Public-Key Authenticated Encryption With Keyword Search Supporting Constant Trapdoor Generation and Fast Search

Hongbo Li, Qiong Huang, Jianye Huang, and Willy Susilo, *Fellow, IEEE*

***Abstract*— To improve the quality of medical care and reduce unnecessary medical errors, electronic medical records (EMRs) are widely applied in hospital information systems. However, rapidly increasing EMRs bring heavy storage burden to hospitals. Professional data management service provided by cloud server can save the hospital local storage, and meanwhile, realize EMRs sharing among external researchers. However, the risk of leaking information of patients discourages hospitals to outsource patients' EMRs to the remote cloud server. In this paper, a secure and efficient cloud storing and sharing method can be achieved by applying the proposed public key authenticated encryption with ciphertext update and keyword search (PAUKS). The proposed PAUKS scheme enables EMRs to be encrypted and queried without decryption, and is secure against inside keyword guessing attacks. Compared with the recently proposed PAEKS in literature, the PAUKS scheme enjoys smaller computation and communication overheads. The required number of trapdoors per query is constant in PAUKS scheme, instead of the linearly expanding as the number of senders increases in PAEKS. Furthermore, an inverted index can be built safely in PAUKS scheme to accelerate the query procedure. Experiment results show that our PAUKS scheme owns a comparable running overhead, but enjoys a higher query efficiency after ciphertexts update.**

***Index Terms*— Searchable encryption, keyword guessing attacks, electronic medical record, light overhead, fast search.**

I. Introduction

ELECTRONIC medical record (EMR) plays an important role in the hospital information system, which managing information of patients including name, age, gender, allergic history, social security information, admission diagnosis, operation note, etc. EMR not only facilitates technical exchanges and medical case studies between patients, families and medical researchers, but also provides faster and more convenient services to patients, reduces the workload of medical staff, and reduces medical risks. However, the two problems in electronic EMR system, namely insufficient storage space and information security issues, arouse people's attention.

Although each electronic medical record takes up a small amount of storage space, the accumulated electronic medical records also take up a large amount of space due to the large number of patients. Especially in some famous hospitals, the number of patients is very large, and it is more difficult for hospitals to manage electronic medical records; On the other hand, once the hospital's network system is threatened by viruses, it is very likely to cause the network system to collapse or even cause the electronic medical records to be leaked, greatly threatening patients privacy.

To solve the above problems, an effective solution is to use sophisticated cloud computing and cloud storage technologies to ensure data consistency and loss prevention. Hospitals can outsource encrypted electronic medicals records to the cloud. The procedure of this solution is shown in Figure 1(a). At the beginning, the attending doctor encrypts patients' EMRs with the public key of the hospital data administrator and sends the encrypted EMRs to the administrator. The administrator checks the validity of the electronic medical records and outsources them to the cloud. The administrator is responsible for the data access control and grants relevant doctors or medical researchers access to the outsourced encrypted data. The drawback of this solution is that utilizing general public key encryption schemes cannot meet the demand of frequent queries from doctors and medical researchers.

Public key encryption with keyword search (PEKS) firstly proposed by Boneh et al. [1] probably satisfies the requirement. As shown in figure 1(b), a user is able to generate a trapdoor with a keyword and let the cloud server test whether there are ciphertexts matching with the trapdoor, i.e. ciphertexts embedded with the same keyword as the trapdoor. However, Boneh et al.'s PEKS scheme and a number of PEKS schemes were pointed out insecure under the special attacks named off-line keyword guessing attacks (KGAs) [2]. Once the keyword space is not large, a probabilistic polynomial time (PPT) adversary could detect which keyword is queried. Unfortunately, the keyword space in real world is small enough for a PPT adversary to launch KGAs.

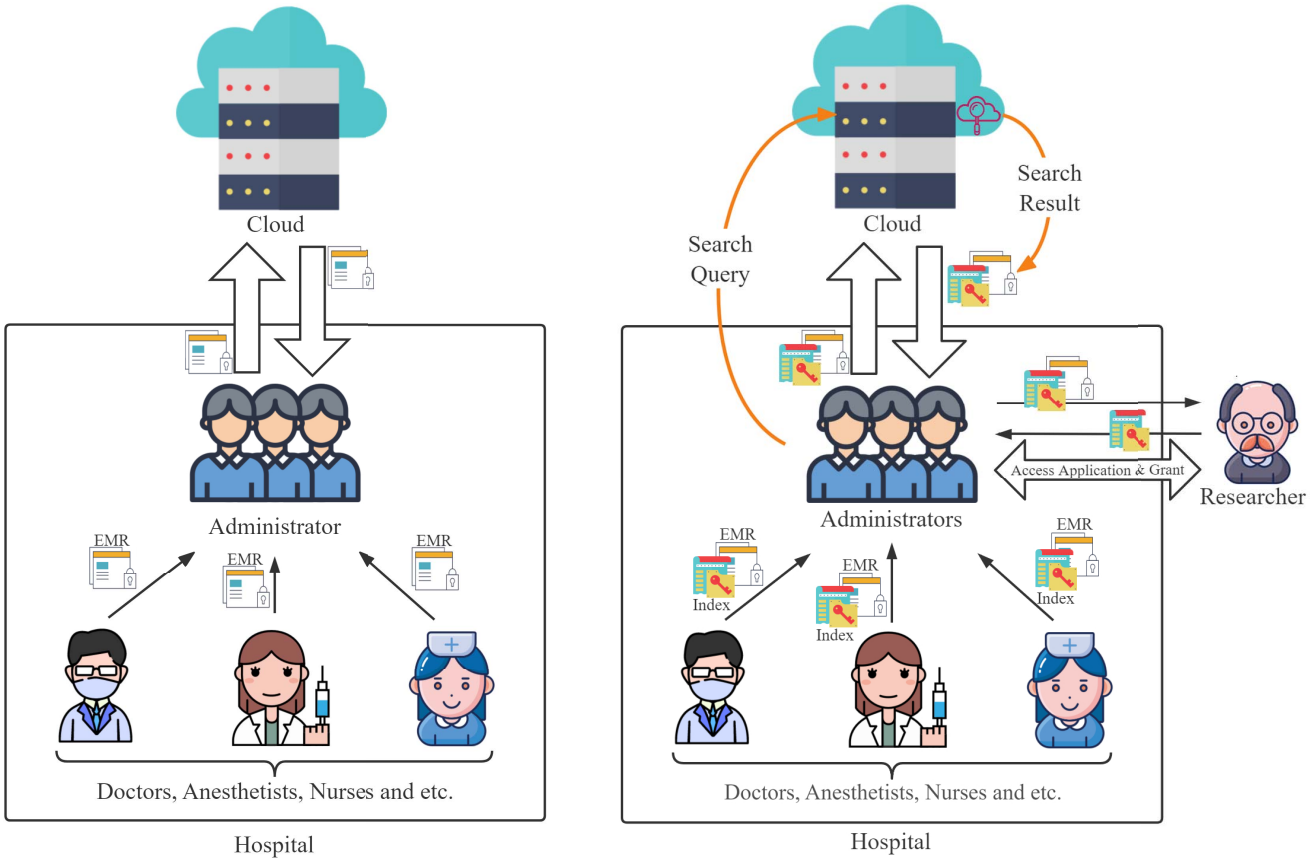
Manuscript received 5 March 2022; revised 11 September 2022; accepted 12 November 2022. Date of publication 23 November 2022; date of current version 7 December 2022. This work was supported in part by the Major Program of Guangdong Basic and Applied Research under Grant 2019B030302008, in part by the National Natural Science Foundation of China under Grant 61872152 and Grant 62272174, and in part by the Science and Technology Program of Guangzhou under Grant 201902010081. The associate editor coordinating the review of this manuscript and approving it for publication was Mr. Frederik Armknecht. *(Corresponding author: Qiong Huang.)*

Hongbo Li is with the College of Mathematics and Informatics, South China Agricultural University, Guangzhou 510642, China (e-mail: hongbo@scau.edu.cn).

Qiong Huang is with the College of Mathematics and Informatics and Guangzhou Key Laboratory of Intelligent Agriculture, South China Agricultural University, Guangzhou 510642, China (e-mail: qhuang@scau.edu.cn).

Jianye Huang and Willy Susilo are with the School of Computing and Information Technology, University of Wollongong, Wollongong, NSW 2522, Australia (e-mail: jh207@uow.edu.au; wsusilo@uow.edu.au).

Digital Object Identifier 10.1109/TIFS.2022.3224308



(a) System of Encrypted EMR System

(b) System of Encrypted EMR System with PEKS Index

Fig. 1. Electronic medical record system based on cloud service.

Dual-servers based PEKS schemes [3], [4], [5] were proposed to counter the KGAs, but limited by the application scenario and with higher cost of cloud service. Secure channel-free PEKS schemes [6] resist outside adversaries from launching KGAs, but are unable to intercept KGAs from inside adversaries, e.g. attacks from staff of the CSP. Public key authenticated encryption with keyword search (PAEKS) [7] could be a safer alternative of PEKS to counterattack KGAs regardless of whether from outside or inside PPT adversaries. A number of references [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], and [19] on PAEKS demonstrate its scalability, however, a main shortcoming of PAEKS remains unresolved. The number of required trapdoors is linearly expanded as the number of senders increases. With each query, the user needs to generate multiple trapdoors for the same keyword. In the EMR system, the more doctors there are, the more trapdoors are required, which hinders the application and has motivated our work.

A. Motivation, Challenge and Contributions

This work aims to build a provably secure and practically efficient searchable public key encryption scheme for cloud computing and cloud storage applications with privacy protection requirements, represented by the hospital EMR system.

1) Motivation: The public key authenticated encryption with keyword search (PAEKS) [7] satisfies the security requirements and counterattacks the off-line keyword guessing attacks launched by semi-trusted cloud server, but are still some drawbacks to be concerned.

- The spatial complexity of the required trapdoor for retrieval is linear with the number of senders, which is not practical in scenarios of multiple senders. To query a keyword, the receiver needs to generate multiple trapdoors, one per sender, in case there are multiple senders. It is thus not efficient and of heavy trapdoor communication overhead if the number of senders is large. Besides, it is complex to generate different trapdoors corresponding to different senders in each query and for the same keyword. Overall, it is important to implement constant-size trapdoor generation for PAEKS to reduce the communication overhead and management complexity.
- The searching time per query is linear with the number of encrypted keywords as well, which is not efficient if there are a large number of files to be encrypted and each file contains multiple keywords. Therefore, it is reasonable and necessary to further improve the retrieval efficiency.

PAEKS counterattacks the inside KGAs benefiting from the necessary of the sender's secret key during the encryption.

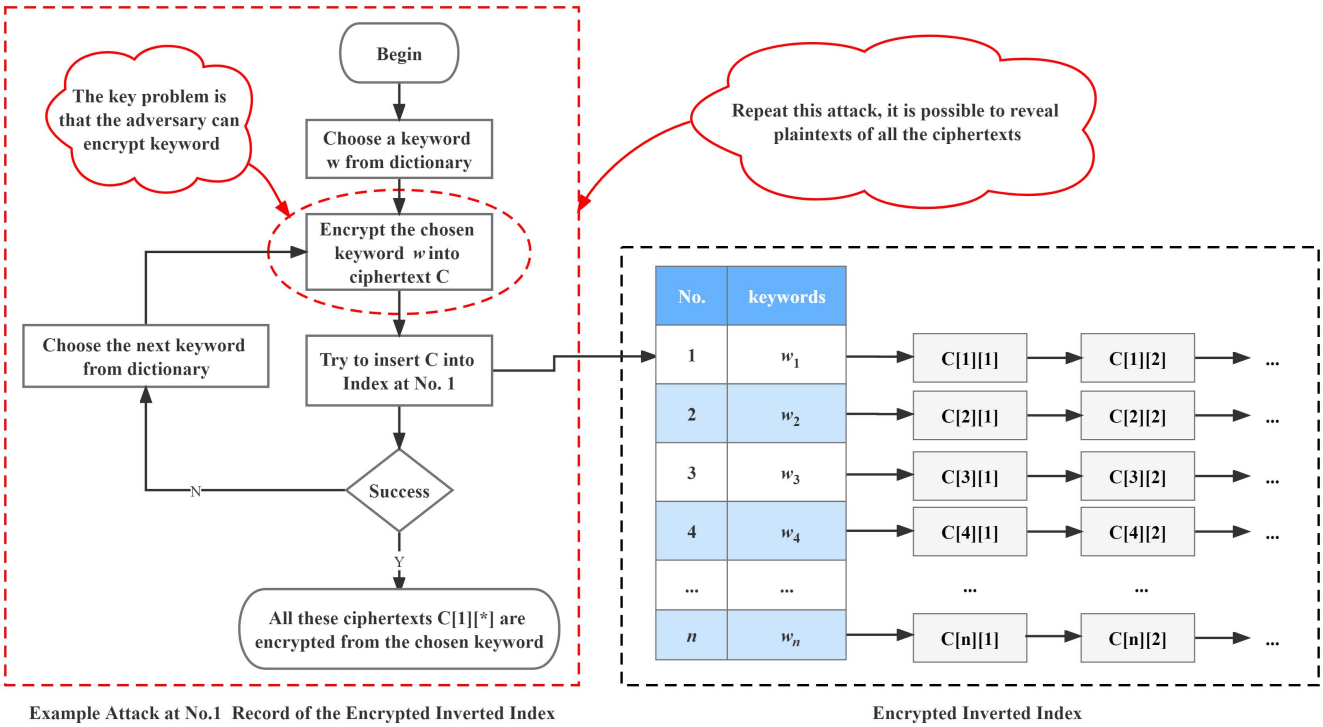


Fig. 2. Ciphertext inserting attacks on PEKS with inverted index.

However, it decreases the computation efficiency of trapdoor generation and increases the communication overhead of trapdoor transmission.

To the best of our knowledge, existing PAEKS schemes in literature fail to support constant size trapdoor generation for multiple senders. Han et al. [20] proposed a PAEKS scheme which supports logarithmic searching efficiency, but does not support constant size trapdoor generation. It is interesting and important to design a PAEKS scheme to support constant size trapdoor and sub-linearly fast retrieval, which motivates this work.

2) Challenge: In the original definition of PAEKS [7], the trapdoor generation algorithm takes as input both the receiver's secret key and the sender's public key. Thus, it is an inherent issue of PAEKS that the trapdoor size per query would linearly expand as the number of senders increases. The issue does not exist in those PEKS schemes which are vulnerable to inside KGAs, because neither encryption nor trapdoor generation include the sender's information in the algorithm, and the trapdoor is generated independently of the sender. On the other hand, although inverted index is often used in symmetric-key based secure data searching schemes, it is still challenging to achieve sub-linear searching efficiency in PEKS via applying inverted index.

- To reduce the communication overhead of trapdoor in PAEKS, it is natural to consider how to transform ciphertexts from different senders into a unified form so that the searching process does not need to consider the sender's information of a ciphertext. In the meanwhile, the transform should reserve the security of inside KGAs resistance.
- While applying the inverted index into PEKS, it would be vulnerable to the ciphertext inserting attacks, as illustrated in Figure 2.

Hence, it is not a trivial job to come up with a PAEKS scheme achieving the aforementioned goals.

3) Contributions: In this paper, we present an efficient solution to the above problem. The main idea is to add a non-collusion proxy to update the received ciphertexts and convert ciphertexts from different senders into a unified form. The contributions of this paper are summarized as follows.

- **PAUKS.** We introduce a new notion called *public key authenticated encryption with ciphertext update and keyword search* (PAUKS) in order to reduce the trapdoor communication overhead. The proposed PAUKS meets the needs of secure electronic medical record system of hospitals.
- **Security.** We give the threat model and the formally defined security model for PAUKS. The security defined in this paper requires a PAUKS scheme should counterattacks attacks launched by both the outside adversary and the inside adversary. Besides, the proxy in PAUKS is also considered as a threatening party and can be covered in the security model with the restriction that the proxy cannot collude with senders nor the cloud server.
- **Concrete Scheme.** We give a concrete PAUKS scheme based on the CDH assumption and a variant DLIN assumption. The proposed PAUKS scheme inherits features of PAEKS and meanwhile supports constant trapdoor communication overhead per query. We give formal security analysis of the proposed PAUKS scheme

shown in appendix and prove it to be secure in the proposed security model.

- **Efficiency.** It is worth noting that the proposed PAUKS scheme supports sub-liner searching efficiency, which is a rare property of PEKS or PAEKS schemes in the literature. The updated ciphertext can be securely sorted and inserted into an inverted index. searching based on inverted index will be much faster than searching directly over the whole encrypted-keyword space. Note that, leveraging inverted index to accelerate searching process is a natural method in scope of plaintext search, but challenging in PEKS or PAEKS, because it may face ciphertext inserting attacks. The proposed PAUKS scheme is proved secure with the inverted index.
- **Experimental Evaluation.** We evaluate the performance of our PAUKS scheme and HL-PAEKS scheme. The running efficiency of **Enc**, **Trapdoor** and **Test** algorithms of our PAUKS scheme are comparable with that of HL-PAEKS scheme. Differently, our PAUKS scheme enables a proxy to update the received ciphertexts. Before update, the running time of trapdoor generation per query is linear with the number of senders in both HL-PAEKS and our proposed PAUKS scheme. After update, the overhead of generating trapdoor will be decreased to constant which is no longer related to the number of senders. Experiments show that, when the number of senders is greater than a value (approximately 20), the running time of trapdoor generation before update is significantly greater than the time after update. Moreover, in our PAUKS scheme, the server can build a secure inverted index for fast search after update. With the inverted index, the search overhead is just linear with the size of keyword space and sub-linear with the total number of received searchable ciphertexts. Once the keyword space is much smaller than the received ciphertexts, the searching time based on inverted index will be significantly faster than the directly searching time.

B. Related Work

Boneh et al. [1] firstly introduced the notion of searchable encryption into the public key settings, and proposed the first public key encryption with keyword search (PEKS) scheme denoted by BDOP-PEKS. Byun et al. [2] pointed out the recent PEKS schemes were vulnerable against the proposed offline keyword guessing attacks (KGAs). Baek et al. [21] revisited the BDOP-PEKS scheme and gave a secure-channel free PEKS scheme which resists the outsider adversaries launching KGAs Xu et al. [22] proposed a fuzzy keyword search scheme based on PEKS and resists KGAs from outsid adversaries. However, the aforementioned schemes are still insecure under KGAs by insider adversaries.

Chen et al. [4] applied two server to counterattack the inside KGAs. In their scheme, the Keyword Server (KS) is separated from the Storage Server (SS). Based on the deterministic blind signature, Chen et al. [4] achieved the security against KS and SS. However, in their scheme, senders and receivers have to interact with the KS before storing ciphertexts or requesting searching results. Chen et al. [3] proposed another dual-server PEKS scheme which is secure against KGAs from both of the two servers. Based on a new proposed linear-and-homomorphic smooth projective hash function, Chen et al.'s scheme [3] lets a front server to preprocess the trapdoor and ciphertexts and deliver an internal test state to the back server. The back server finally returns the testing results to the receiver.

Huang and Li [7], based on PEKS, proposed a new primitive named public authenticated encryption with keyword search (PAEKS), which is secure against inside KGAs in the single-server setting. Noroozi and Eslami [23] pointed out their PAEKS scheme [7] fails to resist inside KGAs. Qin et al. [11] revisited the PAEKS scheme and proposed new security definitions, e.g. multi-ciphertext indistinguishability and multi-trapdoor privacy. They also provided a concrete scheme meeting the new definitions. Pan and Li [24] followed Qin et al.'s work [11] and proposed another PAEKS scheme. Chen et al.. [25] proposed a PAEKS scheme to apply in specific scenarios, which is based on dual servers and enjoys a higher security. Lu et al. [26] proposed a secure-channel-free PAEKS scheme to prevent outside adversaries from obtaining any information about the search pattern of users. Guo et al. [27] improved the security of secure-channel-free PAEKS scheme and achieved multi-ciphertext indistinguishability.

Li et al. [10] introduced PAKES into the identity-based cryptographic settings to resolve the complex public-key certification problem and proposed an identity-based authenticated encryption scheme with keyword search (IBAEKS). Liu et al. [28] proposed another IBAEKS scheme which enjoys a higher running efficiency. He et al. [8] introduced PAEKS into certificateless public key setting and proposed a CLPAEKS scheme, removing the barrier of complex key management. Liu et al. [29] reviewed He et al.'s CLPAEKS scheme and improved the security. Wu et al. [9] proposed an improved CLPAEKS scheme which only allows the designated server to do the test and enjoys a better running efficiency than the preceding CLPAEKS scheme. Yang et al. [16] and Lu et al. [30] proposed PAEKS schemes without using bilinear pairings, suitable for Industrial Internet of Things scenarios.

Behnia et al. [31] proposed an efficient lattice-based PEKS scheme to resist quantum computing attacks. However, it does not secure against inside KGAs. Liu et al. [32] recently proposed a PAEKS scheme based on lattice assumption to resist inside KGAs and quantum computing attacks.

Proxy re-encryption (PRE), introduced by Blaze et al. [33] in 1998, is an important technique used in our scheme. It allows a proxy to re-encrypt a ciphertext for receiver Alice into another ciphertext for receiver Bob without decryption. Shao et al. [34] introduced PRE into public key searchable encryption settings. Chen et al. [35] designed access control strategy for the PRE based PEKS scheme. Hoang et al. [36] leveraged a secure enclave on the cloud server to securely perform the proxy work and minimize the bottleneck of network. Hoang et al. [37] proposed a searchable based the secure enclave supporting oblivious search and update.

Xu et al. [38] applied the PRE based PEKS scheme into the EMR system. Deng et al. [39] achieved a secure-channel-free PEKS scheme supporting proxy re-encryption.

II. Preliminaries

Before introducing our proposed searchable encryption scheme, in this section, we some necessary preliminaries used in this paper.

A. Bilinear Pairing

Assume p is a large prime and $\mathbb{G}, \mathbb{G}_T$ are two groups with order p. Bilinear pairing [1] is a map $\hat{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ satisfying the following conditions:

- Bilinearity: $\forall x, y \in \mathbb{Z}_p, g \in \mathbb{G}, \hat{e}(g^x, g^y) = \hat{e}(g, g)^{xy}$;
- Non-degeneracy: $\forall g \neq 1, h \neq 1 \in \mathbb{G}, \hat{e}(g, h) \neq 1$;
- Computability: $\forall g, h \in \mathbb{G}$, $\hat{e}(g, h)$ is efficiently computable.

B. CDH Assumption

Let $\mathbb{G}$ be a group with a large prime order p. Given a CDH tuple (g, g^a, g^b), the CDH problem [40], [41] is to compute g^{ab}, where $g \in \mathbb{G}$ is a generator and $(a, b) \in \mathbb{Z}_p^2$ are randomly selected [40], [41].

CDH Assumption: The above CDH problem is intractable for any probabilistic polynomial time (PPT) adversary.

C. DLIN Assumption

Let $\hat{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ be a bilinear pairing. Given a tuple $(g, g^x, g^y, g^{x \cdot r}, g^{y \cdot s}, Z \in \mathbb{G})$, the DLIN problem [40], [41] is to distinguish whether $Z = g^{r+s}$ or is a random element of $\mathbb{G}$, where $g \in \mathbb{G}$ is a generator and $(x, y) \in \mathbb{Z}_p^2$ are randomly selected.

DLIN Assumption: The above DLIN problem is intractable for any PPT adversary.

D. A Variant of DLIN Assumption

Let $\hat{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ be a bilinear pairing. Given a modified DLIN tuple $(g, g^x, g^y, g^{x \cdot r}, g^{r+s}, Z \in \mathbb{G})$, the modified DLIN problem [40], [41] is to distinguish whether $Z = g^{y \cdot s}$ or not, where $g \in \mathbb{G}$ is a generator and $(x, y) \in \mathbb{Z}_p^2$ are randomly selected.

mDLIN Assumption: The above modified DLIN problem is intractable same as the DLIN problem.

III. The Public Key Authenticated Encryption With Ciphertext Update and Keyword Search

In this section, we propose a new cryptography encryption primitive named *public key authenticated encryption with ciphertext update and keyword search* (PAUKS) and introduce the framework, system model, threat model and security model of the proposed PAUKS.

A. PAUKS Framework

A basic PAUKS scheme contains ten algorithms as follows. The first six algorithms are inherited from the PAEKS scheme which guarantee any inside PPT adversary cannot successfully break the scheme via off-line keyword guessing attacks; The last four algorithms are designed to update ciphertexts and to support constant trapdoor communication overhead.

- **Setup**(1^λ): Given the security parameter 1^λ, this algorithm initializes the system and generates the public parameter $\mathbb{PP}$.
- **KeyGen$_\mathbf{R}$**$(\mathbb{PP})$: Given the public parameter $\mathbb{PP}$, this algorithm returns a receiver's public/secret key pair $(pk_\mathbf{R}, sk_\mathbf{R})$.
- **KeyGen$_\mathbf{S}$**$(\mathbb{PP})$: Given the public parameter $\mathbb{PP}$, this algorithm returns a sender's public/secret key pair $(pk_\mathbf{S}, sk_\mathbf{S})$.
- **Enc**$(\mathbb{PP}, sk_\mathbf{S}, pk_\mathbf{R}, w)$: Given the public parameter $\mathbb{PP}$, a sender's secret key $sk_\mathbf{S}$, a receiver's public key $pk_\mathbf{R}$ and a keyword w, this algorithm returns a PAEKS ciphertext C.
- **Trapdoor**$(\mathbb{PP}, sk_\mathbf{R}, pk_\mathbf{S}, w')$: Given the public parameter $\mathbb{PP}$, a receiver's secret key $sk_\mathbf{R}$, a sender's public key $pk_\mathbf{S}$ and a keyword w', this algorithm returns a trapdoor $T_{w'}$.
- **Test**$(\mathbb{PP}, pk_\mathbf{S}, C, T_w)$: Given the public parameter $\mathbb{PP}$, a sender's public key $pk_\mathbf{S}$, the ciphertext C and trapdoor T_w generated with $pk_\mathbf{S}$, this algorithm returns 1 if C and T_w are generated from the same keyword, 0, otherwise.
- **UpdKeyGen**$(\mathbb{PP}, sk_\mathbf{R}, pk_\mathbf{S})$: Given the public parameter $\mathbb{PP}$, the receiver's secret key $sk_\mathbf{R}$ and a candidate sender's public key $pk_\mathbf{S}$, this algorithm returns an updating key $uk_\mathbf{S}$ attached to the sender.
- **UpdEnc**$(\mathbb{PP}, C_\mathbf{S}, uk_\mathbf{S})$: Given the public parameter $\mathbb{PP}$, a ciphertext $C_\mathbf{S}$ sent from sender $\mathbf{S}$, and the updating key $uk_\mathbf{S}$ grant for $\mathbf{S}$, this algorithm updates $C_\mathbf{S}$ and returns an updated ciphertext $\hat{C}$.
- **ConstTrapdoor**$(\mathbb{PP}, sk_\mathbf{R}, w')$: Given the public parameter $\mathbb{PP}$, the receiver's secret key $sk_\mathbf{R}$ and a keyword w', this algorithm returns a constant trapdoor $\hat{T}_{w'}$.
- **UpdTest**$(\mathbb{PP}, \hat{C}, \hat{T}_{w'})$: Given the public parameter $\mathbb{PP}$, an updated ciphertext $\hat{C}$ and a constant $\hat{T}_{w'}$, this algorithm returns 1 if $\hat{C}$ and $\hat{T}_{w'}$ contain the same keyword, 0, otherwise.

B. System Model

The system model of the PAUKS, as shown in Figure 3, contains five types of parties: multiple data senders (Alice, Bob, Cindy, etc.), a data receiver (an administrator of a hospital), a proxy (a sub-administrator), a cloud server and an external data user (e.g. researchers).

1) Ability of Parties: We demonstrate the ability of each party of PAUKS as follows.

- **Data senders**: There are multiple senders, e.g. Alice, Bob, Cindy shown in Figure 3.
  - Each sender can encrypt keywords with the secret key of itself and the public key of the receiver.
  - ach sender is able to monitor the channel and capture ciphertexts sent from other senders and trapdoors submitted by the receiver.

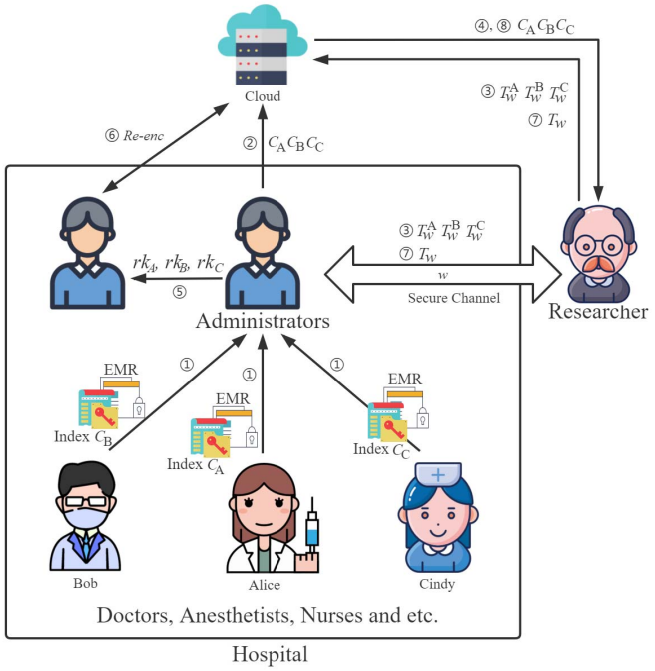


Fig. 3. System model of PAUKS.

- **Data receiver**: We consider only one receiver in PAUKS system for simplicity.
  - The receiver is the only user who can generate trapdoor using the secret key of itself and the public keys of senders.
  - The receiver in the proposed scenario could be an administrator of the hospital, who roles a fully trusted party.
- **Cloud server**: There is a cloud server in PAUKS system.
  - The server provides almost unlimited storage space and stores all the received ciphertexts.
  - The server will test whether there is any ciphertext matching with the submitted trapdoor.
  - The server is semi-trusted. It may try to reveal the information behind ciphertexts and trapdoors.
- **Proxy**: There is a proxy in PAUKS system to update the ciphertexts stored in the cloud.
  - The proxy could be a sub-administrator of the hospital and it help the administrator to update the ciphertexts.
  - The proxy could be semi-trusted. It means that the proxy will honestly perform the given task but may be curious about the plaintext information behind ciphertexts or trapdoors.
  - The restriction is that the proxy cannot collude with any sender nor the cloud server.
- **External user**: There could be one or more external users trying to utilize the EHR system to do medical research or data analysis. The external users are given limited authorization to access the data on most occasions.
  - The external user could be a researcher who would like to access some data.
  - The external user will be given a trapdoor containing the candidate keywords to search.
  - The external user can delegate the server to search files with the trapdoor and move on the next process, e.g. querying decryption keys of the files from the administrator, which is out of scope of this work.

2) System Flow: The PAUKS system runs after setup procedure supervised by a trusted certification authority (CA). All users' public keys are certificated and broadcasted by the CA. The system flow is shown as follows.

- At the beginning of the PAUKS system, a sender, e.g. Alice, encrypts patient EMRs with some kind of encryption scheme and encrypts index with the **Enc** algorithm of PAUKS. Alice concatenates the encrypted file and the encrypted keywords as a ciphertext, and finally sends the ciphertext to the administrator. Same with Alice, more senders send ciphertexts to the administrator (step ①). The administrator uploads ciphertexts to the cloud (step ②), to save local storage space.
- The administrator can generate trapdoors, e.g. T_w^A, T_w^B, T_w^C, with any candidate keyword w. Doctors in the hospital can query ciphertexts stored in cloud given trapdoors by the administrator. Researcher who was granted access of the ciphertexts can also obtain trapdoors from the administrator and query the matching ciphertexts stored in the cloud (step ③ ④). However, as shown in step ③, the overhead of generating trapdoors and the number of trapdoors per query per keyword is linear with the number of senders.
- The administrator can generate updating key for each sender (step ⑤) and delegate a sub-administrator to update the stored ciphertexts in the cloud (step ⑥). The update process will increase search efficiency and reduce communication overhead, meanwhile, it reserves the format of the original ciphertexts and still supports PAEKS-like test.
- After update, the administrator can generate constant size trapdoor, e.g. T_w, per query per keyword, and the server still returns the matching ciphertexts (step ⑦,⑧). The communication and computation overhead of search queries will be theoretically reduced.
- As an extension, after update, it is possible to classify the ciphertexts by the embedded keywords without decryption and against keyword guessing attacks. It means that an inverted index could be build to accelerate the search process.

C. Threat Model

Based on the system model of PAUKS, we analyse the threats from different adversaries.

1) Adversaries: The receiver is assumed honest naturally, and the external user with limited access permission can be considered as low risky. Three types of adversaries as follows are considered in the PAUKS.

- **Outside adversary**: it could be one of the senders trying to reveal plaintexts from trapdoors or ciphertexts encrypted by other senders.
- **Inside adversary**: it could be an adversary who can access in the inside network of the cloud storage service.

The inside adversary would try to reveal plaintexts from trapdoors, ciphertexts or updated ciphertexts.

- **Non-collusion proxy**: It could be a sub-administrator and should be a semi-trusted and non-collusion party. The proxy will honestly execute the proposed protocol but try to reveal information from ciphertexts or trapdoors; The proxy cannot collude with senders nor the cloud server, which is a reasonable as a sub-administrator in the real-life world.

2) Security Requirements: According to the threat model, the security requirements are as follows.

IKGA Resistance: We consider security against keyword guessing attacks launched by an inside adversary or the semi-trusted collusion-free proxy. Notice that security against KGA from an outside adversary could be implied by that from an inside adversary. Therefore, we do not discuss it in the rest of the paper.

- For any PPT adversary who does not have the knowledge of the secret keys and updating keys of the corresponding senders and receiver, it is of a negligible advantage to distinguish whether or not two ciphertexts (before update) contain the same keyword unless the matching trapdoors are given.
- For any PPT adversary who does not have knowledge of the updating keys and the receiver's secret key, the probability of revealing keyword from a trapdoor is negligible, even under the off-line keyword guessing attacks.

Non-collusion Proxy: Another security requirement is that the proxy with the updating keys cannot obtain more information than outside adversaries from the ciphertexts or trapdoors. That is, for any PPT adversary who does not have knowledge of the receiver's secret key, the probability of revealing the plaintext from the updated ciphertexts is negligible.

Remark: The secure enclave proposed by Hoang et al. [36] could be an alternative of a trusted proxy residing on the server side. To weaken the security assumption, it is enough to employ a semi-honest proxy who will not collude with the server and any sender for the proposed system. It is also possible to delegate the semi-honest cloud server as the proxy in the future work.

D. Security Model

Based on the threat model and security requirements, we define the security of PAUKS as follow.

Definition 1: A PAUKS scheme is semantically secure against the off-line keyword guessing attacks if for any PPT adversary $\mathcal{A}$, the advantage to win the following games is negligible.

Security Game 1 (Ciphertext-Indistinguishability Without Update, IND-CCA): In this game, the adversary $\mathcal{A}$ tries to distinguish ciphertexts without update.

- **Setup**: Given the security parameter 1^λ, return the public parameter $\mathbb{PP}$ and the public keys $pk_{\mathbf{S}}$, $pk_{\mathbf{R}}$ of a random sender and the receiver.
- **Phase 1**: Given the public parameter $\mathbb{PP}$, the adversary $\mathcal{A}$ can do the following queries.
  - *Ciphertext Query*: Given a keyword, return a corresponding PAUKS ciphertext encrypted with $sk_{\mathbf{S}}$ and $pk_{\mathbf{R}}$.
  - *Trapdoor Query*: Given a keyword, return a corresponding PAUKS trapdoor generated with $pk_{\mathbf{S}}$ and $sk_{\mathbf{R}}$.
- **Challenge**: $\mathcal{A}$ submits two keywords w_0^*, w_1^* which have not been queried for trapdoors. The simulator tosses a coin $b \leftarrow \{0, 1\}$, encrypts w_b^* with $sk_{\mathbf{S}}$ and $pk_{\mathbf{R}}$, and returns the ciphertext C_b^*.
- **Phase 2**: $\mathcal{A}$ can also do the queries same as in Phase 1, except that w_0^*, w_1^* could not appear in trapdoor queries.
- **Guess**: Finally, $\mathcal{A}$ returns a bit $b' \leftarrow \{0, 1\}$ and wins the game if $b' = b$. The advantage that $\mathcal{A}$ wins the game is defined as $Adv_{\mathcal{A}}^{C} = \left|\mathbf{Pr}[\mathcal{A}_{\text{win}}] - \frac{1}{2}\right| = \left|\mathbf{Pr}[b' = b] - \frac{1}{2}\right|$.

Security Game 2 (Updated-Ciphertext-Indistinguishability, IND-U-CCA): In this game, the adversary $\mathcal{A}$ tries to reveal plaintexts from the updated ciphertexts.

- **Setup**: Same as above.
- **Phase 1**: Given the public parameter $\mathbb{PP}$, the adversary $\mathcal{A}$ could issue ciphertext and trapdoor queries as in the above game.
- **Challenge**: $\mathcal{A}$ submits two keywords w_0^*, w_1^* which have not been queried for trapdoors. The simulator tosses a coin $b \leftarrow \{0, 1\}$, encrypts w_b^* with $sk_{\mathbf{S}}$ and $pk_{\mathbf{R}}$ to get a ciphertext C_b^*. With the updating key $uk_{\mathbf{S}}$, the simulator updates the ciphertext C_b^*, and returns the updated ciphertext $\hat{C}_b^*$.
- **Phase 2**: $\mathcal{A}$ continues to query as in Phase 1, except that w_0^*, w_1^* cannot appear in trapdoor queries.
- **Guess**: Finally, $\mathcal{A}$ returns a bit $b' \leftarrow \{0, 1\}$ and wins the game if $b' = b$. The advantage that $\mathcal{A}$ wins the game is defined as $Adv_{\mathcal{A}}^{\hat{C}} = \mathbf{Pr}[\mathcal{A}_{\text{win}}] - \frac{1}{2} = \mathbf{Pr}[b' = b] - \frac{1}{2}$.

Security Game 3 (Trapdoor Privacy, IND-TP-CCA): In this game, the adversary $\mathcal{A}$ tries to reveal plaintexts from the trapdoors (both with and without update).

- **Setup**: Same as above.
- **Phase 1**: Given the public parameter $\mathbb{PP}$, the adversary $\mathcal{A}$ can do the ciphertext and trapdoor queries same the above game. Differently, the returned ciphertexts have been updated.
- **Challenge**: $\mathcal{A}$ submits two keywords w_0^*, w_1^* which have not been queried for ciphertexts nor trapdoors. The simulator tosses a coin $b \leftarrow \{0, 1\}$, returns two trapdoors $T_{w_b^*}, \hat{T}_{w_b^*}$.
- **Phase 2**: $\mathcal{A}$ continues to query as in Phase 1, except that w_0^*, w_1^* could not appear in both ciphertext and trapdoor queries.
- **Guess**: Finally, $\mathcal{A}$ returns a bit $b' \leftarrow \{0, 1\}$ and wins the game if $b' = b$. The advantage that $\mathcal{A}$ wins the game is defined as $Adv_{\mathcal{A}}^{T} = \left|\mathbf{Pr}[\mathcal{A}_{\text{win}}] - \frac{1}{2}\right| = \left|\mathbf{Pr}[b' = b] - \frac{1}{2}\right|$.

IV. Our PAUKS Scheme

- **Setup**(1^λ) : Given the security parameter 1^λ, output the public parameter $\mathbb{PP} = \{p, g, \mathbb{G}, \mathbb{G}_T, \hat{e}, H, H_1, H_2, H_3, H_4\}$ shown as follows:
  - p is a large prime,
  - $\mathbb{G}, \mathbb{G}_T$ are groups with order p,

- $\hat{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a bilinear pairing,
- H, H_1, H_2, H_3, H_4 are hash functions:
  - $H : \mathbb{G} \times \mathbb{G} \times \mathbb{G} \times \mathbb{G} \to \mathbb{G}$,
  - $H_1 : \mathbb{G} \times \{0,1\}^* \to \mathbb{Z}_p$,
  - $H_2 : \mathbb{G} \to \mathbb{Z}_p$,
  - $H_3 : \mathbb{G} \to \mathbb{Z}_p$,
  - $H_4 : \{0,1\}^* \to \mathbb{G}$.

- **KeyGen$_\mathbf{R}$**($\mathbb{PP}$) : Given $\mathbb{PP}$, the algorithm randomly selects x_1, x_2, x_3, x_4 from $\mathbb{Z}_p$ and outputs the receiver's public key $pk_\mathbf{R} = (pk_{\mathbf{R_1}}, pk_{\mathbf{R_2}}, pk_{\mathbf{R_3}})$ and secret key $sk_\mathbf{R} = (sk_{\mathbf{R_1}}, sk_{\mathbf{R_2}}, sk_{\mathbf{R_3}}, sk_{\mathbf{R_4}})$ as follows:
  - $pk_{\mathbf{R_1}} = g^{x_1}$, $pk_{\mathbf{R_2}} = g^{x_2}$, $pk_{\mathbf{R_3}} = g^{x_3}$,
  - $sk_{\mathbf{R_1}} = x_1, sk_{\mathbf{R_2}} = x_2, sk_{\mathbf{R_3}} = x_3, sk_{\mathbf{R_4}} = x_4$.
- **KeyGen$_\mathbf{S}$**($\mathbb{PP}$) : Given the public parameter $\mathbb{PP}$, the algorithm randomly selects $y \xleftarrow{\$} \mathbb{Z}_p$ and outputs the sender's public key $pk_\mathbf{S} = g^y$ and secret key $sk_\mathbf{S} = y$.
- **Enc**($\mathbb{PP}, sk_\mathbf{S}, pk_\mathbf{R}, w$) : Given the public parameter $\mathbb{PP}$, a sender's secret key $sk_\mathbf{S}$ and the receiver's public key $pk_\mathbf{R}$, the algorithm randomly selects $r_1, r_2 \leftarrow \mathbb{Z}_p$ and outputs the ciphertext $C = (C_1, C_2, C_3, C_4)$ as follows:

$$C_1 = \left(pk_{\mathbf{R_2}}^{H_1(pk_{\mathbf{R_1}}^{sk_\mathbf{S}}, w)} \cdot pk_{\mathbf{R_3}} \right)^{r_1}, \quad C_2 = g^{r_1},$$
$$C_3 = \left(pk_{\mathbf{R_2}}^{H_2(pk_{\mathbf{R_1}}^{sk_\mathbf{S}})} \cdot pk_{\mathbf{R_3}} \right)^{r_2} \cdot g^{H_3(pk_{\mathbf{R_1}}^{sk_\mathbf{S}}) \cdot r_1},$$
$$C_4 = H_4(w)^{r_2}, \quad C_5 = H(C_1, C_2, C_3, C_4)^{r_1}.$$

- **Trapdoor**($\mathbb{PP}, sk_\mathbf{R}, pk_\mathbf{S}, w'$) : Given the public parameter $\mathbb{PP}$, the receiver's secret key $sk_\mathbf{R}$ and a sender's public key $pk_\mathbf{S}$, the algorithm randomly selects $r_3 \leftarrow \mathbb{Z}_p$ and returns a trapdoor $T_w = (T_{w,1}, T_{w,2})$ as follows:

$$T_{w,1} = g^{\frac{r_3}{sk_{\mathbf{R_2}} \cdot H_1(pk_\mathbf{S}^{sk_{\mathbf{R_1}}}, w') + sk_{\mathbf{R_3}}}}, \quad T_{w,2} = g^{r_3}.$$

- **Test**($\mathbb{PP}, C, T_w$) : Given the public parameter $\mathbb{PP}$, a ciphertext C and a trapdoor T_w, the algorithm returns 1 if the following equation holds, otherwise, returns 0:

$$\hat{e}(C_1, T_{w,1}) = \hat{e}(C_2, T_{w,2}).$$

- **UpdKeyGen**($\mathbb{PP}, sk_\mathbf{R}, pk_\mathbf{S}$) : Given $\mathbb{PP}$, the receiver's secret key $sk_\mathbf{R}$ and a sender's public key $pk_\mathbf{S}$, the algorithm parses $sk_\mathbf{R}$ as $(sk_{\mathbf{R_1}}, sk_{\mathbf{R_2}}, sk_{\mathbf{R_3}}, sk_{\mathbf{R_4}})$ and generates an updating key $uk_\mathbf{S} = (uk_{\mathbf{S_1}}, uk_{\mathbf{S_2}})$ as follows:

$$uk_{\mathbf{S},1} = H_3(pk_\mathbf{S}^{sk_{\mathbf{R_1}}}),$$
$$uk_{\mathbf{S},2} = \frac{sk_{\mathbf{R_4}}}{sk_{\mathbf{R_2}} \cdot H_2(pk_\mathbf{S}^{sk_{\mathbf{R_1}}}) + sk_{\mathbf{R_3}}}.$$

- **UpdEnc**($\mathbb{PP}, C, uk_\mathbf{S}$) : Given $\mathbb{PP}$, a ciphertext C sent from sender $\mathbf{S}$ and the updating key $uk_\mathbf{S}$ assigned for the sender $\mathbf{S}$, the algorithm parses C as $(C_1, C_2, C_3, C_4, C_5)$ and returns $\perp$ if the following equation (1) does not hold, which means the ciphertext was tampered by adversaries.

$$\hat{e}(H(C_1, C_2, C_3, C_4), C_2) = \hat{e}(C_5, g). \quad (1)$$

Otherwise, this algorithm returns an updated ciphertext $\hat{C} = (C, C_6)$, where

$$C_6 = (C_3 / C_2^{uk_{\mathbf{S_1}}})^{uk_{\mathbf{S_2}}} = g^{r_2 \cdot sk_{\mathbf{R_4}}}.$$

No.	LabelCipher	Pointer
Null	Null	Null

Fig. 4. Header $\mathcal{H}$ of the empty index $\mathcal{I}$.

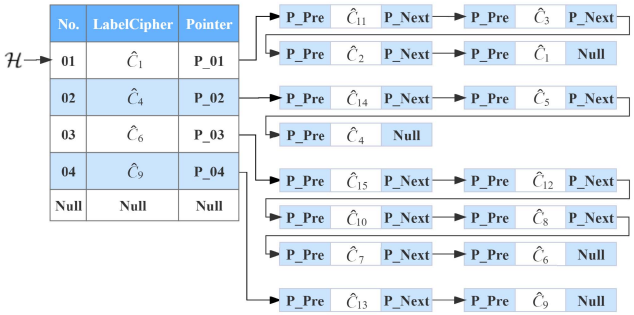


Fig. 5. Encrypted fast searching index.

- **ConstTrapdoor**($\mathbb{PP}, sk_\mathbf{R}, w'$) : Given $\mathbb{PP}$, the receiver's secret key $sk_\mathbf{R}$ and a keyword w', the algorithm randomly selects $r \leftarrow \mathbb{Z}_p$ and returns a constant trapdoor $\hat{T}_{w'} = (\hat{T}_{w',1}, \hat{T}_{w',2})$ applicable to matching ciphertexts sent from different senders, where

$$\hat{T}_{w',1} = g^{sk_{\mathbf{R_4}} \cdot r}, \quad \hat{T}_{w',2} = H_4(w')^r.$$

- **UpdTest**($\mathbb{PP}, \hat{C}, \hat{T}_{w'}$) : Given $\mathbb{PP}$, an updated ciphertext $\hat{C}$ and a constant trapdoor $\hat{T}_{w'}$, the algorithm returns 1 if the following equation holds, otherwise, returns 0:

$$\hat{e}(C_4, \hat{T}_{w',1}) = \hat{e}(C_6, \hat{T}_{w',2}).$$

A. Fast Searching Index for PAUKS

The updated ciphertexts can be inserted into an encrypted inverted index without decryption.

- **EqualityTest**($\mathbb{PP}, \hat{C}_1, \hat{C}_2$): Given $\mathbb{PP}$, and two updated ciphertexts $\hat{C}_1, \hat{C}_2$, this algorithm parses $\hat{C}_1, \hat{C}_2$ as $(C_{1,\ldots,}^{(1)} C_6^{(1)})$ and $(C_{1,\ldots,}^{(2)} C_6^{(2)})$, and returns 1 if the following equation holds, and 0 otherwise:

$$\hat{e}(C_4^{(1)}, C_6^{(2)}) = \hat{e}(C_4^{(2)}, C_6^{(1)}).$$

- **InitIndex**($\mathbb{PP}$): Given $\mathbb{PP}$, this algorithm initializes an empty index $\mathcal{I}$ with header

$$\mathcal{H} = \langle \text{No.}, \text{LabelCipher}, \text{Pointer} \rangle,$$

illustrated in Figure 4.
- **InsertIndex**($\mathbb{PP}, \hat{C}, \mathcal{H}$): Given $\mathbb{PP}$, an updated ciphertext $\hat{C}$ and the header $\mathcal{H}$ of the fast searching index, this algorithm inserts $\hat{C}$ into the index illustrated in Algorithm 1. After several rounds of insertion, the index $\mathcal{I}$ can be exampled in Figure 5.
- **FastSearch**($\mathbb{PP}, \mathcal{H}, \hat{T}_{w'}$): Given $\mathbb{PP}$, the header $\mathcal{H}$ of index $\mathcal{I}$ and a constant trapdoor $\hat{T}_{w'}$, this algorithm returns the a pointer (Pointer) if the ciphertexts pointed by the Pointer contain the same keyword as $\hat{T}_{w'}$, and $\perp$ otherwise. The detail algorithm is shown in Algorithm 2.

Algorithm 1 InsertIndex

Input: $\mathbb{PP}, \hat{C}, \mathcal{H} = \langle$No., LabelCipher, Pointer$\rangle$
Output: $\mathcal{H}$

ptr $= \mathcal{H}$;
isEqual $= 0$;
while ptr $\neq$ Null && isEqual $\neq 0$ **do**
  isEqual $=$ **EqualityTest**($\mathbb{PP}$, ptr.LabelCipher, $\hat{C}$);
  if isEqual **then**
    insert $\hat{C}$ *into* ptr.Pointer;
    return $\mathcal{H}$;
  end if
  ptr $=$ ptr.NextNode;
end while
if !isEqual **then**
  insert $\hat{C}$ *into* ptr.Pointer;
end if
return $\mathcal{H}$;

Algorithm 2 FastSearch

Input: $\mathbb{PP}, \hat{T}_{w'}, \mathcal{H} = \langle$No., LabelCipher, Pointer$\rangle$
Output: Pointer or $\perp$

ptr $= \mathcal{H}$;
isMatch $= 0$;
while ptr $\neq$ Null && isMatch $\neq 0$ **do**
  isMatch $=$ **UpdTest**($\mathbb{PP}$, ptr.LabelCipher, $\hat{T}_{w'}$);
  if isMatch **then**
    return ptr.Pointer.;
  end if
  ptr $=$ ptr.NextNode;
end while
if !isMatch **then**
  return $\perp$;
end if

B. Correctness of the PAUKS Scheme

Assuming the hash function is collision resistant, the **Test** algorithm returns 1 if and only if the hash inputs are the same. The correctness can be proved by the following equations:

$$\begin{aligned}&\hat{e}(C_1, T_{w,1})\\ &= \hat{e}\left((pk_{\mathbf{R_2}}^{H_1(pk_{\mathbf{R_1}}^{sk_{\mathbf{S}}}, w)} \cdot pk_{\mathbf{R_3}})^{r_1}, g^{\frac{r_2}{sk_{\mathbf{R_2}} \cdot H_1(pk_s^{sk_{\mathbf{R_1}}}, w') + sk_{\mathbf{R_3}}}}\right)\\ &\overset{\text{iff}(pk_{\mathbf{R_1}}^{sk_{\mathbf{S}}}, w) = (pk_s^{sk_{\mathbf{R_1}}}, w')}{=\!=\!=\!=\!=\!=\!=\!=\!=\!=\!=\!=\!=} \hat{e}(g^{r_1}, g^{r_2}) = \hat{e}(C_2, T_{w,2}).\end{aligned}$$

Parse $\hat{T}_{w',1}, \hat{T}_{w',2}$ and C_4, C_6 as follows:

$$\begin{aligned}&\hat{T}_{w',1} = g^{sk_{\mathbf{R_4}} \cdot r}, \quad \hat{T}_{w',2} = H_2(w)^r, \quad C_4 = H_4(w)^{r_2},\\ &C_6 = \left(pk_{\mathbf{R_2}}^{H_2(pk_{\mathbf{R_1}}^{sk_{\mathbf{S}}})} \cdot pk_{\mathbf{R_3}}\right)^{\frac{r_2 \cdot sk_{\mathbf{R_4}}}{sk_{\mathbf{R_2}} \cdot H_2(pk_{\mathbf{S}}^{sk_{\mathbf{R_1}}}) + sk_{\mathbf{R_3}}}}\\ &\quad = g^{r_2 \cdot sk_{\mathbf{R_4}}}.\end{aligned}$$

The **UpdTest** algorithm returns 1 if and only if the keyword embedded in the constant trapdoor is identical to that in the updated encrypted ciphertext, which can be proved by the following equations:

$$\begin{aligned}\hat{e}(C_4, \hat{T}_{w',1}) &= \hat{e}(H_4(w)^{r_2}, g^{sk_{\mathbf{R_4}}})\\ &\overset{\text{iff}(w'=w)}{=\!=\!=\!=\!=\!=\!=\!=} \hat{e}(g^{r_2 \cdot sk_{\mathbf{R_4}}}, H_4(w')^r)\\ &= \hat{e}(C_6, \hat{T}_{w,2}).\end{aligned}$$

TABLE I
COMMUNICATION OVERHEAD EVALUATION

Schemes	C	T_w	$\hat{C}$	$\hat{T}_w$
HL-PAEKS [7]	$2\|\mathbb{G}\|$	$m \cdot \|\mathbb{G}_T\|$	—	—
Our PAUKS	$5\|\mathbb{G}\|$	$m \cdot (2\|\mathbb{G}\|)$	$6\|\mathbb{G}\|$	$2\|\mathbb{G}\|$

$|\mathbb{G}|$: the size of an element in group $\mathbb{G}$.
$|\mathbb{G}_T|$: the size of an element in group $\mathbb{G}_T$.
m: the number of senders.

C. Security of the PAUKS Scheme

Theorem 1 (IKGA): The proposed PAUKS scheme is semantically secure against off-line keyword guessing attacks from any PPT adversary in the random oracle model if the CDH and mDLIN assumptions hold.

Similar to the HL-PAEKS scheme [7], the proposed PAUKS scheme prevents the adversary from encrypting keywords on behalf of other senders, which counterattacks the inside keyword guessing attacks. After update, the ciphertexts can only be tested with the constant trapdoor generated by the receiver, and the original ciphertexts without update cannot be tested with the constant trapdoor. Theorem 1 follows from the lemma 1, 2, and 3, which are given the formal proof in Appendix A.

V. PERFORMANCE EVALUATION

We evaluate the communication and computation overhead of our PAUKS scheme compared with HL-PAEKS scheme [7], as shown in Table I and Table II. Before ciphertext update, the proposed PAUKS scheme has a higher but still comparable communication overhead in terms of the ciphertext and the trapdoor. However, after ciphertext update, the trapdoor communication overhead of our PAUKS scheme reduces to constant complexity and is significantly lower in comparison with [7] when the number of senders increases. On the other hand, although our PAUKS scheme has slightly larger computation overhead than HL-PAEKS scheme in terms of **Enc**, **Trapdoor** and **Test** algorithms, it enjoys a lower computation complexity when using **ConstTrapdoor** and **UpdTest** algorithms to realize ciphertext retrieval.

We perform experiments on a desktop computer with an Intel Core i7-6700 CPU (3.40 GHz), 8GB Memory. The operating system is the Ubuntu 20.04.3 LTS. We instantiated our scheme using C programming language with GNU Multiple Precision Arithmetic (GMP) library and Pairing-Based Cryptography (PBC) library. The initial HL-PAEKS scheme [7] was instantiated for comparison. The results are shown in Table III, Table IV, Table V, Figure 6 and Figure 7.

TABLE II
COMPUTATION OVERHEAD EVALUATION

Schemes	**Enc**	**Trapdoor**	**Test**	**UpdEnc**	**ConstTrapdoor**	**UpdTest**
HL-PAEKS [7]	$3\text{Exp} + \text{Hash}$	$m \cdot (\text{Exp} + \text{Pair} + \text{Hash})$	$n \cdot (2\text{Exp} + \text{Mul})$	—	—	—
Our PAUKS	$9\text{Exp} + 5\text{Hash}$ $+2\text{Mul}$	$m \cdot (3\text{Exp} + \text{Mul}$ $+\text{Div} + \text{Add})$	$2n \cdot \text{Exp}$	$n \cdot (2\text{Pair} + \text{Hash}$ $+2\text{Exp} + \text{Div})$	$2\text{Exp} + \text{Hash} + \text{Mul}$	2Pair

Exp: computation overhead of an exponential operation. Pair: computation overhead of a bilinear pairing.
Hash: computation overhead of a hash operation. Mul: computation overhead of a multiplication operation.
Div: computation overhead of a division operation. Add: computation overhead of an addition operation.
n: the number of received ciphertexts. m: the number of senders.

TABLE III
STORAGE COST OF KEYS, CIPHERTEXTS, AND TRAPDOORS

Schemes	Public Keys (Bytes)		Secret Keys (Bytes)		UpdKey	Ciphertexts (Bytes)		Trapdoors (Bytes)	
	Sender	Receiver	Sender	Receiver	(Bytes)	**Enc**	**UpdEnc**	**Trapdoor**	**ConstTrapdoor**
HL-PAEKS [7]	128	128	20	20	—	256	—	128	—
Our PAUKS	128	384	20	80	40	640	768	256	256

TABLE IV
AVERAGE RUNNING TIME OF SETUP AND KEY GENERATION

Schemes	Setup Time (ms)	KeyGen Time (ms)		
		Sender	Receiver	UpdKeyGen
HL-PAEKS [7]	4.546	1.243	1.153	—
Our PAUKS	4.546	1.117	3.457	1.025

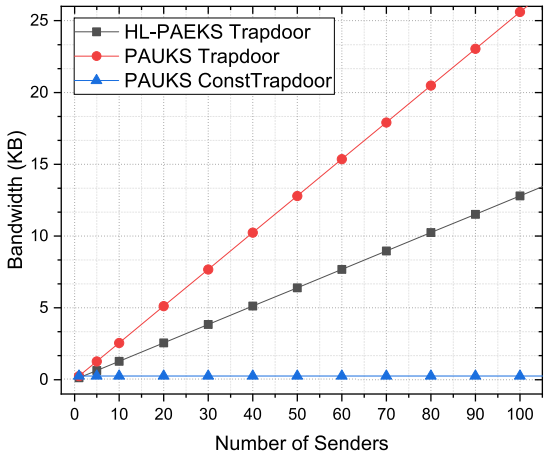


Fig. 6. Trapdoor bandwidth cost comparison.

Table III shows the storage cost of each key, ciphertext and trapdoor. The storage cost of the sender's keys in both HL-PAEKS and our PAUKS are the same. Due to extensional functions, our PAUKS scheme needs larger storage space than HL-PAEKS scheme in terms of the receiver's keys, ciphertexts and trapdoors. However, as shown in Figure 6, our PAUKS scheme supports constant trapdoor transmission after ciphertext update and needs much less bandwidth when the number of senders is large.

Table IV shows the running time of system initialization. The **Setup** algorithms are the same in both HL-PAEKS and our PAUKS, thus the running time are the same. The key generation algorithms for senders and receivers in HL-PAEKS scheme and for senders in our PAUKS scheme enjoy similar overhead; The key generation algorithm for receivers in our PAUKS scheme approximately runs three times as long. The **ReKeyGen** algorithm in our PAUKS scheme is as efficient as the key generation algorithm in HL-PAEKS scheme. Overall, compared with HL-PAEKS scheme, the system initialization overhead of our PAUKS scheme is higher, fortunately, each algorithm only needs to be run once and the total running time is negligible for users.

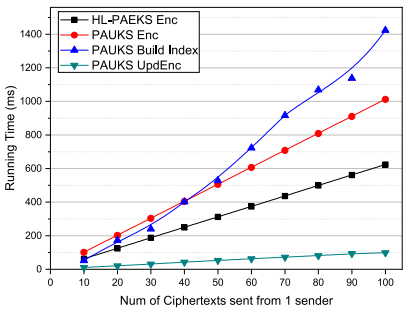
Table V shows the average running time of **Enc**, **Trapdoor**, **Test**, **UpdEnc** and **ConstTrapdoor** algorithms in the aforementioned two schemes. The data related to **UpdEnc** and **ConstTrapdoor** of HL-PAEKS is empty due that there is no such algorithms in HL-PAEKS scheme. In the comparison of **Enc** algorithms, the HL-PAEKS scheme enjoys a higher operation efficiency; In the comparison of **Trapdoor** algorithms, our PAUKS scheme leads a slight advantage. In the comparison of **Test** algorithms, the two schemes are almost tied. In our PAUKS scheme, the average running time of **UpdEnc** and **ConstTrapdoor** algorithms are close to that of **Enc** and **Trapdoor** algorithms, respectively.

From the experiment results shown in Figure 7(a)(b), the overhead of running **Enc** and **Trapdoor** algorithms are comparable between HL-PAEKS scheme and our PAUKS scheme. The overhead of the **Enc** algorithm in our PAUKS scheme is higher than but comparable with that in HL-PAEKS scheme because of longer ciphertext length and more functionality. The newly added algorithm **UpdEnc** enjoys a high running efficiency compared with the **Enc** algorithms. The PAUKS Build Index in Figure 7(a) shows the total overhead of building fast searching index and inserting index procedure, which is a bit higher than the **Enc** algorithms but reasonable.

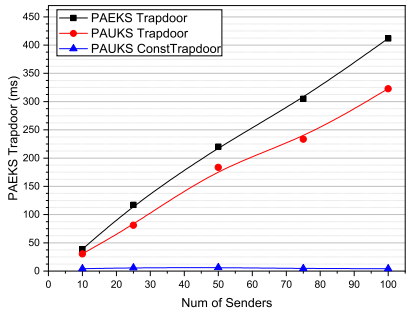
The running time of the **Trapdoor** algorithm in our PAUKS scheme is less than that in HL-PAEKS scheme. The **Trapdoor** running time in the two schemes are both linearly increase as the number of senders increases. The **ConstTrapdoor** algorithm is much more efficient and enjoys an almost constant running overhead as the number of senders increases.

TABLE V
ALGORITHMS AVERAGE RUNNING TIME COMPARASION

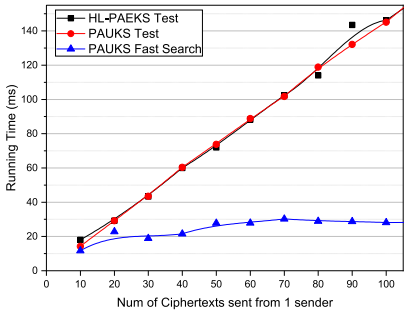
Schemes	**Enc** (ms)	**Trapdoor** (ms)	**Test** (ms)	**UpdEnc** (ms)	**ConstTrapdoor** (ms)
HL-PAEKS [7]	6.236	4.120	1.411	—	—
Our PAUKS	10.122	3.295	1.424	9.8514	4.302



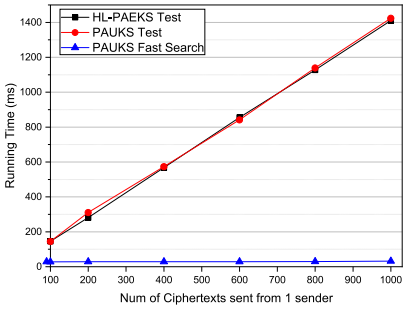
(a) Comparison of Encryption Time



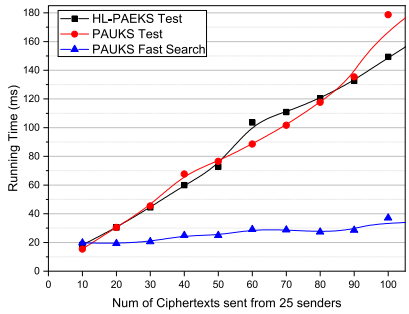
(b) Comparison of Trapdoor Generation Time



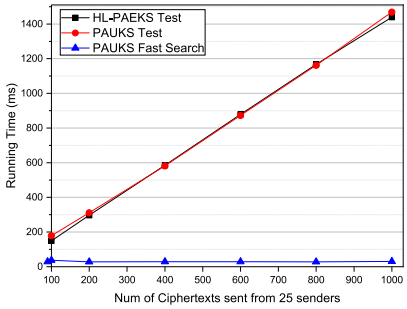
(c) Test Time with only 1 Sender



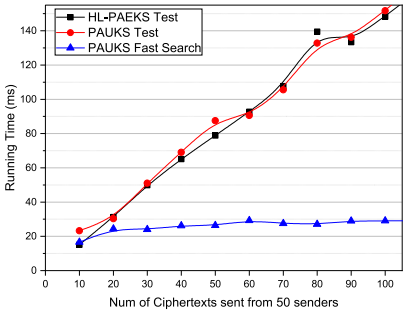
(d) Test Time with 1 Sender (more ciphertexts)



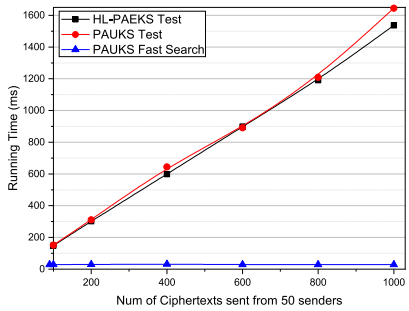
(e) Test Time with 25 Senders



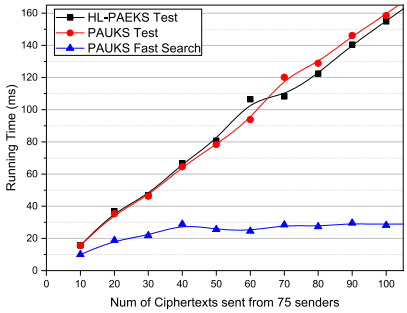
(f) Test Time with 25 Senders (more ciphertexts)



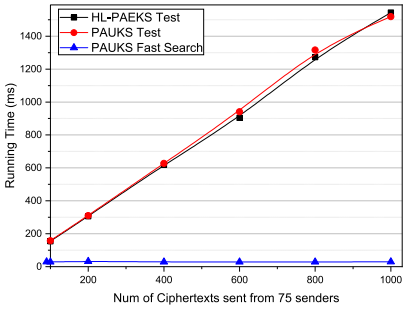
(g) Test Time with 50 Senders



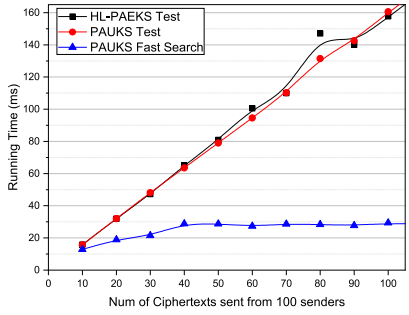
(h) Test Time with 50 Senders (more ciphertexts)



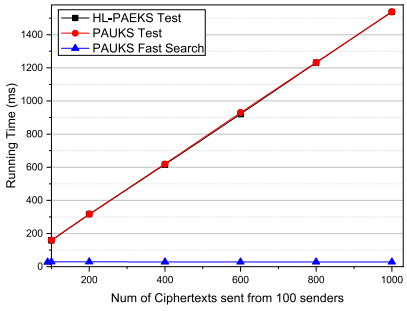
(i) Test Time with 75 Senders



(j) Test Time with 75 Senders (more ciphertexts)



(k) Test Time with 100 Senders



(l) Test Time with 100 Senders (more ciphertexts)

Fig. 7. Computation overhead comparison: **Enc**, **Trapdoor** and **Test**.

Figure 7(c)-(l) show the comparison of searching overhead of HL-PAEKS scheme and our PAUKS scheme. To ensure the impartiality, the keywords in each encryption and query are randomly selected from the experimental keyword space; The sender in each encryption is also randomly chosen. Subfigure (c)(d), (e)(f), (g)(h), (i)(j) and (k)(l) show the test time varies with the total number of received ciphertexts, when there are 1, 25, 50, 75 and 100 senders, respectively. Each ciphertext is labeled with the sender's public key, hence, the running time of the **Test** algorithms in the two schemes are linear with the total received ciphertexts shown in subfigure (d)(f)(h)(j)(l) but not entirely linear fit shown in subfigure (c)(e)(g)(i)(k). This is because the randomness of the senders and plaintexts leads to noteworthy random errors in a small sample space but no influence in a larger sample space.

Observing subfigure (c)-(l), with the same number of ciphertexts, there is no significant difference in the running time of **Test** algorithms. This is because all ciphertexts have been classified based on the label of senders and the common operations caused by different number of senders are much more efficient than the **Test** algorithms, which have minimal effect on the final experimental results.

Illustrated in Figure 7(c)-(l), compared with the "linear-cost" **Test** algorithms, the fast search of our PAUKS scheme enjoys an almost constant running overhead as the number of ciphertexts increases. The experimental results show that the "constant" fast search of our PAUKS scheme has a huge advantage in search efficiency when the number of stored ciphertexts is greater than some threshold (50, approximately).

Remark: The source code of the experiment has been uploaded to the public website, which can be found at https://github.com/PAEKSMAKER/Crypto_PAUK.git.

VI. Conclusion and Future Work

In this paper, we proposed an EMR system by introducing a public key authenticated encryption with ciphertext update and keyword search (PAUKS) scheme. Different from preceding PAEKS schemes, the proposed PAUKS scheme not only resists KGAs from both outside and inside adversaries, but also reduces trapdoor communication overhead from linear to sub-linear with the help of a non-collusion proxy. As a side result, the proposed PAUKS scheme is compatible with a secure inverted index, which achieves a fast searching efficiency.

In the system model of our PAUKS, the proxy should not collude with senders nor the cloud server. It would be an interesting work to propose a PAUKS scheme to remove the proxy from the system model and delegate the cloud computing server as the proxy. Another future work is to design a long-term secure PAUKS scheme. For many scenarios, it is important to guarantee the long-term security and support key update or forward security.

With the development of quantum computing technique, pairing based schemes might be vulnerable under the quantum computing in the future. It is important to design schemes based on quantum-resistant hard problems, e.g. LWE and SIS problems. In addition, quantum-secure lattice-based schemes might be faster than pairing in certain cases, even though larger space might be needed for trapdoors and keys. Our PAUKS scheme proves the feasibility of achieving both KGA security and lightweight in application. In the future we consider to design lattice-based PAEKS schemes supporting constant trapdoor generation and fast retrieval efficiency.

Appendix I
Security Analysis of the PAUKS Scheme

Lemma 1 (IND-CCA): For any PPT adversary $\mathcal{A}$, the advantage $\mathbf{Adv}^{C}_{\mathcal{A}}(1^{\lambda})$ to break the IND-CCA security of our PAUKS scheme is negligible if the CDH and mDLIN assumptions hold.

Proof: Let $\mathcal{A}$ be a PPT adversary to break the trapdoor privacy of the proposed PAUKS scheme. There is an algorithm $\mathcal{B}$ to solve the CDH problem based on the following game played with $\mathcal{A}$. The advantage of winning the following game for $\mathcal{A}$ is negligible if the CDH and mDLIN assumptions hold.

Game 1: The algorithm $\mathcal{B}$ takes as input a CDH instance, e.g. (g, g^x, g^y), and then runs $\mathcal{A}$ as a subroutine and plays the following game with $\mathcal{A}$.

- **Setup**: Given the security parameter 1^{λ}, the algorithm $\mathcal{B}$ sets the public parameter parameter $\mathbb{PP} = (\mathbb{G}, \mathbb{G}_T, \hat{e}, p, g)$ and sets a pair of users (U^*_S, U^*_R) as the target sender and target receiver. Let $pk^*_S = (g^y)$ be the public key of U^*_S and $pk^*_R = (g^x, g^{x_2}, g^{x_3})$ be the public key of U^*_R, where x_2, x_3 are randomly selected from $\mathbb{Z}_p$.
- **Phase 1**: Given the public key parameter $\mathbb{PP}$, the adversary $\mathcal{A}$ is able to query the following oracles. The oracles will log all the inputs and outputs, and preferentially returns the output value if the input appears in the log. If the input is a new record, then the oracles return as follows.
  - **Hash Oracles**:
    - $\mathcal{O}_{H_1}$: Given an input (T, w), this oracle randomly selects $h_1 \leftarrow \mathbb{Z}_p$ and returns $H_1(T, w) = h_1$;
    - $\mathcal{O}_{H_2}$: Given an input (T), this oracle randomly selects $h_2 \leftarrow \mathbb{Z}_p$ and returns $H_2(T) = h_2$;
    - $\mathcal{O}_{H_3}$. Given an input (T), this oracle randomly selects $h_3 \leftarrow \mathbb{Z}_p$ and returns $H_3(T) = h_3$.
  - **Ciphertext Oracle** $\mathcal{O}_C$: Given a keyword w, this oracle returns ciphertext $C = C_1, C_2, C_3, C_4, C_5$ as follows:

$$C_1 = \left((g^{x_2})^{H_1(T^*, w)} \cdot g^{x_3}\right)^{r_1}, \quad C_2 = g^{r_1}$$
$$C_3 = \left((g^{x_2})^{H_2(T^*)} \cdot g^{x_3}\right)^{r_2} \cdot {C_2}^{H_3(T^*)},$$
$$C_4 = H_4(w)^{r_2}, C_5 = H(C_1, C_2, C_3, C_4)^{r_1},$$

    where r_1 and r_2 are randomly selected from $\mathbb{Z}_p$, and $H_1(T^*, w)$, $H_2(T^*)$, $H_3(T^*)$ are generated randomly by the aforementioned hash oracles. Note that even though the input $T^* = g^{xy}$ is unknown by $\mathcal{B}$, the hash values are known, which are randomly selected.

– **Trapdoor Oracle** $\mathcal{O}_T$: Given a keyword w, this oracle returns trapdoor $T_w = T_{w,1}, T_{w,2}$ as follows:

$$T_{w,1} = g^{\frac{r_3}{x_2 \cdot H_1(T^*, w') + x_3}}, \quad T_{w,2} = g^{r_3},$$

where $H_1(T^*, w')$ is randomly generated by oracle $\mathcal{O}_H$ and r_2 is randomly selected from $\mathbb{Z}_p$.

- **Challenge**: $\mathcal{A}$ submits two keywords w_0^*, w_1^* which have not been queried to oracle $\mathcal{O}_C$. The algorithm $\mathcal{B}$ tosses a coin to decide a bit $b \leftarrow \{0, 1\}$, and returns the following ciphertext $C^* = C_1^*, C_2^*, C_3^*, C_4^*, C_5^*$:

$$C_1^* = \left(g^{x_2 \cdot H_1^* + x_3}\right)^{r_1^*}, \quad C_2^* = g^{r_1^*},$$
$$C_3^* = \left(g^{x_2 \cdot H_2(T^*) + x_3}\right)^{r_2^*} \cdot C_2^{*H_3(T^*)}, \quad C_4^* = H_4(w)^{r_2^*},$$
$$C_5^* = H(C_1, C_2, C_3, C_4)^{r_1^*},$$

where r_1^*, r_2^*, H_1^* are random selected from $\mathbb{Z}_p$, and $H_2(T^*)$ is generated from the hash oracle $\mathcal{O}_{H_2}$.

- **Phase 2**: $\mathcal{A}$ can also query the oracles in phase 1, except that w_0^*, w_1^* cannot appear in $\mathcal{O}_T$.
- **Guess**: Finally, $\mathcal{A}$ returns a bit $b' \leftarrow \{0, 1\}$ and wins the game iff $b' = b$.

We denote by $\mathbf{E}_1$ the event that $\mathcal{A}$ has queried (g^{xy}, w_b^*) to the hash oracle $\mathcal{O}_{H_1}$. In case $\mathbf{E}_1$ happens, the algorithm $\mathcal{B}$ aborts the game and solves the CDH problem based on $\mathcal{A}$'s input. In other case, the game is identical to the game in the security model in $\mathcal{A}$'s view and the ciphertext is indistinguishable if the mDLIN assumption holds of which the proof is omitted here limited by space. Hence, the probability of $\mathcal{A}$ to win this game is:

$$\begin{aligned}\left|\mathbf{Pr}[\mathcal{A}_{\text{win}}^{\mathbf{G}_1}]\right| &= \left|\mathbf{Pr}[\mathcal{A}_{\text{win}}^{\mathbf{G}_1} \wedge \overline{\mathbf{E}_1}] + \mathbf{Pr}[\mathcal{A}_{\text{win}}^{\mathbf{G}_1} \wedge (\mathbf{E}_1)]\right| \\ &= \left|\mathbf{Pr}[\mathcal{A}_{\text{win}}^{\mathbf{G}_1} | \overline{\mathbf{E}_1}] \cdot \mathbf{Pr}[\overline{\mathbf{E}_1}] + \mathbf{Pr}[\mathcal{A}_{\text{win}}^{\mathbf{G}_1} | \mathbf{E}_1] \cdot \mathbf{Pr}[\mathbf{E}_1]\right| \\ &= \frac{1}{2} \cdot (1 - negl(1^\lambda)) + \mathbf{Pr}[\mathcal{A}_{\text{win}}^{\mathbf{G}_1} | \overline{\mathbf{E}_1}] \cdot negl(1^\lambda) \\ &\leq \frac{1}{2} + negl(1^\lambda).\end{aligned}$$

■

Lemma 2 (IND-U-CCA): The proposed PAUKS scheme IND-U-CCA secure in the random oracle if the CDH and mDLIN assumptions hold.

Proof: The proof is based on the following game played between a PPT adversary $\mathcal{A}$ and an algorithm $\mathcal{B}$.

Game 2: The algorithm $\mathcal{B}$ is given the CDH tuple (g, g^x, g^y) and to solve the CDH problem if $\mathcal{A}$ is able to break the security of the proposed PAUKS scheme within a PPT time.

- **Setup**: Given the security parameter 1^λ, the algorithm $\mathcal{B}$ sets the public parameter parameter $\mathbb{PP} = \{\mathbb{G}, \mathbb{G}_T, \hat{e}, p, g\}$ and sets a pair of users (U_S^*, U_R^*) as target receiver. Let $pk_S^* = (g^y)$ be the public key of U_S^* and $pk_R^* = (g^x, g^{x_2}, g^{x_3})$ be the public key of U_R^*, where the private key x_2, x_3, x_4 are randomly selected from $\mathbb{Z}_p$.
- **Phase 1**: In this phase, the adversary $\mathcal{A}$ can query the following oracles.
  – Hash Oracles:
    * $\mathcal{O}_{H_1}$: Given an input (T, w), this oracle randomly selects $h \xleftarrow{\$} \mathbb{Z}_p$ and returns $H_1(T, w) = h_1$.
    * $\mathcal{O}_{H_2}$: Given an input (T), this oracle randomly selects $h_1 \xleftarrow{\$} \mathbb{Z}_p$ and returns $H_2(T) = h_2$.
    * $\mathcal{O}_{H_3}$: Given an input (T), this oracle randomly selects $h_2 \xleftarrow{\$} \mathbb{Z}_p$ and returns $H_3(T) = h_3$.
  – Encryption Oracle $\mathcal{O}_C$: Given a keyword w, this oracle randomly selects $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_p$ and returns a ciphertext $C = C_1, C_2, C_3, C_4, C_5$ as follow.

$$C_1 = \left(pk_{\mathbf{R_2}}^{H_1(T^*, w)} \cdot pk_{\mathbf{R_3}}\right)^{r_1}, \quad C_2 = g^{r_1},$$
$$C_3 = \left(pk_{\mathbf{R_2}}^{H_2(T^*)} \cdot pk_{\mathbf{R_3}}\right)^{r_2} \cdot C_2^{H_3(T^*)},$$
$$C_4 = H_4(w)^{r_2}, C_5 = H(C_1, C_2, C_3, C_4)^{r_1}.$$

  – Trapdoor Oracle $\mathcal{O}_{\hat{T}}$: Given a keyword w, this oracle randomly selects $r_3 \xleftarrow{\$} \mathbb{Z}_p$ and returns a trapdoor $\hat{T}_w = (\hat{T}_{w,1}, \hat{T}_{w,2})$ as follow.

$$\hat{T}_{w,1} = g^{x_4 \cdot r_3}, \hat{T}_{w,2} = H_2(w)^{r_3}.$$

- **Challenge**: $\mathcal{A}$ submits two keywords w_0^*, w_1^* which have not been queried to oracle $\mathcal{O}_{\hat{T}}$. The algorithm $\mathcal{B}$ randomly selects $b \xleftarrow{\$} \{0, 1\}$ and $r_1^*, r_2^* \xleftarrow{\$} \mathbb{Z}_p$, and returns the challenged ciphertext $C^* = C_1^*, C_2^*, C_3^*, C_4^*, C_5^*, C_6^*$ as follow:

$$C_1^* = \left(pk_{\mathbf{R_2}}^{H_1^*} \cdot pk_{\mathbf{R_3}}\right)^{r_1^*}, \quad C_2^* = g^{r_1},$$
$$C_3^* = \left(pk_{\mathbf{R_2}}^{H_2(T^*)} \cdot pk_{\mathbf{R_3}}\right)^{r_2^*} \cdot C_2^{*H_3(T^*)}, C_4^* = H_4(w^*)^{r_2^*},$$
$$C_5^* = H(C_1, C_2, C_3, C_4)^{r_1^*}, \quad C_6^* = g^{x_4 \cdot r_2^*}$$

where r_1^*, r_2^*, H_1^* are random selected from $\mathbb{Z}_p$, and $H_2(T^*)$ is generated from the hash oracle $\mathcal{O}_{H_2}$.

- **Phase 2**: The adversary $\mathcal{A}$ can query the oracles in phase 1, except that the keywords w_0^*, w_1^* cannot appear in the trapdoor oracle $\mathcal{O}_{\hat{T}}$.
- **Guess**: $\mathcal{A}$ returns a keyword w' and wins the game iff $w' = w^*$.

We denote the event that the adversary $\mathcal{A}$ queries g^{xy} to the hash oracle $\mathcal{O}_{H_1}$, $\mathcal{O}_{H_2}$ or $\mathcal{O}_{H_3}$ by $\mathbf{E}_2$. In case $\mathbf{E}_2$ happens, $\mathcal{B}$ solves the CDH problem by retrieving the log of the hash oracles. In other case, the game is identical to the game in the security model in $\mathcal{A}$'s view and the ciphertext is indistinguishable if the mDLIN assumption holds of which the proof is similar to that in game 1 and omitted here due to the space limitation. Hence, The probability that $\mathcal{A}$ breaks the ciphertext semantic security of the proposed PAUKS scheme is:

$$\begin{aligned}\left|\mathbf{Pr}[\mathcal{A}_{\text{win}}^{\mathbf{G}_2}]\right| &= \left|\mathbf{Pr}[\mathcal{A}_{\text{win}}^{\mathbf{G}_2} \wedge \overline{\mathbf{E}_2}] + \mathbf{Pr}[\mathcal{A}_{\text{win}}^{\mathbf{G}_2} \wedge \mathbf{E}_2]\right| \\ &= \left|\mathbf{Pr}[\mathcal{A}_{\text{win}}^{\mathbf{G}_2} | \overline{\mathbf{E}_2}] \cdot \mathbf{Pr}[\overline{\mathbf{E}_2}] + \mathbf{Pr}[\mathcal{A}_{\text{win}}^{\mathbf{G}_2} | \mathbf{E}_2] \cdot \mathbf{Pr}[\mathbf{E}_2]\right| \\ &= \frac{1}{2} \cdot (1 - negl(1^\lambda)) + \mathbf{Pr}[\mathcal{A}_{\text{win}}^{\mathbf{G}_2} | \mathbf{E}_2] \cdot negl(1^\lambda) \\ &\leq \frac{1}{2} + negl(1^\lambda).\end{aligned}$$

■

Lemma 3 (IND-TP-CCA): The proposed PRAEK scheme satisfies the IND-TP-CCA security in the random oracle if the CDH assumption holds.

The proof of lemma 3 is similar to that of lemma 1 and 2, which is omitted here.

REFERENCES

[1] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Advances in Cryptology—EUROCRYPT*, C. Cachin and J. L. Camenisch, Eds. Berlin, Germany: Springer, 2004, pp. 506–522.

[2] J. W. Byun, H. S. Rhee, H. A. Park, and D. H. Lee, "Off-line keyword guessing attacks on recent keyword search schemes over encrypted data," in *Secure Data Management*, 3rd ed. Seoul, (South) Korea: SDM, Sep. 2006.

[3] R. Chen, Y. Mu, G. Yang, F. Guo, and X. Wang, "Dual-server public-key encryption with keyword search for secure cloud storage," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 4, pp. 789–798, Apr. 2016.

[4] R. Chen et al., "Server-aided public key encryption with keyword search," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 12, pp. 2833–2842, Dec. 2016.

[5] Y. Ling, S. Ma, Q. Huang, and X. Li, "A general two-server framework for ciphertext-checkable encryption against offline message recovery attack," in *Cloud Computing and Security* (Lecture Notes in Computer Science), X. Sun, Z. Pan, and E. Bertino, Eds. Cham, Switzerland: Springer, 2018, pp. 370–382.

[6] L. Fang, W. Susilo, C. Ge, and J. Wang, "Public key encryption with keyword search secure against keyword guessing attacks without random Oracle," *Inf. Sci.*, vol. 238, pp. 221–241, Jul. 2013.

[7] Q. Huang and H. Li, "An efficient public-key searchable encryption scheme secure against inside keyword guessing attacks," *Inf. Sci.*, vols. 403–404, pp. 1–14, Sep. 2017.

[8] D. He, M. Ma, S. Zeadally, N. Kumar, and K. Liang, "Certificateless public key authenticated encryption with keyword search for industrial Internet of Things," *IEEE Trans. Ind. Informat.*, vol. 14, no. 8, pp. 3618–3627, Aug. 2018.

[9] L. Wu, Y. Zhang, M. Ma, N. Kumar, and D. He, "Certificateless searchable public key authenticated encryption with designated tester for cloud-assisted medical Internet of Things," *Ann. Telecommun.*, vol. 74, nos. 7–8, pp. 423–434, Aug. 2019.

[10] H. Li, Q. Huang, J. Shen, G. Yang, and W. Susilo, "Designated-server identity-based authenticated encryption with keyword search for encrypted emails," *Inf. Sci.*, vol. 481, pp. 330–343, May 2019.

[11] B. Qin, Y. Chen, Q. Huang, X. Liu, and D. Zheng, "Public-key authenticated encryption with keyword search revisited: Security model and constructions," *Inf. Sci.*, vol. 516, pp. 515–528, Apr. 2020.

[12] N. Pakniat, D. Shiraly, and Z. Eslami, "Certificateless authenticated encryption with keyword search: Enhanced security model and a concrete construction for industrial IoT," *J. Inf. Secur. Appl.*, vol. 53, Aug. 2020, Art. no. 102525.

[13] M. Noroozi and Z. Eslami, "Public-key encryption with keyword search: A generic construction secure against online and offline keyword guessing attacks," *J. Ambient Intell. Humanized Comput.*, vol. 11, no. 2, pp. 879–890, Feb. 2020.

[14] B. Wu, C. Wang, and H. Yao, "Security analysis and secure channel-free certificateless searchable public key authenticated encryption for a cloud-based Internet of Things," *PLoS ONE*, vol. 15, no. 4, Apr. 2020, Art. no. e0230722.

[15] J. Li, M. Wang, Y. Lu, Y. Zhang, and H. Wang, "ABKS-SKGA: Attribute-based keyword search secure against keyword guessing attack," *Comput. Standards Interfaces*, vol. 74, Feb. 2021, Art. no. 103471.

[16] N. Yang, Q. Zhou, and S. Xu, "Public-key authenticated encryption with keyword search without pairings," *J. Comput. Res. Develop.*, vol. 57, no. 10, p. 2125, 2020.

[17] X. Liu, K. He, G. Yang, W. Susilo, J. Tonien, and Q. Huang, "Broadcast authenticated encryption with keyword search," in *Proc. Australas. Conf. Inf. Secur. Privacy*. Cham, Switzerland: Springer, 2021, pp. 193–213.

[18] Y. Lu and J. Li, "Lightweight public key authenticated encryption with keyword search against adaptively-chosen-targets adversaries for mobile devices," *IEEE Trans. Mobile Comput.*, vol. 21, no. 12, pp. 4397–4409, Dec. 2022.

[19] B. Qin, H. Cui, X. Zheng, and D. Zheng, "Improved security model for public-key authenticated encryption with keyword search," in *Proc. Int. Conf. Provable Practical Secur.* Cham, Switzerland: Springer, 2021, pp. 19–38.

[20] L. Han, J. Guo, G. Yang, Q. Xie, and C. Tian, "An efficient and secure public key authenticated encryption with keyword search in the logarithmic time," *IEEE Access*, vol. 9, pp. 151245–151253, 2021.

[21] J. Baek, R. Safavi-Naini, and W. Susilo, "Public key encryption with keyword search revisited," in *Proc. Int. Conf. Comput. Sci. Appl.*, Perugia, Italy, 2008, pp. 1249–1259.

[22] P. Xu, H. Jin, Q. Wu, and W. Wang, "Public-key encryption with fuzzy keyword search: A provably secure scheme under keyword guessing attack," *IEEE Trans. Comput.*, vol. 62, no. 11, pp. 2266–2277, Nov. 2013.

[23] M. Noroozi and Z. Eslami, "Public key authenticated encryption with keyword search: Revisited," *IET Inf. Secur.*, vol. 13, no. 4, pp. 336–342, Jul. 2019.

[24] X. Pan and F. Li, "Public-key authenticated encryption with keyword search achieving both multi-ciphertext and multi-trapdoor indistinguishability," *J. Syst. Archit.*, vol. 115, May 2021, Art. no. 102075.

[25] B. Chen, L. Wu, S. Zeadally, and D. He, "Dual-server public-key authenticated encryption with keyword search," *IEEE Trans. Cloud Comput.*, vol. 10, no. 1, pp. 322–333, Jan. 2022.

[26] Y. Lu, J. Li, and Y. Zhang, "Secure channel free certificate-based searchable encryption withstanding outside and inside keyword guessing attacks," *IEEE Trans. Services Comput.*, vol. 14, no. 6, pp. 2041–2054, Nov. 2021.

[27] J. Guo, L. Han, G. Yang, X. Liu, and C. Tian, "An improved secure designated server public key searchable encryption scheme with multi-ciphertext indistinguishability," *J. Cloud Comput.*, vol. 11, no. 1, pp. 1–12, Dec. 2022.

[28] Z.-Y. Liu, Y.-F. Tseng, R. Tso, Y.-C. Chen, and M. Mambo, "Identity-certifying authority-aided identity-based searchable encryption framework in cloud systems," *IEEE Syst. J.*, vol. 16, no. 3, pp. 4629–4640, Sep. 2022.

[29] X. Liu, H. Li, G. Yang, W. Susilo, J. Tonien, and Q. Huang, "Towards enhanced security for certificateless public-key authenticated encryption with keyword search," in *Provable Security* (Lecture Notes in Computer Science), R. Steinfeld and T. H. Yuen, Eds. Cham, Switzerland: Springer, 2019, pp. 113–129.

[30] Y. Lu, J. Li, and F. Wang, "Pairing-free certificate-based searchable encryption supporting privacy-preserving keyword search function for IIoTs," *IEEE Trans. Ind. Informat.*, vol. 17, no. 4, pp. 2696–2706, Apr. 2021.

[31] R. Behnia, M. O. Ozmen, and A. A. Yavuz, "Lattice-based public key searchable encryption from experimental perspectives," *IEEE Trans. Dependable Secure Comput.*, vol. 17, no. 6, pp. 1269–1282, Nov. 2020.

[32] Z.-Y. Liu, Y.-F. Tseng, R. Tso, M. Mambo, and Y.-C. Chen, "Public-key authenticated encryption with keyword search: Cryptanalysis, enhanced security, and quantum-resistant instantiation," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, May 2022, pp. 423–436.

[33] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn.* Cham, Switzerland: Springer, 1998, pp. 127–144.

[34] J. Shao, Z. Cao, X. Liang, and H. Lin, "Proxy re-encryption with keyword search," *Inf. Sci.*, vol. 180, no. 13, pp. 2576–2587, Jul. 2010.

[35] Z. Chen, S. Li, Y. Guo, Y. Wang, and Y. Chu, "A limited proxy re-encryption with keyword search for data access control in cloud computing," in *Proc. Int. Conf. Netw. Syst. Secur.* Cham, Switzerland: Springer, 2015, pp. 82–95.

[36] T. Hoang, R. Behnia, Y. Jang, and A. A. Yavuz, "MOSE: Practical multi-user oblivious storage via secure enclaves," in *Proc. 10th ACM Conf. Data Appl. Secur. Privacy*, Mar. 2020, pp. 17–28.

[37] T. Hoang, M. O. Ozmen, Y. Jang, and A. A. Yavuz, "Hardware-supported ORAM in effect: Practical oblivious search and update on very large dataset," *Proc. Privacy Enhancing Technol.*, vol. 2019, no. 1, pp. 172–191, Jan. 2019.

[38] L. Xu, Z. Sun, W. Li, and H. Yan, "Delegatable searchable encryption with specified keywords for EHR systems," *Wireless Netw.*, pp. 1–13, Jul. 2020, doi: 10.1007/s11276-020-02410-3.

[39] M. Deng, W. Song, M. Ma, H. Li, and M. Israr, "Efficient designated-server proxy re-encryption with keyword search for big data," in *Proc. Int. Conf. Artif. Intell. Secur.* Cham, Switzerland: Springer, 2022, pp. 364–377.
[40] X. Boyen, "The uber-assumption family," in *Proc. Int. Conf. Pairing-Based Cryptogr.* Cham, Switzerland: Springer, 2008, pp. 39–56.
[41] F. Bao, R. H. Deng, and H. Zhu, "Variations of Diffie-Hellman problem," in *Proc. Int. Conf. Inf. Commun. Secur.* Cham, Switzerland: Springer, 2003, pp. 301–312.

Hongbo Li received the Ph.D. degree from South China Agricultural University, Guangzhou, China. He currently holds a post-doctoral position with the College of Mathematics and Informatics, South China Agricultural University. His research interests include applied cryptography and cloud security.

Qiong Huang received the Ph.D. degree from the City University of Hong Kong in 2010. He is currently a Professor with the College of Mathematics and Informatics, South China Agricultural University, Guangzhou, China. He has published more than 150 research papers in international conferences and journals. His research interests include cryptography and information security, in particular, cryptographic protocols design and analysis. He served as a program committee member in many international conferences.

Jianye Huang received the B.S. and M.S. degrees from South China Agricultural University, Guangzhou, China, in 2015 and 2018, respectively. He is currently pursuing the Ph.D. degree with the University of Wollongong, Australia. His research interests include cryptography; in particular, leakage-resilient signature, searchable encryption, and k-times anonymous authentication.

Willy Susilo (Fellow, IEEE) is currently a Distinguished Professor with the Faculty of Engineering and Information Sciences, School of Computing and Information Technology, University of Wollongong (UOW), Australia, where he is also the Director of the School of Computing and Information Technology, Institute of Cybersecurity and Cryptology. He has been the Head of the School of Computing and Information Technology, UOW, since 2015. He has also served as the program committee member of several international conferences. He was awarded the prestigious Australian Research Council Future Fellowship in 2009. In 2016, he was awarded the Researcher of the Year at UOW, due to his research excellence and contributions. He is the Editor-in-Chief of the *Computer Standards and Interfaces* (Elsevier) and the *Information* journal (MDPI). He is currently an Associate Editor of IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, *ACM Computing Surveys*, and *Computers and Security* (Elsevier).