StopGuess: A framework for public-key authenticated encryption with keyword search

Tao Xiang [a], Zhongming Wang [a], Biwen Chen [a,b,*], Xiaoguo Li [c], Peng Wang [d], Fei Chen [e]

a College of Computer Science, Chongqing University, Chongqing, 400044, China
b State Key Laboratory of Cryptology, P.O. Box 5159, Beijing, 100878, China
c School of Computing and Information Systems, Singapore Management University, Singapore, 188065, Singapore
d School of Intelligence Technology and Engineering, Chongqing University of Science and Technology, Chongqing, 400044, China
e College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, 518000, China

Published in Computer Standards & Interfaces (2024) 88, 103805. DOI: 10.1016/j.csi.2023.103805

Abstract: Public key encryption with keyword search (PEKS) allows users to search on encrypted data without leaking the keyword information from the ciphertexts. But it does not preserve keyword privacy within the trapdoors, because an adversary (e.g., untrusted server) might launch inside keyword-guessing attacks (IKGA) to guess keywords from the trapdoors. In recent years, public key authenticated encryption with keyword search (PAEKS) has become a promising primitive to counter the IKGA. However, existing PAEKS schemes focus on the concrete construction of PAEKS, making them unable to support modular construction, intuitive proof, or flexible extension. In this paper, our proposal called "StopGuess" is the first elegant framework to achieve the above-mentioned features. StopGuess provides a general solution to eliminate IKGA, and we can construct a bundle of PAEKS schemes from different cryptographic assumptions under the framework. To show its feasibility, we present two generic constructions of PAEKS and their (pairing-based and lattice-based) instantiations in a significantly simpler and more modular manner. Besides, without additional costs, we extend PAEKS to achieve anonymity which preserves the identity of users; we integrate it with symmetric encryption to support data retrieval functionality which makes it practical in resource-constrained applications.

Keywords: Searchable encryption, Keyword guessing attack, Anonymity, Trapdoor privacy, Keyword search

1. Introduction

Public key encryption with keyword search (PEKS) [1] is a cryptographic primitive that allows a user to delegate the search on encrypted data to a third party without data leakage. Unlike searchable symmetric encryption [2], PEKS allows multiple senders to generate ciphertext for a receiver with one public key, which avoids complicated key management in traditional symmetric encryption. PEKS involves three parties: sender, receiver, and server. Briefly, the sender and the receiver can generate a ciphertext and a trapdoor corresponding to their respective keywords, respectively. The server can determine whether the ciphertext and the trapdoor were derived from the same keyword.

Unfortunately, traditional PEKS [1] enforces ciphertext indistinguishability but fails to achieve trapdoor indistinguishability. Thus, it requires a secure channel for receivers to transmit trapdoors to the server. Otherwise, an adversary might launch the keyword-guessing attacks (KGA) [3] to guess the keyword contained in the trapdoors. The KGA is feasible because the keyword space is limited by the word library such that the adversary can guess the keyword using a brute-force search. In detail, the KGA could happen in two cases, i.e., outside KGA OKGA) from an external adversary (e.g., eavesdropper) and inside KGA (IKGA) from an internal adversary (e.g., untrusted server). Compared to an external adversary, an internal adversary always has stronger abilities. For example, the internal adversary can directly access the trapdoors, but the external adversary cannot. So the IKGA actually contains the OKGA.

To resolve the OKGA problem, current solutions can be sorted into two categories according to the methodology: First, enforcing trapdoor indistinguishability in PEKS, e.g., building PEKS from function-private anonymous identity-based encryption (IBE) [4], and then the function-private property ensures the indistinguishability of the trapdoor. Second, authenticating the server in the test phase, as in the designated-tester PEKS (dPEKS) scheme [5] that embeds server's public key in the trapdoor. However, since PEKS allows anyone to test whether the ciphertext and trapdoor contain the same keywords, an internal adversary (e.g., the server) could guess the keyword contained in the trapdoor with the help of the Test algorithm. Thus, while the above approaches prevent information leakage about the keyword from the trapdoor, they are still vulnerable to the IKGA.

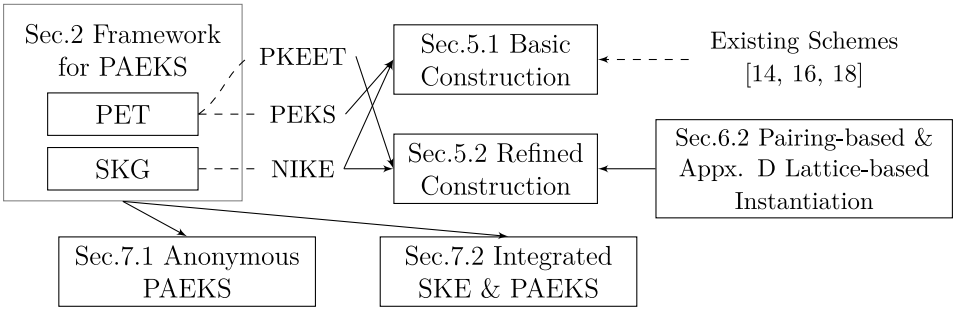

Fig. 1. Illustration of contributions. Our contributions are denoted as bold lines and rectangles.

Table 1
Comparisons of PEKS schemes.

Schemes	OKGA	IKGA	Extra Party	Extra Comm.
BDOP-PEKS [1]	✗	✗	✗	✗
dPEKS [5]	✓	✗	✗	✗
Dual-server PEKS [7]	✓	✓	✓	✗
Server-aided PEKS [6]	✓	✓	✗	✓
PAEKS [11]	✓	✓	✗	✗

OKGA: Secure against outside KGA; IKGA: Secure against inside KGA; Extra Party: Require additional party; Extra Comm.: Require additional communication.

Subsequently, there have been many efforts to achieve IKGA-resistant PEKS [6–10]. The insight in these schemes is to limit the power of a single server by distributing the search ability to multiple parties. However, these schemes either require additional parties [7,8] or additional communication [6], making them rather complicated. Therefore, eliminating IKGA in the context of PEKS without additional parties or communications became a new open problem. To address this problem, public key authenticated encryption with keyword search (PAEKS) is proposed by Huang et al. [11]. We compare PAEKS with prior works in Table 1. In PAEKS, every ciphertext embeds a sender's secret key in the ciphertext. Thus, without the sender's secret key, the inside adversary fails to guess keywords from the trapdoor through the Test algorithm.

Since PAEKS was proposed, much work has been followed, from improvements in security [12–14] to instantiations based on different assumptions [11,15–19]. Despite achieving resistance to IKGA, these schemes are still working on the construction and instantiation of PAEKS, thus lacking a general description and intuitive understanding of its security. Therefore, in this paper, we focus on the following question:

Can we offer a general framework for PAEKS to provide an intuitive understanding of its construction, security, and extensions while eliminating IKGA?

1.1. Our contributions

We present an affirmative answer to this question. Our contributions (also sketched in Fig. 1) are summarized as follows:

- We first present StopGuess for PAEKS to give a general solution to eliminate IKGA. StopGuess is an abstraction of existing schemes and just consists of two modules. Benefiting from StopGuess, the construction of PAEKS is simplified to find proper implementations for the underlying modules. These constructions also feature an intuitive security reduction due to the modular construction.
- To demonstrate the superiority of StopGuess, we present two generic constructions of PAEKS with StopGuess. The basic construction is built from PEKS and non-interactive key exchange (NIKE), which covers all existing constructions through their different instantiations. To further reduce the certificate costs, we introduce a refined construction by reducing the key-pair needed. Moreover, pairing-based and lattice-based instantiations of the refined construction are presented to show the efficiency of the proposed constructions.
- Profiting from StopGuess, we naturally extend PAEKS to support further properties or features without additional assumptions on the underlying modules. Considering the requirements of anonymous applications, we extend the security of PAEKS to anonymity and prove that our refined construction is inherently anonymous. Besides, our construction could easily be integrated with a symmetric encryption scheme to achieve efficient data retrieval on large-scale data.

1.2. Related work

Since the widespread adoption of cloud services, there has been a lot of work on searching for encrypted data [20–27]. One promising technique for this topic is PEKS, but it suffers from keyword guessing attack (KGA). The study of KGA in PEKS is initiated by Byun et al. [3]. Then, Jeong et al. [28] and Shao et al. [29] showed the impossibility of constructing a PEKS scheme that is secure against a malicious server under the conventional PEKS [1].

In a nutshell, the fundamental cause of the KGA is that an adversary could generate ciphertext for arbitrary keywords, and then test it with the trapdoor through the Test algorithm. Due to the limited space of keywords, the adversary can guess the keyword contained in the trapdoor by doing a brute-force search. There are two approaches to block an adversary from performing such a search:

Restrict testing. The first approach is implemented by limiting the test ability of the adversary. Based on the BDOP-PEKS, Rhee et al. [5] add another test server to the system and designate the test ability to the server. It ensures that an adversary cannot run the Test algorithm without the secret key of the server. But the dPEKS only resists the OKGA. To address the IKGA, Chen et al. proposed server-aided [6] and dual-server PEKS [7]. Both two schemes divide the server into two servers, i.e., keyword server and storage server; front server and back server. Although the above two schemes eliminate the IKGA, they require additional parties or communications.

Restrict encryption. The latter approach is implemented by limiting the ability of ciphertext generation to the sender that owns the secret key corresponding to the trapdoor, i.e., PAEKS. After Huang et al. [11] first proposed PAEKS in 2017, Li et al. [16] and He et al. [15] gives two constructions of PAEKS in identity-based and certificateless setting, respectively. The most recent work is due to Liu et al. [18] and Emura et al. [30], who proposed a generic construction using the word-independent smooth projection hash function (WI-SPHF) and PEKS. Additionally, for the security of PAEKS, Noroozi et al. [12] and Qin et al. [13] improved the security model of [11] from single-user setting to multi-user setting. Then, Qin et al. [14] further improved the security model of PAEKS to fully ciphertext indistinguishability, which allows an adversary to query any ciphertext-keywords of its choice.

In summary, [11,15,16] constructs PAEKS schemes in different settings, but is limited to concrete constructions. On the other hand, although [18,30] proposed generic constructions, they fail to cover the

Fig. 2. Ciphertext/Trapdoor generation under StopGuess.

previous concrete constructions. Differing from these works, StopGuess provides a general framework for PAEKS that covers the aforementioned constructions.

2. StopGuess: ready-to-eliminate KGA

Before presenting StopGuess, we first describe the building modules of StopGuess on a high level. These modules are just an abstraction of the required functionalities and properties; they will be instantiated later in Section 3.

Shared key generation (SKG). This module is used to generate a key-pair for users and a shared key between two parties, that is, any two users could generate a shared key by using their own secret key and the other party's public key. For the security of the SKG, we need the shared key to be indistinguishable from a random choice in the key-space, which prevents an adversary from generating a shared key without the corresponding secret key.

Private equality test (PET). The core functionality of PET is to test whether a ciphertext C and a trapdoor T are generated from the same message. The procedure of test needs to be private, i.e., the information about the message M contained in C or T would not leak in the test phase. And the algorithms to generate ciphertext and trapdoor should be probabilistic. Otherwise, the KGA will be possible again.

StopGuess combines the above two modules to build PAEKS in a modular and concise manner. The basic idea of StopGuess is to combine a keyword and a shared key into one message, and then an equality test on the message will authenticate the shared key and test the keyword at the same time. In detail, the sender and the receiver first separately generate a message which is a combination of the shared key and the keyword, then they generate a ciphertext and a trapdoor, respectively, based on their respective message. Finally, the PET module that runs on the server determines whether the ciphertext and trapdoor contain the same message. Since we already defined the underlying modules, the rest of the problem is how to combine a keyword and a shared key. A plausible solution is to take a hash function and program it as a random oracle to ensure the consistency of StopGuess. Based on the above modules, a general description of StopGuess is presented in Fig. 2.

Implementations. To build a generic construction of PAEKS under StopGuess, we only need to properly implement the underlying modules. It is easy to see how the SKG can be implemented using an NIKE scheme. Then, for the PET, we have multi-choices. The classic PEKS scheme (e.g., BDOP-PEKS) may be the preferred option, and it works well in our basic generic construction and existing schemes. However, constructions from PEKS and NIKE require the receiver to store two key-pairs, which means that the system needs to maintain two certificate systems. We introduce a refined construction that achieves a single receiver key-pair without additional costs to address this problem. The comparison between our two constructions is presented in Table 2.

Security. An IKGA-resistant PEKS scheme requires both the ciphertexts and trapdoors to be indistinguishable under IKGA. We focus on realizing the trapdoor indistinguishability, since the security of ciphertexts already satisfies in a classic PEKS scheme. StopGuess achieves the trapdoor indistinguishability through the shared key between two parties, so its security directly follows the underlying modules. In detail, the variant of keyword is pseudorandom for different inputs, since the shared key is pseudorandom and the hash function (programmed as a random oracle) prevents collusion from different inputs.

Table 2
Comparisons between our constructions.

Schemes	SKG	PET	Security	Cert. Cost	Anon.	SKE-PAEKS
Basic	NIKE	PEKS	*fully* CI + TI	×2	✓*	✓
Refined		PKEET	CI + TI	×1	✓	✓

"Anon.": anonymity. "Cert. Cost": means certificate cost that include costs in registration, communication and storage, etc.. "SKE-PAEKS": means integrated symmetric encryption and PAEKS. "✓*": the anonymity of the scheme requires that the underlying PEKS scheme be anonymous.

Then, the semantic security of PET ensures every ciphertext/trapdoor is different even if they contain the same sender/receiver information and keyword. Therefore, the generated ciphertext/trapdoor satisfies indistinguishability in the multi-user setting.

Extensions. We explore the extension of StopGuess on anonymity and data retrieval capabilities, and get the following results: (i) Both of our constructions achieve anonymity, which means that an adversary cannot learn the generator from a ciphertext or trapdoor. More specifically, our refined construction achieves anonymity inherently and the basic construction achieves anonymity when the underlying PEKS is anonymous[1]; (ii) Based on the shared key from the SKG module, both of our constructions can be easily integrated with a symmetric encryption scheme to achieve data retrieval function without additional assumptions or storage costs.

3. Preliminaries

In this section, we instantiate the above modules and introduce definitions for the syntax and security of PAEKS for the remaining sections.

Notations. Let $x \in X$ be a variable. $x \leftarrow\!\$\, X$ denotes the operation of sampling x randomly from X. λ denotes the security parameter in all definitions and constructions. A function $F_k : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ is pseudorandom, if for all PPT distinguishers D, the equation $|\Pr[D^{F_k(\cdot)}(1^\lambda) = 1] - \Pr[D^{f(\cdot)}(1^\lambda) = 1]| \leq \mathsf{negl}(\lambda)$ holds, where $k \leftarrow\!\$\, \{0,1\}^\lambda$ and f is randomly chosen from a set of random functions.

3.1. Cryptographic primitives

Although many schemes allow generating a shared key between two parties, we need the instantiation to be non-interactive. Therefore, non-interactive key exchange (NIKE) becomes the preferred option. Besides, we present two instantiations of the private equality test module, namely public key encryption with keyword search (PEKS) and public key encryption with equality test (PKEET). We recall their definitions as follows:

3.1.1. Non-interactive key exchange (NIKE)

A NIKE scheme [32] allows any two users to generate a shared key between them in a non-interactive way and consists of three algorithms:

- CommonSetup: on input of security parameter λ, output a set of public parameters $\mathcal{PP}$.

[1] An anonymous PEKS scheme can be built from anonymous hierarchical identity-based encryption [31].

- KeyGen: on input of $\mathcal{PP}$ and an identity ID, output a pair of public key and secret key (pk, sk).
- SharedKey: on input of an identity ID_1 and its corresponding secret key sk_1 along with another identity ID_2 and its corresponding public key pk_2, output a shared key shk for the two identities.

Correctness: We say a NIKE scheme is correct if for any two identities ID_1, ID_2 and corresponding key-pairs, the following equation holds: $\mathsf{SharedKey}(ID_1, pk_1, ID_2, sk_2) = \mathsf{SharedKey}(ID_2, pk_2, ID_1, sk_1)$.

CKS-light Security: We say a NIKE scheme is CKS-light secure, if for any PPT adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ holds in the game of CKS-light satisfies: $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{CKS-light}}(\lambda) = \left|\Pr\left[b = b'\right] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda)$, where the game is defined in Appendix A.

3.1.2. Public key encryption with keyword search (PEKS)

A public key encryption with keyword search [1] scheme PEKS = (Setup, KeyGen, PEKS, Trapdoor, Test) consists of four algorithms:

- Setup: on input of security parameter λ, output public parameters $\mathcal{PP}$.
- KeyGen: on input of $\mathcal{PP}$, output a pair of keys (pk, sk).
- PEKS: on input of a public key pk and a word w, output a searchable ciphertext C corresponding to w.
- Trapdoor: on input of a secret key sk and a word w', output a trapdoor T_w.
- Test: on input of the public key pk, a searchable ciphertext $C = \mathsf{PEKS}(pk, w)$ and a trapdoor $T_w = \mathsf{Trapdoor}(pk, w')$, output '1' if $w = w'$ and '0' otherwise.

Correctness: For any ciphertext $C = \mathsf{PEKS}(pk, w)$ and trapdoor $T = \mathsf{Trapdoor}(sk, w')$, if the keyword $w = w'$, the probabilistic of $\Pr[\mathsf{Test}(C, T) = 1]$ is overwhelming.

3.1.3. Public key encryption with equality test (PKEET)

A PKEET [33] scheme allows anyone to check whether two ciphertexts encrypt the same message without information leakage and consists of six algorithms:

- Setup: on input of security parameter λ, output public parameters $\mathcal{PP}$.
- KeyGen: on input of an identity ID, output a pair of keys (pk, sk).
- Enc: on input of pk and a message m, output a ciphertext CT.
- Dec: on input of the secret key sk and a ciphertext CT, output a message m' or $\perp$.
- Trapdoor: on input of the secret key sk, output a trapdoor td.
- Test: on input of two trapdoors and two ciphertexts for two users respectively, output '1' if two ciphertexts encrypt the same message and '0' otherwise.

Correctness: We say PKEET is correct if for any ciphertext and trapdoor pairs $(C_1 = \mathsf{Enc}(pk_1, m_1), T_1 = \mathsf{Trapdoor}(sk_1))$ and $(C_2 = \mathsf{Enc}(pk_2, m_2), T_2 = \mathsf{Trapdoor}(sk_2))$, the equation holds:

$$\begin{cases} \Pr[\mathsf{Test}(C_1, T_1, C_2, T_2) = 1] \geq 1 - \mathsf{negl}(\lambda), & \text{if } m_1 = m_2 \\ \Pr[\mathsf{Test}(C_1, T_1, C_2, T_2) = 1] \leq \mathsf{negl}(\lambda), & \text{otherwise.} \end{cases}$$

4. Definitions

In this section, we present the definition and security model of PAEKS. Besides, the system model of PAEKS is presented in Fig. 3.

4.1. Definitions of syntax

We now formally define a public key authenticated encryption with keyword search (PAEKS) scheme. Differing to the definition from [11], the SkGen/RkGen algorithm in our definition requires an input of identity. This difference facilitates StopGuess to cover instantiations of PAEKS in both identity-based and public key settings. Note that, for a public key system, identities are merely used to track the binding relationship between a user and a public key.[2] Specifically, the algorithms are defined as follows.

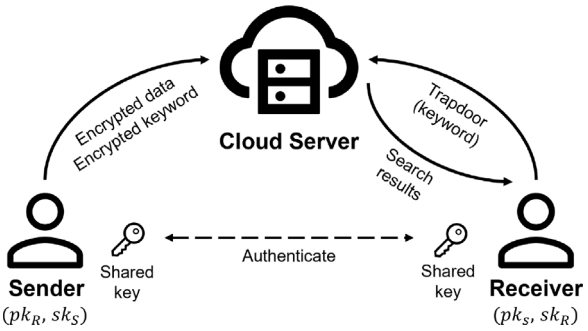

Fig. 3. System model of PAEKS.

- $\mathsf{Setup}(1^\lambda)$: on input of a security parameter λ, generate public parameters $\mathcal{PP}$. We assume that the public parameters will be the input of the following algorithms implicitly.
- $\mathsf{SkGen}(\mathcal{PP}, ID_S)$: given a sender's identity ID_S, generate a key-pair (pk_S, sk_S) for the sender. The identity of the sender is assumed to be included in the public key implicitly; so do the public key of the receiver.
- $\mathsf{RkGen}(\mathcal{PP}, ID_R)$: given a receiver's identity ID_R, generate a key-pair (pk_R, sk_R) for the receiver.
- $\mathsf{PAEKS}(w, sk_S, pk_R)$: for a keyword w, a sender's secret key sk_S and a receiver's public key pk_R, produce a searchable ciphertext C for keyword w.
- $\mathsf{Trapdoor}(w', sk_R, pk_S)$: for a keyword w', a receiver's secret key sk_R and a sender's public key pk_S, produce a trapdoor $T_{w'}$ for keyword w'.
- $\mathsf{Test}(C, T_{w'})$: given a ciphertext $C = \mathsf{PAEKS}(w, sk_S, pk_R)$ and trapdoor $T_w = \mathsf{Trapdoor}(w', sk_R, pk_S)$, output '1' if $w = w'$ and '0' otherwise.

Correctness. For any ciphertext $C = \mathsf{PAEKS}(w, sk_S, pk_R)$ and trapdoor $T = \mathsf{Trapdoor}(w', sk_R, pk_S)$, a PAEKS scheme is correct if $\Pr[\mathsf{Test}(C, T) = 1] \geq 1 - \mathsf{negl}(\lambda)$ holds, when $w = w'$.

4.2. Definitions of security

The security of PAEKS involves two parts, i.e., ciphertext security and trapdoor security. We first define the security of the ciphertext by fully ciphertext indistinguishability (fully CI). As for the security of the trapdoor, we use trapdoor indistinguishability (TI) which is similar to the definition in [12]. Both fully CI-security and TI-security are defined in the multi-user setting.

Actually, the fully CI-security and the TI-security are defined in the same manner. By replacing the challenge ciphertext in Game $\mathrm{G}_{\mathcal{E},\mathcal{A}}^{\mathrm{fCI}}(1^\lambda)$ with trapdoor, we can immediately obtain the TI-security game from the fully CI-security game.

Definition 1 (*Fully CI-Security*). A PAEKS scheme $\mathcal{E}$ satisfies fully ciphertext indistinguishability if the advantage of any PPT adversary

[2] In the rest of this paper, all the definitions, security models, and constructions are in the public key setting.

Game $\mathrm{G}^{\mathrm{fCI}}_{\mathcal{E},\mathcal{A}}(1^\lambda)$	Game $\mathrm{G}^{\mathrm{TI}}_{\mathcal{E},\mathcal{A}}(1^\lambda)$
$\mathcal{PP} \leftarrow \mathsf{Setup}(1^\lambda)$	$\mathcal{PP} \leftarrow \mathsf{Setup}(1^\lambda)$
$(pk_S, sk_S) \leftarrow \mathsf{KeyGen}_S(\mathcal{PP}, ID_S)$	$(pk_S, sk_S) \leftarrow \mathsf{KeyGen}_S(\mathcal{PP}, ID_S)$
$(pk_R, sk_R) \leftarrow \mathsf{KeyGen}_R(\mathcal{PP}, ID_R)$	$(pk_R, sk_R) \leftarrow \mathsf{KeyGen}_R(\mathcal{PP}, ID_R)$
$(w_0^*, w_1^*) \leftarrow \mathcal{A}_1^{\mathcal{O}_C, \mathcal{O}_T}(\mathcal{PP}, pk_S, pk_R)$	$(w_0^*, w_1^*) \leftarrow \mathcal{A}_1^{\mathcal{O}_C, \mathcal{O}_T}(\mathcal{PP}, pk_S, pk_R)$
$b \leftarrow\$ \{0,1\}$	$b \leftarrow\$ \{0,1\}$
$C \leftarrow \underline{\mathsf{PAEKS}(w_b^*, sk_S, pk_R)}$	$T \leftarrow \underline{\mathsf{Trapdoor}(w_b^*, sk_R, pk_S)}$
$b' \leftarrow \mathcal{A}_2^{\mathcal{O}_C, \mathcal{O}_T}(\mathcal{PP}, pk_S, pk_R, C)$	$b' \leftarrow \mathcal{A}_2^{\mathcal{O}_C, \mathcal{O}_T}(\mathcal{PP}, pk_S, pk_R, T)$
return $b = b'$	**return** $b = b'$

Fig. 4. The fully ciphertext indistinguishability and trapdoor indistinguishability for a PAEKS scheme $\mathcal{E}$ and an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. The oracles are defined as $\mathcal{O}_C(\cdot,\cdot) := \mathsf{PAEKS}(\cdot, sk_S, \cdot)$, $\mathcal{O}_T(\cdot,\cdot) := \mathsf{Trapdoor}(\cdot, sk_R, \cdot)$. The difference between the two games is underlined.

Table 3
Difference between CI-Security and fully CI-Security.

Model	Ciphertext security	
	Ciphertext Oracle	Trapdoor oracle
CI	$(pk, w) \neq (pk_R, w_b^*)$	$(pk, w) \neq (pk_S, w_b^*)$
fully-CI	$(pk, w) = (\star, \star)$	$(pk, w) \neq (pk_S, w_b^*)$

$\star$: Arbitrary choice.

$\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ in breaking the fully CI-security game satisfies the equation $\mathsf{Adv}^{\mathrm{fCI}}_{\mathcal{A}}(\lambda) = \left|\Pr\left[\mathrm{G}^{\mathrm{fCI}}_{\mathcal{E},\mathcal{A}}(1^\lambda) = 1\right] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda)$, where the game is depicted in Fig. 4.

To prevent trivial wins for the adversary $\mathcal{A}$, it is restricted to making queries about (pk_S, w_0^*) and (pk_S, w_1^*) to the trapdoor oracle. Otherwise, the ciphertext can be distinguished trivially.

Definition 2 (*TI-Security*)**.** A PAEKS scheme $\mathcal{E}$ satisfies trapdoor indistinguishability if the advantage of PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ in breaking the TI-security game satisfies $\mathsf{Adv}^{\mathrm{TI}}_{\mathcal{A}}(\lambda) = \left|\Pr\left[\mathrm{G}^{\mathrm{TI}}_{\mathcal{E},\mathcal{A}}(1^\lambda) = 1\right] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda)$, where the security game is depicted in Fig. 4.

Similar to the restriction in the above game, the adversary $\mathcal{A}$ is restricted to making queries about (w_b^*, pk_R) to the ciphertext oracle and (w_b^*, pk_S) to the trapdoor oracle, where $b \in \{0,1\}$.

The definition of ciphertext indistinguishability security (CI-Security) is identical to that of fully CI-security, as shown in Table 3, except that the adversary cannot issue queries about the challenge keyword to the ciphertext oracle.

Definition 3 (*CI-Security*)**.** A PAEKS scheme satisfies ciphertext indistinguishability if the advantage of any PPT adversary $\mathcal{A}$ in breaking the CI-security game is negligible in λ.

5. Constructions

In this section, we present a basic generic construction of PAEKS, followed by a refined generic construction.

5.1. Our basic generic construction

We first implement SKG and PET by NIKE and PEKS, and present the basic construction from these two implementations below. In the basic construction, the keyword that inputs into the PEKS algorithm is replaced by a hash of the keyword and a shared key. Therefore, the Test algorithm would test keyword and authenticate the identity at the same time. The proposed generic construction consists of the following algorithms:

- $\mathsf{Setup}(1^\lambda)$: on input of a security parameter λ, generate public parameters through $\mathcal{PP}_{\mathcal{NIKE}} \leftarrow \mathsf{NIKE.CommonSetup}(1^\lambda)$ and $\mathcal{PP}_{\mathcal{PEKS}} \leftarrow \mathsf{PEKS.Setup}(1^\lambda)$. Choose a hash function $H : \mathcal{KS}_{word} \times \mathcal{KS}_{share} \rightarrow \mathcal{KS}_{word}$, where $\mathcal{KS}_{share}$ and $\mathcal{KS}_{word}$ denote the space of shared key and keyword, respectively. Output the public parameters $\mathcal{PP} = (\mathcal{PP}_{NIKE}, \mathcal{PP}_{PEKS}, H)$.
- $\mathsf{SkGen}(ID_S)$: on input of an identity ID_S, generate a key-pair $(pk_S, sk_S) \leftarrow \mathsf{NIKE.KeyGen}(\mathcal{PP}, ID_S)$ and output the key-pair.
- $\mathsf{RkGen}(ID_R)$: on input of an identity ID_R, generate two key-pairs as $(pk_R, sk_R) \leftarrow \mathsf{NIKE.KeyGen}(\mathcal{PP}, ID_R)$ and $(pk_{PEKS}, sk_{PEKS}) \leftarrow \mathsf{PEKS.KeyGen}(\mathcal{PP}, ID_R)$. Output the key-pairs as $(pk_R, sk_R, pk_{PEKS}, sk_{PEKS})$.
- $\mathsf{PAEKS}(w, pk_{PEKS}, sk_S, pk_R)$: on input of a keyword w, a PEKS public key pk_{PEKS}, a sender's secret key sk_S and a receiver's public key pk_R, run as follows:
  - Compute a shared key $shk = \mathsf{NIKE.SharedKey}(ID_R, pk_R, ID_S, sk_S)$ between the sender S and the receiver R, and generate a variant of keyword $kw = H(w, shk)$.
  - Compute the searchable ciphertext $C = \mathsf{PEKS.PEKS}(pk_{PEKS}, kw)$, and then output the ciphertext.
- $\mathsf{Trapdoor}(w', sk_{PEKS}, sk_R, pk_S)$: on input of a keyword w', a PEKS secret key sk_{PEKS}, a sender's public key pk_S and a receiver's secret key sk_R, run as follows:
  - Compute the shared key $shk' = \mathsf{NIKE.SharedKey}(ID_S, pk_S, ID_R, sk_R)$ and generate the variant of keyword $kw' = H(w', shk')$.
  - Compute a trapdoor $T = \mathsf{PEKS.Trapdoor}(sk_{PEKS}, kw')$ and then output the trapdoor.
- $\mathsf{Test}(C, T)$: on input of a ciphertext $C = \mathsf{PEKS.PEKS}(pk_{PEKS}, kw)$ and a trapdoor $T = \mathsf{PEKS.Trapdoor}(sk_{PEKS}, kw')$, run as follows:
  - Compute $b \leftarrow \mathsf{PEKS.Test}(C, T)$, and output b as the result.
    - $b = 1$: if $kw = kw'$, that is, the keyword and identities contained in both the ciphertext C and trapdoor T match each other.
    - $b = 0$: otherwise.

Correctness. The correctness of this construction is demonstrated as follows: If $\mathsf{NIKE.SharedKey}(ID_S, pk_S, ID_R, sk_R) = \mathsf{NIKE.SharedKey}(ID_R, pk_R, ID_S, sk_S)$ and $w = w'$, then we have $\mathsf{PAEKS.Test}(C, T) = \mathsf{PEKS.Test}(C, T) = 1$ with overwhelming probability.

Theorem 1 (*Fully CI-Security*). *Our PAEKS satisfies fully CI-security if the NIKE scheme is CKS-light secure in corrupt key registration (CKR) setting and the PEKS scheme is semantic secure.*

Proof Sketch. We prove the above statement by two lemmas. First, we prove the variant of keyword satisfies CI-security in Lemma 1. Then, in Lemma 2, we prove the semantic security of the underlying PEKS scheme could boost the CI-security to fully CI-security.

Lemma 1. *If the NIKE scheme is CKS-light secure in CKR setting, then the variant of keyword kw satisfies CI-security.*

Proof Sketch. This proof is based on contradictions. Suppose an adversary $\mathcal{A}$ could break the game $\mathrm{G}^{\mathrm{CI}}_{\mathcal{PAEKS},\mathcal{A}}(\lambda)$ with a non-negligible advantage. Then we could construct an adversary $\mathcal{B}$ which could break the game $\mathrm{G}^{\mathrm{CKS-light}}_{\mathcal{NIKE},\mathcal{B}}(\lambda)$ with a non-negligible advantage. The complete proof can be found in Appendix B.

Lemma 2. *If the variant of keyword is CI-secure and the PEKS is semantic secure, then the above PAEKS scheme satisfies fully CI-security.*

Proof Sketch. We prove this lemma using a game hop from Game_0 to Game_1. We detail the two games as follows:

Game_0: The first game is identical to the game of CI-security. In this game, the ciphertext is generated by PAEKS with the input of the variant of keyword. Since we already prove that the variant of keyword satisfies CI-security in Lemma 1, we have that the advantage of adversary in breaking this game satisfies $\mathsf{Adv}^{\mathsf{Game}_0}_{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$.

Game_1: This game is identical to Game_0 except that the adversary $\mathcal{A}$ can query ciphertext-keyword for the challenge keyword. The semantic security of the PEKS scheme guarantees that the adversary cannot get any knowledge about the challenge keyword from prior queries for the challenge keyword. Therefore, we have $\left|\mathsf{Adv}^{\mathsf{Game}_1}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{\mathsf{Game}_0}_{\mathcal{A}}(\lambda)\right| \leq \mathsf{negl}(\lambda)$. The details of the proof can be found in Appendix C.

Finally, with the above games, we have $\mathsf{Adv}^{\mathsf{Game}_1}_{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$, which completes the proof.

Theorem 2 (*TI-Security*). *Our PAEKS satisfies TI-security if the NIKE scheme is CKS-light secure in CKR setting and the PEKS scheme is semantic secure.*

Proof Sketch. The security can be proved in the same way as Theorem 1, except that the requirement for the PEKS algorithm now relates to the Trapdoor algorithm.

Discussions. While the above construction is secure against IKGA, it is inefficient in practice, since each receiver needs to maintain two public-key certificates. A naive solution for improving its efficiency is to reuse the key-pair of the underlying NIKE scheme in the PEKS scheme. However, the key reuse solution requires that the key-pair of the NIKE scheme and the PEKS scheme have the same form, thus limiting the instantiation of the construction. Besides, it could undermine the security guarantees of the PAEKS scheme. For example, [11] generates a shared secret through DH-NIKE, and then reuses the sender's key-pair to generate ciphertexts, which allows an adversary to easily check whether two ciphertexts contain the same keyword and results in a weak security in ciphertext indistinguishability.

5.2. Our refined generic construction

The refined generic construction implements PET and SKG by public key encryption with equality test (PKEET) and NIKE. Unlike the basic construction, a receiver only stores one key-pair in this construction, which reduces the certificate cost on receivers to one instead of two. This is because the key-pair of PKEET is generated only at the time of use, and thus does not need to be retained. However, due to the incompatibility of the ciphertext test and indistinguishability, PKEET only achieves weak indistinguishability under chosen ciphertext attacks (W-IND-CCA2) [33] and cannot achieve semantic security. Therefore, our refined construction only considers the CI-security and the TI-security. That means, in its instantiations, although all ciphertexts with the same message are different, the server can distinguish them through the Test algorithm.

- $\mathsf{Setup}(1^\lambda)$: on input of a security parameter λ, generate public parameters by running $\mathcal{PP}_{PKEET} \leftarrow \mathsf{PKEET.Setup}(1^\lambda)$ and $\mathcal{PP}_{NIKE} \leftarrow \mathsf{NIKE.CommonSetup}(1^\lambda)$. Then choose hash function $H : \{0,1\}^n \rightarrow \mathcal{M}$ and output the public parameters $\mathcal{PP} = (\mathcal{PP}_{NIKE}, \mathcal{PP}_{PKEET}, H)$.
- $\mathsf{SkGen/RkGen}(ID)$: on input of an identity ID, generate a key-pair (pk, sk) by running $\mathsf{NIKE.KeyGen}(ID)$.
- $\mathsf{PAEKS}(w, sk_S, pk_R)$: on input of a keyword w, a sender's secret key sk_S and a receiver's public key pk_R, run as follows:
  - Compute a shared key $shk = \mathsf{NIKE.SharedKey}(ID_R, pk_R, ID_S, sk_S)$ and generate a variant of keyword $kw = H(w, shk)$.
  - Generate a PKEET key-pair $(pk_{Enc}, sk_{Enc}) \leftarrow \mathsf{PKEET.KeyGen}(ID_S)$. Then, compute ciphertext $ct_S = \mathsf{PKEET.Enc}(pk_{Enc}, kw)$ and trapdoor $td_S = \mathsf{PKEET.Td}(sk_{Enc})$. Output the searchable ciphertext $C = (ct_S, td_S)$ for keyword kw.
- $\mathsf{Trapdoor}(w', sk_R, pk_S)$: on input of a keyword w', a sender's public key pk_S and a receiver's secret key sk_R, run as follows:
  - Compute the shared key $shk' = \mathsf{NIKE.SharedKey}(ID_S, pk_S, ID_R, sk_R)$ and generate a variant of keyword $kw' = H(w', shk')$.
  - Generate a PKEET key-pair $(pk_{Td}, sk_{Td}) \leftarrow \mathsf{PKEET.KeyGen}(ID_R)$. Then, compute ciphertext $ct_R = \mathsf{PKEET.Enc}(pk_{Td}, kw')$ and trapdoor $td_R = \mathsf{PKEET.Td}(sk_{Td})$. Output the trapdoor $T = (ct_R, td_R)$ for keyword kw'.
- $\mathsf{Test}(C, T)$: on input of a ciphertext $C = (ct_S, td_S)$ and a trapdoor $T = (ct_R, td_R)$, compute $b = \mathsf{PKEET.Test}(ct_S, ct_R, td_S, td_R)$ and output b as the result.

The correctness of the above construction follows the correctness of the underlying schemes. The security analysis of the above construction is similar to the proof of Theorem 1. Therefore, we omit the details to conserve space.

Theorem 3 (*CI-Security*). *Our PAEKS satisfies CI-security if the NIKE scheme is CKS-light secure in CKR setting and the Enc and Td algorithm of the PKEET scheme is W-IND-CCA2 secure.*

Theorem 4 (*TI-Security*). *Our PAEKS satisfies TI-security if the NIKE scheme is CKS-light secure in CKR setting and the Enc and Td algorithm of the PKEET scheme is W-IND-CCA2 secure.*

6. Instantiations

In this section, we present instantiations of our basic and refined constructions.

6.1. Instantiation of basic construction

Our basic construction can be instantiated by a variety of NIKE schemes and PEKS schemes, such as DH-NIKE [34], SOK-IB-NIKE [35], KV-PAKE[3] [36] and BDOP-PEKS [1]. By combining these instantiations,

[3] Actually, the one-round PAKE scheme requires a password as input; by putting the password into the public parameter, it can be used as an NIKE scheme.

Table 4
Related work comparisons.

Schemes	Setting	NIKE	PEKS
QCZZ21 [14]	Public-key	$\mathcal{DH}-\mathcal{NIKE}$ [34]	$\mathcal{BDOP}-\mathcal{PEKS}$ [1]
LHS+19 [16]	ID-based	$\mathcal{SOK}-\mathcal{IB}-\mathcal{NIKE}$ [35]	
LTT+22 [18]	–	$\mathcal{KV}-\mathcal{PAKE}$ [36]	$\mathcal{PEKS}$

we can construct existing schemes of PAEKS [14,16,18]. So we do not actually instantiate our basic construction here, but present how our basic construction covers the existing schemes in Table 4.

Obliviously, there are other PAEKS schemes that do not appear in Table 4. We omit these schemes since they do not satisfy *fully* CI [11–13,15,17] or employ designated-user setting [30].

6.2. Instantiation of refined construction

We first instantiate our refined construction by Diffie–Hellman NIKE [34] and the pairing-based PKEET scheme in [33] as below. In addition, we present a lattice-based instantiation in Appendix D to demonstrate the flexibility of our construction.

- $\mathsf{Setup}(1^\lambda)$: given a security parameter λ, choose two cyclic groups $\mathbb{G}$, $\mathbb{G}_T$ with prime order q and generator g, and bilinear map $e : \mathbb{G}\times\mathbb{G}\to\mathbb{G}_T$. Select hash function $H : \{0,1\}^* \to \{0,1\}^\lambda$, $H_1 : \{0,1\}^* \to \mathbb{G}$. Finally output public parameters as $\mathcal{PP} = (\mathbb{G}, \mathbb{G}_T, q, g, e, H, H_1)$.
- $\mathsf{SkGen}(ID_S)$: choose $x \leftarrow\!\$\, \mathbb{Z}_q$ and generate a key-pair $(pk_S, sk_S) = (g^x, x)$.
- $\mathsf{RkGen}(ID_R)$: choose $y \leftarrow\!\$\, \mathbb{Z}_q$ and generate a key-pair $(pk_R, sk_R) = (g^y, y)$.
- $\mathsf{PAEKS}(w, sk_S, pk_R)$: given a keyword w, secret key sk_S and public key pk_R, generate a shared key as $shk = H(ID_S \| ID_R \| pk_R^{sk_S})$. Then, generate ciphertext as $C = (C_1, C_2) = (g^{r_c}, H_1(w, shk)^{r_c})$, where $r_c \leftarrow\!\$\, \mathbb{Z}_q$. Finally, output the ciphertext.
- $\mathsf{Trapdoor}(w', sk_R, pk_S)$: given a keyword w', secret key sk_R and public key pk_S, the algorithm is the same as the PAEKS algorithm. Output the trapdoor as $T = (T_1, T_2) = (g^{r_t}, H_1(w', shk')^{r_t})$, where $shk' = H(ID_S \| ID_R \| pk_S^{sk_R})$.
- $\mathsf{Test}(C, T)$: given a ciphertext C and trapdoor T, compute the equation $e(C_1, T_2) = e(T_1, C_2)$ and output '1' if it holds. Otherwise, output '0'.

Correctness and Security. Following from the underlying PKEET and DH-NIKE, the correctness and security of the above PAEKS scheme can be easily proven. Details are omitted to conserve space.

6.3. Comparison and evaluation

We first provide a theoretical analysis of the efficiency and size of our instantiation (Section 6.2); we compare it with BDOP-PEKS [37], QCH+20 [13], and QCZZ21 [14]. As shown in Table 5, compared to BDOP-PEKS, other PAEKS schemes achieve IKGA-resistant with slightly higher costs. Compared to the other two schemes, our instantiation reduces the certificate cost from two to one, and achieves anonymity without additional assumption (See in Section 7). Note that we ignore the costs of the general hash function in the efficiency analysis.

Additionally, to evaluate the performance of our scheme in practice, we implement the above schemes in C based on the PBC library [38], where the elliptic curve is generated by the Type-A parameters. The hash function that maps string to group element uses the hash function in the PBC library, and the general hash function uses SHA-256. The experiments are conducted on a PC with an Intel Core i5-8500 CPU@3.00 GHz and 8 GB of RAM. As shown in Fig. 5, our scheme has similar efficiency with other schemes in encryption. The trapdoor and test algorithms in our scheme consume more time than other schemes, but our scheme has lower costs for certification and the running time of the two algorithms is still low in practice.

Table 5
Computation and communication efficiency comparison.

Schemes	Efficiency				Size		
	RkGen	Encryption	Trapdoor	Test	RK (sk|pk)	Ciphertext	Trapdoor
BDOP04 [1]	E	2E+P+H	E+H	P	$\mathbb{Z}_p\|\mathbb{G}$	$\mathbb{G}$+$\mathbb{H}$	$\mathbb{G}$
QCH+20 [13]	E	3E+P+H	2E+H	P	$\mathbb{Z}_p\|\mathbb{G}$	$\mathbb{G}$+$\mathbb{H}$	$\mathbb{G}$
QCZZ21 [14]	2E	3E+P+H	2E+H	P	$2\mathbb{Z}_p\|2\mathbb{G}$	$\mathbb{G}$+$\mathbb{H}$	$\mathbb{G}$
Section 6.2	E	3E+H	3E+H	2P	$\mathbb{Z}_p\|\mathbb{G}$	$2\mathbb{G}$	$2\mathbb{G}$

P: Bilinear pairing; E: Exponentiation in the group; H: Hash function that maps a string to a group element.
$\mathbb{G}$: Ordinary elliptic group element; $\mathbb{H}$: General hash function output.

7. Extensions

We present two extensions of PAEKS, i.e., anonymous PAEKS and integrated symmetric encryption and PAEKS (shorted as SKE-PAEKS).

7.1. Anonymous PAEKS

Anonymous PAEKS is a generalization of a PAEKS scheme in which the identities of both parties are hidden. We extend PAEKS to anonymous PAEKS, due to the importance of anonymity in both theoretical research and real-world applications. For example, Bellare et al. [37] proposed the concept of anonymity in public key encryption; anonymous bulletin board is another notable application. The syntax of anonymous PAEKS is the same as PAEKS, but different in security.

Security. Compared to Definitions 1 and 2, the definition of anonymous PAEKS adds protection to the privacy of users' identities, which ensures an adversary cannot guess the identities of both parties from the ciphertexts or the trapdoors. Corresponding to the identity privacy, the security games of anonymous PAEKS add the identities and two key generation oracles (SkGen and RkGen) to the challenge and the query phase, respectively. This additional part in the game allows the adversary to query key-pairs for any identities except the challenge identities, and challenge any identities as it wants. Besides, to prevent trivial wins, the adversary is restricted to queries on the secret key of the challenge identities; another restriction is the same as discussed in Definition 1.

Construction. Cryptographic schemes always require additional components or assumptions to achieve anonymity. But in our construction, both parties' identities are hidden in the shared key whose security follows the security of NIKE. So, the anonymity of the refined construction is inherent in the anonymity of the underlying NIKE. Although the basic construction is also constructed from NIKE, its anonymity additionally requires the instantiation of the underlying PEKS to be anonymous due to its key-pair of PEKS.

Theorem 5 (*Anonymous PAEKS*)**.** *If a PAEKS scheme is constructed under StopGuess (and the underlying PEKS scheme is anonymous), then it satisfies anonymity.*

The above theorem actually covers both our basic and refined constructions. The anonymity of the basic construction is easy to get from the proof of the refined construction and the anonymity of the underlying PEKS scheme. Therefore, we only sketch the proof for the anonymity of the refined construction as follows:

Proof Sketch. The proof is divided into three cases covering different choices of keywords and public keys contained in the challenge. For each case, we prove that the advantage of the adversary is negligible.

Case 1. The keywords are different. Then the security can be reduced to the fully-CI security, and the proof is the same as Theorem 1.

Case 2. The keywords are the same, but the public keys are different. In this case, the challenge can be transformed into a challenge

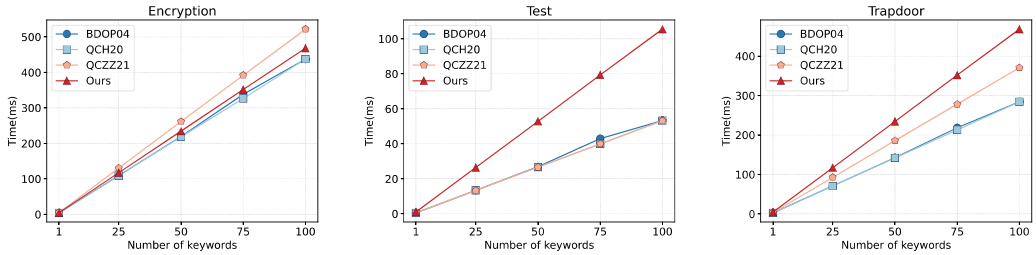

Fig. 5. Comparison of performance with other PAEKS schemes.

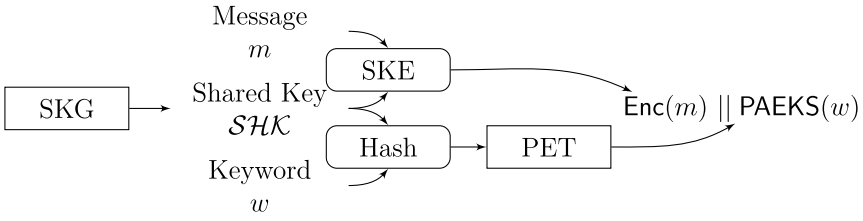

Fig. 6. Generation of a ciphertext in SKE-PAEKS.

to the pseudorandomness of the shared key. Therefore, the advantage of the adversary is identical to the advantage in breaking the pseudorandomness of NIKE, which is negligible.

Case 3. Both keywords and public keys are the same. This case is nonsense because there is no difference in challenge ciphertext whether b equals '0' or '1'.

7.2. Integrated SKE and PAEKS

PAEKS only provides the functionality of keyword searching. However, as pointed out by the aforementioned work [1], a PEKS scheme is meaningless without a public key encryption (PKE) scheme to provide a data retrieval function. This problem inspires a series of researches on integrated PKE and PEKS (PKE-PEKS) [39]. But, due to the efficiency gap between PKE and symmetric encryption (SKE), we integrate SKE and PAEKS to improve efficiency.

Construction. Existing schemes are often complicated when supporting data retrieval due to security requirements. In contrast, we could construct a SKE-PAEKS scheme more concisely and more efficiently by reusing the shared key; because our proposed constructions of PAEKS rely on a shared key between the sender and the receiver. That is, in the SKE-PAEKS scheme, both parties first separately generate a shared key $\mathcal{SHK}$ between them through an NIKE scheme. Then the sender encrypts messages and keywords with the $\mathcal{SHK}$ through SKE and PAEKS, respectively. For the receiver, the same shared key is used in both the search and decryption phases. We present the generation of a ciphertext of SKE-PAEKS as in Fig. 6.

Security and Anonymity. The joint chosen plaintext attack (CPA)-security of our SKE-PAEKS scheme is easy to prove based on the security of the underlying schemes. Furthermore, the anonymity of SKE and PAEKS has been established in [40] and the discussion in the last section, respectively. Combining them, we could have that the SKE-PAEKS scheme built from our construction satisfies anonymity.

8. Conclusion

In this paper, we have proposed StopGuess which provides an elegant solution to eliminating IKGA and features modular construction, intuitive proof, and flexible extensions. Under StopGuess, we modularly construct two generic constructions with intuitive security analysis. By instantiating with existing public-key building blocks, StopGuess is practical efficient. We also extend StopGuess to anonymous PAEKS and SKE-PAEKS to make it more useful. In the future, one work is to extend the proposed construction in multi-keyword setting, which supplements the proposed construction in the single-keyword setting. Another future work is to support fine-grained access control policy in multi-user environments.

CRediT authorship contribution statement

Tao Xiang: Writing – review & editing, Supervision. **Zhongming Wang:** Writing – original draft, Experiment, Formal analysis. **Biwen Chen:** Methodology, Formal analysis. **Xiaoguo Li:** Formal analysis, Methodology. **Peng Wang:** Methodology. **Fei Chen:** Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grants 62102050, U20A20176,62072062, the Natural Science Foundation of Chongqing, China, under Grants cstc2022ycjh-bgzxm0031 and CSTB2022NSCQMSX0582, Sichuan Science and Technology Program, China under Grant 2021YFQ0056, CCF-AFSG Research Fund, China under Grant CCF-AFSG RF20220009, the Science and Technology Research Program of Chongqing Municipal Education Commission, China under Grant KJQN202201520.

Appendix A. CKS-light Game

The CKS-light game is defined between adversary $\mathcal{A}$ and challenger $\mathcal{C}$ as follows:

- **Setup:** The challenger $\mathcal{C}$ takes input of security parameter λ and runs NIKE.CommonSetup to generate public parameters $\mathcal{PP}$. Then it gives $\mathcal{PP}$ to the adversary.

- **Query Phase:** The adversary adaptively issues queries and the challenger C answers $\mathcal{A}$'s queries as follows (Note that in the CKS-light security, $\mathcal{A}$ can only query **Register Honest ID** twice and **Test** once):
  - **Register Honest ID:** Given an identity ID, C runs NIKE.KeyGen to generate a key-pair (pk, sk) and records the tuple $(honest, ID, pk, sk)$. Then C returns pk to $\mathcal{A}$.
  - **Register Corrupt ID:** Given an identity ID and a public key pk, C records the tuple $(corrupt, ID, pk, \perp)$.
  - **Corrupt Reveal:** Given two identities ID_1, ID_2 that satisfies at least one of the two identities was registered as honest, C runs NIKE.SharedKey and returns the shared key to $\mathcal{A}$.
  - **Test:** Given two identities ID_1, ID_2 that both of them were registered as honest, C randomly choose $b \leftarrow \{0,1\}$. If $b = 0$, C runs NIKE.SharedKey and returns the shared key to $\mathcal{A}$. Otherwise, C randomly chooses a key and returns to $\mathcal{A}$.
- **Challenge:** $\mathcal{A}$ finally outputs $b' \in \{0,1\}$ as its guess.

Appendix B. Proof of Lemma 1

Proof. Supposed $\mathcal{A}$ is an attack algorithm that has advantage $\mathsf{Adv}^{\mathrm{CI}}_{\mathcal{A}}(\lambda) \geq \epsilon_1$ in breaking the Game $\mathrm{G}^{\mathrm{CI}}_{\mathcal{E},\mathcal{A}}(1^\lambda)$ of CI-security. Then we construct an algorithm $\mathcal{B}$ that breaking the security of NIKE with advantage $\mathsf{Adv}^{\mathrm{CKS-light}}_{\mathcal{B}}(\lambda) \geq \frac{\epsilon_1}{2}$. $\mathcal{B}$ simulates Game $\mathrm{G}^{\mathrm{CI}}_{\mathcal{E},\mathcal{A}}(1^\lambda)$ and interacts with $\mathcal{A}$ as follows:

Setup phase. The algorithm $\mathcal{B}$ first receives public parameter $\mathcal{PP}_{NIKE}$ from the challenger C. And then it registers ID_S and ID_R as honest users to C and receives pk_S, pk_R as response. Finally, $\mathcal{B}$ returns $(\mathcal{PP}_{NIKE}, pk_S, pk_R)$ to $\mathcal{A}$.

Query phase 1. $\mathcal{B}$ answer $\mathcal{A}$'s queries as follows:

- Hash queries. In this queries, $\mathcal{B}$ maintains a list $\mathcal{L}_H$ of tuple $\langle w, shk, h\rangle$. When $\mathcal{A}$ supplies a query on keyword and shared key (w_i, shk_i), the algorithm $\mathcal{B}$ first checks the list $\mathcal{L}_H$. If the query already appears on the list, $\mathcal{B}$ returns value from $\mathcal{L}_H$. Otherwise, it randomly choose $h_i \leftarrow \mathcal{KS}_{word}$ and adds the tuple $\langle w_i, shk_i, h_i\rangle$ to the list. Then it returns the tuple to $\mathcal{A}$.
- Ciphertext-keyword queries. $\mathcal{A}$ supplies a keyword w_i and a receiver's public key pk_{R_i}. $\mathcal{B}$ registers (ID_{Ri}, pk_{Ri}) to C as a corrupt user, makes reveal query of (ID_S, ID_{Ri}) to C and receives shk_i as response. Then $\mathcal{B}$ returns $C_i = kw_i$ to $\mathcal{A}$, where $kw_i = H(w_i, shk_i)$.
- Trapdoor-keyword queries. $\mathcal{A}$ supplies a keyword w_i and a sender's public key pk_{S_i}. $\mathcal{B}$ registers (ID_{Si}, pk_{Si}) to C as a corrupt user, makes reveal query of (ID_{Si}, ID_R) to C and receives shk_i as response. Then $\mathcal{B}$ returns $T_i = kw_i$ to $\mathcal{A}$, where $kw_i = H(w_i, shk_i)$.

Challenge. $\mathcal{A}$ supplies a pair of keywords (w_0^*, w_1^*) as its challenge. $\mathcal{B}$ makes test queries on (ID_S, ID_R) to C and receives shk^* as response. Then $\mathcal{B}$ computes $kw^* = H(w_b^*, shk^*)$ and returns $C^* = kw^*$ as the challenge ciphertext to $\mathcal{A}$, where $b \leftarrow\!\!\$\, \{0,1\}$.

Query phase 2. C answer $\mathcal{A}$'s queries as the same as phase 1.

Guess phase. Eventually, $\mathcal{A}$ supplies b' as its guess to Game $\mathrm{G}^{\mathrm{CI}}_{\mathcal{E},\mathcal{A}}(1^\lambda)$. If $b = b'$, then $\mathcal{B}$ returns $\hat{b} = 0$ as guess to C. Otherwise, $\mathcal{B}$ returns $\hat{b} = 1$ to C.

Since shk_i is the response from C and kw_i is generated as a real game, the response to $\mathcal{A}$ are perfectly simulated by $\mathcal{B}$. Therefore, there is no abort in the simulation.

Consider the response shk^* from C in the challenge phase which can be either a real key or a random bit string. If shk^* is a real key, $\mathcal{A}$'s view in the Game $\mathrm{G}^{\mathrm{CI}}_{\mathcal{E},\mathcal{A}}(1^\lambda)$ is identical to a real game. Thus the advantage $\mathsf{Adv}^{\mathrm{CKS-light}}_{\mathcal{B}}(\lambda)$ is equal to the advantage $\mathsf{Adv}^{\mathrm{CI}}_{\mathcal{A}}(\lambda)$. Else if shk^* is a random bit string, $\mathcal{A}$'s probability in guessing b is $1/2$. Thus, in this case, the advantage $\mathsf{Adv}^{\mathrm{CKS-light}}_{\mathcal{B}}(\lambda) = 0$.

Hence, from the above discussion, we have B breaks the game of CKS-light in advantage $\frac{\epsilon_1}{2}$, which completes the proof.

Appendix C. Proof of Game_1 in Lemma 2

Proof. Supposed $\mathcal{A}$ is an attack algorithm that has advantage $\mathsf{Adv}^{\mathrm{fCI}}_{\mathcal{A}}(\lambda) \geq \epsilon_1$ in breaking the game of fully CI-security. Then we construct an algorithm $\mathcal{B}$ that breaks the semantic security of the PEKS scheme with advantage $\mathsf{Adv}^{\mathrm{ss}}_{\mathcal{B}}(\lambda) \geq (q_{ck} \cdot \epsilon_1)/|\mathcal{KS}_{word}|$, where q_{ck} and $\mathcal{KS}_{word}$ denote the number of ciphertext-keyword queries and the space of keyword respectively. $\mathcal{B}$ simulates Game $\mathrm{G}^{\mathrm{fCI}}_{\mathcal{E},\mathcal{A}}(1^\lambda)$ and interacts with $\mathcal{A}$ as follows:

Setup phase. The algorithm $\mathcal{B}$ receives public parameter $\mathcal{PP}$ from the challenger C, registers ID_S, ID_R as honest users to C and receives pk_S, pk_R as response. Finally, $\mathcal{B}$ returns $(\mathcal{PP}, pk_{PEKS}, pk_S, pk_R)$ to $\mathcal{A}$.

Query phase 1. $\mathcal{B}$ answers $\mathcal{A}$'s queries as follows:

- Hash queries. This query is the same as in the proof of Lemma 1.
- Ciphertext-keyword queries. $\mathcal{A}$ supplies a keyword w_i and a receiver's public key (pk_{R_i}, pk_{PEKSi}). $\mathcal{B}$ registers (ID_{Ri}, pk_{Ri}) to C as a corrupt user, makes reveal query of (ID_S, ID_{Ri}) to C and receives shk_i as response. Then $\mathcal{B}$ computes $C_i = PEKS.\mathsf{PEKS}(kw_i, pk_{PEKSi})$ and returns C_i to $\mathcal{A}$, where $kw_i = H(w_i, shk_i)$.
- Trapdoor-keyword queries. $\mathcal{A}$ supplies a keyword w_i and a sender's public key pk_{S_i}. $\mathcal{B}$ registers (ID_{Si}, pk_{Si}) to C as a corrupt user, makes reveal query of (ID_{Si}, ID_R) to C and receives shk_i as response. Then $\mathcal{B}$ computes $T_i = PEKS.\mathsf{Trapdoor}(kw_i, sk_{PEKSi})$ and returns T_i to $\mathcal{A}$, where $kw_i = H(w_i, shk_i)$.

Challenge. $\mathcal{A}$ supplies a pair of keywords (w_0^*, w_1^*) as its challenge. $\mathcal{B}$ first makes test queries on (ID_S, ID_R) to C and receives shk^* as response. Then it computes $kw^* = H(w_b^*, shk^*)$ and $C^* = PEKS.\mathsf{PEKS}(kw_i^*, pk_{PEKS})$, where $b \leftarrow\!\!\$\, \{0,1\}$ and $kw^* = H(w_b^*, shk^*)$. Finally, $\mathcal{B}$ returns C^* to $\mathcal{A}$.

Query phase 2. C answers $\mathcal{A}$'s queries in the same way as phase 1.

Guess phase. Eventually, $\mathcal{A}$ supplies b' as its guess to Game $\mathrm{G}^{\mathrm{CI}}_{\mathcal{E},\mathcal{A}}(1^\lambda)$. If $b = b'$, then $\mathcal{B}$ returns $\hat{b} = 0$ as guess to C; otherwise, $\mathcal{B}$ returns $\hat{b} = 1$ to C.

Similar to the proof in Lemma 1, it is easy to see that the response to $\mathcal{A}$ is perfectly simulated by $\mathcal{B}$. Therefore, there is no abort in the simulation.

Next, in the challenge phase, the challenge keyword can be divided into two cases through the different choices of $\mathcal{A}$. First, w_b^* is not queried in the query phase, then this game is identical to Game_0. Thus, we have $\mathsf{Adv}^{\mathrm{CI}}_{\mathcal{A}}(\lambda) = \mathsf{Adv}^{\mathrm{fCI}}_{\mathcal{A}}(\lambda)$ which is negligible. Second, w_b^* is queried before, the advantage $\mathsf{Adv}^{\mathrm{ss}}_{\mathcal{B}}(\lambda)$ is identical to the advantage $\mathsf{Adv}^{\mathrm{fCI}}_{\mathcal{A}}(\lambda)$. Since the probability of this case occurring is at least $q_{ck}/|\mathcal{KS}|$, we have $\mathsf{Adv}^{\mathrm{ss}}_{\mathcal{B}}(\lambda) \geq (q_{ck} \cdot \epsilon_1)/|\mathcal{KS}|$.

From the above discussions, we have $\mathcal{B}$ breaks the semantic security of the PEKS scheme in non-negligible advantage. This completes the proof.

Appendix D. Lattice-based instantiation

Due to the impossibility of (ring) LWE-based NIKE [41], we build the lattice-based instantiation from the one-round PAKE scheme in [42] and the PKEET scheme in [43]. Before describing the details of the instantiation, we first introduce some underlying notations.

- $\mathcal{R}$ and $\mathcal{U}$ are a ring and a subset of $\mathcal{R}^\times$ of invertible elements respectively.
- $\{G = I_n \otimes g^\top\} \in \mathbb{Z}_q^{n \times nk}$ is a gadget matrix [44], where $g = [1, 2, \ldots, 2^{k-1}] \in \mathbb{Z}_q^k$.
- $\mathsf{Encode}(\mu) = \mu \cdot (0, \ldots, 0, \left\lceil \frac{q}{2} \right\rceil)^\top$ is an encode function, where $\mu \in \{0,1\}$ and $\mathsf{Encode}(\mu) \in \mathbb{Z}_q^n$.
- $P(x) = \left\lfloor \frac{2x}{q} \right\rceil \mod 2$ is a deterministic rounding function [36].
- h is an injective ring homomorphism $h : \mathcal{R} \to \mathbb{Z}_q^{n \times m}$.

- $d_S \in \mathbb{Z}_q^t$ is a vector, where $d_{S,j} \in \mathbb{Z}_q$ denotes the jth element of it. And the same is true for d_R.

The algorithms of the lattice-based instantiation are detailed as follows:

- Setup(1^λ): given a security parameter λ, randomly choose an K-bit string $\mathsf{m} = \mathsf{m}_1\mathsf{m}_2 \ldots \mathsf{m}_K \leftarrow \{0,1\}^K$, $u \leftarrow \mathcal{U}$ and $V \in \mathbb{Z}_q^{n\times t}$. Select $n\times m$ matrices $A_1, A_2, \ldots, A_l$, $B \in \mathbb{Z}_q^{n\times m}$ and $A_0 \leftarrow \mathbb{Z}_q^{n\times m'}$. Set matrix $A = [A_0 | - A_0 Q]$, where $Q \leftarrow D_{\mathbb{Z},\sigma}^{m'\times m''}$ and $m = m' + m''$. Choose hash functions $H : \{0,1\}^* \to \{0,1\}^t$, $H_1 : \{0,1\}^* \to \{-1,1\}^l$ and output public parameters $\mathcal{PP} = (A, A_1, \ldots, A_l, B, V, H, H_1, \mathsf{m}, u)$.
- SkGen(ID_S): given identity ID_S, randomly choose $k_S \leftarrow D_{\mathbb{Z},\sigma}^m$, $s_{S,i} \leftarrow \mathbb{Z}_q^n$ and $e_{S,i} \leftarrow \overline{\Psi}_\alpha^m$. Compute vector $p_S = A_u \cdot k_S$ and $c_{S,i} = A_u^\top s_{S,i} + e_{S,i} + \mathsf{Encode}(\mathsf{m}_i) \mod q$, where matrix $A_u = [A_0| - A_0 Q + h(u)G] \in \mathbb{Z}_q^{n\times m}$. Output a key-pair as $(pk_S = (p_S, \{c_{S,i}\}_{i=1}^K), sk_S = (k_S, \{s_{S,i}\}_{i=1}^K))$.
- RkGen(ID_R): given identity ID_R, the RkGen algorithm is the same as SkGen. It outputs the key-pair as $(pk_R = (p_R, \{c_{R,i}\}_{i=1}^K), sk_R = (k_R, \{s_{R,i}\}_{i=1}^K))$.
- PAEKS(w, sk_S, pk_R): given keyword w, secret key sk_S and public key pk_R, proceed as follows:
  - For $i = 1, 2, \ldots, K$, compute $y_{S,i} = h_{S,i} \cdot p_{S,i}$, where $h_{S,i} \leftarrow P(c_{R,i}^\top \cdot k_S (\mathrm{mod}\ q))$ and $p_{S,i} \leftarrow P(s_{S,i}^\top \cdot p_R (\mathrm{mod}\ q))$. Set K-bit string $y_S = y_{S,1} y_{S,2} \ldots y_{S,K} \in \{0,1\}^K$ and compute the variant of keyword $kw = H(w, y_S)$. Randomly choose $s_1 \leftarrow \mathbb{Z}_q^n$, $x_1 \leftarrow \overline{\Psi}_\alpha^t$, and compute $C_1 = V^\top s_1 + x_1 + kw \left\lfloor \frac{q}{2} \right\rfloor \in \mathbb{Z}_q^t$.
  - Use the TrapGen algorithm to generate matrix A' with trapdoor $T_{A'}$ as $(A', T_{A'}) \leftarrow \mathsf{TrapGen}(q, n)$. Compute $F = (A'|B + \sum b_i A_i)$ and $J = \sum b_i J_i$, where $b = H_1(C_1) \in \{-1,1\}^l$ and $J_i \in \{-1,1\}^{m\times m}$. Randomly choose $y_1 \in \overline{\Psi}_\alpha^m$ and set $z_1 = J^\top y_1$. Compute $C_2 = F^\top s_1 + [y_1^\top | z_1^\top]^\top \in \mathbb{Z}_q^{2m}$. Finally, output $Ct = (C_1, C_2, C_3 = A', td_S = T_{A'})$.
- Trapdoor(w, sk_R, pk_S): the Trapdoor algorithm is the same as the PAEKS algorithm except the value of the input; we omit the details. It outputs trapdoor $Td = (T_1, T_2, T_3, td_R)$.
- Test(Ct, Td): given ciphertext Ct and trapdoor Td, proceed as follows:
  - Compute $e_S = \mathsf{SampleLeft}(C_3, B + \sum b_{S,i} A_i, td_S, V, \sigma)$ and $d_S = C_1 - e_S^\top \cdot C_2 \in \mathbb{Z}_q^t$, where $b_S = H_1(C_1) = (b_{S,1} \ldots b_{S,l})$.
  - Compute $e_R = \mathsf{SampleLeft}(T_3, B + \sum b_{R,i} A_i, td_R, V, \sigma)$ and $d_R = T_1 - e_R^\top \cdot T_2 \in \mathbb{Z}_q^t$, where $b_R = H_1(T_1) = (b_{R,1} \ldots b_{R,l})$.
  - For $j = 1, 2, \ldots, t$, compare $d_{S,j}$ with $\left\lfloor \frac{q}{2} \right\rfloor$. Output $h_{S,j} = 1$ if they are close; otherwise, output $h_{S,j} = 0$. Similarly, follow the same procedure to generate h_R according to d_R.
  - Output '1' if $h_S = h_R$ and '0' otherwise.

Correctness. If the one-round PAKE scheme and the PKEET scheme are correct, then our lattice-based instantiation is correct.

Theorem 6 (*Security*). *If the one-round PAKE scheme is secure and the PKEET scheme is W-IND-CCA2 secure, then the above PAEKS is CI-secure and TI-secure.*

Since our lattice-based instantiation is an implementation of the refined construction in StopGuess, its security follows the refine construction. The security of the refined construction relies on the security underlying PAKE and PKEET scheme. The procedure of reduction is similar to Theorems 3 and 4.

References

[1] D. Boneh, G. Di Crescenzo, R. Ostrovsky, G. Persiano, Public key encryption with keyword search, in: C. Cachin, J.L. Camenisch (Eds.), EUROCRYPT 2004, Springer, Berlin, Heidelberg, 2004, pp. 506–522.

[2] D.X. Song, D. Wagner, A. Perrig, Practical techniques for searches on encrypted data, in: Proceeding 2000 IEEE Symposium on Security and Privacy. S P 2000, 2000, pp. 44–55.

[3] J.W. Byun, H.S. Rhee, H.-A. Park, D.H. Lee, Off-line keyword guessing attacks on recent keyword search schemes over encrypted data, in: W. Jonker, M. Petković (Eds.), Secure Data Management, Springer, Berlin, Heidelberg, 2006, pp. 75–83.

[4] D. Boneh, A. Raghunathan, G. Segev, Function-private identity-based encryption: Hiding the function in functional encryption, in: R. Canetti, J.A. Garay (Eds.), CRYPTO 2013, in: LNCS, 8043, Springer, 2013, pp. 461–478.

[5] H.S. Rhee, J.H. Park, W. Susilo, D.H. Lee, Improved searchable public key encryption with designated tester, in: W. Li, W. Susilo, U.K. Tupakula, R. Safavi-Naini, V. Varadharajan (Eds.), ASIACCS 2009, ACM, 2009, pp. 376–379.

[6] R. Chen, Y. Mu, G. Yang, F. Guo, X. Huang, X. Wang, Y. Wang, Server-aided public key encryption with keyword search, IEEE Trans. Inf. Forensics Secur. 11 (12) (2016) 2833–2842.

[7] R. Chen, Y. Mu, G. Yang, F. Guo, X. Wang, Dual-server public-key encryption with keyword search for secure cloud storage, IEEE Trans. Inf. Forensics Secur. 11 (4) (2016) 789–798.

[8] P. Jiang, Y. Mu, F. Guo, Q.-Y. Wen, Private keyword-search for database systems against insider attacks, J. Comput. Sci. Tech. 32 (3) (2017) 599–617.

[9] J. Li, M. Wang, Y. Lu, Y. Zhang, H. Wang, ABKS-SKGA: Attribute-based keyword search secure against keyword guessing attack, Comput. Stand. Interfaces 74 (2021) 103471.

[10] Y. Miao, Q. Tong, R.H. Deng, K.-K.R. Choo, X. Liu, H. Li, Verifiable searchable encryption framework against insider keyword-guessing attack in cloud storage, IEEE Trans. Cloud Comput. 10 (2) (2022) 835–848.

[11] Q. Huang, H. Li, An efficient public-key searchable encryption scheme secure against inside keyword guessing attacks, Inform. Sci. 403–404 (2017) 1–14.

[12] M. Noroozi, Z. Eslami, Public key authenticated encryption with keyword search: Revisited, IET Inf. Secur. 13 (4) (2019) 336–342.

[13] B. Qin, Y. Chen, Q. Huang, X. Liu, D. Zheng, Public-key authenticated encryption with keyword search revisited: Security model and constructions, Inform. Sci. 516 (2020) 515–528.

[14] B. Qin, H. Cui, X. Zheng, D. Zheng, Improved security model for public-key authenticated encryption with keyword search, in: Q. Huang, Y. Yu (Eds.), ProvSec, Springer International Publishing, Cham, 2021, pp. 19–38.

[15] D. He, M. Ma, S. Zeadally, N. Kumar, K. Liang, Certificateless public key authenticated encryption with keyword search for industrial internet of things, IEEE Trans. Ind. Inform. 14 (8) (2018) 3618–3627.

[16] H. Li, Q. Huang, J. Shen, G. Yang, W. Susilo, Designated-server identity-based authenticated encryption with keyword search for encrypted emails, Inform. Sci. 481 (2019) 330–343.

[17] Q. Fan, D. He, J. Chen, C. Peng, L. Wang, Isoga: An isogeny-based quantum-resist searchable encryption scheme against keyword guessing attacks, IEEE Syst. J. (2022) 1–12.

[18] Z.-Y. Liu, Y.-F. Tseng, R. Tso, M. Mambo, Y.-C. 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, ASIA CCS '22, 2022, pp. 423–436.

[19] V.B. Chenam, S.T. Ali, A designated cloud server-based multi-user certificateless public key authenticated encryption with conjunctive keyword search against IKGA, Comput. Stand. Interfaces 81 (2022) 103603.

[20] W. Zhang, B. Qin, X. Dong, A. Tian, Public-key encryption with bidirectional keyword search and its application to encrypted emails, Comput. Stand. Interfaces 78 (2021) 103542.

[21] W. Li, L. Xu, Y. Wen, F. Zhang, Conjunctive multi-key searchable encryption with attribute-based access control for EHR systems, Comput. Stand. Interfaces 82 (2022) 103606.

[22] H. Wang, Y. Li, W. Susilo, D.H. Duong, F. Luo, A fast and flexible attribute-based searchable encryption scheme supporting multi-search mechanism in cloud computing, Comput. Stand. Interfaces 82 (2022) 103635.

[23] M. Miao, Y. Wang, J. Wang, X. Huang, Verifiable database supporting keyword searches with forward security, Comput. Stand. Interfaces 77 (2021) 103491.

[24] C. Ge, Z. Liu, J. Xia, L. Fang, Revocable identity-based broadcast proxy re-encryption for data sharing in clouds, IEEE Trans. Dependable Secure Comput. 18 (3) (2019) 1214–1226.

[25] C. Ge, W. Susilo, Z. Liu, J. Xia, P. Szalachowski, L. Fang, Secure keyword search and data sharing mechanism for cloud computing, IEEE Trans. Dependable Secure Comput. 18 (6) (2020) 2787–2800.

[26] C. Ge, W. Susilo, J. Baek, Z. Liu, J. Xia, L. Fang, Revocable attribute-based encryption with data integrity in clouds, IEEE Trans. Dependable Secure Comput. 19 (5) (2021) 2864–2872.

[27] C. Ge, Z. Liu, W. Susilo, L. Fang, H. Wang, Attribute-based encryption with reliable outsourced decryption in cloud computing using smart contract, IEEE Trans. Dependable Secure Comput. (2023).

[28] I.R. Jeong, J.O. Kwon, D. Hong, D.H. Lee, Constructing PEKS schemes secure against keyword guessing attacks is possible? Comput. Commun. 32 (2) (2009) 394–396.

[29] Z.-Y. Shao, B. Yang, On security against the server in designated tester public key encryption with keyword search, Inform. Process. Lett. 115 (12) (2015) 957–961.

[30] K. 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, APKC '22, 2022, pp. 39–49.
[31] X. Boyen, B. Waters, Anonymous hierarchical identity-based encryption (without random oracles), in: C. Dwork (Ed.), Advances in Cryptology - CRYPTO 2006, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 290–307.
[32] E.S.V. Freire, D. Hofheinz, E. Kiltz, K.G. Paterson, Non-interactive key exchange, in: K. Kurosawa, G. Hanaoka (Eds.), PKC 2013, in: LNCS, vol. 7778, Springer, 2013, pp. 254–271.
[33] G. Yang, C.H. Tan, Q. Huang, D.S. Wong, Probabilistic public key encryption with equality test, in: J. Pieprzyk (Ed.), CT-RSA 2010, in: LNCS, vol. 5985, Springer, 2010, pp. 119–131.
[34] W. Diffie, M. Hellman, New directions in cryptography, IEEE Trans. Inform. Theory 22 (6) (1976) 644–654.
[35] R. Sakai, K. Ohgishi, M. Kasahara, Cryptosystems based on pairings, in: The 2000 Symposium on Cryptography and Information Security, Vol. 45, Japan, 2000, pp. 26–28.
[36] J. Katz, V. Vaikuntanathan, Round-optimal password-based authenticated key exchange, in: Y. Ishai (Ed.), Theory of Cryptography, Springer, Berlin, Heidelberg, 2011, pp. 293–310.
[37] M. Bellare, A. Boldyreva, A. Desai, D. Pointcheval, Key-privacy in public-key encryption, in: C. Boyd (Ed.), ASIACRYPT 2001, Springer, Berlin, Heidelberg, 2001, pp. 566–582.
[38] B. Lynn, et al., The pairing-based cryptography library, 2006, Internet: crypto. stanford. edu/pbc/[Mar. 27, 2013]. https://crypto.stanford.edu/pbc/.
[39] Y. Chen, J. Zhang, D. Lin, Z. Zhang, Generic constructions of integrated PKE and PEKS, Des. Codes Cryptogr. 78 (2) (2016) 493–526.
[40] F. Banfi, U. Maurer, Anonymous symmetric-key communication, in: C. Galdi, V. Kolesnikov (Eds.), SCN 2020, in: LNCS, vol. 12238, Springer, 2020, pp. 471–491.
[41] S. Guo, P. Kamath, A. Rosen, K. Sotiraki, Limits on the efficiency of (ring) LWE-Based non-interactive key exchange, J. Cryptol. 35 (1) (2022) 1.
[42] Z. Li, D. Wang, Achieving one-round password-based authenticated key exchange over lattices, IEEE Trans. Serv. Comput. 15 (1) (2022) 308–321.
[43] D.H. Duong, K. Fukushima, S. Kiyomoto, P.S. Roy, W. Susilo, A lattice-based public key encryption with equality test in standard model, in: J. Jang-Jaccard, F. Guo (Eds.), ACISP 2019, in: LNCS, vol. 11547, Springer, 2019, pp. 138–155.
[44] D. Micciancio, C. Peikert, Trapdoors for lattices: Simpler, tighter, faster, smaller, in: D. Pointcheval, T. Johansson (Eds.), EUROCRYPT 2012, in: LNCS, vol. 7237, Springer, 2012, pp. 700–718.