf{KeyExt}(\lambda)$, a label label, a randomness ρ, a ciphertext generated by $\mathsf{ct_{PKE}} \leftarrow \mathsf{Encrypt}$ ($\mathsf{pk_{PKE}}$, label, $\mathsf{pt_{PKE}}, \rho$), the PKE scheme is correct if $\Pr[\mathsf{Decrypt}(\mathsf{sk_{PKE}}, \mathsf{label}, \mathsf{ct_{PKE}}) = \mathsf{pt_{PKE}}] \approx 1$.
(2) IND-CPA/IND-CCA security: A secure PKE protocol satisfies the indistinguishability against chosen-plaintext attacks (IND-CPA) if it does not exist an adversary $\mathcal{A}$ can obtain any information of a challenge plaintext $\mathsf{pt_{PKE}}$. In addition, it realizes indistinguishability against chosen-ciphertext attacks (IND-CCA) if $\mathcal{A}$ is permitted to access the decryption query for any ciphertext $\mathsf{ct_{PKE}}$ excepting for querying the challenge ciphertext.

2.3 Basic Knowledge of Lattice and Trapdoors

Definition 1 (Lattice). *[41] 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). *[42] 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: $\Lambda_q(\mathbf{A}) := \{\mathbf{e} \in \mathbb{Z}^m | \exists \mathbf{s} \in \mathbb{Z}_q^n, \mathbf{A}^\top \mathbf{s} = \mathbf{e} \bmod q\}$, $\Lambda_q^\perp(\mathbf{A}) := \{\mathbf{e} \in \mathbb{Z}^m | \mathbf{A}\mathbf{e} = 0 \bmod q\}$, and $\Lambda_q(\mathbf{A^u}) := \{\mathbf{e} \in \mathbb{Z}^m | \mathbf{A}\mathbf{e} = \mathbf{u} \bmod q\}$.*

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 $\mathcal{D}_{\sigma,\mathbf{c}} = \frac{\rho_{\sigma,\mathbf{c}(\mathbf{x})}}{\rho_{\sigma,\mathbf{c}(\Lambda)}}$ for $\forall \mathbf{x} \in \Lambda$ as the Discrete Gaussian Distribution over Λ with a center $\mathbf{c}$, where $\rho_{\sigma,\mathbf{c}(\mathbf{x})} = \exp(-\pi \frac{\|\mathbf{x}-\mathbf{c}\|^2}{\sigma^2})$ and $\rho_{\sigma,\mathbf{c}(\Lambda)} = \Sigma_{\mathbf{x} \in \Lambda} \rho_{\sigma,\mathbf{c}(\mathbf{x})}$. Specially, we say $\mathcal{D}_{\sigma,0}$ abbreviated as $\mathcal{D}_\sigma$ when $\mathbf{c} = 0$.*

Definition 4. *[43] 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)). *[44] 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. *$\{\mathbf{A} : (\mathbf{A}, \mathbf{T_A}) \leftarrow \mathrm{TrapGen}(n, m, q)\}$ 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$(\mathbf{A}, \mathbf{T_A}, \mathbf{u}, \sigma)$). *[45] 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 $\sigma \le \|\tilde{\mathbf{T_A}}\| \cdot \omega(\sqrt{\log(m)})$, where $m \ge 2n\lceil \log q \rceil$, this* PPT *algorithm publishes a sample $\mathbf{e} \in \mathbb{Z}_q^m$ statistically distributed in $\mathcal{D}_{\Lambda_q^{\mathbf{u}}(\mathbf{A}),\sigma}$ s.t. $\mathbf{Ae} = \mathbf{u} \mod q$.*

Lemma 3 (NewBasisDel$(\mathbf{A}, \mathbf{R}, \mathbf{T_A}, \sigma)$). *[43] Taking a parameter $\sigma \in \mathbb{R}$, a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, a $\mathbb{Z}_q$-invertible matrix $\mathbf{R}$ sampled from the distribution $\mathcal{D}_{m\times m}$, and trapdoor $\mathbf{T_A}$ as input, this* PPT *algorithm will output a short lattice basis $\mathbf{T_B}$ of $\Lambda_q^{\perp}(\mathbf{B})$, where $\mathbf{B} = \mathbf{AR^{-1}}$.*

Lemma 4 (SampleLeft$(\mathbf{A}, \mathbf{M}, \mathbf{T_A}, \mathbf{u}, \sigma)$). *[46] After input a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and its corresponding trapdoor $\mathbf{T_A} \in \mathbb{Z}_q^{m\times m}$, a matrix $\mathbf{M} \in \mathbb{Z}_q^{n\times m_1}$, a vector $\mathbf{u} \in \mathbb{Z}_q^n$, and a parameter $\sigma \le \|\tilde{\mathbf{T_A}}\| \cdot \omega(\sqrt{\log(m+m_1)})$, 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. $[\mathbf{A}|\mathbf{M}] \cdot t = \mathbf{u} \mod q$.*

Lemma 5 (ExtBasis$(\mathbf{A}''$,$\mathbf{S})$). *[25] For an input matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, a 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 and Security Models of FS-PAEKS

This sector presents syntax and security models of FS-PAEKS. Our scheme prohibits the use of a token to search for ciphertexts generated after the time period in which the token was generated.

3.1 Syntax of FS-PAEKS Scheme

We formalize the syntax of FS-PAEKS primitive (including seven algorithms), $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}_S, \mathsf{KeyGen}_R, \mathsf{KeyUpdate}, \mathsf{FS\text{-}PAEKS}, \mathsf{Trapdoor}, \mathsf{Test})$.

- $\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda, \mathsf{d})$: Given a security parameter λ and a depth d, this algorithm returns a public parameter pp.
- $(\mathsf{pk}_S, \mathsf{sk}_S) \leftarrow \mathsf{KeyGen}_S(\mathsf{pp})$: Given a public parameter pp, this algorithm publishes a public-secret key pair for a sender $(\mathsf{pk}_S, \mathsf{sk}_S)$.
- $(\mathsf{pk}_R, \mathsf{sk}_R) \leftarrow \mathsf{KeyGen}_R(pp)$: Given a public parameter pp, this algorithm outputs a public-secret key pair for a receiver $(\mathsf{pk}_R, \mathsf{sk}_R)$.
- $\mathsf{sk}_R(t+1) \leftarrow \mathsf{KeyUpdate}(\mathsf{pp}, \mathsf{pk}_R, \mathsf{sk}_R, t, \mathsf{d})$: Given a public parameter pp, a public key of a receiver pk_R, a secret key of a sender $\mathsf{sk}_R(t)$ at time period t, and the depth of binary tree d as input, this algorithm outputs a new secret key of the sender $\mathsf{sk}_R(t+1)$ at time period $t+1$. Moreover, the former secret key of the receiver $\mathsf{sk}_R(t)$ has been deleted.

– $\mathsf{ct} \leftarrow \mathsf{FS\text{-}PAEKS}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{sk}_S, \mathsf{pk}_R, kw, t, \mathsf{d})$: Given a public parameter pp, a public key pk_S and a secret key sk_S of a sender, a public key pk_R, any keyword kw at time period t, and the depth of binary tree d, this algorithm returns a ciphertext ct of kw with time t as output.
– $\mathbf{Trap} \leftarrow \mathsf{Trapdoor}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{pk}_R, \mathsf{sk}_R(t), kw')$: Given a public parameter pp, a public key of a sender pk_S, a public key and a secret key of a receiver sk_R with time t, and a keyword kw', this algorithm outputs a trapdoor **Trap** of kw'.
– $(1 \text{ or } 0) \leftarrow \mathsf{Test}(\mathsf{pp}, \mathsf{ct}, \mathbf{Trap})$: Given a public parameter pp, a ciphertext ct and a trapdoor **Trap**, this algorithm returns 1 if the ct and **Trap** is related to a same keyword, that is, $kw = kw'$ holds; Otherwise, it returns 0.

3.2 Security Models

The security criteria are 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 [4,47]. We define ciphertext indistinguishability (CI) of forward-secure PAEKS under indistinguishability against chosen keywords attack (IND-CKA), the trapdoor privacy of forward-secure PAEKS under indistinguishability against inside keyword guessing attack (IND-IKGA), and the multi-ciphertext indistinguishability (MCI) of forward-secure PAEKS under indistinguishability against chosen multi-keywords attack (IND-Multi-CKA).

IND-CKA Game of Forward-Secure PAEKS

– **Setup**: After input a 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 $(\mathsf{pk}_S, \mathsf{sk}_S)$ and $(\mathsf{pk}_R, \mathsf{sk}_R)$. Ultimately, $\mathcal{C}$ sends $\mathsf{pp}, \mathsf{pk}_S$ 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 polynomial times.
  • **KeyUpdate Oracle** $\mathcal{O}_{KU}$: If the time period $t < T - 1$, $\mathcal{C}$ will update 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^* selected by an adversary. Given any keyword kw, $\mathcal{C}$ calls $\mathsf{FS\text{-}PAEKS}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{sk}_S, \mathsf{pk}_R, kw, t, \mathsf{d})$ 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 $\mathsf{Trapdoor}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{pk}_R, \mathsf{sk}_R(t), kw')$ algorithm to obtain the trapdoor **Trap** in time period t and transmits it to $\mathcal{A}$. When $\mathcal{A}$ accesses $\mathcal{O}_{KU}$, $\mathcal{A}$ is forbidden to issue $\mathcal{O}_T$ for the past time periods.

- **Challenge**: In time period t^*, which has not been queried 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 $\mathsf{FS\text{-}PAEKS}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{sk}_S, \mathsf{pk}_R, kw_b^*, t^*, \mathsf{d})$ 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 a 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 $Adv_{\mathcal{A}}^{IND\text{-}CKA}(\lambda) := |\Pr[b = b'] - \frac{1}{2}|$.

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 $Adv_{\mathcal{A}}^{IND\text{-}CKA}(\lambda)$ is negligible.*

IND-IKGA Game of Forward Secure PAEKS

- **Setup**: This process is the same as the **IND-CKA Game**.
- **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-CKA Game**) in some polynomial times.
- **Challenge**: In time period t^*, which has not been queried 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 $\mathsf{Trapdoor}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{pk}_R, \mathsf{sk}_R(t'), kw_b')$ algorithm to calculate the challenge trapdoor $\mathbf{Trap}^*$. Finally, $\mathcal{C}$ sends $\mathbf{Trap}^*$ 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 Forward Secure PAEKS

- **Setup**: This process is the same as the **IND-CKA Game**.
- **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-CKA Game**) 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 s.t. $kw^*_{0,i} \neq kw^*_{1,i}$. After that, $\mathcal{C}$ selects a bit $b \in \{0,1\}$ randomly and calls $\mathsf{FS\text{-}PAEKS}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{sk}_S, \mathsf{pk}_R, kw^*_b, t^*, \mathsf{d})$ algorithm to calculate the challenge ciphertext ct^*. Moreover, $\mathcal{C}$ selects $n-1$ ciphertexts from the output space of 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. Specifically, we define $\mathcal{KS}_{\mathsf{PEKS}}$ as the keyword space and a standard PEKS scheme includes four algorithms (PEKS.KeyGen, PEKS.PEKS, PEKS.Trapdoor, PEKS.Test). Moreover, we define $\mathcal{PKS}_{\mathsf{PKE}}$ and $\mathcal{PS}_{\mathsf{PKE}}$ as the public key and plaintext space, respectively. Finally, we utilize a binary tree structure and the smallest minimal cover set to realize a secret key update for a receiver and we also employ ExtBasis algorithm to fulfill one-way secret key evolution.

A labelled PKE scheme consists of three algorithms (PKE.KeyGen, PKE. Encrypt, PKE.Decrypt). A SPHF protocol incorporates four algorithms $(\mathsf{SPHF.KeyGen}_{\mathsf{Hash}}, \mathsf{SPHF.KeyGen}_{\mathsf{ProjHash}}, \mathsf{SPHF.Hash}, \mathsf{SPHF.ProjHash})$. We first define the language of ciphertext as $(Para_l, Trap_l) = (\mathsf{pk}_{\mathsf{PKE}}, \mathsf{sk}_{\mathsf{PKE}})$, where $\mathsf{pk}_{\mathsf{PKE}} \in \mathcal{PKS}_{\mathsf{PKE}}$, $\tilde{\mathcal{L}} := \{(\mathsf{label}, \mathsf{ct}_{\mathsf{PKE}}, \mathsf{m}_{\mathsf{PKE}}) | \exists \rho, \mathsf{ct}_{\mathsf{PKE}} \leftarrow \mathsf{Encrypt}(\mathsf{pk}_{\mathsf{PKE}}, \mathsf{label}, \mathsf{m}_{\mathsf{PKE}}, \rho)\}$, and $\mathcal{L} := \{(\mathsf{label}, \mathsf{ct}_{\mathsf{PKE}}, \mathsf{m}_{\mathsf{PKE}}) | \mathsf{Decrypt}(\mathsf{sk}_{\mathsf{PKE}}, \mathsf{label}, \mathsf{ct}_{\mathsf{PKE}}) = \mathsf{m}_{\mathsf{PKE}}\}$. Besides, we also define the witness relation $\tilde{\mathcal{K}}((\mathsf{label}, \mathsf{ct}_{\mathsf{PKE}}, \mathsf{m}_{\mathsf{PKE}}), \rho) = 1$ if and only if we have $\mathsf{ct}_{\mathsf{PKE}} \leftarrow \mathsf{Encrypt}(\mathsf{pk}_{\mathsf{PKE}}, \mathsf{label}, \mathsf{m}_{\mathsf{PKE}}, \rho)\}$.

- $\mathsf{Setup}(\lambda, \mathsf{d})$: Given a security parameter λ and a depth d, this algorithm processes:
  - Calculates $(\mathbf{pk}_{\mathsf{PKE}}, \mathbf{sk}_{\mathsf{PKE}}) \leftarrow \mathsf{PKE.KeyExt}(\lambda)$.
  - Selects a plaintext $\mathsf{m}_{\mathsf{PKE}} \xleftarrow{\$} \mathcal{PKS}_{\mathsf{PKE}}$ and a label $\mathsf{label} \xleftarrow{\$} \{0,1\}^*$ randomly.

- Selects two hash functions:

$$H_1 : \mathcal{PKS}_{\mathsf{PKE}} \times \mathcal{PS}_{\mathsf{PKE}} \times \{0,1\}^* \rightarrow \mathcal{PKS}_{\mathsf{PKE}};\ H_2 : \mathcal{KS}_{\mathsf{PEKS}} \times \{0,1\}^* \rightarrow \mathcal{KS}_{\mathsf{PEKS}}.$$

- Selects 2d matrices from $\mathbb{Z}_q^{n\times m}$ as $\mathsf{Matrices}$.
- Outputs $\mathsf{pp} := (\lambda, \mathsf{mpk}, \mathsf{pk}_{\mathsf{PKE}}, \mathsf{m}_{\mathsf{PKE}}, \mathsf{label}, H_1, H_2, \mathsf{Matrices})$ as a public parameter.

– $\mathsf{KeyGen}_S(\mathsf{pp})$: Given a public parameter pp, this algorithm processes these operations:
  - Calculates $\mathbf{h}_S \leftarrow \mathsf{SPHF.KeyGen}_{\mathsf{Hash}}(\mathsf{mpk})$ and $\mathbf{p}_S \leftarrow \mathsf{SPHF.KeyGen}_{\mathsf{Proj}}(\mathbf{h}_\mathsf{S}, \mathsf{mpk})$.
  - 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 a sender, respectively.

– $\mathsf{KeyGen}_R(\mathsf{pp})$: Given a public parameter pp, this algorithm processes the following operations:
  - Calculates $\mathbf{h}_R \leftarrow \mathsf{SPHF.KeyGen}_{\mathsf{Hash}}(\mathsf{mpk})$ and $\mathbf{p}_R \leftarrow \mathsf{SPHF.KeyGen}_{\mathsf{Proj}}(\mathbf{h}_\mathsf{R}, \mathsf{mpk})$.
  - 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 $(\mathsf{pk}_{\mathsf{PEKS}}, \mathsf{sk}_{\mathsf{PEKS}}) \leftarrow \mathsf{PEKS.KeyGen}(\lambda)$.
  - 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, this algorithm processes as below:
  - 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 $\mathsf{sk}_R(t) := (\mathsf{sk}_R, \mathbf{S}_{\Theta^{(i)}})$, where $\Theta^{(i)} \in \mathsf{node}(t)$.
  - Defines and outputs $\mathsf{sk}_R(t+1) := (\mathsf{sk}_R, \mathbf{S}_{\Theta^{(i)}})$, where $\Theta^{(i)} \in \mathsf{node}(t+1)$.

– $\mathsf{FS\text{-}PAEKS}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{sk}_S, \mathsf{pk}_R, kw, t, \mathsf{d})$: Given a public parameter pp, a public key pk_S and a secret key sk_S of a sender, a public key pk_R of a receiver, a keyword $kw \in \mathcal{KS}_{\mathsf{FS\text{-}PAEKS}}$ the time period t, and the depth d, this 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 $\mathsf{ct} \leftarrow \mathsf{PEKS.PEKS}(\mathsf{pk}_{\mathsf{PEKS}}, kw_S)$.

– $\mathsf{Trapdoor}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{pk}_R, \mathsf{sk}_R(t), kw')$: Given a public parameter pp, a public key pk_S of a sender, a public key pk_R and a secret key $\mathsf{sk}_R(t)$ of a receiver, a keyword $kw' \in \mathcal{KS}_{\mathsf{FS\text{-}PAEKS}}$, this 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 $\mathbf{Trap_1} \leftarrow \mathsf{PEKS.Trapdoor}(\mathsf{sk_{PEKS}}, kw'_R)$, $\mathbf{Trap_2} \leftarrow \mathsf{SamplePre}$ $(\mathbf{S}_{\Theta^{(t)}}, H_3(kw'), \sigma_3)$.
- Defines and outputs $\mathbf{Trap} := (\mathbf{Trap_1}, \mathbf{Trap_2})$.

– $\mathsf{Test}(\mathsf{pp}, \mathsf{ct}, \mathbf{Trap})$: Given a public parameter pp, a ciphertext ct, and a trapdoor $\mathbf{Trap}$, this algorithm outputs $\mathsf{PEKS.Test}(\mathsf{ct}, \mathbf{Trap})$.

5 Security Analysis

This section illustrates that the proposed FS-PAEKS construction satisfies CI under IND-CKA, TP under IND-IKGA, and MCI under IND-Multi-CKA. We specify the proofs of two theorems and give the analysis of a corollary.

Theorem 1. *The proposed* FS-PAEKS *scheme satisfies* CI *under* 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.
$\widehat{\mathbf{Game\ 0}}$: We simulate a real security game for the adversary $\mathcal{A}$ and define $Adv_{\mathcal{A}}^{\widehat{\mathbf{Game\ 0}}}(\lambda) := \epsilon$. $\mathcal{A}$ 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 $\mathsf{sk}_R(t+1) \leftarrow \mathsf{KeyUpdate}(\mathsf{pp}, \mathsf{pk}_R, \mathsf{sk}_R, t, \mathsf{d})$ and returns $\mathsf{sk}_R(t+1)$ 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 a keyword kw, $\mathcal{C}$ calculates $\mathsf{ct} \leftarrow \mathsf{FS\text{-}PAEKS}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{sk}_S, \mathsf{pk}_R, kw, t, \mathsf{d})$ and returns ct to $\mathcal{A}$.
– $\mathcal{O}_{\mathcal{T}}$: Given a keyword kw, $\mathcal{C}$ calculates $\mathbf{Trap} \leftarrow \mathsf{Trapdoor}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{pk}_R, \mathsf{sk}_R(t), kw')$ and returns $\mathbf{Trap}$ to $\mathcal{A}$.

$\widehat{\mathbf{Game\ 1}}$: This game is identical to $\widehat{\mathbf{Game\ 0}}$, except changing the calculation method of ct^* in the **Challenge** query. To be more specific, $\mathcal{C}$ selects $\mathsf{Hash}_S \xleftarrow{\$} \mathcal{OS}_{\mathsf{Hash}_S}$ randomly ($\mathcal{OS}_{\mathsf{Hash}_S}$ is the output space of Hash_S) instead of calculating $\mathsf{Hash}_S \leftarrow \mathsf{SPHF.Hash}(\mathsf{h}_S, \mathsf{mpk}, (\mathsf{ct}_{\mathsf{PKE},R}, \mathsf{m_{PKE}}))$. For the view of $\mathcal{A}$, $\widehat{\mathbf{Game\ 1}}$ and $\widehat{\mathbf{Game\ 0}}$ are statistically indistinguishable due to the fact that the output of Hash_S satisfies pseudo-randomness. Hence, we acquire: $|Adv_{\mathcal{A}}^{\widehat{\mathbf{Game\ 1}}}(\lambda) - Adv_{\mathcal{A}}^{\widehat{\mathbf{Game\ 0}}}(\lambda)| \leq \mathbf{negl}(\lambda)$.
$\widehat{\mathbf{Game\ 2}}$: This game is identical to $\widehat{\mathbf{Game\ 1}}$, 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 $kw_S \xleftarrow{\$} \mathcal{KS}_{\mathsf{PEKS}}$ randomly ($\mathcal{KS}_{\mathsf{PEKS}}$ is the keyword space of $\mathsf{PEKS}(\mathsf{pk_{PEKS}}, kw)$ algorithm), instead of calculating $kw_S \leftarrow H_2(kw_b, \mathsf{Hash}_S \oplus \mathsf{ProjHash}_S)$. In this way, the output of $H_2(kw_b, \mathsf{Hash}_S \oplus \mathsf{ProjHash}_S)$ is random since Hash_S is randomly selected 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: $|Adv_{\mathcal{A}}^{\widehat{\textbf{Game 2}}}(\lambda) - Adv_{\mathcal{A}}^{\widehat{\textbf{Game 1}}}(\lambda)| \leq \textbf{negl}(\lambda)$.

$\widehat{\textbf{Game 3}}$: Till now, the keyword is generated by $kw_S \xleftarrow{\$} \mathcal{KS}_{\mathsf{PEKS}}$ at random, the challenge ciphertext $\mathsf{ct}^* = \mathsf{ct}_{\mathsf{PEKS},kw}$ is obtained from $\mathsf{PEKS.PEKS}(\mathsf{pk}_{\mathsf{PEKS}}, kw_S)$ and $kw_S \xleftarrow{\$} \mathcal{KS}_{\mathsf{PEKS}}$. 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: $|Adv_{\mathcal{A}}^{\widehat{\textbf{Game 3}}}(\lambda)| = 0$.

Theorem 2. *The proposed* FS-PAEKS *scheme satisfies* TP *under* 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 $\mathcal{A}$.

$\widehat{\textbf{Game 1}}$: 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 $\mathsf{Hash}_R \xleftarrow{\$} \mathcal{OS}_{\mathsf{Hash}_R}$ randomly ($\mathcal{OS}_{\mathsf{Hash}_R}$ is the output space of Hash_R) instead of calculating $\mathsf{Hash}_R \leftarrow \mathsf{SPHF.Hash}(\mathsf{h}_R, \mathsf{mpk}, (\mathsf{ct}_{\mathsf{PKE},S}, \mathsf{m}_{\mathsf{PKE}}))$. For $\mathcal{A}$, $\widehat{\textbf{Game 1}}$ and $\widehat{\textbf{Game 0}}$ are statistically indistinguishable due to the fact that the output of Hash_R satisfies pseudo-randomness. Hence, we acquire: $|Adv_{\mathcal{A}}^{\widehat{\textbf{Game 1}}}(\lambda) - Adv_{\mathcal{A}}^{\widehat{\textbf{Game 0}}}(\lambda)| \leq \textbf{negl}(\lambda)$.

$\widehat{\textbf{Game 2}}$: 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 $b \in \{0,1\}$ and samples $kw'_R \xleftarrow{\$} \mathcal{KS}_{\mathsf{PEKS}}$ randomly, instead of calculating $kw'_R \leftarrow H_2(kw'_b, \mathsf{Hash}_R \oplus \mathsf{ProjHash}_R)$. In this way, the output of $H_2(kw'_b, \mathsf{Hash}_R \oplus \mathsf{ProjHash}_R)$ is random since Hash_R is randomly selected and H_2 is 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: $|Adv_{\mathcal{A}}^{\widehat{\textbf{Game 2}}}(\lambda) - Adv_{\mathcal{A}}^{\widehat{\textbf{Game 1}}}(\lambda)| \leq \textbf{negl}(\lambda)$.

$\widehat{\textbf{Game 3}}$: Till now, the keyword is generated by $kw'_R \xleftarrow{\$} \mathcal{KS}_{\mathsf{PEKS}}$ at random, the challenge trapdoor $\textbf{Trap}^* = (\textbf{Trap}_\textbf{1}{}^*, \textbf{Trap}_\textbf{2}{}^*)$ is generated from $\mathsf{Trapdoor}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{pk}_R, \mathsf{sk}_R(t), kw')$. 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: $|Adv_{\mathcal{A}}^{\widehat{\textbf{Game 3}}}(\lambda)| = 0$.

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 involves PEKS.PEKS algorithm. To the best of our knowledge, the existing PEKS.PEKS algorithm satisfies probabilistic [1,24]. 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.

6 Lattice-Based Instantiation of FS-PAEKS

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

- $\mathsf{Setup}(\lambda, d)$: Given a security parameter λ, the depth d of a 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 are preimage sample parameters, α is a gaussian distribution parameter and $T = 2^d$ is 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 an element $u \xleftarrow{\$} \mathcal{U}$ randomly as the label of PKE.
  - Selects three Hash functions

$$H_1 : \mathbb{Z}^{n\times m} \times \{0,1\}^\kappa \times \mathcal{U} \to \mathbb{Z}_q^{n\times m}; H_2 : \{1,-1\}^\ell \times \{0,1\}^\kappa \to \{1,-1\}^\ell; H_3 : \{1,-1\}^\ell \to \mathbb{Z}_q^n.$$

  - Selects an Injective function $H_4 : \mathcal{R} \to \mathbb{Z}_q^{n\times n}$.
  - Calculates the master public key of PKE: $\mathbf{A} \leftarrow H_1(\mathbf{T_{A_0}}, \mathbf{m}, u) \in \mathbb{Z}_q^{n\times m}$.
  - 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 $\mathbf{G} := \mathbf{I}_n \otimes \mathbf{g}^\top$, $\mathbf{g}^\top = [1, 2, \cdots, 2^k], k = \lceil \mathsf{log} q \rceil - 1$.
  - Defines and calculates $\mathbf{A}_{\mathsf{label}} = \mathbf{A} + \begin{bmatrix} 0 \\ \mathbf{G}H_4(u) \end{bmatrix} = \mathbf{A} + \begin{bmatrix} 0 \\ (\mathbf{I}_n \otimes \mathbf{g}^\top)H_4(u) \end{bmatrix}$.
  - Selects a matrix $\mathbf{h}_S \xleftarrow{\$} D_{\mathbb{Z},s}^m$ at random, and calculates the matrix $\mathbf{p}_S = \mathbf{A}_{\mathsf{label}} \cdot \mathbf{h}_S \in \mathbb{Z}_q^n$.
  - For $i = 1, 2, \cdots, \kappa$, selects vectors $\mathbf{s}_i \xleftarrow{\$} \mathbb{Z}_q$ and vectors $\mathbf{e}_{S,i} \xleftarrow{\$} D_{\mathbb{Z},t}^m$ randomly s.t. $\|\mathbf{e}_{S,i}\| \le 2t\sqrt{m}$ and then calculates $\mathbf{c}_{S,i} = \mathbf{A}_{\mathsf{label}}^\top \cdot \mathbf{s}_i + \mathbf{e}_{S,i} + m_i[0, 0, \cdots, 0, \lceil \frac{q}{2} \rceil]^\top \bmod q$.

  - 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 a 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 a 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 $\mathbf{G} := \mathbf{I}_n \otimes \mathbf{g}^\top$, $\mathbf{g}^\top = [1, 2, \cdots, 2^k], k = \lceil \mathsf{log} q \rceil - 1$.
  - Defines and calculates $\mathbf{A}_{\mathsf{label}} = \mathbf{A} + \begin{bmatrix} 0 \\ \mathbf{G}H_4(u) \end{bmatrix} = \mathbf{A} + \begin{bmatrix} 0 \\ (\mathbf{I}_n \otimes \mathbf{g}^\top)H_4(u) \end{bmatrix}$.
  - Selects a matrix $\mathbf{h}_R \xleftarrow{\$} D_{\mathbb{Z},s}^m$ at random, and calculates the matrix $\mathbf{p}_R = \mathbf{A}_{\mathsf{label}} \cdot \mathbf{h}_R \in \mathbb{Z}_q^n$.
  - For $i = 1, 2, \cdots, \kappa$, selects vectors $\mathbf{r}_i \xleftarrow{\$} \mathbb{Z}_q$ and vectors $\mathbf{e}_{R,i} \xleftarrow{\$} D_{\mathbb{Z},t}^m$ randomly s.t. $\|\mathbf{e}_{R,i}\| \le 2t\sqrt{m}$ and then calculates $\mathbf{c}_{R,i} = \mathbf{A}_{\mathsf{label}}^\top \cdot \mathbf{r}_i + \mathbf{e}_{R,i} + m_i[0, 0, \cdots, 0, \lceil \frac{q}{2} \rceil]^\top \bmod q$.
  - 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.

- $\mathsf{KeyUpdate}(pp, pk_R, sk_R, t, d)$: Given a public parameter pp, time t, initial public key pk_R, and initial secret key sk_R, this algorithm processes 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 an 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 $sk_R(1011) = (\mathbf{h}_R, \{\mathbf{r}_{R,1}\}, \{\mathbf{r}_{R,2}\}, \cdots, \{\mathbf{r}_{R,\kappa}\}, \mathbf{S}_{1011}, \mathbf{S}_{11})$.

– FS-PAEKS$(pp, pk_S, sk_S, pk_R, kw, t, d)$: Given a public parameter pp, the sender's public key and secret key pk_S, sk_S, the receiver's public key pk_R, a 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
$$h_{S,i} \leftarrow \lfloor \frac{2(\mathbf{c}_{R,i}^\top \cdot \mathbf{h}_S (\text{mod} q))}{q} \rceil, p_{S,i} \leftarrow \lfloor \frac{2(\mathbf{s}_i^\top \cdot \mathbf{p}_R (\text{mod} q))}{q} \rceil.$$
  - Defines $y_{S,i} = h_{S,i} \cdot p_{S,i}$, and $\mathbf{y}_S = y_{S,1} y_{S,2} \cdots y_{S,\kappa} \in \{0,1\}^\kappa$.
  - Defines and calculates $\mathbf{dk}_S = dk_{S,1} dk_{S,2} \cdots dk_{S,\ell} \leftarrow H_2(kw, \mathbf{y}_S) \in \{1, -1\}^\ell$.
  - Defines and calculates $\mathbf{M}_{dk} = \mathbf{C}_R + \sum_{i=1}^{\ell} dk_{S,i} \mathbf{M}_{R,i}$.
  - Calculates $\mathbf{F}_{dk} = [\mathbf{M}_R \parallel \mathbf{M}_{dk}] = [\mathbf{M}_R \parallel \mathbf{C}_R + \sum_{i=1}^{\ell} dk_{S,i} \mathbf{M}_{R,i}]$.
  - 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 $\bar{\mathbf{R}}_j = \sum_{i=1}^{\ell} dk_{S,i} \mathbf{R}_{i_j} \in \{-\ell, -\ell + 1, \cdots, \ell\}^{\frac{(d+3)m}{2} \times \frac{(d+3)m}{2}}$;
    * 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 $\mathbf{z}_j \leftarrow \bar{\mathbf{R}}_j^\top \mathbf{y}_j \in \mathbb{Z}_q^{\frac{(d+3)m}{2}}$, and $c_{0_j} = (\mathbf{r}_R^\top + H_3(kw)^\top)\mathbf{s}_j + x_j + b_j \lfloor \frac{q}{2} \rfloor \in \mathbb{Z}_q$.
    * Calculates $\mathbf{c}_{\mathbf{1}_j} = (\mathbf{F}_{dk} \parallel \mathbf{F}_t)^\top \mathbf{s}_j + \begin{bmatrix} \mathbf{y}_j \\ \mathbf{z}_j \end{bmatrix} \in \mathbb{Z}_q^{(d+3)m}$.
  - Outputs a forward-secure searchable ciphertext $\mathsf{ct} := (\{c_{0_j}, \mathbf{c}_{\mathbf{1}_j}, b_j\}_{j=1}^{\rho})$.

– Trapdoor$(pp, pk_S, pk_R, sk_R(t), kw')$: After input a 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 a keyword $kw' \in \{1, -1\}^\ell$, this algorithm will process the following steps.
  - For $i = 1, 2, \cdots, \kappa$, calculates
$$h_{R,i} \leftarrow \lfloor \frac{2(\mathbf{c}_{S,i}^\top \cdot \mathbf{h}_R (\text{mod} q))}{q} \rceil, p_{R,i} \leftarrow \lfloor \frac{2(\mathbf{s}_{R,i}^\top \cdot \mathbf{p}_S (\text{mod} q))}{q} \rceil.$$
  - Defines $y_{R,i} = h_{R,i} \cdot p_{R,i}$, and $\mathbf{y}_R = y_{R,1} y_{R,2} \cdots y_{R,\kappa} \in \{0,1\}^\kappa$.
  - Defines and calculates $\mathbf{dk}_R = dk_{R,1} dk_{R,2} \cdots dk_{R,\ell} \leftarrow H_2(kw', \mathbf{y}_R)$.
  - Defines and calculates $\mathbf{M}_{dk} = \mathbf{C}_R + \sum_{i=1}^{\ell} dk_{R,i} \mathbf{M}_{R,i}$.
  - Invokes 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 ExtBasis$(F_{\Theta^{(t)}}, \mathbf{S}_{\Theta^{(i)}})$ to generate it and then continue the remainder procedures.

- Invokes $\mathsf{SamplePre}(\mathbf{S}_{\Theta^{(t)}}, H_3(kw'), \sigma_3)$ 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 $v_j = c_{0_j} - \begin{pmatrix} \mathbf{Trap_1} \\ \mathbf{Trap_2} \end{pmatrix}^\top \mathbf{c}_{1_j}$.
- Checks whether it satisfies $\lfloor v_j - \lfloor \frac{q}{2} \rfloor \rfloor$: If it 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.

7 Parameters and Correctness

7.1 Parameters Setting

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 let $\mathsf{SampleLeft}(\mathbf{A}, \mathbf{M}, \mathbf{T_A}, \mathbf{u}, \sigma)$ algorithm execute properly.
5. $m \geq 2n\lceil \log q \rceil$, $\sigma_3 \geq \| \tilde{\mathbf{B}} \| \cdot \omega(\sqrt{\log n})$ to let $\mathsf{SamplePre}(\mathbf{A}, \mathbf{T_A}, \mathbf{u}, \sigma)$ algorithm operate properly.
6. $\frac{(d+3)m}{2}$ is an integer to make $\mathsf{FS\text{-}PAEKS}(pp, pk_S, sk_S, pk_R, kw, t, d)$ 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. $\alpha < [\sigma_2 \ell m \omega(\sqrt{\log n})]^{-1}$, $q = \Omega(\sigma_2 m^{\frac{3}{2}})$ to make second error term is bounded legitimately.

7.2 Correctness

Theorem 3. *We initially consider the condition mentioned by Lemma 6.1 in reference [48] and $\epsilon = \mathsf{negl}(n)$ is negligible. That is, if the keywords hold $kw = kw'$ 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 $\frac{\epsilon \cdot q}{8}$ with overwhelming probability, then we obtain the equality $\mathbf{dk}_S = \mathbf{dk}_R$.*

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

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

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

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 4. *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 1, and* b_j *is correct.*

Proof.

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

Therefore, as mentioned in Lemma 22 of reference [46], for $j = 1, 2, \cdots, \rho$, if the given keywords are absolutely identical, we can conclude that $v_j = b_j$.

8 Theoretical Comparison

We cryptanalyze and compare eight PEKS and PAEKS schemes with regards to six security properties in Table 1. Then, we compare the computational complexity and communication overhead with several post-quantum PEKS and PAEKS primitives in Tables 2 and 3.

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. [5]	×	×	×	×	×	✓
Behnia et al. [24]	×	✓	✓	×	✓	✓
Zhang et al. [49]	×	✓	✓	×	✓	×
Zhang et al. [19]	✓	✓	✓	×	✓	✓
Liu et al. [12]	×	✓	✓	✓	✓	✓
Emura [50]	×	✓	✓	✓	✓	✓
Cheng et al. [35]	×	✓	✓	✓	✓	✓
Our scheme	✓	✓	✓	✓	✓	✓

Notes. **PQ**: Post-quantum. **WTA**: Without trusted authority.

As for Table 2, the abbreviations are multiplication (T_{Mul}), hash function (T_{HF}), SampleLeft (T_{SL}), SamplePre(T_{SP}), and BasisDel(T_{BD}) algorithms. With regard to Table 3, we analyze the communication overhead in terms of ciphertext size and trapdoor size. d is the depth of a binary tree, ℓ is the length of a keyword kw, ρ, κ are related to the security parameter.

9 Potential Applications of FS-PAEKS

(1) Combining with Electronic Medical Records (EMRs). Numerous scholars have utilized PEKS primitive for doctors (data receiver) to search EMRs and protect the privacy of patients (data sender) [20,51,52]. However, a malicious attacker may recover the keyword kw from the previous search trapdoor **Trap** through keyword guessing attacks. Besides, if secret keys have been compromised, sensitive medical data may be disclosed. Compared with the existing schemes, our FS-PAEKS protocol completely avoids those problems and provides better security.

(2) Combining with Industrial Internet of Things (IIoTs). The PAKES protocol has been employed to safeguard the privacy of IIoTs while simultaneously achieving CI and TP security [37]. However, they failed to account for the potential risks of quantum computing attacks and the likelihood of secret key leakage during communication. Our FS-PAEKS primitive offers enhanced security features such as quantum resistance and elimination of secret key leakage. Besides, we realize MCI security, which addressed a previously unresolved issue of their work.

Table 2. Computational complexity comparison

Schemes	Ciphertext Generation	Trapdoor Generation	Test Generation
Behnia et al. [24]	$\rho(m^2+2nm+n+\ell+1)T_{Mul}$	$\ell T_{Mul}+T_{SL}$	$2\rho m T_{Mul}$
Zhang et al. [19]	$T_{HF}+(\rho n+nm^2+\rho)T_{Mul}+T_{SP}$	$T_{HF}+nm^2 TMul$ $+T_{BD}+T_{SP}$	$T_{HF}+(\ell m+nm)T_M$
Liu et al. [12]	$T_{HF}+(\kappa(m+n+1)$ $+\rho(m^2+2nm+n+\ell+1))T_{Mul}$	$T_{HF}+(\kappa(m+n+1)$ $+\ell)T_{Mul}+T_{SL}$	$2\rho m T_M$
Our scheme	$(\rho+1)T_{HF}+(\kappa(m+n+1)$ $+\rho(\frac{(d+3)^2m^2}{4}+(d+3)nm$ $+2n+\ell+1))T_{Mul}$	$2T_{HF}+(\kappa(m+n+1)$ $+\ell)T_{Mul}+T_{SL}+T_{SP}$	$(d+3)\rho m T_M$

Table 3. Communication overhead comparison

Schemes	Ciphertext Size	Trapdoor Size
Behnia et al. [24]	$\kappa(\lvert q\rvert+2m\lvert q\rvert+1)$	$2m\lvert q\rvert$
Zhang et al. [19]	$(\ell+m\ell+m)\lvert q\rvert$	$m\lvert q\rvert$
Liu et al. [12]	$\rho(\lvert q\rvert+2m\lvert q\rvert+1)$	$2m\lvert q\rvert$
Our scheme	$\rho(\lvert q\rvert+(d+3)m\lvert q\rvert+1)$	$(d+3)m\lvert q\rvert$

10 Conclusion

In this paper, we generalize the first post-quantum public-key authenticated searchable encryption with forward security primitive, namely FS-PAEKS. The proposed scheme addresses the challenge of secret key exposure while enjoying quantum-safe security without trusted authorities. Technically speaking, we introduce 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 analyze it satisfies IND-CKA, IND-IKGA, and IND-Multi-CKA in a quantum setting. Besides, we also elaborate on the theoretical comparisons. Ultimately, we show two applications for FS-PAEKS to illustrate its feasibility. We hereby address an open problem of how to construct a post-quantum FS-PAEKS scheme without random oracle models.

Acknowledgements. Xue Chen interned as a Summer Research Assistant at HKU. This work is partially supported by HKU-SCF FinTech Academy, Shenzhen-Hong Kong-Macao Science and Technology Plan Project (Category C Project: SGDX20210823103537030), and Theme-based Research Scheme of RGC, Hong Kong (T35-710/20-R).

References

1. Boneh, D., Di Crescenzo, G., Ostrovsky, R., Persiano, G.: Public key encryption with keyword search. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 506–522. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24676-3_30
2. Gang, X., et al.: A searchable encryption scheme based on lattice for log systems in blockchain. Comput. Mater. Continua **72**(3), 5429–5441 (2022)
3. Gang, X., et al.: PPSEB: a postquantum public-key searchable encryption scheme on blockchain for e-healthcare scenarios. Secur. Commun. Netw. **2022**, 13 (2022)
4. Byun, J.W., Rhee, H.S., Park, H.-A., Lee, D.H.: Off-line keyword guessing attacks on recent keyword search schemes over encrypted data. In: Jonker, W., Petković, M. (eds.) SDM 2006. LNCS, vol. 4165, pp. 75–83. Springer, Heidelberg (2006). https://doi.org/10.1007/11844662_6
5. Huang, Q., Li, H.: An efficient public-key searchable encryption scheme secure against inside keyword guessing attacks. Inf. Sci. **403**, 1–14 (2017)
6. Baodong Qin, Yu., Chen, Q.H., Liu, X., Zheng, D.: Public-key authenticated encryption with keyword search revisited: security model and constructions. Inf. Sci. **516**, 515–528 (2020)
7. Noroozi, M., Eslami, Z.: Public key authenticated encryption with keyword search: revisited. IET Inf. Secur. **13**(4), 336–342 (2019)
8. Qin, B., Cui, H., Zheng, X., Zheng, D.: Improved security model for public-key authenticated encryption with keyword search. In: Huang, Q., Yu, Yu. (eds.) ProvSec 2021. LNCS, vol. 13059, pp. 19–38. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-90402-9_2
9. Yang, L., Li, J.: Lightweight public key authenticated encryption with keyword search against adaptively-chosen-targets adversaries for mobile devices. IEEE Trans. Mob. Comput. **21**(12), 4397–4409 (2021)

10. Pan, X., Li, F.: Public-key authenticated encryption with keyword search achieving both multi-ciphertext and multi-trapdoor indistinguishability. J. Syst. Architect. **115**, 102075 (2021)
11. Huang, Q., Huang, P., Li, H., Huang, J., Lin, H.: A more efficient public-key authenticated encryption scheme with keyword search. J. Syst. Architect. **137**, 102839 (2023)
12. Liu, Z.-Y., Tseng, Y.-F., Tso, R., Mambo, M., Chen, Y.-C.: 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, pp. 423–436 (2022)
13. Bellare, M., Miner, S.K.: A forward-secure digital signature scheme. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 431–448. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_28
14. Cao, Y., Shiyuan, X., Chen, X., He, Y., Jiang, S.: A forward-secure and efficient authentication protocol through lattice-based group signature in vanets scenarios. Comput. Netw. **214**, 109149 (2022)
15. Chen, X., Xu, S., He, Y., Cui, Y., He, J., Gao, S.: LFS-AS: lightweight forward secure aggregate signature for e-health scenarios. In: ICC 2022-IEEE International Conference on Communications, pp. 1239–1244. IEEE (2022)
16. Canetti, R., Halevi, S., Katz, J.: A forward-secure public-key encryption scheme. J. Cryptol. **20**, 265–294 (2007)
17. Chen, X., Shiyuan, X., Cao, Y., He, Y., Xiao, K.: AQRS: anti-quantum ring signature scheme for secure epidemic control with blockchain. Comput. Netw. **224**, 109595 (2023)
18. Xu, S., Chen, X., Kong, W., Cao, Y., He, Y., Xiao, K.: An efficient blockchain-based privacy-preserving authentication scheme in VANET. In: 2023 IEEE 97th Vehicular Technology Conference (VTC2023-Spring), pp. 1–6. IEEE (2023)
19. Zhang, X., Chunxiang, X., Wang, H., Zhang, Y., Wang, S.: FS-PEKS: lattice-based forward secure public-key encryption with keyword search for cloud-assisted industrial Internet of Things. IEEE Trans. Dependable Secure Comput. **18**(3), 1019–1032 (2021)
20. Zhe, J., Kai, Z., Liangliang, W., Jianting, N.: Forward secure public-key authenticated encryption with conjunctive keyword search. Comput. J. **66**, 2265–2278 (2022)
21. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Rev. **41**(2), 303–332 (1999)
22. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th Annual Symposium on Foundations of Computer Science, pp. 124–134. IEEE (1994)
23. Liu, Z.-Y., Tseng, Y.-F., Tso, R., Mambo, M., Chen, Y.-C.: Public-key authenticated encryption with keyword search: a generic construction and its quantum-resistant instantiation. Comput. J. **65**(10), 2828–2844 (2022)
24. Behnia, R., Ozmen, M.O., Yavuz, A.A.: Lattice-based public key searchable encryption from experimental perspectives. IEEE Trans. Depend. Secure Comput. **17**(6), 1269–1282 (2020)
25. Cash, D., Hofheinz, D., Kiltz, E., Peikert, C.: Bonsai trees, or how to delegate a lattice basis. J. Cryptol. **25**, 601–639 (2012)
26. Cramer, R., Shoup, V.: Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 45–64. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46035-7_4

27. Katz, J., Vaikuntanathan, V.: Smooth projective hashing and password-based authenticated key exchange from lattices. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 636–652. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10366-7_37
28. Canetti, R., Dachman-Soled, D., Vaikuntanathan, V., Wee, H.: Efficient password authenticated key exchange via oblivious transfer. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 449–466. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30057-8_27
29. Abdalla, M., Benhamouda, F., MacKenzie, P.: Security of the j-pake password-authenticated key exchange protocol. In: 2015 IEEE Symposium on Security and Privacy, pp. 571–587. IEEE (2015)
30. Jarecki, S., Krawczyk, H., Xu, J.: OPAQUE: an asymmetric PAKE protocol secure against pre-computation attacks. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10822, pp. 456–486. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78372-7_15
31. Erwig, A., Hesse, J., Orlt, M., Riahi, S.: Fuzzy asymmetric password-authenticated key exchange. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020. LNCS, vol. 12492, pp. 761–784. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64834-3_26
32. Abdalla, M., Eisenhofer, T., Kiltz, E., Kunzweiler, S., Riepel, D.: Password-authenticated key exchange from group actions. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology-CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, 15–18 August 2022, Proceedings, Part II, pp. 699–728. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-15979-4_24
33. Katz, J., Vaikuntanathan, V.: Round-optimal password-based authenticated key exchange. J. Cryptol. **26**, 714–743 (2013)
34. Zhang, R., Imai, H.: Generic combination of public key encryption with keyword search and public key encryption. In: Bao, F., Ling, S., Okamoto, T., Wang, H., Xing, C. (eds.) CANS 2007. LNCS, vol. 4856, pp. 159–174. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-76969-9_11
35. Cheng, L., Meng, F.: Public key authenticated encryption with keyword search from LWE. In: Atluri, V., Di Pietro, R., Jensen, C.D., Meng, W. (eds.) Computer Security – ESORICS 2022. ESORICS 2022. Lecture Notes in Computer Science, vol. 13554, pp. 303–324. Springer, 2022. https://doi.org/10.1007/978-3-031-17140-6_15
36. Yao, L., et al.: Scalable CCA-secure public-key authenticated encryption with keyword search from ideal lattices in cloud computing. Inf. Sci. **624**, 777–795 (2023)
37. Pu, L., Lin, C., Chen, B., He, D.: User-friendly public-key authenticated encryption with keyword search for industrial internet of things. IEEE Internet of Things J. **10**, 13544–13555 (2023)
38. Zeng, M., Qian, H., Chen, J., Zhang, K.: Forward secure public key encryption with keyword search for outsourced cloud storage. IEEE Trans. Cloud Comput. **10**(1), 426–438 (2019)
39. Yang, X., Chen, X., Huang, J., Li, H., Huang, Q.: FS-IBEKS: forward secure identity-based encryption with keyword search from lattice. Comput. Stand. Interfaces **86**, 103732 (2023)
40. Abdalla, M., Benhamouda, F., Pointcheval, D.: Public-key encryption indistinguishable under plaintext-checkable attacks. IET Inf. Secur. **10**(6), 288–303 (2016)

41. Ajtai, M.: Generating hard instances of lattice problems. In: Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, pp. 99–108 (1996)
42. Peikert, C.: An efficient and parallel gaussian sampler for lattices. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 80–97. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_5
43. Agrawal, S., Boneh, D., Boyen, X.: Lattice basis delegation in fixed dimension and shorter-ciphertext hierarchical IBE. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 98–115. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_6
44. Micciancio, D., Peikert, C.: Trapdoors for lattices: simpler, tighter, faster, smaller. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 700–718. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_41
45. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing, pp. 197–206 (2008)
46. Agrawal, S., Boneh, D., Boyen, X.: Efficient lattice (H)IBE in the standard model. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 553–572. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_28
47. Rhee, H.S., Park, J.H., Susilo, W., Lee, D.H.: Trapdoor security in a searchable public-key encryption scheme with a designated tester. J. Syst. Software **83**(5), 763–771 (2010)
48. Li, Z., Wang, D.: Achieving one-round password-based authenticated key exchange over lattices. IEEE Trans. Serv. Comput. **15**(1), 308–321 (2019)
49. Zhang, X., Tang, Y., Wang, H., Chunxiang, X., Miao, Y., Cheng, H.: Lattice-based proxy-oriented identity-based encryption with keyword search for cloud storage. Inf. Sci. **494**, 193–207 (2019)
50. Emura, K.: 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, pp. 39–49 (2022)
51. Xu, G., et al.: AAQ-PEKS: an attribute-based anti-quantum public-key encryption scheme with keyword search for e-healthcare scenarios. Cryptology ePrint Archive (2023)
52. Li, H., Huang, Q., Huang, J., Susilo, W.: Public-key authenticated encryption with keyword search supporting constant trapdoor generation and fast search. IEEE Trans. Inf. Forensics Secur. **18**, 396–410 (2022)