Efficient Privacy-Preserving Conjunctive Searchable Encryption for Cloud-IoT Healthcare Systems

JIADI MA, College of Computer Science, Beijing University of Technology, Beijing, China
TIANQI PENG, College of Computer Science, Beijing University of Technology, Beijing, China
GONG BEI, College of Computer Science, Beijing University of Technology, Beijing, China
MUHAMMAD WAQAS, School of Computing and Mathematical Sciences, University of Greenwich, London, United Kingdom of Great Britain and Northern Ireland and School of Engineering, Edith Cowan University, Joondalup, Australia
HISHAM ALASMARY, Department of Computer Science, King Khalid University, Abha, Saudi Arabia
SHENG CHEN, University of Southampton, Southampton, United Kingdom of Great Britain and Northern Ireland

In cloud-Internet of Things (IoT) healthcare systems, private medical data leakage is a serious concern as the cloud server is not fully trusted. Dynamic searchable symmetric encryption (DSSE), with necessary forward and backward privacy security properties, enables doctors to retrieve ciphertexts while guaranteeing data privacy. However, existing forward and backward private DSSE schemes are not well-suited for cloud-IoT healthcare systems with attribute-value type databases. To this end, we propose an efficient privacy-preserving conjunctive searchable encryption scheme for cloud-IoT healthcare systems, called PC-SE. It is the first conjunctive DSSE scheme designed for attribute-value type databases. Specifically, we design flexible search capabilities for PC-SE to address users' various search requirements. It can not only achieve precise conjunctive search based on keywords but also realize broad attribute search. Moreover, our scheme achieves fine-grained search for attribute values while maintaining forward and Type-I^{-} backward privacy. This approach reduces the communication burden and minimizes the risk of privacy exposure. To ensure that users with different authorities can only access the corresponding attribute values, we introduce an attribute access control mechanism in PC-SE. Finally, security analysis and experimental results demonstrate that PC-SE is secure and effective.

CCS Concepts: • **Security and privacy → Management and querying of encrypted data;**

Additional Key Words and Phrases: Cloud-IoT healthcare system, dynamic symmetric searchable encryption, forward and backward security, attribute-value type databases

The authors extend their appreciation to the Deanship of Scientific Research at King Khalid University for funding this work through Large Groups Project under grant number RGP.2/637/46.
Authors' Contact Information: Jiadi Ma, College of Computer Science, Beijing University of Technology, Beijing, Beijing, China; e-mail: 13718021152@163.com; Tianqi Peng, College of Computer Science, Beijing University of Technology, Beijing, China; e-mail: tianqi_peng@emails.bjut.edu.cn; Gong Bei, College of Computer Science, Beijing University of Technology, Beijing, China; e-mail: gongbei@bjut.edu.cn; Muhammad Waqas (corresponding author), School of Computing and Mathematical Sciences, University of Greenwich, London, United Kingdom of Great Britain and Northern Ireland, School of Engineering and Edith Cowan University, Joondalup, Western Australia, Australia; e-mail: engr.waqas2079@gmail.com; Hisham Alasmary, Department of Computer Science, King Khalid University, Abha, Saudi Arabia; e-mail: alasmary@kku.edu.sa; Sheng Chen, University of Southampton, Southampton, United Kingdom of Great Britain and Northern Ireland; e-mail: sqc@ecs.soton.ac.uk.

ACM 2471-2566/2025/11-ART1
https://doi.org/10.1145/3769425

ACM Reference Format:
Jiadi Ma, Tianqi Peng, Gong Bei, Muhammad Waqas, Hisham Alasmary, and Sheng Chen. 2025. Efficient Privacy-Preserving Conjunctive Searchable Encryption for Cloud-IoT Healthcare Systems. *ACM Trans. Priv. Sec.* 29, 1, Article 1 (November 2025), 27 pages. https://doi.org/10.1145/3769425

1 Introduction

With the rapid development of **Internet of Things** (**IoTs**) and cloud computing, integrating these technologies in cloud-IoT healthcare systems has found widespread applications in real life [1–4]. Wearable IoT devices aggregate collected patient data into **electronic health records** (**EHRs**) and upload them to cloud servers [5]. This not only facilitates doctors' prompt access to medical data for diagnosis and treatment but also reduces the cost of storing a large number of EHRs files. However, due to the fact that cloud servers may not always be trusted, the sensitivity and privacy of medical data impose a challenge for cloud-IoT healthcare systems [6]. The leakage of EHRs can lead to reputation damage and unnecessary discrimination against patients [7, 8]. A common solution is to encrypt EHRs before uploading them to the cloud server, but this introduces another problem, i.e., encrypted EHRs cannot be retrieved by users based on keywords.

Symmetric searchable encryption (**SSE**) was first introduced by Song et al. [9] and can successfully achieve ciphertext retrieval. To support the dynamic update of databases, Kamara et al. [10] proposed **dynamic SSE** (**DSSE**) by introducing additional leakage. Unfortunately, these additional leakages can severely compromise data privacy in healthcare systems [11–13]. To this end, Stefanov et al. [14] introduced the concepts of **forward and backward** (**FB**) privacy to mitigate the additional privacy concerns introduced by supporting update operations. Forward privacy severs the connection between newly added files and previous search queries, ensuring that previous search queries cannot match the newly added files. On the other hand, backward privacy severs the connection between the current search queries and files that have been deleted, ensuring that subsequent search queries on keyword w cannot match files that were added and later deleted. Bost et al. [15, 16] formally defined forward privacy and three types of backward privacy, gradually decreasing in strength from Type-I to Type-III. Based on [16], Zuo et al. [17] defined backward privacy Type-I^{-}, which is more secure than backward privacy Type-I. Most DSSE schemes have drawn inspiration from and incorporated these security definitions [18, 20–22, 24, 25].

Single-keyword DSSE scheme with FB privacy has limited expressiveness of queries and is unsuitable for healthcare systems. A practical DSSE scheme for healthcare systems should support conjunctive search [26]. Patranabis and Mukhopadhyay [27] designed an oblivious computation protocol and constructed a practical DSSE scheme for conjunctive search with FB privacy, named ODXT. Zuo et al. [28] proposed FB-DSSE-CQ, a conjunctive DSSE scheme using an extended bitmap index. Chen et al. [29] and Guo et al. [30] implemented a conjunctive searchable encryption scheme with FB privacy using inner product matching and puncturable pseudorandom functions, respectively.

In cloud-IoT healthcare systems, EHRs files follow an attribute-value data format, and a representative resulting database is shown in Table 1. Each row represents an EHRs file with a corresponding file identifier that contains several attributes. Each column contains the attribute values for the respective EHRs files. However, existing FB privacy conjunctive DSSE schemes are not directly applicable to cloud-IoT healthcare systems [31]. The reasons are as follows: Doctors perform various tasks such as diagnosis, research, and statistics based on EHRs. They not only perform conjunctive searches based on keywords to precisely obtain query results but also search for files containing a specific attribute. Unlike the former, the latter type of search is not dependent on keywords but rather is a broad attribute-based search [32]. Existing schemes only support fixed keyword-based search functionality, which fails to meet users' various practical query requirements. Second, when

Table 1. EHRs Database

ID \ Attribute	Name	Age	Family History	...	Past History	Medications
700756	***	42	Father has Diabetes	...	Hypertension (diagnosed in 2015)	Lisinopril (10mg daily) for hypertension
749939	***	53	⊥	...	Asthma (diagnosed in childhood), Seasonal allergies	Loratadine for seasonal allergies
...	...	...	...	...	...	...
739221	***	60	⊥	...	Type 2 Diabetes (diagnosed in 2016), Hypertension	Metformin (250mg daily) for diabetes

applied to attribute-based databases, existing conjunctive search schemes introduce additional communication burdens and privacy concerns. Specifically, different attributes may contain the same keywords. When a doctor seeks files with "hypertension" in the "past history" attribute and "diabetes" in the "family history" attribute, the correct result should be the file with ID = 700756. However, using previous DSSE schemes, the search results would include both ID = 700756 and ID = 739221 files. The file with ID = 739221 also contains "hypertension" and "diabetes", but it is not the required file for the doctor, resulting in unnecessary communication burden and information leakage. A straightforward solution for exsiting scheme is to flatten the multi-dimensional structure of attribute-value databases by treating attributes as part of the keyword, that is, by concatenating the attribute name and the keyword into a single new keyword for search. However, this approach still faces several challenges in terms of both search granularity and privacy. Specifically, it can only support coarse-grained, document-level search, meaning that users can retrieve only entire documents containing the matching keyword, rather than specific content under a certain attribute. In practice, doctors may wish to obtain fine-grained information pertaining to a particular attribute (e.g., "past history" or "medications") rather than accessing the full EHR. This increases the risk of unnecessary privacy exposure and also leads to higher communication overhead. Furthermore, in real-world healthcare systems, the attribute values of EHR files often contain highly sensitive personal information. Different users (e.g., attending physicians, nurses, and medical interns) should have distinct access rights to specific attribute fields. However, when attributes are embedded within keyword strings, the system loses the ability to identify and authorize access to individual attributes. As a result, access control can only be enforced at the keyword level, not at the attribute level. This issue becomes even more pronounced when attribute names overlap with keyword values, making precise, attribute-level access control even more difficult to implement. This approach causes DSSE schemes to overlook the need for attribute-level access control and fine-grained attribute-value retrieval. In summary, designing a FB privacy conjunctive DSSE scheme with flexible search capabilities and high private security for cloud-IoT healthcare systems remains a challenge.

To address the above challenges, we design PC-SE, a FB) privacy-preserving conjunctive DSSE scheme specifically tailored for attribute-value structured EHR databases in cloud-IoT healthcare systems. At a high level, the scheme integrates oblivious computation protocol and symmetric encryption with homomorphic addition with a structured secure index to support precise conjunctive keyword search across multiple attributes, while enabling fine-grained attribute-value retrieval instead of returning entire documents. An attribute access mapping mechanism is further introduced to achieve fast attribute-specific queries and enforce attribute-level access control, ensuring that users with different roles can only access authorized fields. The scheme supports dynamic updates while preserving forward privacy and Type-I^{-} backward privacy, and is proven secure under the

defined security model. Experimental results demonstrate that PC-SE achieves a practical balance between strong privacy guarantees, expressive query capabilities, and low computation overhead. Our contributions are summarized as follows:

— We propose the first FB privacy conjunctive DSSE scheme for EHRs file databases. By integrating an oblivious computation protocol and symmetric encryption with homomorphic addition, the scheme enables precise and efficient conjunctive keyword search across multiple attributes. Instead of returning full documents, the scheme allows users to retrieve only specific attribute values that satisfy the query. This enables users to obtain accurate search results and addresses the widespread issues of additional communication burden and privacy leakage present in existing schemes.
— To support flexible search, we introduce an attribute access mapping mechanism, which facilitates efficient broad attribute queries and enforces attribute-level access control. This ensures that users with different roles can only search and access the attributes they are authorized to see, satisfying fine-grained privacy requirements in real-world healthcare systems.
— We formally prove that our scheme achieves forward privacy and Type-I^{-} backward privacy, the strongest defined level in existing literature. We also conduct extensive experiments to demonstrate that our scheme achieves a practical balance between security, search efficiency, and functionality, outperforming related works in both expressiveness and performance.

2 Related Works

Song et al. [9] pioneered the first static SSE scheme in 2000. The search time of the scheme is linearly related to the size of the database, but its security and expressive power were still lacking. Following this, SSE has attracted extensive research and attention. Goh et al. [34] and Chang et al. [35] respectively constructed security models for SSE and designed corresponding SSE schemes. In particular, Goh et al. [34] proposed the first SSE scheme based on inverted indexing. However, the security models they proposed still had flaws. Curtmola et al. [36] introduced an adaptive security model and constructed an SSE scheme with sub-linear complexity based on inverted index. However, these static database-based SSE schemes do not meet the real-time data update requirements in many scenarios.

To enable SSE schemes to support dynamic updates, Kamara et al. [10] proposed the concept of DSSE and designed the first DSSE scheme supporting dynamic data updates. A series of subsequent works focused on enhancing the functionality, efficiency, and security of DSSE schemes. In particular, leveraging the strong expressiveness of **attribute-based encryption** (**ABE**), Li et al. [37] proposed a new attribute-based encryption scheme (KSF-OABE) that outsources key issuance and decryption to enable keyword search, reducing computation costs and ensuring the cloud service provider cannot access the plaintext, while proving its security against chosen-plaintext attacks. Gao et al. [38] and Yu et al. [39] integrated ABE with blockchain technology to achieve fine-grained search and revocable functionalities. Similarly, Yin et al. [40] proposed a novel secure index based on access policies and an attribute-based search token, allowing for fine-grained search capabilities along with access control. While these approaches have successfully minimized the overhead associated with decryption and revocation, DSSE schemes that rely on ABE still face challenges when applied in resource-constrained network environments. In addition, some work focused on addressing common query and result pattern leakage in SSE protocols. Li et al. [41] proposed a **verifiable searchable encryption** (**VSE**) scheme that supports conjunctive keyword search while hiding both access and volume patterns, using additively symmetric homomorphic encryption and private set intersection protocols. Yang et al. [42] used the **oblivious polynomial evaluation** (**OPE**) protocol to construct a verifiable searchable encryption scheme named OpenSE, to preserve query, and result

pattern privacy. Ji et al. [43] proposed a leakage-suppressed **verifiable searchable symmetric encryption** (**VSSE**) scheme that hides search, access, and response length patterns, allows the client to verify the server's response. Xu et al. [44] introduced the concept of **keyword pair result pattern** (**KPRP**) leakage and proposed a DSSE scheme to address this issue. Chen et al. [45] introduced MFSSE, an SSE scheme that conceals search patterns by modifying the search trapdoor for each query and introduces random errors to resist access pattern leakage. Jiang et al. [46] leveraged trusted hardware Intel SGX to deploy a new tree-based SSE scheme that hides result patterns.

There is no doubt that dynamic updates introduce more security threats to the scheme. In particular, Zhang et al. [13] proposed a file injection attack in 2016 that could easily compromise the security of DSSE. Forward and backward privacy effectively control leakage in DSSE and resist file injection attacks. These concepts were first introduced by Stefanov et al. [14]. Bost et al. [16] formally defined forward privacy and three security levels of backward privacy (Type-I to Type-III) and proposed DSSE schemes with three levels of security. Song et al. [47] designed a state chain structure based on symmetric cryptographic primitives to construct a fast forward privacy scheme, named Fast. Li et al. [18] adopted a triple dictionary structure to design a forward privacy scheme for healthcare systems, but the scheme is limited by its cumbersome update operations. Li et al. [19] proposed a **multi-user dynamic searchable symmetric encryption** (**MDSSE**) scheme for attribute-value type databases. The scheme also uses a triple dictionary structure to support keyword / attribute-based searches and adopts blind storage to enhance privacy. Although the scheme is practical in dynamic multi-user settings, it only supports single-keyword / attribute queries and lacks support for conjunctive searches, backward privacy, and fine-grained access control. Liu et al. [20] and Wang et al. designed a more efficient forward privacy scheme for both search and update using a hash chain structure. The lack of consideration for backward privacy still fails to protect privacy security comprehensively. Bost et al. [16] proposed Janus, a DSSE scheme with forward and Type-III backward privacy, using puncturable encryption. Later, Sun et al. [48] found that the public-key-based puncturable encryption used in Janus had low update efficiency. As a result, they proposed a symmetric cryptographic primitive-based puncturable encryption and designed an optimized scheme called Janus++. After analyzing the Type-I backward privacy proposed by Bost [16], Zuo et al. [17] proposed a more secure Type-I^{-} backward privacy and construct a scheme with both forward and backward privacy using bitmap indexes and homomorphic encryption techniques. In order to reduce client storage, Demertzis et al. [11] proposed the scheme Qos, which is the first quasi-optimal Type-III backward privacy scheme as far as we know. Zhao et al. [49] introduced an efficient **cumulative commitment verification structure** (**AC-VS**) with constant-size storage to reduce the search cost of the scheme while ensuring forward and backward privacy. Dou et al. [50] proposed a robust forward and backward privacy scheme to adapt to more complex update and query processes. Chen et al. [51] utilized blockchain and hash-proof chain technologies to build a publicly verifiable DSSE scheme and designed a new data hiding structure that provides both forward and backward privacy.

It is clear that single-keyword search cannot meet the query requirements in real-world systems. The capability of conjunctive keyword search enhances the application of DSSE schemes in practical scenarios [52]. Cash et al. [53] introduced the first static conjunctive search scheme, called OXT, which lacks the functionality for data updates. To this end, Patranabis et al. [27] proposed three dynamic conjunctive searchable encryption schemes: Mitra$_{Conj}$, BDXT, and ODXT. Mitra$_{Conj}$ is an extension of the single-keyword DSSE scheme Mitra [22]. Its idea is to run single-keyword searches multiple times and take the intersection to achieve conjunctive search, but its computational and communication costs are proportional to the number of updates for each keyword in the conjunction. This overhead is significant when the number of keywords or updates is high. BDXT optimized Mitra$_{Conj}$'s search cost by making its computational cost only related to the keyword

Table 2. Comparison of Existing Schemes with Proposed Scheme

Scheme	Query Type	Atrribute Search	Fine-grained Search	User Update	Attribute Access Contol	Update cost	Search cost	Forward Privacy	Backward Privacy
[16]	Single	✗	✗	✗	✗	$\widetilde{O}(\log^2 P)$	$\widetilde{O}(a_w \log P + \log^3 P)$	✓	I
[16]	Single	✗	✗	✗	✗	$O(1)$	$O(n_w d_w)$	✓	III
[50]	Single	✗	✗	✗	✗	$O(\|D\|)$	$O(a_w \|D\|)$	✓	I$^-$
[24]	Single	✗	✓	✗	✓	$O(1)$	$O(\|D\|^2)$	✓	II
[29]	Conjunctive	✗	✗	✗	✗	$O(a_f)$	$O(a_{w_{min}})$	✓	I$^-$
[18]	Single	✓	✓	✗	✗	$O(P)$	$O(a_w)$	✓	✗
[20]	Single	✗	✓	✓	✓	$O(\|s\|)$	$O(a_w)$	✓	✗
[19]	Single	✓	✗	✓	✗	$O(P)$	$O(a_w)$	✓	✗
[30]	Conjunctive	✗	✗	✗	✗	$O(1)$	$O(na_{w_{min}})$	✓	✗
[54]	Conjunctive	✗	✗	✗	✗	$O(\log P \log_k P)$	$O(na_{w_{min}} \log P \log_k P)$	✓	I
[44]	Conjunctive	✗	✗	✗	✗	$O(1)$	$O(na_{w_{min}})$	✓	II
[27]	Conjunctive	✗	✗	✗	✗	$O(1)$	$O(na_{w_{min}})$	✓	II
Ours	Conjunctive	✓	✓	✓	✓	$O(\|s\|)$	$O(na_{w_{min}})$	✓	I$^-$

P is the number of keyword/document pairs, $|D|$ is the number of total files, and $|s|$ is the number of authorized user. For keyword w, n_w is the number of results currently matching w, a_w is the total number of keyword updates, d_w is the number of deleted operations for w, $a_{w_{min}}$ is the number of the least update keyword in $q = (w_1 \wedge w_2 \wedge, \ldots, \wedge w_n)$, and n is the number of keyword in conjunctive query q. a_f is the number of updates for f. The notion $\widetilde{O}$ hides polylog factors, and hence $\widetilde{O}(A) > O(A)$.

with the least updates, but still has issues with high latency and multi-round communication. ODXT further improved BDXT by using oblivious computation protocols. It is currently an effective scheme for conjunctive search with forward and backward privacy. Chen et al. [29] designed a conjunctive search DSSE scheme DSSE-DC with a revocation mechanism using the idea of inner product matching. Guo et al. [30] constructed a forward index using a t-puncturable pseudorandom function and combined it with an inverted index to achieve conjunctive keyword search. Li et al. [54] introduced the notion of an update counter to construct a new bi-directional index structure, which enables conjunctive queries over bipartite graphs. Their scheme also proposed a new oblivious data structure to store the bi-directional index and used a semantically secure encryption scheme to encrypt node information, providing the proposed scheme with forward privacy and Type-I backward privacy. Yuan et al. [55] introduced the first sub-linear KPRP-hiding conjunctive DSSE scheme with forward and backward privacy, enabled by a novel cryptographic primitive, **Attribute-updatable Hidden Map Encryption** (**AUHME**). Li et al. [56] designed a highly balanced binary tree data structure called the **Indistinguishable Binary Tree** (**IBtree**) to achieve conjunctive keyword search. In addition, many other works have focused on extending DSSE with various query functionalities [57–59]. For instance, to enable fuzzy search, Li et al. [57] introduced a two-dimensional inner product relation to construct a wildcard searchable encryption scheme. Liu et al. [58] proposed a partitioning strategy to implement range searches on large-scale databases while using order-weighted inverted indexes and bitmap structures to enhance the efficiency and security of the scheme.

However, as mentioned before, these existing schemes cannot be directly applied to cloud-IoT healthcare systems based on attribute value-type databases. Table 2 presents a comparison of our scheme with existing advanced DSSE schemes in terms of security, functionality, and efficiency.

3 Preliminaries

3.1 System Model

Figure 1 depicts the system model we considered, which consists of four entities: patients, cloud server, doctors, and hospital.

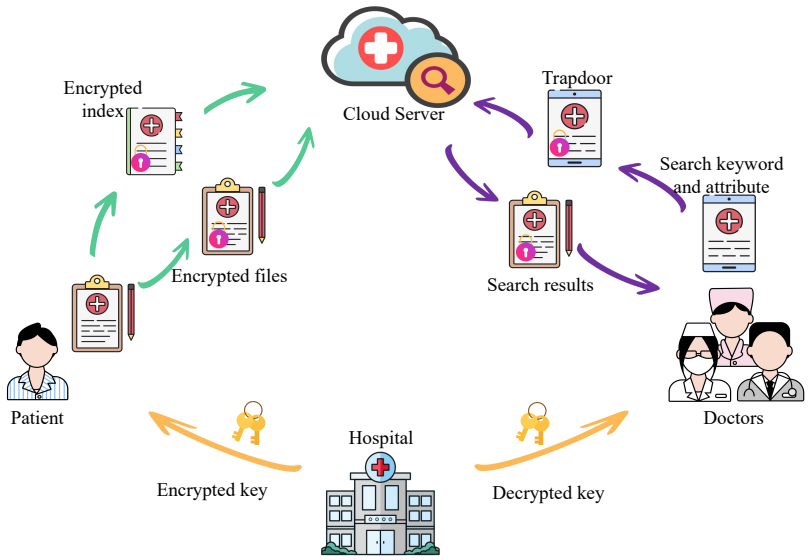


Fig. 1. Cloud-IoT healthcare system model of our proposed scheme.

Patients: As data owners, patients upload the EHRs collected through wearable IoT devices to the cloud server. To protect data privacy and facilitate subsequent search operations by doctors, patients first encrypt the EHRs using a symmetric encryption algorithm and construct the corresponding encrypted index. Then, they upload the encrypted files along with the secure index to the cloud server. Finally, when doctors perform search operations, patients share the relevant keys with the doctors through the hospital.

Doctors: As data users, doctors retrieve EHRs files stored on the cloud server to study and diagnose patients' conditions. To retrieve the required EHRs, doctors need to generate a search trapdoor and send them to the cloud server. It is necessary to filter out the unauthorized access portion from the search results locally and prepare for subsequent fine-grained searches.

Cloud Server: The cloud server is responsible for storing data and performing retrieval operations. Upon receiving a trapdoor sent by the doctor, the server executes the search protocol based on the secure index and returns the final search results to the doctor.

Hospital: The hospital is responsible for generating encrypted and decrypted keys and distributing them to patients and doctors, respectively. Only patients and hospital-authorized doctors can perform encryption and decryption operations on the files.

Note: Our goal is to design a forward and backward conjunctive DSSE scheme with flexible search capabilities and high privacy for cloud-IoT healthcare systems. The key point is how to design a secure index architecture for the attribute-based database in the cloud-IoT health system, so that doctors can implement multiple search functions while minimizing the privacy leakage of patients during the update and search process. Like all existing DSSE schemes, the security of our scheme relies on a "strong" assumption that the client's key can always be protected and will not be compromised [60]. Therefore, we did not discuss the key distribution and sharing issues in the article.

3.2 Threat Model

In our scheme, the patients and doctors are legal. To protect private information, they execute the protocol honestly and never expose the keys to any unauthorized entities. We also assume the hospital is a fully trusted third party. The hospital's functions are to generate and distribute

Table 3. Notations and their Descriptions

Notations	Descriptions
f_i	The i-th EHRs file
λ	The security parameter
DB	The attribute-value-type database
EDB	The encrypted database
w	The keyword
att	The attribute in EHRs file
W_{att}	The set of keywords under the attribute att $W_{att} = \{W_{att,f_1}, \dots, W_{att,f_m}\}$
$W_{att,f}$	The set of keywords under the attribute att in f
$\mathcal{W}$	The set of keyword $\mathcal{W} = \{W_{att_1}, \dots, W_{att_v}\}$
$\lvert\mathcal{W}\rvert$	The total number of keywords
ATT	The set of attributes $ATT = \{att_1, \dots, att_v\}$
bs	The bit string of bitmap index
e	The bit string of encrypted bitmap index
Sum_e	The sum of bit strings for encrypted bitmap index
m	The maximum number of EHRs files that the database DB can hold
$V_{f,att}$	The value of attribute att in file f
q	The conjunctive search query $q = \{(w_1 \lVert att_1) \wedge, \dots, \wedge (w_n \lVert att_n)\}$
$\lVert$	The concatenation of strings
$x \to X$	x is uniformly and randomly sampled from X
$c_{w\lVert att}$	The counter for the update frequency of w under att
c_{att}	The counter for the update frequency of attribute att
C	The map stores the counter
T	The map stores the secure index and corresponding encrypted file
X	The map stores the cross-tag

keys, authenticate identities, reduce the burden of key storage, and provide secure communication within the system. The cloud server is assumed to be honest but curious. This means the server will perform updates and retrieval operations honestly and return the correct search results to the doctor. However, it may also attempt to learn more valuable information during these processes.

3.3 Notations

Table 3 introduces the notations used in this article.

3.4 Bitmap Index

We use a bitmap index to represent the file identifiers in our scheme. Specifically, within a bit string bs of length l (l is the maximum number of files the database can support), the ith bit being 1 indicates the existence of file f_i, while 0 indicates the file's absence. Figure 2 shows an example supporting six files, i.e., $l = 6$. Figure 2(a) shows the initial state of the bit string bs, which indicates that files f_1 and f_3 exist. If we wish to add file f_0, we need to perform the operation as shown in Figure 2(b). Specifically, we need to generate the bit string for f_0, which is $2^0 = (000001)_2$, and add it to the initial state of the bit string bs, $(001010)_2$, resulting in an updated state of $(001011)_2$. If we want to delete file f_3 from the current state of the bit string, we must perform the operation as shown in Figure 2(c). We need to generate the bit string for f_3, which is $-2^3 = -(001000)_2$. Since

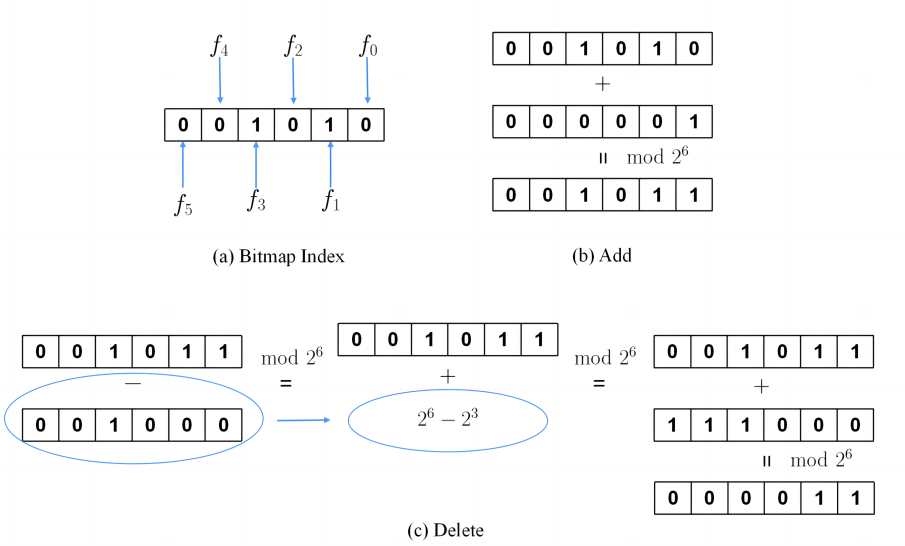


Fig. 2. Construction and operation rules of bitmap index.

we are performing a modulo 2^6 operation (the modulus is related to the length of the bit string l), -2^3 is converted to $2^6 - 2^3 = (111000)_2$.[1] Then, $(111000)_2$ is added to the current state $(001011)_2$. Please note that the addition and deletion operations of the bitmap index can be implemented through modular addition. The reason for this design is to enable the bitmap index to be encrypted (homomorphically) and updated (to reflect the addition and deletion of files) using encryption with homomorphic properties.

Remark. In the proposed scheme, we use one bit to indicate whether f_i exists. Multiple bits can also be used to indicate the existence of f_i in different applications. For details, please refer to the reference [28].

3.5 Dynamic Symmetric Searchable Encryption (DSSE)

A DSSE scheme Σ runs between the user and the server and consists of one algorithm **Setup** and two protocols **Update**, **Search**. The specific definitions are as follows:

Setup$(\lambda, DB) \leftarrow (K, \sigma, EDB)$: The algorithm takes security parameters λ and the original database DB as input and outputs the secret key K, a local state σ for the user, and an encrypted database EDB stored in the server.

Search$(K, q, \sigma; EDB) \leftarrow (\sigma', R; EDB')$: The search protocol runs between the user and the server and is responsible for executing keyword-based search queries on the database. Specifically, the user inputs (K, q, σ) and outputs a modified state σ' and a search result R returned by the server, where q is query request. The server takes the encrypted database EDB as input and outputs a modified database EDB.

Update$(K, \sigma, op, in; EDB) \leftarrow (\sigma'; EDB')$: The update protocol runs between the user and the server and is responsible for updating the database EDB. Specifically, the user inputs (K, σ, op, in) and outputs a modified state σ', where op denotes the update type and in denotes the update index information. The server takes the EDB as input and outputs a modified encrypted database EDB'.

[1]Modular operations typically return a non-negative result. When dealing with modular operations involving negative numbers, the result can be ensured to be non-negative by adding the negative number to the modulus and then performing the modular operation.

The adaptive security of a DSSE scheme $\sum$ = (**Setup, Update, Search**) is determined by a real-ideal game model. The real game $\mathrm{R}_{\mathrm{EAL}}{}^{\Sigma}_{\mathcal{A}}(\lambda)$ is consistent with the DSSE scheme itself, while the ideal game $\mathrm{I}_{\mathrm{DEAL}}{}^{\Sigma}_{\mathcal{A},\mathcal{S}}(\lambda)$ reflects the behavior of treating the information leaked by DSSE as input to the simulator $\mathcal{S}$. Adversary $\mathcal{A}$ can learn the information leaked by the DSSE scheme through a leakage function $\mathcal{L} = (\mathcal{L}^{Stp}, \mathcal{L}^{Updt}, \mathcal{L}^{Srch})$, where $\mathcal{L}^{Stp}$, $\mathcal{L}^{Updt}$, and $\mathcal{L}^{Srch}$ are used to capture the leakages in the **Setup**, **Update**, and **Search** protocols, respectively. The definitions of $\mathrm{R}_{\mathrm{EAL}}{}^{\Sigma}_{\mathcal{A}}(\lambda)$ and $\mathrm{I}_{\mathrm{DEAL}}{}^{\Sigma}_{\mathcal{A},\mathcal{S}}(\lambda)$ are as follows:

$\mathrm{R}_{\mathrm{EAL}}{}^{\Sigma}_{\mathcal{A}}(\lambda)$: It runs **Setup** when the adversary $\mathcal{A}$ chooses a database DB. After $\mathcal{A}$ obtains encrypted database EDB, the game calls **Update** or **Search** to respond to the adversary's queries. Finally, $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$.

$\mathrm{I}_{\mathrm{DEAL}}{}^{\Sigma}_{\mathcal{A},\mathcal{S}}(\lambda)$: The simulator $\mathcal{S}$ takes $\mathcal{L}^{Stp}$ as input and executes. When $\mathcal{A}$ generates update or search query, $\mathcal{S}$ replies by running $\mathcal{L}^{Updt}$ or $\mathcal{L}^{Srch}$. Finally, $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$.

Definition 1 (L-adaptive security). A DSSE scheme is $\mathcal{L}$-adaptively-secure, if for any **probabilistic polynomial-time** (**PPT**) adversary, there exists an efficient simulator $\mathcal{S}$ contenting

$$\left|\Pr\left(\mathrm{R}_{\mathrm{EAL}}{}^{\Sigma}_{\mathcal{A}}(\lambda)=1\right) - \Pr\left(\mathrm{I}_{\mathrm{DEAL}}{}^{\Sigma}_{\mathcal{A},\mathcal{S}}(\lambda)=1\right)\right| \leq negl(\lambda). \tag{1}$$

Forward privacy. Zhang et al. [13] proposed a file injection attack against the DSSE scheme. This attack leverages matching between previous search tokens and leakage about newly added files during the update to recover keywords. Forward privacy ensures that the newly inserted files cannot be linked with the previous search queries. This means that the newly added data will not reveal the previous query pattern during the update, and the adversary cannot infer the query content by correlating the previous and new data. Therefore, a DSSE scheme with forward privacy can effectively resist file injection attacks.

Definition 2 (Forward privacy). An $\mathcal{L}$-adaptive security DSSE scheme is forward private, if leakage function $\mathcal{L}^{Updt}$ can be written as following form:

$$\mathcal{L}^{Updt}(op, (w, bs)) = \mathcal{L}'(op, bs), \tag{2}$$

where $\mathcal{L}'$ is stateless functions, $op = \{add/del\}$ is operation type.

Backward privacy. Backward privacy ensures that the subsequent search queries cannot match files that have been deleted previously. That is, subsequent search queries should not reveal the existence or contents of the file that has already been deleted from the database. There are four types of backward privacy: Type-I$^-$, Type-I, Type-II, and Type-III. The security decreases from Type-I$^-$ to Type-III [17, 29]. We focus on Type-I$^-$ backward privacy defined in [17]. This kind of backward privacy allows the scheme to leak the files currently containing w, the total number of keyword updates and the timestamp for each update. Based on the leakage of Type-I$^-$ backward privacy, Type-I and Type-II additionally leak that the insertion times of matching files, and Type-III further reveals the timestamps of each deletion operation as well as the corresponding insertion operation. Before formally defining Type-I$^-$ backward privacy, we introduce several functions.

For a list of all queries Q, search pattern $sp(w) = \{t|(t, w) \in Q\}$ reveals the timestamps of all search queries on w, where t is the timestamp of the query, and (t, w) is a search query. $rp(w) = \{bs\}$ reveals the files containing w that have not yet been deleted. $Time(w) = \{t|(t, op, (w, bs)) \in Q\}$ leaks the timestamps of each update for w, where $(t, op, (w, bs))$ is update query.

Definition 3 (Backward privacy). An $\mathcal{L}$-adaptive security DSSE scheme is Type-I$^-$ Backward private, if leakage function $\mathcal{L}^{Updt}$, $\mathcal{L}^{Srch}$ can be written as following form:

$$\mathcal{L}^{Updt}(op, (w, bs)) = \mathcal{L}'(op), \tag{3}$$

$$\mathcal{L}^{Srch}(w) = \mathcal{L}''(sp(w), rp(w), Time(w)), \tag{4}$$

where $\mathcal{L}''$ is stateless functions.

3.6 Design Goals

To achieve efficient conjunctive searchable encryption for healthcare systems, the following design goals should be met.

Privacy preserving: In cloud-IoT healthcare systems, the privacy preserving is essential. The proposed scheme should ensure that apart from some necessary leakage, the cloud server cannot learn any additional information during the update and search. Forward and backward privacy are crucial to prevent the cloud server from inferring whether newly added files contain previously searched keywords as well as matching files that have been added and subsequently deleted. Combining forward privacy and backward privacy provides comprehensive protection for data, ensuring that sensitive information is not leaked during the addition, deletion, and search. Forward privacy and backward privacy are essential security properties for a DSSE scheme for cloud-IoT healthcare systems.

Flexible searchability: Flexible search capabilities are crucial for practical cloud-IoT healthcare systems. The proposed scheme should support exact conjunctive search and broad attribute search to meet the various requirements of users.

- Exact conjunctive search: The scheme allows for precise data retrieval by enabling searches over combinations of multiple attribute/keyword pairs.
- Broad attribute search: Users can perform a fast attribute search using only the attribute element. This search mode enables users to obtain a class of data.
- Fine-grained attribute-value search: Based on the above, beyond document-level retrieval, the scheme should also support fine-grained searches over specific attribute values.

These features should be integrated into the design of the scheme to provide flexibility and privacy protection in practical data retrieval scenarios.

Dynamic update: The proposed scheme should support dynamic update capabilities so that both EHRs files and users can be updated.

Access control: Given the sensitivity of EHRs, only authorized doctors with relevant permissions should be able to access them. Therefore, the proposed scheme should have access control capabilities.

Efficiency: Along with rich functionality and high security, search efficiency is also an important indicator to evaluate whether a DSSE scheme can be truly applied in practice.

4 PC-SE: Efficient Privacy-preserving Conjunctive Searchable Encryption

In this section, we present our PC-SE, an efficient conjunctive searchable encryption for cloud-IoT healthcare systems.

4.1 Overview of the Proposed Scheme

Before presenting the detailed construction, we briefly outline the main flow of our scheme. In PC-SE, we utilizes bitmap index as data structure to efficiently support both conjunctive keyword queries and attribute-value queries. The workflow begins with the data owner encrypting each EHR file using a symmetric encryption with homomorphic addition combined with modular addition, which conceals both the access pattern and the update type while preserving the ability to perform necessary computations directly over encrypted data. Note that in the scheme, we assume that all key management and distribution are handled by a trusted third-party hospital. Therefore, this issue is not discussed in the detailed construction. The index design further integrates an

oblivious computation protocol to protect query privacy and incorporates an attribute access mapping to enable efficient attribute-level queries and enforce access control policies. When a doctor submits a search request, the hospital sends the relevant key to the user and encrypts the query to generate a search token. The cloud server processes this token to locate encrypted entries in the index and returns only the matching results. This process supports expressive conjunctive searches across multiple attributes, such as retrieving records where attribute "past history" contains hypertension and attribute "family history" contains diabetes, as well as fine-grained attribute-value retrieval, ensuring that only the queried attribute's content is revealed while preventing unnecessary exposure of unrelated fields. Access control is enforced throughout the process by the attribute access mapping, which guarantees that each doctor can only retrieve information from attributes they are authorized to access, thereby enabling attribute-level restrictions. The scheme also supports secure dynamic updates, allowing insertion and deletion of EHR files while maintaining forward privacy and Type-I^{-} backward privacy. Through this workflow, PC-SE achieves strong privacy protection, expressive and flexible search capabilities, and practical efficiency for large-scale cloud-IoT healthcare systems.

4.2 Our Construction

To achieve the design goals presented in section 3.6, our scheme consists of three protocols: **Setup**, **Update**, and **Search**. Among them, **Update** includes file updates and user updates, while **Search** includes broad attribute search and exact conjunctive keyword-attribute search. Before presenting the details of these protocols, we first clarify the frequently used symbols and functions in the literature.

F_1, F_2, F_3, and F_p are **pseudorandom functions** (**PRFs**), H_1 and H_2 are hash functions. k_t, k_s, and k_{u_i} are λ-bit keys generated for F_1, F_2, and F_3, respectively. k_x, k_y, and k_z are keys generated for F_p. Let g be a uniformly sampled generator for the $p = p(\lambda)$ order cyclic group $\mathbb{G}$. $Setup$, Enc, Dec, and Add are algorithms in symmetric encryption with homomorphic addition Π [17, 29]. Specifically,

$\Pi.Setup(1^\lambda)$: A setup algorithm that takes the security parameter λ as inputs. It produces a message space N, where $N = 2^\mu$ and μ is the maximum number of files in the database.

$\Pi.Enc(k_e, I, N)$: An encryption algorithm that takes the random key k_e $(0 \le k_e < N)$, message I $(0 \le I < N)$ and a message space N as inputs. It produces the ciphertext $e = k_e + I \mod N$ as output, where the key k_e can be used only once and needs to be retained for decryption.

$\Pi.Add(e_1, e_2, N)$: A homomorphic addition algorithm that takes ciphertexts e_1, e_2 and a message space N as inputs. It produces $Sum_e = e_1 + e_2 \mod N$ as output, where $e_1 = Enc(k_{e_1}, I_1, N)$, $e_2 = Enc(k_{e_2}, I_2, N)$, $0 \le k_{e_1}, k_{e_2} < N$ and $0 \le I_1, I_2 < N$.

$\Pi.Dec(k_e, e, N)$: A decryption algorithm that takes the key k_e, ciphertext e and a message space N as inputs. It produces $I = e - k_e \mod N$ as output.

Correctness. The correctness of symmetric encryption with homomorphic addition Π means that given two ciphertexts $e_1 = k_{e_1} + I_1 \mod N$ and $e_2 = k_{e_2} + I_2 \mod N$, one can compute $Sum_e = e_1 + e_2 \mod N$. Meanwhile, one need to know the key $\hat{k_e} = k_{e_1} + k_{e_2} \mod N$ to decrypt Sum_e. In other words,

$$Dec(\hat{k_e}, Sum_e, N) = Sum_e - \hat{k_e} \mod N = I_1 + I_2 \mod N. \tag{5}$$

It is easy to see that Π achieves perfect security (**PS**) if each secret key is used only once. In this setting, a fresh secret key is chosen for every message encryption, which is conceptually analogous to the **one-time pad** (**OTP**). As a result, any ciphertext reveals no information about the corresponding plaintext to an adversary $\mathcal{A}$. The formal definition is presented below.

Perfect Security [61]. If for any PPT adversary $\mathcal{A}$, its advantage

$$\mathbf{Adv}^{\mathrm{PS}}_{\Pi,\mathcal{A}}(\lambda) = |\Pr[\mathcal{A}(\Pi.Enc(k_e, I_0, N)) = 1] - \Pr[\mathcal{A}(\Pi.Enc(k_e, I_1, N)) = 1], \tag{6}$$

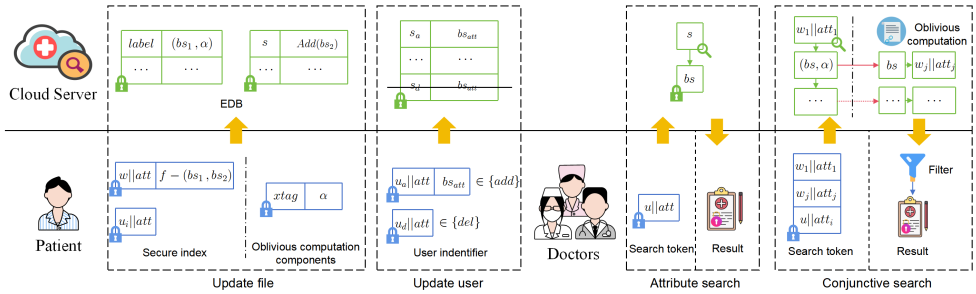


Fig. 3. The major function of PC-SE.

ALGORITHM 1: Setup

Input: Security parameter λ, and attribute file database DB.
Output: Security keys k_s, k_t, k_x, k_y, k_z, empty maps C, T, X and U, encrypted file database EDB.

$k_s, k_t, k_u, k_x, k_y, k_z \leftarrow \{0,1\}^\lambda$
$k_v \leftarrow Sym.Gen(1^\lambda)$
for $att \in DB$ **do**
  for $f \in DB$ **do**
    Construct keyword set W_{att}
    $Cip_{f,att} \leftarrow Sym.Enc(k_v, V_{f,att})$
  end for
end for
$C, T, X, U \leftarrow$ empty map
return $k_s, k_t, k_u, k_x, k_y, k_z, C, T, X, U$ and $\{Cip_{f,att}\}$

is negligible, then Π is perfectly secure, where $N \leftarrow \Pi.Setup(1^\lambda)$, the key k_e $(0 \le k_e < N)$ is kept secret and $\mathcal{A}$ chooses I_0, I_1 s.t. $0 \le I_0, I_1 < N$.

Algorithms 1–3, and 4-5 summarize our proposed protocols for **Setup**, **Update**, and **Search** respectively. Figure 3 shows the major function of our scheme in high level. The detailed explanations are as follows:

Setup. In this protocol, a patient generates a set of λ-bit keys $\{k_s, k_t, k_u, k_x, k_y, k_z\}$. Specifically, k_s is used to encrypt keywords and attributes, k_t is used to generate secure indexes, k_u is used to encrypt user identities and accessible attributes, and k_x, k_y, and k_z are used to generate elements related to oblivious shared computation. The hospital needs to generate a symmetric key k_v to encrypt/decrypt value in EHRs file. Then, the keyword set W_{att} of each attribute att in EHRs is extracted, and each value of EHRs is encrypted by symmetric encryption algorithm $Sym.Enc(\cdot)$. In addition, the patient initializes four empty maps C, T, X, U. Specifically, C is used to store counters for the update frequency of keyword and attribute, T and X store the secure index, and U keeps access authority (i.e., the bit string of file) for different users under different attributes, which can support users to quickly complete broad attribute search. Among them, C is held by the patient locally, and T, X, U are maintained by the cloud server.

Update. This protocol includes both file and user updates. Algorithm 2 is for file update. When updating an attribute-keyword/file pair $(w||att, f)$, the patient first checks map C and updates the update frequency counters $c_{w||att}$, c_{att} for the keyword-attribute $w||att$ and the attribute att, respectively. Next, the patient runs PRFs F_1, F_2, and F_3 as well as hash functions H_1 and H_2. Specifically, F_1 with key k_t is used to compute the location *label* for this update. F_1 with key k_s is

ALGORITHM 2: Update File

Input: Keyword w, attribute att, file f, bit string of file bs and user u_i.
Output: Fresh maps C, T, X and U.
Patient:
1: $c_{w||att} \leftarrow C[w||att]$, $c_{att} \leftarrow C[att]$
2: $c_{w||att} \leftarrow c_{w||att} + 1, c_{att} \leftarrow c_{att} + 1$
3: $label \leftarrow F_1(k_t, w||att||c_{w||att})$
4: $k_{w||att} \leftarrow F_1(k_s, w||att)$, $k_{att}||k'_{att} \leftarrow F_2(k_s, att)$
5: $k_{e_1} \leftarrow H_1(k_{w||att}, c_{w||att})$, $k_{e_2} \leftarrow H_2(k'_{att}, c_{att})$
6: $\{s\} \leftarrow \emptyset$
7: **for** all user u_i can access this attribute **do**
8: $s_i \leftarrow F_3(k_{u_i}, u_i||att))$, $\{s\} \leftarrow \{s\} \cup s_i$
9: **end for**
10: $e_1 \leftarrow Enc(k_{e_1}, bs_1, l_{bs_1})$, $e_2 \leftarrow Enc(k_{e_2}, bs_2, l_{bs_2})$
11: $\alpha \leftarrow F_p(k_y, f) \cdot (F_p(k_z, w||att||c_{w||att}))^{-1}$
12: $xtag \leftarrow g^{F_p(k_x, w||att) \cdot F_p(k_y, f)}$
13: $C[w||att] \leftarrow c_{w||att}$, $C[att] \leftarrow c_{att}$
14: Send $(label, e_1, e_2, \alpha, xtag, \{s\})$ to cloud server
Cloud Server:
15: $T[label] \leftarrow (e_1, \alpha)$, $X[xtag] \leftarrow 1$
16: **for** $s_i \in \{s\}$ **do**
17: $Sum_{e_2} \leftarrow Add(U[s_i], e_2, l_{bs_2})$
18: $U[s_i] \leftarrow Sum_{e_2}$
19: **end for**

also used to encrypt the keyword $w||att$ to generate the key $k_{w||att}$, and then H_1 is used to generate the homomorphic encryption key k_{e_1}. Similarly, F_2 with key k_s is used to encrypt the attribute att to generate the key k'_{att} and the encrypted attribute k_{att}, where k'_{att} is used to generate the homomorphic encryption key k_{e_2}, and k_{att} serves as an identifier for each attribute in the attribute value type database, used to locate specific attribute values in fine-grained search. F_3 with key k_u is used to compute the identity s of the authorized user associated with attribute att and adds it to the set $\{s\}$ (lines 1-9). Note that $s_i \in \{s\}$ acts as an index, allowing the server to find the attribute access authority corresponding to the user when the user issues a search query, and the authority are represented by bit string. This mechanism enables fine-grained access control without requiring users to store or manage individual versions of the database. Then, the patient adopts symmetric encryption with homomorphic addition with different keys k_{e_1} and k_{e_2} to encrypt the bit strings bs_1 and bs_2 of the updated file f, resulting in encrypted entries e_1 and e_2, where e_1 is related to the update frequency of the keyword-attribute $c_{w||att}$, and e_2 is related to the update frequency of the attribute c_{att} (lines 10-11). Note that bs_1 and bs_2 are identical, and $l_{bs_1}, l_{bs_2} = 2^{\mu}$, where μ is the maximum number of files the database can support. To implement conjunctive search, the patient also needs to prepare for the subsequent oblivious computation protocol. The protocol requires the method of blind exponentiation in a cyclic group of prime order (e.g., based on the Diffie-Hellman oblivious PRF). Specifically, F_p with keys k_x, k_y, and k_z is used to compute the blinding factor α and cross-tag $xtag$ so that the cloud server can learn whether the attribute in a fixed file contains all the keywords in conjunctive query (lines 11-12). Please note that conceptually, α is divided into two parts to be multiplied, one part is related to the file identifier f, and the other part is related to the keyword-attribute $w||att$ and its update count $c_{w||att}$. The $xtag$ is also divided into two parts to be multiplied by the exponent of g, one part is related to the keyword-attribute, and the other

ALGORITHM 3: Update User

Input: User u_d to be deleted and user u_a to be added.
Output: Updated map U.
Patient:
$\{del\}, \{add\}, \{Sum_e\} \leftarrow \emptyset$
for $att \in ATT$ **do**
  $s_d \leftarrow F_3(k_{u_d}, u_d \| att)), \{del\} \leftarrow \{del\} \cup s_d$
  $s_a \leftarrow F_3(k_{u_a}, u_a \| att)), \{add\} \leftarrow \{add\} \cup s_a$
  Generates bit string bs_{att} of the file that can be accessed, and encrypt it to obtain $Sum_{e_{att}}$
  $\{Sum_e\} \leftarrow \{Sum_e\} \cup Sum_{e_{att}}$
end for
Send $\{add\}, \{del\}, \{Sum_e\}$ to cloud server
Cloud Server:
for $s_a \in \{add\}, Sum_{e_{att}} \in \{Sum_e\}$ **do**
  $U[s_a] \leftarrow Sum_{e_{att}}$
end for
for $s_d \in \{del\}$ **do**
  Remove $U[s_d]$
end for
return U

part is related to the file identifier. This is a deliberate design choice, the effect of which will be manifested in the process of conjunctive search.

Upon receiving $(label, e_1, e_2, \alpha, xtag, \{s\})$ from the patient, the cloud server stores e_1 along with α in $T[label]$ and sets $X[xtag]$ to 1 (line 16). Unlike e_1, which is related to the keyword-attribute $w\|att$, e_2 is only related to the attribute att. Therefore, for each authorized user s_i, the cloud server adds the access authority information e_2 for the current update of the attribute att to Sum_{e_2} using homomorphic addition. Finally, the cloud server stores the file set Sum_{e_2} that user s_i currently has access to in map $U[s_i]$.

In addition, attributes serve as classification indicators that help organize database records into meaningful categories (e.g., patient name, age, disease type). The attribute set is initialized and considered static during the setup phase, meaning that the structure and types of attributes are predefined and known before the secure index is constructed. While it is technically possible to update the attribute set, such updates are not handled dynamically like keyword insertions or deletions. Instead, any addition or removal of attributes would typically occur during system reconfiguration or reinitialization. This design choice aligns with practical use cases, such as healthcare systems, where the attribute (e.g., medical record structure) is usually stable once deployed. By treating the attribute set as static, the system avoids frequent restructuring of secure indexes and ensures consistent query performance.

Algorithm 3 focuses on user update. When the patient needs a new doctor u_a to search the values of certain attributes in the EHRs files for diagnosis, the patient needs to send u_a's user identifier s_a for each attribute along with the relevant access permission information to the cloud server, which is then stored in U. If the doctor u_d is no longer responsible for the patient's treatment, the patient needs to send u_d's user identifier s_d for each attribute to the server, and then the server removes the corresponding access authority information from U.

Search. The search protocol includes broad attribute search and exact conjunctive keyword-attribute search. Algorithm 4 is for attribute search. When the doctor u wants to search for the value of attribute att, he/she needs to run F_3 with key k_u to compute the user's identifier s for att and send it to the cloud server. Then, the server directly retrieves the encrypted entries Sum_{e_2} from

ALGORITHM 4: Attribute Search

Input: Search attribute att and user u.
Output: The attribute value $Cip_{f_i,att}$.
Doctor:
1: $s \leftarrow F_3(k_u, u||att))$
2: Send s to cloud server
Cloud Server:
3: $Sum_{e_2} \leftarrow U[s]$
4: Send Sum_{e_2} to doctor
Doctor:
5: $Sum_{k_{e_2}} \leftarrow 0$, $k_{att}||k'_{att} \leftarrow F_2(k_s, att)$
6: **for** $i = 1$ to c_{att} **do**
7: $\quad k_i \leftarrow H_2(k'_{att}, i)$
8: $\quad Sum_{k_{e_2}} \leftarrow Sum_{k_{e_2}} + k_i \bmod l_{bs_2}$
9: **end for**
10: $bs_2 \leftarrow Dec(Sum_{k_{e_2}}, Sum_{e_2}, l_{bs_2})$
11: Send bs_2, k_{att} to cloud server
Cloud Server:
12: **for** $f \in bs_2$ **do**
13: $\quad$ Intersection of the row in database located by file identifier of f_i and the column in database located by encrypted attribute k_{att} is the search value $Cip_{f_i,att}$
14: **end for**
15: **return** $Cip_{f_i,att}$

$U[s]$ and returns it to the doctor (lines 3-4). Next, the doctor computes the key $Sum_{k_{e_2}}$ based on the counter c_{att} and the hash function H_2, and runs F_2 to obtain the encrypted attribute k_{att}. He/She uses $Sum_{k_{e_2}}$ to decrypt the encrypted entries Sum_{e_2} and obtain the access authority information bs_2 (lines 5-11). To perform fine-grained search, upon receiving (bs_2, k_{att}) from the doctor, the server uses the file identifiers contained in bs_2 to locate the corresponding rows in the EHRs database and uses the encrypted attribute k_{att} to locate the corresponding column in the EHRs database (refer to Table 1). The intersection of these rows and this column in EHRs database is the search value $Cip_{f_i,att}$ (lines 12-14). Finally, the cloud server returns the encrypted value to doctor (line 15), and the doctor can decrypt it with symmetric key k_v locally.

Algorithm 5 is for exact conjunctive keyword-attribute search. When doctor u issues a search query $q = (w_1||att_1) \wedge (w_2||att_2) \wedge \cdots \wedge (w_n||att_n)$, he/she first identifies the keyword-attribute with the least update frequency in the query q based on map C. Assuming that $(w_1||att_1)$ fulfill this condition, for each update involving $(w_1||att_1)$, the doctor first runs F_1 with key k_t to obtain the corresponding updated location $label_i$ and adds it to the set $\{label\}$. Then, to support the cloud server in completing the subsequent oblivious shared computation protocol, the doctor runs F_p with keys k_x and k_z to obtain the cross-token $xtoken_{i,j}$ of the remaining keyword-attribute pairs in the query q for this update and stores them in the set $\{xtoken_i\}$ (lines 1-10). The doctor also runs F_3 with key k_u to obtain the user's identifier associated with each attribute in q, and adds them to the set $\{s\}$ one by one. Finally, the doctor sends $(\{label\}, \{xtoken_i\}_{i=1,\ldots,c_{w_1||att_1}}, \{s\})$ to the cloud server.

Upon receiving the search trapdoor, the server performs the oblivious shared computation protocol to implicitly determine whether each update of $(w_1||att_1)$ contains the other terms from the query q. Specifically, the server first retrieves the corresponding encrypted entry e_1 and blinding factor α_i from map T based on each update location $label$ of $(w_1||att_1)$. To determine whether the each update of $(w_1||att_1)$ involves other terms in q, the cloud server performs an exponential operation

ALGORITHM 5: Conjunctive Keyword-Attribute Search

Input: Search query $q = \{(w_1||att_1) \wedge (w_2||att_2) \wedge, \ldots, \wedge(w_n||att_n)\}$ and user u.
Output: The attribute value.
Doctor:
1: Assume $(w_1||att_1)$ with the least updates
2: $c_{w_1||att_1} \leftarrow C[w_1||att_1]$, $\{label\} \leftarrow \emptyset$
3: $\{xtoken_1\}, \ldots, \{xtoken_{c_{w_1||att_1}}\} \leftarrow \emptyset$
4: **for** $i = 1$ to $c_{w_1||att_1}$ **do**
5: $label_i \leftarrow F_1(k_t, w_1||att_1||i)$, $\{label\} \leftarrow \{label\} \cup label_i$
6: $\{label\} \leftarrow \{label\} \cup label_i$
7: **for** $j = 2$ to n **do**
8: $xtoken_{i,j} \leftarrow g^{F_p(k_x, w_j||att_j) \cdot F_p(k_z, w_1||att_1||i)}$, $\{xtoken_i\} \leftarrow \{xtoken_i\} \cup xtoken_{i,j}$
9: **end for**
10: **end for**
11: **for** $i = 1$ to n **do**
12: $s_i \leftarrow F_3(k_u, u||att_i))$, $\{s\} \leftarrow \{s\} \cup s_i$
13: **end for**
14: Send $(\{label\}, \{xtoken_i\}_{i=1,\ldots,c_{w_1||att_1}}, \{s\})$ to server
Cloud Server:
15: $Sum_{e_1} \leftarrow 0$, $Num, \{Sum_{e_2}\} \leftarrow \emptyset$
16: **for** $i = 1$ to $|\{label\}|$ **do**
17: $cnt_i \leftarrow 1$, $label_i \leftarrow \{label\}[i]$, $(e_{1_i}, \alpha_i) \leftarrow T[label_i]$
18: **for** $j = 2$ to n **do**
19: $xtoken_{i,j} \leftarrow \{xtoken_i\}[j]$, $xtag_{i,j} \leftarrow (xtoken_{i,j})^{\alpha_i}$
20: **if** $X[xtag_{i,j}] = 1$ **then**
21: $cnt_i \leftarrow cnt_i + 1$
22: **end if**
23: **end for**
24: **if** $cnt_i = n$ **then**
25: $Sum_{e_1} \leftarrow Add(Sum_{e_1}, e_{1_i}, l_{bs_1})$, $Num \leftarrow Num \cup i$
26: **end if**
27: **end for**
28: **for** $i = 1$ to $|\{s\}|$ **do**
29: $Sum_{e_{2_i}} \leftarrow U[s_i]$, $\{Sum_{e_2}\} \leftarrow \{Sum_{e_2}\} \cup Sum_{e_{2_i}}$
30: **end for**
31: Send $(Sum_{e_1}, Num, \{Sum_{e_2}\}$ to doctor
Doctor:
32: $Sum_{k_{e_1}}, Sum_{k_{e_2}} \leftarrow 0$
33: **for** $i = 1$ to $|Num|$ **do**
34: $k_i \leftarrow H_1(k_{w_1||att_1}, Num[i])$, $Sum_{k_{e_1}} \leftarrow Sum_{k_{e_1}} + k_i \bmod l_{bs_1}$
35: **end for**
36: $Res \leftarrow Dec(Sum_{k_{e_1}}, Sum_{e_1}, l_{bs_1})$
37: **for** $i = 1$ to $|\{Sum_{e_2}\}|$ **do**
38: **for** $j = 1$ to c_{att_i} **do**
39: $k_{i,j} \leftarrow H_2(k'_{att_i}, j)$, $Sum_{k_{e_2}} \leftarrow Sum_{k_{e_2}} + k_{i,j} \bmod l_{bs_2}$
40: **end for**
41: $bs_i \leftarrow Dec(Sum_{k_{e_2}}, Sum_{e_{2_i}}, l_{bs_2})$
42: $Res \leftarrow bs_i \cap Res$
43: **end for**
44: $\forall att \in q$ run Algorithm 4 (lines 11-15)

on the cross-token $xtoken_{i,j}$ along with the blinding factor α_i to obtain the cross-tag $xtag_{i,j}$. Note that this operation allows the cloud server to obliviously compute the $xtag$ that may have appeared during the file update, without learning the file identifier and the keyword-attribute. Whether the xtag exists in the map X will determine the search results. If $X[xtag_{i,j}] = 1$, it means that the current update of $(w_1 \| att_1)$ contains $(w_j \| att_j) \in q_{,j=2,\ldots,n}$ and the corresponding counter cnt_i will also be updated. Then, for the updates with $cnt_i = n$, the server adds the corresponding encrypted entry to Sum_{e_1} using homomorphic addition and adds the update timestamp to the set Num (lines 16-27). The server also retrieves the corresponding access authority information from U based on the user's identifier s_i from $\{s\}$ and returns $(Sum_{e_1}, Num, \{Sum_{e_2}\}$ to the doctor (lines 28-31). Finally, the doctor decrypts Sum_{e_1} and Sum_{e_2} to obtain Res and bs_i, where Res represents the search results in response to query q, and bs_i represents the files that are accessible for each attribute in q. To restrict to only the files that the user is authorized to access, it is also necessary to perform intersection operations on Res and bs_i locally (lines 32-43). To obtain the values corresponding to any attribute in q within the EHRs database, the subsequent fine-grained search interaction follows the same process as described in lines 11-15 of Algorithm 4.

4.3 Suitability of IoT Devices

In our system model, wearable IoT devices play a supporting role by collecting patient data and uploading it to the cloud server. The configuration of IoT devices does not impact the core search performance of our scheme, but discussing their suitability could affect the practical deployment of the scheme. We can analyze the suitability of the proposed scheme on IoT devices from the perspectives of storage, communication, and computational overhead during the data update.

The patient's IoT device stores the doctor's key k_{u_i} to compute for different doctors u_i who have access to authorized attributes att, $s_i = F_3(k_{u_i}, u_i \| att)$. We use AES-128 to implement a pseudo-random function $F_3(\cdot)$ in this process, so the key k_{u_i} size is 128 bits. According to the literature [62] and data published by the World Health Organization [63], the number of medical staff in a large general hospital averages between 2500-4000. Based on this data, the maximum key storage space required is only 39.1KB to 62.5KB. This storage requirement is minimal and well within the capabilities of modern IoT devices, which can easily handle this load. For computational, the primary tasks during data updates involve executing AES-128 algorithm, homomorphic encryption algorithm and group exponentiation operation. By selecting an efficient elliptic curve and using fixed-base exponentiation, the computational cost of group exponentiation is feasible for databases of a certain scale [27]. AES-128 algorithm and homomorphic encryption algorithm are designed based on symmetric cryptographic primitives, ensuring high efficiency and minimal computational overhead. They also have a computational complexity of $O(1)$, and due to its speed and simplicity, it is ideal for real-time data encryption and decryption on IoT devices. The lightweight nature of these operations ensures that our protocol can be implemented on a wide range of IoT devices without significant performance degradation. For communication, the patient's IoT device uploads the update trapdoor to the cloud server, which includes the secure index, ciphertext, and user identifier. The communication load is proportional to the size of the user identifier $O(|s|)$, where $|s|$ is the number of users. With 128-bit user identifiers and a hospital staff size of up to 4000, the communication load for each update is approximately 62.5KB. This is well within the communication capacity of typical IoT devices used in healthcare settings.

Based on the above analysis, our scheme can be well deployed on modern IoT devices, from low-power wearable health monitors to more powerful mobile medical devices, making it an ideal choice for healthcare systems.

4.4 Security Analysis

According to the definitions of security [17], our PC-SE scheme achieves forward and Type-I$^-$ backward privacy. Specifically, the cloud server learns $(label, e_1, e_2, \alpha, xtag, \{s\})$ during the update. Due to the pseudorandomness of the PRF F and the distinct inputs for each update, the values of $label, e_1, e_2$, and $xtag$ are indistinguishable from random values. In addition, the cloud server cannot obtain the information about the updated files through $\{s\}$. Therefore, our scheme leaks no useful information during the update, thereby ensuring forward privacy. For backward privacy, the cloud server learns a series of locations set $\{label\}$ related to $(w_1||att_1)$ during the search. These locations were observed by the server in the previous update, enabling it to obtain the timestamp of each update for $(w_1||att_1)$. In addition, for each update $(w_1||att_1, f)$, once the server computes $xtag$ observed in the previous updates, it can obtain the number of updates and corresponding timestamp for each $w_i||att_i, i = 2, \ldots, n$ in the file f. Apart from the above leakage, the server cannot learn any other valuable information. Therefore, our scheme achieves Type-I$^-$ backward privacy defined in [17].

We use the leakage function $\mathcal{L} = (\mathcal{L}^{Stp}, \mathcal{L}^{Updt}, \mathcal{L}^{Srch})$ defined in section 2.5 to describe the leakages as mentioned above. After the leakage is captured, the formal definition of our scheme's leakage functions are as follows:

$$\mathcal{L}^{Updt}(op, (w, bs)) = (\perp),$$
$$\mathcal{L}^{Srch}(q) = (sp(q), rp(q), Time(q)). \tag{7}$$

According to *Definition 2* of forward privacy and *Definition 3* of backward privacy, our scheme achieves forward privacy and Type-I$^-$ backward privacy.

Formally, we have the following theorem for PC-SE security.

Theorem 1. *(Security of PC-SE) Assume that F_1, F_2, and F_3 are secure PRFs, the decisional Diffie-Hellman (DDH) assumption [27] holds over the group $\mathcal{G}$, and H_1, H_2 are hash functions. The leakage functions $\mathcal{L}^{Stp} = (\perp)$, $\mathcal{L}^{Updt}(op, (w, bs)) = (\perp)$, and $\mathcal{L}^{Srch}(q) = (sp(q), rp(q), Time(q))$, where $sp(q) = \{i|(i, w)_{w \in q}\}$ and $rp(q) = \{bs\}$ are two common leakages in most existing schemes, known as search pattern and result pattern, while $Time(q) = \{i|(i, (w, bs)_{w \in q})\}$ denotes the update timestamps leaked during the search. The proposed PC-SE is an $\mathcal{L}$-adaptive-secure [16] SSE scheme with forward and backward privacy.*

Proof. We prove Theorem 1 using a real-ideal game model. This model starts with the real game $\mathrm{R}_{\mathrm{EAL}}()$ and reaches an ideal game $\mathrm{I}_{\mathrm{DEAL}}()$ by constructing a series of slightly different games. We need to demonstrate the indistinguishability between adjacent games in the sequence to show that the real game and the ideal game are also indistinguishable.

$Game_1$: The difference between $Game_1$ and the real game $\mathrm{R}_{\mathrm{EAL}}()$ is that $Game_1$ uses random functions G_1, G_2, G_3, G_x, G_y, and G_z, instead of the PRFs F_1, F_2, F_3 ,$F_p(k_x, \cdot)$, $F_p(k_y, \cdot)$ and $F_p(k_z, \cdot)$, to generate the relevant transcripts. Since the adversary cannot distinguish between the PRFs and the truly random functions, $Game_1$ and the real game are indistinguishable.

$Game_2$: The difference between $Game_2$ and $Game_1$ is that $Game_2$ uses a random oracle, instead of hash functions H_1 and H_2, to generate keys k_{e_1} and k_{e_2}. Taking H_1 and k_{e_1} as an example, k_{e_1} is randomly generated from $\{0, 1\}^\lambda$ and stored in map L during the update. Then, $\mathbf{H_1}[k_{w||att}||c_{w||att}] \leftarrow \mathbf{L}[k_{w||att}||c_{w||att}]$ is executed during the search, where $\mathbf{H_1}$ is the table for random oracle. Since $\mathbf{H_1}$ is not updated in time before the search query, if the adversary accesses $\mathbf{H_1}[k_{w||att}||c_{w||att}]$ at this point, it will obtain a random value $k^*_{e_1}$ generated by $\mathbf{H_1}$, which is different from k_{e_1}. When the adversary queries $\mathbf{H_1}[k_{w||att}||c_{w||att}]$ again after the search query, it will obtain k_{e_1}. Receiving different responses for two identical queries would make the adversary realize that it is in $Game_2$.

However, the probability of this situation is negligible. Specifically, the adversary does not know the λ-bit key $k_{w\|att}$, and the probability of guessing it correctly is $\frac{1}{2^\lambda} + negl(\lambda)$, where $negl(\cdot)$ is negligible function. Assuming a PPT adversary can make at most $p = poly(\lambda)$ queries, where $poly(\cdot)$ is polynomial function, the probability of guessing correctly is $p(\frac{1}{2^\lambda} + negl(\lambda))$, which is negligible. H_2 and k_{e_2} are similarly processed in this game. Therefore, $Game_2$ and $Game_1$ are indistinguishable.

$Game_3$: The difference between $Game_3$ and $Game_2$ is that $Game_3$ changes the way to generate *xtoken* during the search. Specifically, the challenger first collects update information about $w_1\|att_1$ from the history of update queries issued by the adversary. Then, it computes $xtag$ and α of the remaining $w_i\|att_i \in q$ for each update operation. Finally, it obtains $xtoken = xtag^{1/\alpha}$. In $Game_2$, $xtoken = g^{G_x(\cdot)\cdot G_z(\cdot)}$. It is clear that the distribution of each *xtoken* value in $Game_2$ is identical to the distribution of each *xtoken* value in $Game_3$. Therefore, $Game_3$ and $Game_2$ are indistinguishable.

$Game_4$: The difference between $Game_4$ and $Game_3$ is that $Game_4$ changes the way to generate α during the update. Specifically, the challenger generates blinding factor from $\mathbb{Z}_p^*$ using random sampling, i.e., $\alpha \xrightarrow{\$} \mathbb{Z}_p^*$. In $Game_3$, α is computed by $G_y(\cdot)$ and the inverse of $G_z(\cdot)$, where $G_y(\cdot)$ and $G_z(\cdot)$ are also uniformly sampled from the set of all random functions on $\mathbb{Z}_p^*$. Therefore, $Game_4$ and $Game_3$ are indistinguishable.

$Game_5$: The difference between $Game_5$ and $Game_4$ is that $Game_5$ changes the way to generate $xtag$ during the update. Specifically, the challenger computes cross-tag as $xtag = g^\gamma$, where g is an uniformly sampled generator for the group $\mathbb{G}$ and $\gamma \xrightarrow{\$} \mathbb{Z}_p^*$. In $Game_4$, $xtag = g^{G_x(\cdot)\cdot G_y(\cdot)}$. Since the DDH assumption holds in the group $\mathbb{G}$, the probability of a PPT adversary distinguishing $xtag = g^{G_x(\cdot)\cdot G_y(\cdot)}$ from $xtag = g^\gamma$ is negligible. Therefore, $Game_5$ and $Game_4$ are indistinguishable.

$Game_6$: The difference between $Game_6$ and $Game_5$ is that $Game_6$ changes the way to compute *label* during the update and search. Specifically, the challenger uses $G_1(t)$ to compute *label*, instead of $G_1(w\|att\|c_{w\|att})$ in $Game_5$, where t is the timestamp of update operation. Noting that the counter $c_{w\|att}$ and timestamp t are both monotonically increasing, the value of G_1 is computationally indistinguishable in $Game_6$ and $Game_5$. Therefore, $Game_6$ and $Game_5$ are indistinguishable.

$Game_7$: The difference between $Game_7$ and $Game_6$ is that $Game_7$ replaces *bs* with an all 0 bit string. Because of the perfect security of symmetric encryption with homomorphic addition, $Game_7$ and $Game_6$ are indistinguishable.

Simulator $\mathcal{S}$: $\mathcal{S}$ simulates the ideal game $\mathrm{I}_{\mathrm{DEAL}}(\lambda)$, but unlike $Game_7$, it generates a view based on the leakage functions $\mathcal{L} = (\mathcal{L}^{Stp}, \mathcal{L}^{Updt}, \mathcal{L}^{Srch})$. Specifically, $\mathcal{S}$ cannot gain any information about the update operations based on $\mathcal{L}^{Updt}$ and generates a series of variables in the same way as in $Game_7$. In the search phase, $\mathcal{S}$ first uses $\hat{q} \leftarrow \min \mathbf{sp}(q)$ to denote the first timestamp of search query q, and then it obtains the information leaked by $rp(q)$ and $Time(q)$. Note that the view generated by $\mathcal{S}$ using the above information is indistinguishable from the view in $Game_7$.

This completes the proof. □

4.5 Emergency

Our scheme aims at providing a more efficient and flexible search mode for medical staff in their daily work while ensuring patient privacy. That said, the scheme is not specifically designed for real-time emergency response. However, particularly urgent or severe situations are common in hospitals. Therefore, we propose some additional strategies to address emergencies.

— When a patient's life is at risk, the proposed scheme can also operate in a degraded mode. Specifically, we supports doctors to perform single-keyword searches on a small set of critical data (e.g., patient vitals or emergency records). By omitting resource-intensive operations

such as group exponentiation and oblivious computation protocols, this method significantly reduces computational overhead, ensuring faster data retrieval during critical moments. This degraded mode strikes a good balance between maintaining security and achieving real-time response.

— For some critical data attributes (e.g., recent medical history, allergies, prescriptions), they can be pre-cached on edge devices or IoT devices to ensure rapid retrieval when communication with the cloud is limited. Once the network is restored, the devices will automatically sync data with the cloud. This approach reduces reliance on real-time encrypted searches in the cloud, thereby minimizing delays and allowing the system to continue functioning effectively in cases of network constraints or outages, ensuring timely access to critical medical information.

5 Performance Evaluation

In this section, we evaluate the efficiency of our scheme on a simulated dataset, in comparison with existing state-of-the-arts.

5.1 Implementation Details

To evaluate the efficiency of PC-SE, we implement it in Python 3.10 using the PyCrypto library[2] and Sagemath library[3] for symmetric cryptographic operations and group-based operations. Specifically, we use AES-128/256 as PRF, SHA-256 as the hash function, and the elliptic curve Curve25519 [64] for group operations. For the implementation of oblivious computation protocols, we referred to the code provided by the authors of [27].[4] In addition, we reconstructed schemes in [18] and [20], which are forward-private DSSE schemes designed for cloud-IoT healthcare systems. Both schemes have the ability to perform fine-grained retrieval based on attribute type databases, and scheme in [18] can also achieve fast attribute retrieval. We compared them in terms of conjunctive search efficiency, attribute search efficiency, and update efficiency.

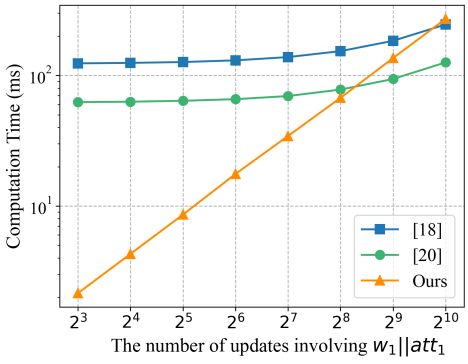
Platform and Dataset. The goal of our scheme is to enable efficient and privacy-preserving ciphertext retrieval, data updates, and access control on the cloud. IoT devices act as auxiliary roles to collect and upload patient data, and their configuration does not affect the core search performance of our scheme. All experiments are run on workstations configured with CPU Intel(R) Core(TM) i7-14700K 3.40 GHz RAM 32 GB and 16 GB, and the operating system is Windows 11 (64-bit), which are used to simulate cloud servers, users, and IoT devices.

We extended a larger simulated attribute base database based on the **Indian Liver Patient Dataset (ILPD)**.[5] The ILPD database includes 583 instances and 11 attributes. We increased the number of attributes to 20 to test the performance of each scheme on a larger database. All the experiments are repeated 10 times and the results are averaged over the ten runs.

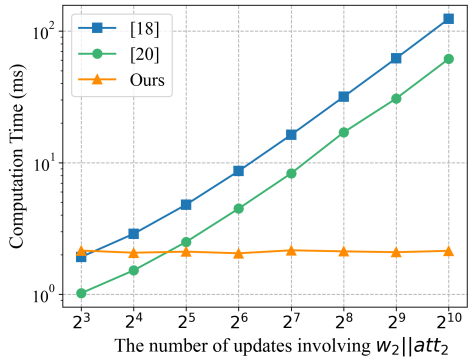
5.2 Efficiency of Conjunctive Search

We compare our PC-SE with the schemes [18, 20], which are advanced DSSE schemes designed for healthcare systems. Figures 4 and 5 compare the conjunctive search efficiency of the three schemes for $q = \{(w_1||att_1) \wedge (w_2||att_2)\}$ and $q = \{(w_1||att_1) \wedge \cdots \wedge (w_4||att_4)\}$, respectively, where $w_1||att_1$ has the least updates. Specifically, Figure 4(a) shows the impact of $w_1||att_1$ update frequency on search efficiency, by fixing $w_2||att_2$ update frequency to 2^{10}, while Figure 4(b) investigates the influence of $w_2||att_2$ update frequency on search efficiency, by fixing $w_1||att_1$ update frequency to 2^3. Similarly,

[2] https://pycryptodome.readthedocs.io/en/latest/
[3] http://www.sagemath.org/
[4] https://github.com/appmonster007/dsse-odxt-implementation
[5] http://archive.ics.uci.edw/ml/index.php

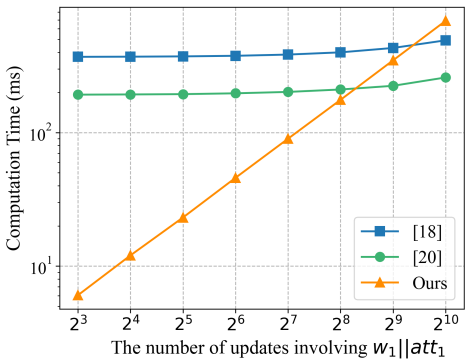


(a) $w_2||att_2$ updates fixed to 2^{10}

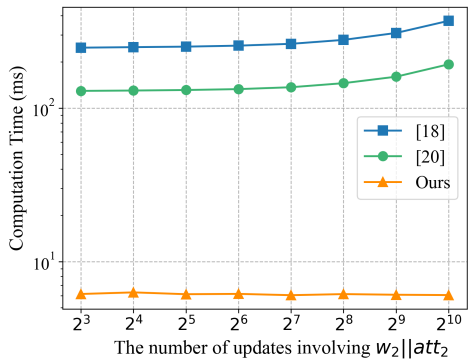


(b) $w_1||att_1$ updates fixed to 2^3

Fig. 4. Comparison of the efficiency for two-terms conjunctive search.



(a) $w_i||att_i$, $i = 2, 3, 4$, updates fixed to 2^{10}

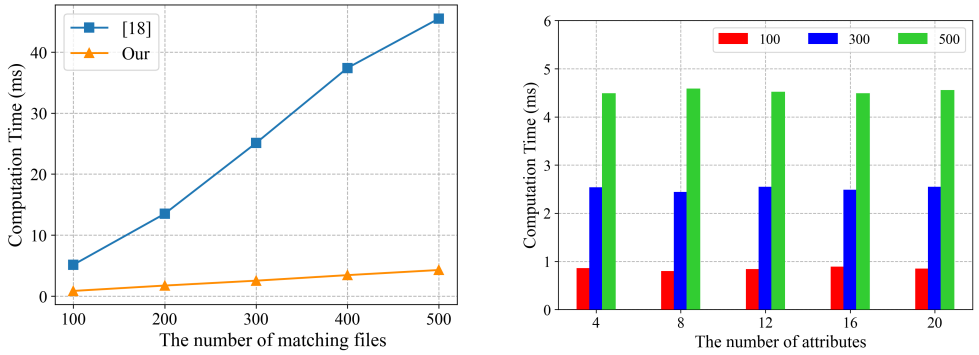


(b) $w_1||att_1$ updates fixed to 2^3, $w_i||att_i$, $i = 3, 4$, updates fixed to 2^{10}

Fig. 5. Comparison of the efficiency for four-terms conjunctive search.

Figure 5(a) illustrates the effect of $w_1||att_1$ update frequency on search efficiency, when the update frequencies of $w_2||att_2$, $w_3||att_3$ and $w_4||att_4$ are fixed at 2^{10}, while Figure 5(b) depicts the impact of $w_2||att_2$ update frequency on search efficiency, when $w_1||att_1$ update frequency is fixed at 2^3, and the update frequencies of $w_3||att_3$ and $w_4||att_4$ are fixed at 2^{10}.

The results of Figures 4 and 5 reveal that our scheme significantly outperforms the schemes of [18] and [20] in most cases. For example, for the two-terms conjunctive search given $w_1||att_1$ update frequency 2^3 and $w_2||att_2$ update frequency 2^{10}, our scheme takes about 2.2 ms, while the scheme [20] takes 65.5 ms and the scheme [18] requires 123.8 ms. Moreover, for the four-terms conjunctive search with $w_1||att_1$ update frequency fixed to 2^3 and $w_i||att_i$, $i = 3, 4$, update frequencies fixed to 2^{10}, our PC-SE is 21 to 32 times faster than the scheme [20] and 40 to 61 times faster than the scheme [18]. Only in some extreme and rare query cases when $w_1||att_1$ has a very high update frequency or $w_2||att_2$ has a very low update frequency, the schemes of [18] and [20] may match to or slightly outperform our scheme. This is because the burden generated by the group exponential computation of our scheme becomes evident in the above extreme cases. However, in extreme scenarios, we can enhance the performance of PC-SE by executing single keyword search for each keyword in the conjunction in parallel.



(a) Comparison of two schemes (b) PC-SE for different number of attributes

Fig. 6. Efficiency of attribute search.

5.3 Efficiency of Attribute Search

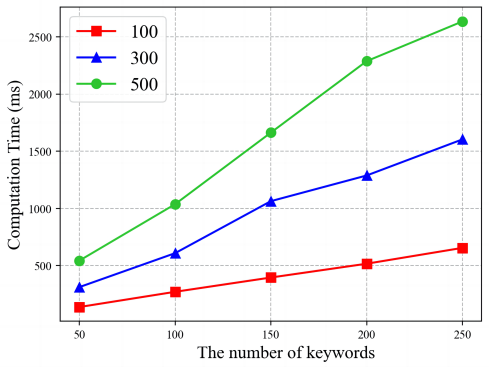
Since the scheme [20] lacks attribute search, we only compare the attribute search efficiency with the scheme [18]. Figure 6(a) depicts the efficiency of attribute search for our PC-SE and the scheme [18]. It is evident that the search time of our scheme is 5 to 10 times faster than that of the scheme [18]. For example, when the number of matching files is 500, our scheme takes about 4.3 ms, while the scheme [18] takes 45.5 ms. Figure 6(b) shows the search performance of our scheme when the database contains different numbers of attributes, where the different colored bars represent different numbers of matching files. It can be observed that when the number of matching files is fixed, increasing the number of attributes has almost no affect on the search efficiency of the proposed scheme.

5.4 Efficiency of Update

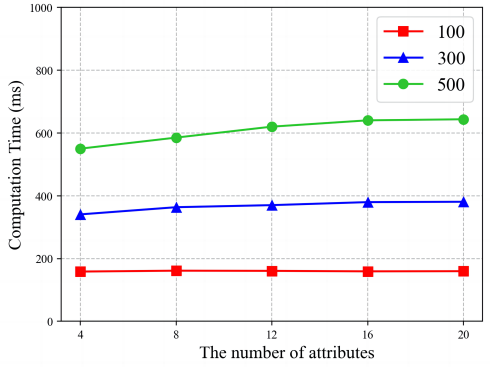
Figure 7(a) shows the update time of our scheme as the function of the number of keywords when the number of attributes is fixed, given three different numbers of updated files. As expected, the update time increases with the number of keywords or the number of files. Figure 7(b) shows the impact of the number of attributes on the update time when the number of keywords is fixed. The results of Figure 7(b) indicates that changing the number of attributes only has a slight effect on update time. This is due to the fact that the update operation cannot be performed when there are only attributes but lacking keywords. Figure 7(c) shows the efficiency of user update as the function of the number of users, given four different numbers of attributes/numbers of files. It is evident that update time scales linearly with the number of users, and both changing the number of attributes and changing the number of files have a significant impact on update time.

6 Conclusions and Future Works

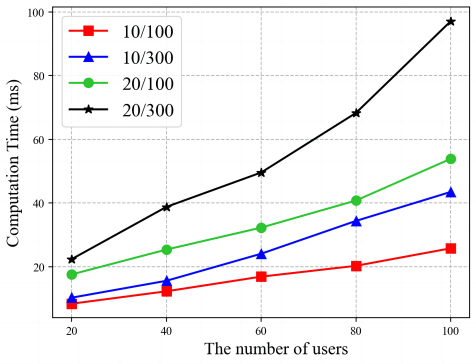
In this work, we explore how to design a DSSE scheme suited for cloud-IoT healthcare systems. We proposed the first conjunctive DSSE scheme based on attribute-value type database, named PC-SE. Our scheme not only can performs conjunctive keyword search and attribute search based on users' requirements, but also can maintain forward and backward privacy. To further address the additional communication burden and privacy concerns in attribute-value type databases, which widely existed in the previous DSSE scheme, our scheme also supports fine-grained search. Moreover, we realize the access control mechanism in PC-SE. The experimental results indicate that, compared to other latest DSSE schemes, PC-SE is effective and reliable.



(a) given three numbers of updated files



(b) given three numbers of updated files



(c) given four numbers of attributes/numbers of files

Fig. 7. The efficiency of updates for our scheme.

Regarding future work, we aim at enhancing the security and adaptability of DSSE schemes in more practical settings. First, after reviewing the latest and previous literature on SSE, we found that existing SSE schemes are all based on the assumption that the client key is always secure, and do not consider the security issues in key sharing between the client and the user. This is lacking in practical application scenarios. If the client's key is exposed, the adversary can easily compromise the encrypted database and observe the update and search process. Second, existing SSE schemes focus on how to reduce access pattern leakage without considering the impact of keyword guessing attacks that often appear in **public key searchable encryption** (**PEKS**) schemes on SSE schemes. Third, we will continue improving the flexibility of the search functionality, including the integration of fuzzy attribute-based search, as discussed in Section 3.6 Supporting partial attribute matching and wildcard queries will further enhance usability in real-world cloud-IoT healthcare scenarios. Finally, further research and optimization of DSSE's degraded mode in emergency situations will help the scheme handle more complex healthcare scenarios, making the proposed scheme an ideal choice for deployment in cloud-IoT healthcare systems.

These directions will help build a more practical, secure, and flexible DSSE framework for cloud-IoT healthcare systems.

References

[1] Y. Zhang, C. Xu, H. Li, K. Yang, J. Zhou, and X. Lin. 2018. HealthDep: An efficient and secure deduplication scheme for cloud-assisted eHealth systems. *IEEE Trans. Industrial Informatics* 14, 9 (2018), 4101–4112.

[2] G. Yang, L. Xie, M. Mäntysalo, X. Zhou, Z. Pang, L. D. Xu, S. Kao-Walter, Q. Chen, and L. R. Zheng. 2014. A health-IoT platform based on the integration of intelligent packaging, unobtrusive bio-sensor, and intelligent medicine box. *IEEE Trans. Industrial Informatics* 10, 4 (2014), 2180–2191.

[3] X. Fu, L. T. Yang, J. Li, X. Yang and Z. Yang 2024. A Searchable Symmetric Encryption-Based Privacy Protection Scheme for Cloud-Assisted Mobile Crowdsourcing. *IEEE Internet of Things Journal* 11, 2 (2024), 1910–1924.

[4] Muhammad Waqas, Shanshan Tu, Zahid Halim, Sadaqat Ur Rehman, Ghulam Abbas, and Ziaul Haq Abbas. 2022. The role of artificial intelligence and machine learning in wireless networks security: Principle, practice and challenges. *Artificial Intelligence Review* 55, 2022 (2022), 5215–5261.

[5] Q. Wang. Q. Jiang, Y. Yang and J. Pan 2022. The burden of travel for care and its influencing factors in China: An inpatient-based study of travel time. in *Journal of Transport and Health* 25, 2022 (2022), 101353.

[6] B. Gong, C. Guo, C. Guo, Y. Sun, M. Waqas, and S. Chen, 2024. SLIM: A secure and lightweight multi-authority attribute-based signcryption scheme for IoT. In *IEEE Transactions on Information Forensics and Security* 19 (2024), 1299–1312.

[7] J. Shu, X. Jia, K. Yang, and H. Wang 2021. Privacy-preserving task recommendation services for crowdsourcing. *IEEE Trans. Services Computing* 14, 1 (2021), 235–247.

[8] C. Zhang, L. Zhu, C. Xu, J. Ni, C. Huang, and X. Shen. 2022. Location privacy-preserving task recommendation with geometric range query in mobile crowdsensing. *IEEE Trans. Mobile Computing* 21, 12 (2022), 4410–4425.

[9] D. X. Song, D. Wagner, and A. Perrig 2000. Practical techniques for searches on encrypted data. In *Proceedings of the 2000 IEEE Symp. Security and Privacy.* (Berkeley, CA, USA), 44–55.

[10] S. Kamara, C. Papamanthou, and T. Roeder 2012. Dynamic searchable symmetric encryption. In *Proceedings of the CCS 2012.* (Raleigh, NC, USA), 965–976.

[11] I. Demertzis, J. G. Chamani, D. Papadopoulos, and C. Papamanthou 2020. Dynamic searchable encryption with small client storage. In *Proceedings of the NDSS 2020.* (San Diego, CA, USA), 1–17.

[12] S. F. Sun, R. Steinfeld, S. Lai, X. Yuan, A. Sakzad, J. K. Liu, S. Nepal, and D. Gu. 2012. Practical non-interactive searchable encryption with forward and backward privacy. In *Proceedings of the Usenix Network and Distributed System Security Symposium 2021.* 1–18.

[13] Y. Zhang, J. Katz and C. Papamanthou 2016. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In *Proceedings of the 25th USENIX Security Symposium.* (Austin, TX, USA), 707–720.

[14] E. Stefanov, C. Papamanthou, and E. Shi 2014. Practical dynamic searchable encryption with small leakage. In *Proceedings of the NDSS 2014.* (San Diego, CA, USA), 1–15.

[15] R. Bost 2016. $\sum o\phi o\xi$: Forward secure searchable encryption. In *Proceedings of ACM SIGSAC Conference on Computer and Communications Security.* (Vienna, Austria), 1143–1154.

[16] R. Bost, B. Minaud, and O. Ohrimenko 2017. Forward and backward private searchable encryption from constrained cryptographic primitives. In *Proceedings of the CCS 2017.* (Dallas, TX, USA), 1465–1482.

[17] Q. Gan, C. Zuo, J. Wang, S. F. Sun, and X. Wang. 2019. Dynamic searchable symmetric encryption with forward and stronger backward privacy. In *Proceedings of the ESORICS 2019. (Luxembourg).* 283–303.

[18] S. Li, C. Xu, Y. Zhang, Y. Du, X. Wen, K. Chen, and J. Ma. 2022. Efficient data retrieval over encrypted attribute-value type databases in cloud-assisted ehealth systems. *IEEE Systems Journal.* 16, 2 (2022), 3096–3107.

[19] S. Li, C. Xu, Y. Zhang, and X. Wen 202. Multi-user dynamic symmetric searchable encryption for attribute-value type database in cloud storage. In *Proceedings of the Security and Privacy in Digital Economy. Communications in Computer and Information Science, Springer* 1268 (2020), 355–368.

[20] Y. Liu, J. Yu, J. Fan, P. Vijayakumar, and V. Chang. 2022. Achieving privacy-preserving DSSE for intelligent IoT healthcare system. *IEEE Trans. Industrial Informatics* 18, 3 (2022), 2010–2020.

[21] K. Wang, C. M. Chen, Z. Tie, M. Shojafar, S. Kumar, S. Kumari. 2022. Forward privacy preservation in IoT-enabled healthcare systems. *IEEE Trans. Industrial Informatics* 18, 3 (2022), 1991–1999.

[22] J. G. Chamani, D. Papadopoulos, C. Papamanthou, and R. Jalili 2018. New constructions for forward and backward private symmetric searchable encryption. In *Proceedings of the CCS 2018.* (Toronto, ON, Canada), 1038–1055.

[23] F. Li, J. Ma, Y. Miao, Z. Liu, K. K. R. Choo, X. Liu, and R. H. Deng. 2023. Towards efficient verifiable boolean search over encrypted cloud data. *IEEE Transactions on Cloud Computing* 11, 1 (2023), 839–853.

[24] H. Li, Y. Yang, Y. Dai, S. Yu, and Y. Xiang. 2020. Achieving secure and efficient dynamic searchable symmetric encryption over medical cloud data. *IEEE Trans. Cloud Computing* 8, 2 (2020), 484–494.

[25] S. Tu, M. Waqas, A. Badshah, M. Yin and G. Abbas 2023. Network intrusion detection system (NIDS) based on pseudo-siamese stacked autoencoders in fog computing. In *IEEE Transactions on Services Computing* 16, 6 (2023), 4317-4327.

[26] X. Wu, B. Zou, C. Lu, L. Wang, Y. Zhang, and H. Wang. 2025. Dynamic security computing framework with zero trust based on privacy domain prevention and control theory. In *IEEE Journal on Selected Areas in Communications* 43, 6 (2025), 2266–2278.

[27] S. Patranabis and D. Mukhopadhyay 2021. Forward and backward private conjunctive searchable symmetric encryption. In *Proceedings of the NDSS 2021*. 1–18.
[28] C. Zuo, S. F. Sun, J. K. Liu, J. Shao, J. Pieprzyk, and G. Wei. 2020. Forward and backward private dynamic searchable symmetric encryption for conjunctive queries. *Cryptology ePrint Archive* 1357 (2020), 1–13.
[29] L. Chen, J. Li and J. Li 2023. Toward forward and backward private dynamic searchable symmetric encryption supporting data deduplication and conjunctive queries. In *IEEE Internet of Things Journal* 10, 19 (2023), 17408–17423.
[30] C. Guo, W. Li, X. Tang, K. -K. R. Choo and Y. Liu 2023. Forward private verifiable dynamic searchable symmetric encryption with efficient conjunctive query. *IEEE Trans. Dependable and Secure Computing* 21, 2 (2023), 746–763.
[31] P. Hao, Z. Yan and H. Wen 2025. Privacy-preserving NILM: A self-alignment source-aware domain adaptation approach. In *IEEE Transactions on Instrumentation and Measurement* 74, (2025), 1-12.
[32] Y. Gong, H. Yao, X. Liu, M. Bennis, A. Nallamathan, Z. Han 2024. Computation and privacy protection for satellite-ground digital twin networks. In *IEEE Transactions on Communications* 72, 9 (2024), 5532–5546.
[33] S. Tu, A. Badshah, H. Alasmary and M. Waqas 2024. EAKE-WC: Efficient and anonymous authenticated key exchange scheme for wearable computing. In *IEEE Transactions on Mobile Computing* 23, 5 (2024), 4752–4763.
[34] E. J. Goh 2003. Secure indexes. *Cryptology ePrint Archive*, (2003). Retrieved from https://eprint.iacr.org/2003/216
[35] Y. C. Chang and M. Mitzenmacher 2005. Privacy preserving keyword searches on remote encrypted data. In *Proceedings of the ACNS 2005*. (New York, NY, USA), 442–455.
[36] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky 2011. Searchable symmetric encryption: Improved definitions and efficient constructions. *J. Computer Security* 19, 5 (2011), 895–934.
[37] J. Li, X. Lin, Y. Zhang and J. Han 2017. KSF-OABE: Outsourced attribute-based encryption with keyword search function for cloud storage. *IEEE Transactions on Services Computing* 10, 5 (2017), 715–725.
[38] H. Gao, H. Huang, L. Xue, F. Xiao and Q. Li 2023. Blockchain-enabled fine-grained searchable encryption with cloud–edge computing for electronic health records sharing. *IEEE Internet of Things Journal* 10, 20 (2023), 18414–18425.
[39] J. Yu, S. Liu, M. Xu, H. Guo, F. Zhong and W. Cheng 2023. An efficient revocable and searchable MA-ABE scheme with blockchain assistance for C-IoT. *IEEE Internet of Things Journal* 10, 3 (2023), 2754–2766.
[40] H. Yin, W. Zhang, H. Deng, Z. Qin and K. Li 2023. An attribute-based searchable encryption scheme for cloud-assisted IIoT. *IEEE Internet of Things Journal* 10, 12 (2023), 11014–11023.
[41] J. Li, L. Ji, Y. Zhang, Y. Lu and J. Ning 2025. Response-hiding and volume-hiding verifiable searchable encryption with conjunctive keyword search. *IEEE Transactions on Computers* 74, 2 (2025), 455–467.
[42] Y. Yang, Y. Hu, X. Dong, J. Shen, Z. Cao, G. Yang, and R. H. Deng. 2024. OpenSE: Efficient verifiable searchable encryption with access and search pattern hidden for cloud-IoT. *IEEE Internet of Things Journal* 11, 8 (2024), 13793–13809.
[43] L. Ji, J. Li, Y. Zhang and Y. Lu 2025. Verifiable searchable symmetric encryption over additive homomorphism. *IEEE Transactions on Information Forensics and Security* 20, (2025), 1320–1332.
[44] C. Xu, R. Wang, L. Zhu, C. Zhang, R. Lu and K. Sharif 2023. Efficient strong privacy-preserving conjunctive keyword search over encrypted cloud data. *IEEE Transactions on Big Data* 9, 3 (2023), 805–817.
[45] D. Chen, Z. Liao, Z. Xie, R. Chen, Z. Qin, M. Cao, H. N. Dai, and K. Zhang, 2024. MFSSE: Multi-keyword fuzzy ranked symmetric searchable encryption with pattern hidden in mobile cloud computing. In *IEEE Transactions on Cloud Computing* 12, 4 (2024), 1042–1057.
[46] Q. Jiang, E. Chang, Y. Qi, S. Qi, P. Wu and J. Wang 2022. Rphx: Result pattern hiding conjunctive query over private compressed index using Intel SGX. *IEEE Transactions on Information Forensics and Security* 17, (2022),1053–1068.
[47] X. Song, C. Dong, D. Yuan, Q. Xu, and M. Zhao. 2020. Forward private searchable symmetric encryption with optimized I/O efficiency. *IEEE Trans. Dependable and Secure Computing* 17, 5 (2020), 912–927.
[48] S. F. Sun, X. Yuan, J. K. Liu, R. Steinfeld, A. Sakzad, V. Vo, and S. Nepal. 2018. Practical backward-secure searchable encryption from symmetric puncturable encryption. In *Proceedings of the CCS 2018*. (Toronto, ON, Canada), 763–780.
[49] C. Zhao, R. Du, K. He, J. Chen, J. Li, X. Liu, and J. Ning. 2025. Efficient verifiable dynamic searchable symmetric encryption with forward and backward security. *IEEE Internet of Things Journal* 12, 3 (2025), 2633–2645.
[50] H. Dou, Z. Dan, P. Xu, W. Wang, S. Xu, T. Chen, and H. Jin, 2024. Dynamic searchable symmetric encryption with strong security and robustness. In *IEEE Transactions on Information Forensics and Security* 19 (2024), 2370–2384.
[51] B.Chen, T. Xiang, D. He, H. Li and K.-K. R. Choo 2023. BPVSE: Publicly verifiable searchable encryption for cloud-assisted electronic health records. *IEEE Transactions on Information Forensics and Security* 18, (2023), 3171–3184.
[52] M. Zhang, E. Wei, R. Berry and J. Huang 2024. Age-dependent differential privacy. In *IEEE Transactions on Information Theory* 70, 2 (2024), 1300–1319.
[53] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M. C. Roşu, and M. Steiner. 2013. Highly-scalable searchable symmetric encryption with support for Boolean queries. In *Proceedings of the CRYPTO 2013*. (Santa Barbara, CA, USA), 353–373.
[54] M. Li, C. Jia, R. Du and W. Shao 2023. Forward and backward secure searchable encryption scheme supporting conjunctive queries over bipartite graphs. *IEEE Transactions on Cloud Computing* 11, 1 (2023), 1091–1102.

[55] D. Yuan, C. Zuo, S. Cui, and G. Russello. 2023. Result-pattern-hiding conjunctive searchable symmetric encryption with forward and backward privacy. *Proceedings on Privacy Enhancing Technologies* 2023, 2 (2023), 40–58.
[56] R. Li and A. X. Liu 2022. Adaptively secure and fast processing of conjunctive queries over encrypted data. *IEEE Transactions on Knowledge and Data Engineering* 34, 4 (2022), 1588–1602.
[57] Y. Li, J. Ning and J. Chen 2023. Secure and practical wildcard searchable encryption system based on inner product. *IEEE Transactions on Services Computing* 16, 3 (2023), 2178–2190.
[58] F. Liu, K. Xue, J. Yang, J. Zhang, Z. Huang, J. Li, and D. S. L. Wei. 2024. Volume-hiding range searchable symmetric encryption for large-scale datasets. *IEEE Transactions on Dependable and Secure Computing* 21, 4 (2024), 3597–3609.
[59] M. Xie, X. Yang, H. Hong, G. Wei, and Z. Zhang. 2024. A novel verifiable chinese multi-keyword fuzzy rank searchable encryption scheme in cloud environments. *Future Generation Computer Systems* 153 (2024), 287–300.
[60] T. Chen, P. Xu, S. Picek, B. Luo, W. Susilo, H. Jin, and K. Liang 2023. The power of bamboo: On the post-compromise security for searchable symmetric encryption. In *Proceedings of the NDSS 2023.* (San Diego, California, USA).
[61] C. Castelluccia, E. Mykletun and G. Tsudik 2005. Efficient aggregation of encrypted data in wireless sensor networks. In *Proceedings of the 2nd Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services.* San Diego, CA, USA, (2005), 109–117.
[62] C. R. Butler, L. B. Webster, and D. S. Diekema. 2024. Staffing crisis capacity: A different approach to healthcare resource allocation for a different type of scarce resource. *Journal of Medical Ethics* 50, 9 (2024), 647–649.
[63] Global Health Workforce statistics database - World Health Organization. Retrieved from https://www.who.int/data/gho/data/themes/topics/health-workforce
[64] D. J. Bernstein 2006. Curve25519: New diffie-hellman speed records. In *Proceedings of the PKC 2006.* (New York, NY, USA), 207–228.

Received 15 October 2024; revised 7 August 2025; accepted 4 September 2025