Quick Draw Queries: Lightweight Searchable Public-key Ciphertexts with Hidden Structures via Non-Interactive Key Exchange

Keita Emura*[1,2], Toshihiro Ohigashi[3], and Nobuyuki Sugio[4]

[1.]Kanazawa University, Japan.
[2.]AIST, Japan.
[3.]Tokai University, Japan.
[4.]Hokkaido University of Science, Japan.

October 12, 2026

Abstract

Basically, public-key searchable encryption schemes require a linear search time with respect to the total number of ciphertexts. Xu, Wu, Wang, Susilo, Domingo-Ferrer, and Jin (IEEE Transactions on Information Forensics and Security, 2015) introduced Searchable Public-Key Ciphertexts with Hidden Structure (SPCHS). In SPCHS, ciphertexts associated with the same keyword are linked through a hidden structure that can be revealed using a trapdoor. This enables efficient extraction of matching ciphertexts and achieves optimal asymptotic search efficiency by traversing only the result set. However, a drawback of the SPCHS scheme and its subsequent works is that they are all pairing-based or rely on identity-based encryption (IBE) as a foundational component. To improve efficiency, this paper proposes a generic construction of SPCHS. Our core building block is Non-Interactive Key Exchange (NIKE), and the remaining components are symmetric-key primitives, namely secret-key encryption and a pseudorandom function (PRF). In particular, our search algorithm depends solely on these symmetric-key primitives and is independent of any public-key primitives. We implement the most efficient instantiation of the generic construction using Diffie-Hellman NIKE, an HMAC-based PRF, and AES. Our evaluation demonstrates that the search time of the DH-based instantiation is over 250 times faster than that of the asymmetric pairing-based scheme by Wang et al. (IEEE Transactions on Industrial Informatics, 2020). Moreover, we reduce the ciphertext size by a factor of approximately 10. In addition to this practical contribution of significantly improving efficiency, the proposed generic construction also makes a theoretical contribution by enabling the realization of the first set of post-quantum SPCHS schemes based on lattices, isogenies, and codes.

1 Introduction

1.1 Background

In typical usage of Public Key Encryption with Keyword Search (PEKS) [6], a ciphertext and a trapdoor (corresponding to a search query) are associated with a keyword. Essentially, PEKS

*Corresponding Author: k-emura@se.kanazawa-u.ac.jp

enables equality-matching search functionality between an encrypted keyword and a search keyword. Consequently, the search complexity depends linearly on the number of ciphertexts.

Xu, Wu, Wang, Susilo, Domingo-Ferrer, and Jin [52] introduced Searchable Public-key Ciphertexts with Hidden Structure (SPCHS), which achieve optimal asymptotic search efficiency by traversing only the result set. In SPCHS, all ciphertexts corresponding to the same keyword have hidden relations. They employed a hidden star-like structure to accelerate the search process (which is more explained later). Due to the hidden structure, the search complexity (in terms of cryptographic operations) for searching the i-th keyword kw_i is the number of ciphertexts contained in the i-th hidden structure n_{kw_i}, that traverses only the result set. Wang and Chow called it optimal search [46].

1.2 Our Motivation

Although SPCHS achieves optimal asymptotic search efficiency, the SPCHS scheme proposed by Xu et al. and its subsequent works [14,30,49–51,56] are all pairing-based. Pairings are widely regarded as computationally expensive compared to other cryptographic primitives, making it critical to minimize the number of pairing operations. Furthermore, pairing-based schemes do not provide post-quantum security because they are vulnerable to Shor's quantum algorithm. We argue that it is important to develop a generic construction that supports instantiations secure under a broad range of complexity assumptions, including those resistant to quantum attacks. In addition to pairing-based SPCHS schemes, Xu et al. [52] proposed a generic construction of SPCHS based on identity-based encryption (IBE) and a collision-free, full-identity malleable Identity-Based Key Encapsulation Mechanism (IBKEM) with anonymity. They also proved that the Verifiable Random Function (VRF)-suitable IBKEM instance (proposed in Appendix A.2 of [2]) satisfies the collision-free, full-identity malleability property. This VRF-suitable IBKEM instance is based on the Boneh-Franklin IBE scheme [7]. In summary, all existing SPCHS schemes rely on pairings or a variant of IBE as a foundational component, and these components are currently instantiated using pairings only. Thus, we explore a natural question:

Can efficient pairing-free and/or post-quantum SPCHS schemes be constructed?

Here, we use the term "pairing-free" to refer specifically to discrete logarithm (DL)-based schemes. This usage does not imply other complexity assumptions such as those based on lattices, isogenies, codes, and so on.

Difficulty. First, PEKS is known to be equivalent to IBE [6]. Moreover, no black-box construction from public-key encryption (PKE) to IBE exists [9]. Although IBE can be constructed under the Computational Diffie-Hellman (CDH) assumption [18, 19, 28], its efficiency is significantly lower than that of pairing-based IBE due to the non-black-box approach based on garbled circuits. These results indicate that constructing an efficient pairing-free PEKS scheme based on a discrete-logarithm-related assumption is highly challenging.

Beyond linear search, more efficient public-key searchable encryption schemes have been proposed. Wang and Chow [45] introduced Hybrid Searchable Encryption (HSE), which aims to combine the advantages of symmetric searchable encryption (sublinear search) and public-key searchable encryption (support for multiple writers). They first presented a conceptually simple generic construction based on dynamic symmetric searchable encryption (DSSE) and anonymous IBE. To achieve forward privacy and compact tokens, they further improved the construction by employing epoch-based DSSE (E-DSSE) and identity-coupling key-aggregate encryption. HSE incurs

Table 1: Comparison in Terms of Building Blocks

	Building Block	How to Implement Search functionality
Previous SPCHEs	IBE	IBE Decryption
Our Construction	NIKE, SKE, PRF	SKE Decryption

an additive overhead of public-key operations that scales linearly with the number of active keywords. Later, Wang and Chow [46] optimized these public-key operations and proposed Delegatable Searchable Encryption (DSE), which achieves optimal search time in the multi-writer, multi-reader setting. We refer their explanation: *A writer IBE-encrypts the secret key material once, which will then be used for encrypting subsequent updates within the encrypted index of SSE. The traversal only takes one IBE decryption to retrieve the first result from each writer. The remaining operations in the asymptotical-optimal search use only symmetric keys.* Although the number of public-key operations has been optimized, IBE is still employed as a core building block. Aronesty et al. [3] introduced an approach based on encapsulated VRFs, which, to date, have only been instantiated using pairing-based constructions.

To enable expressive search functionality, key-policy attribute-based encryption (KP-ABE) can be employed, e.g. [39, 42]. Hayata et al. [31] proved that if PEKS supports logical disjunctions and conjunctions as search conditions, then it implies anonymous KP-ABE. Since ABE implies IBE [32], the impossibility result mentioned above [9] still hinders the construction of efficient discrete-logarithm-based pairing-free schemes. Wildcard searchable encryption schemes have also been proposed [29, 40, 41, 47], but all of them remain pairing-based. Le et al. [35] proposed a multi-user searchable encrypted database, called MUSES, based on a key-homomorphic pseudorandom function [8] constructed from lattices. A notable difference is that their padding and shuffle strategies require an additional communication round during the search procedure, whereas SPCHS supports search in a non-interactive manner.

To sum up, following previous construction methodologies does not provide a viable approach to overcoming the IBE barrier or to constructing pairing-free SPCHS schemes based on discrete-logarithm-related assumptions.

1.3 Our Contribution

In this paper, we propose a new generic construction of SPCHS that breaks the IBE-barrier. Our core building block is Non-Interactive Key Exchange (NIKE) [24] and remaining components are all symmetric key primitives (secret key encryption (SKE) and pseudorandom function (PRF)). Our contributions are organized as follows, distinguishing between practical and theoretical aspects.

Practical Contribution:

- Our search algorithm depends only on these symmetric key primitives and is independent of any public key primitives whereas those of previous SPCHE schemes rely on IBE decryption (See Table 1).
  - We implement the most efficient instantiation of the generic construction from DH NIKE [17], HMAC-based PRF, and AES (in counter mode), and demonstrate that the search time of the DH-based instantiation is over 250 times faster than that of the asymmetric pairing-based Wang et al. scheme [49].
- A ciphertext consists of a random value and a SKE ciphertext whereas that of previous schemes is an IBE ciphertext.

– Compared to the Wang et al. scheme [49], the DH-based instantiation can reduce the ciphertext size by a factor of approximately 10.

Theoretical Contribution:

- We found that the IBE-related component of SPCHSs can be replaced by SKE without any degradation in security (See Table 1).
- We can instantiate the first set of post-quantum SPCHS schemes.
  – We can employ NIKEs (from lattices [26], isogenies [13], and codes [43]) as a building block. We also assume to employ SHA512 (for HMAC-based PRF) and AES256 due to the Grover algorithm.
- We instantiate an SPCHS scheme secure in the standard model, for example using the pairing-based NIKE [24]. As a feasibility result, Xu et al. [52] proposed a multilinear-map-based SPCHS scheme in the standard model via a standard-model variant of the BF IBE scheme [25]. Thus, ours is the first SPCHS instantiation in the standard model with reasonable efficiency.
  – Although the pairing-based approach does not meet our pairing-free goal, it is notable for avoiding the random oracle (as shown in [11]) since all prior SPCHS schemes except the above multilinear-map-based one rely on it.

Our basic construction idea is as follows. We focus on the following facts:

1. The encryption algorithm of SPCHS requires a secret value that allows us to bypass the impossibility result [9].
2. The syntax is reminiscent of public key authenticated encryption with keyword search (PAEKS) [33].
3. PAEKS can be constructed from several complexity assumptions via the generic construction from NIKE (and symmetric-key Equality-Predicate Encryption (EPE)) given by Li and Boyen [38].

We emphasize that SPCHS is not directly constructed by the Li-Boyen generic construction due to the *decryption and plaintext connection* procedures (which are explained later). We adopt the *methodology* of the generic construction of PEKS from anonymous IBE [1] and find that SKE is sufficient to implement the decryption procedure. We give an overview of our construction in Sec. 1.4.

Remark. Regarding the term "lightweight", as described in our practical contribution, our main result lies in demonstrating that our DH-based instantiation achieves more than 250 times higher search efficiency compared with existing schemes, and that our search algorithm depends solely on symmetric-key primitives and is completely independent of any public-key primitives. In particular, while existing schemes realize search using an algorithm equivalent to IBE decryption, our construction implements search using only the decryption algorithm of a SKE scheme. This allows us to claim that our construction is sufficiently lightweight compared to prior works. Although, as described in the theoretical contribution, we do provide post-quantum instantiations, we do not claim these are lightweight. We would like to emphasize that the term lightweight is included in the paper's title to highlight our primary result.

Beyond the scope of this paper. To make clear what this paper addresses and does not address, we outline the areas excluded from our contributions.

As mentioned by Wang and Chow, SPCHS does not provide forward privacy (it requires the past search not to compromise the privacy of future data). In SPCHS, if a keyword to be encrypted has been encrypted before, then the ciphertext shares the hidden structure and is added to the leaf of the star-like structure. Recognizing that forward privacy in SSE is a de facto standard [10,34,44], and forward secure P(A)EKS schemes also have been proposed [21, 54]. Although Chen et al. [14] considered forward privacy with hidden structures, the scheme introduces an update algorithm that makes the SPCHS scheme complex. For PAEKS, a generic compiler that adds forward privacy to any PAEKS scheme has been proposed [21]. However, this compiler does not seem directly applicable to SPCHS due to its hidden structure. As previously mentioned, the primary goal of this paper is to break through the IBE barrier and construct a SPCHS scheme using building blocks at the PKE level, and avoiding the use of IBE in a PEKS context itself is a significant achievement. We position this result as a stepping stone toward incorporating important security properties such as forward privacy, which will not be addressed further in this paper.

As another demerit of our construction, a trapdoor (search query) depends on a sender (as in PAEKS) unlike the original syntax of SPCHS. This sender-dependent trapdoor may affect a negative impact on search efficiency. To clarify the impact of the demerit, we will consider the number of senders in our implementation in Sec. 6.3 and show that our Diffie-Hellman NIKE based instantiation outperforms the Wang et al. SPCHE scheme in scenarios involving thousands of senders. We leave how to provide sender-independent trapdoors in future work.

In all SPCHE schemes including our construction, the search algorithm seeks a ciphertext having $C[1] = R$. Since this part is independent of searchable encryption, is not our contribution, and any search algorithm could be employed, we used a simple hash table in our implementation and chose not to adopt a more sophisticated search algorithm (e.g., [23]) in this paper.

1.4 Technical Overview

To explain our construction methodology, we revisit the Xu et al. SPCHS scheme. Based on our observations, the Xu et al. SPCHS construction methodology, employing a decryption algorithm of the underlying encryption scheme, is reminiscent of the generic construction of PEKS from anonymous IBE [1]. To help the reader understand the SPCHS construction methodology, we revisit the generic construction as follows.

Generic construction of PEKS from anonymous IBE. A keyword is treated as an identity in the underlying IBE scheme. For a keyword kw, a PEKS ciphertext consists of a pair of an IBE ciphertext and a random plaintext R. We denote a PEKS ciphertext as $(C_{\mathsf{IBE}} = \mathsf{IBE.Enc}(kw, R), R)$. A trapdoor is the decryption key of the underlying IBE scheme for the identity kw, denoted as dk_{kw}. The search algorithm outputs 1 if $\mathsf{IBE.Dec}(C_{\mathsf{IBE}}, \mathsf{dk}_{kw}) = R$. Due to the anonymity of the underlying IBE scheme, no information of kw is leaked from ciphertexts. The generic construction can be instantiated by the Boneh-Franklin (BF) IBE scheme [7] as follows: A PEKS ciphertext is

$$((C^{(1)}_{\mathsf{IBE}}, C^{(2)}_{\mathsf{IBE}}) = (g^r, R \cdot e(H(kw), S)^r), R)$$

where $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a symmetric pairing, $\mathbb{G}$ and $\mathbb{G}_T$ are groups with prime order p, $g \in \mathbb{G}$ is a generator, $s \in \mathbb{Z}_p$ is the master secret key, $S = g^s$ is the master public key, and $r \in \mathbb{Z}_p$ is a randomness for encryption. The trapdoor is $T_{kw} = H(kw)^s \in \mathbb{G}$. The search algorithm runs the decryption algorithm of the BF IBE scheme, i.e., it checks whether $C^{(2)}_{\mathsf{IBE}}/e(C^{(1)}_{\mathsf{IBE}}, T_{kw}) = R \cdot e(H(kw), S)^r / e(g^r, H(kw)^s) = R$.

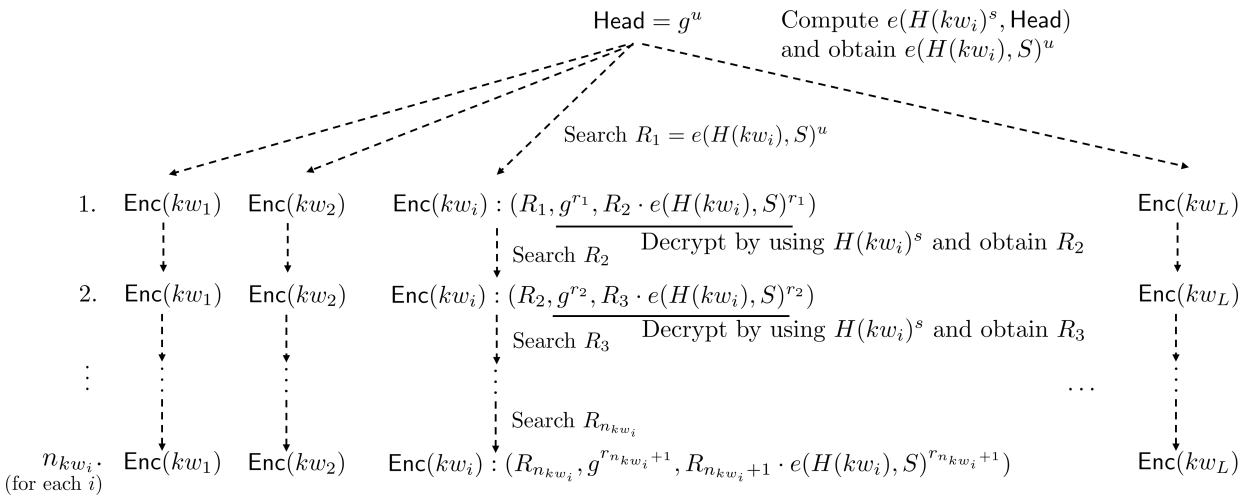


Figure 1: Hidden Star-like Structure and Xu et al. SPCHS [52]

Xu et al. SPCHS scheme. The Xu et al. SPCHS scheme [52] is briefly introduced as follows (See Fig. 1). First, a sender (an encryptor) sets $\mathsf{Head} = g^u$ where u is a secret value of the sender. For each keyword kw_i, the first ciphertext is

$$(R_1 = e(H(kw_i), S)^u, g^{r_1}, R_2 \cdot e(H(kw_i), S)^{r_1})$$

and the second ciphertext is

$$(R_2, g^{r_2}, R_3 \cdot e(H(kw_i), S)^{r_2})$$

Here, $(\mathsf{Head}, e(H(kw_i), S)^u)$ can be regarded as an BF IBE ciphertext for the plaintext 1 and the secret value u is required to compute $e(H(kw_i), S)^u$ ($S = g^s$ is a public key of the receiver and the sender (encryptor) does not know s). By using the trapdoor $H(kw_i)^s$, R_1 can be obtained by decryption: $e(\mathsf{Head}, H(kw_i)^s) = e(H(kw_i), S)^u = R_1$, and the first ciphertext can be found by seeking a ciphertext containing R_1. Since $(g^{r_1}, R_2 \cdot e(H(kw_i), S)^{r_1})$ contained in the first ciphertext is an BF IBE ciphertext for the plaintext R_2, the search algorithm can obtain R_2 by decryption. Now, the search algorithm can find the second ciphertext since R_2 is contained in the second ciphertext. By the same procedure, R_3 is obtained and is used as a key for searching the third ciphertext, and these procedures connect all ciphertexts for the same keyword. Due to the decryption (that obtains R from a ciphertext) and plaintext connection (that finds R from N ciphertexts) procedure described above, the number of *cryptographic operations* (i.e. decryption) is directly proportional to the number of ciphertexts embedded within the corresponding hidden structure. Since this structure traverses only the result set, the computational cost is confined to those ciphertexts that are actually linked through the connection process. Here, the search complexity for locating R among N ciphertexts is $O(N)$ when using a naive exhaustive search. However, more efficient search algorithms, such as hash tables, can reduce the number of comparisons. Thus, we do not focus on the search complexity for comparisons from our analysis, as the cost of cryptographic operations is the dominant factor. Since each ciphertext is an anonymous IBE ciphertext and a random plaintext that is independent to a keyword to be encrypted, no keyword information is revealed from ciphertexts without the decryption key (trapdoor in this context). In other words, the trapdoor reveals the hidden structure among ciphertexts for the same keyword.

Overview of Our Construction. We observe that $R_1 = e(H(kw), S)^u$ in the Xu et al. SPCHS scheme is uniquely determined by a sender's secret key u, the receiver's public key $S = g^s$,

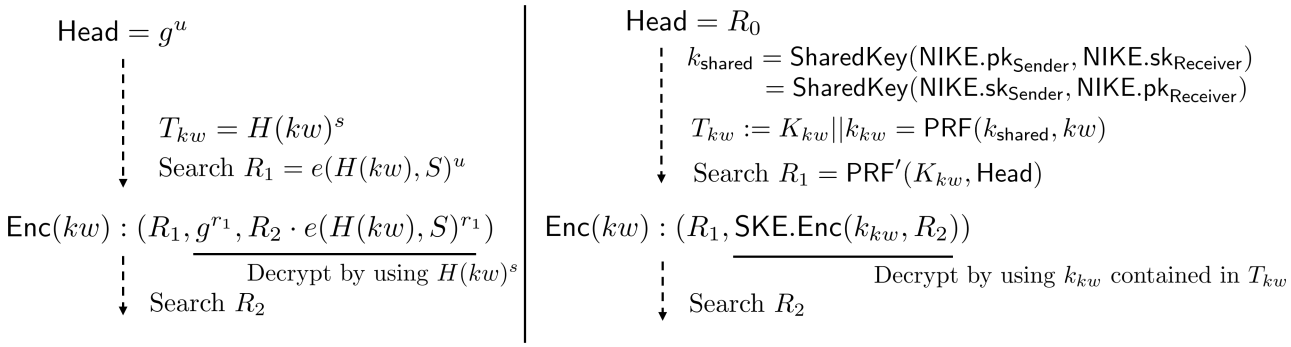


Figure 2: Comparison between Xu et al. SPCHS [52] (left) and ours (right)

and a keyword kw (here, the subscript i is omitted for readability). Moreover, R_1 and other R need to be indistinguishable, i.e., R_1 randomly distributes over the target group as in other R. Here, we adopt the core idea of the Li-Boyen generic construction of PAEKS from NIKE since a NIKE shared key k_{shared} is uniquely determined by a sender's secret (resp. public) key and the receiver's public (resp. secret) key where $k_{\mathsf{shared}} = \mathsf{SharedKey}(\mathsf{NIKE.pk}_{\mathsf{Sender}}, \mathsf{NIKE.sk}_{\mathsf{Receiver}}) = \mathsf{SharedKey}(\mathsf{NIKE.sk}_{\mathsf{Sender}}, \mathsf{NIKE.pk}_{\mathsf{Receiver}})$ (here identities, S and R, are omitted). We set a PRF value of a keyword kw under the NIKE shared key as a trapdoor $T_{kw} = \mathsf{PRF}(k_{\mathsf{shared}}, kw)$. In the Xu et al. scheme, R_1 is uniquely determined by the trapdoor $H(kw_i)^s$ and Head, and is contained in the first ciphertext. By following this methodology, we set

$$\mathsf{Head} = R_0 \in \{0,1\}^\lambda \text{ and } R_1 = \mathsf{PRF}'(K_{kw}, R_0)$$

where $\lambda \in \mathbb{N}$ is a security parameter, and $T_{kw} := K_{kw}||k_{kw}$. We set the second ciphertext as

$$(R_1, \mathsf{SKE.Enc}(k_{kw}, R_2))$$

where $R_2 \in \{0,1\}^\lambda$ is a random plaintext. Repeating these procedures connect all ciphertexts for the same keyword. A comparison between the Xu et al. scheme and our construction is displayed in Fig. 2. We note that, as a demerit of our construction, a trapdoor depends on a sender (as in PAEKS) unlike to the original syntax of SPCHS. To clarify the impact of the demerit, we will consider the number of senders in our implementation in Sec. 6.3. We also remark that a sender also can compute a trapdoor in our construction unlike to previous SPCHS schemes. We emphasize that this difference does not affect the security of SPCHS.

Due to the security of NIKE, k_{shared} is indistinguishable from random. Thus, we can utilize the security of the underlying PRF scheme. Due to the security of PRF (PRF), $T_{kw} = K_{kw}||k_{kw}$ is indistinguishable from random. Then, due to the security of PRF (PRF'), R_1 is indistinguishable from random (and indistinguishable from other Rs). Finally, we can utilize the security of the underlying SKE scheme. Due to the PCPA (pseudo-randomness against chosen plaintext attacks) security [16], a SKE ciphertext is indistinguishable from random. Since all ciphertexts are random, no keyword information is leaked from ciphertexts.

1.5 Related Work

As mentioned in the previous section, one pairing operation is required for each decryption procedure in the Xu et al. pairing-based SPCHS scheme [52], and n_{kw}-times pairing computations are required in total where n_{kw} is the number of ciphertexts contained in a hidden structure. Xu et

al. [51] reduced the number of pairing operations where one pairing operation is required in total, and one multiplication over the target group is required for each decryption procedure. Chen et al. [14] also improved the efficiency where one pairing operation is required in total, and one pseudo-random (invertible) permutation is required for each decryption procedure. Xu et al. [50] considered multi-keyword search by extending the Xu et al. scheme [52]. Han et al. [30] considered dynamic searchable public-key encryption by applying SPCHS. Zhang et al. [56] proposed a fast keyword search scheme for cloud-assisted edge computing. These six SPCHS schemes [14, 30, 50–52, 56] and one instantiation of the generic construction [52] via the VRF-suitable IBKEM instance [2] are constructed over symmetric pairings. To improve the efficiency in terms of implementation, Wang et al. [49] employed asymmetric pairings since asymmetric pairings are suitable for providing an efficient implementation [27] due to the difference of embedded degree. To sum up, currently, all SPCHS schemes rely on pairings or IBE as a foundational component.

2 Preliminaries

2.1 Non-Interactive Key Exchange (NIKE)

Let $\mathsf{NIKE} = (\mathsf{NIKE.Setup}, \mathsf{NIKE.KeyGen}, \mathsf{SharedKey})$ be a NIKE scheme. The system parameter generation algorithm $\mathsf{NIKE.Setup}$ takes a security parameter $\lambda \in \mathbb{N}$ as input, and outputs a system parameter params. The key generation algorithm $\mathsf{NIKE.KeyGen}$ takes params an identity ID as input, and outputs a public key/secret key pair $(\mathsf{NIKE.pk}, \mathsf{NIKE.sk})$. In our construction, we assume that an identity is either "S (Sender)" or "R (Receiver)" (with an index if necessary). Then, we denote $(\mathsf{NIKE.pk}_{\mathsf{Sender}}, \mathsf{NIKE.sk}_{\mathsf{Sender}}) \leftarrow \mathsf{NIKE.KeyGen}(\mathsf{params}, \mathsf{S})$ and $(\mathsf{NIKE.pk}_{\mathsf{Receiver}}, \mathsf{NIKE.sk}_{\mathsf{Receiver}}) \leftarrow \mathsf{NIKE.KeyGen}(\mathsf{params}, \mathsf{R})$, respectively. The shared key generation algorithm $\mathsf{SharedKey}$ takes an identity and a public key along with another identity and a secret key, and outputs either a shared key k_{shared} or a failure symbol $\perp$. We assume that $k_{\mathsf{shared}} \in \{0,1\}^\lambda$. As in the $\mathsf{NIKE.KeyGen}$ algorithm, we basically denote $k_{\mathsf{shared}} \leftarrow \mathsf{SharedKey}(\mathsf{S}, \mathsf{NIKE.pk}_{\mathsf{Sender}}, \mathsf{R}, \mathsf{NIKE.sk}_{\mathsf{Receiver}})$ or $k'_{\mathsf{shared}} \leftarrow \mathsf{SharedKey}(\mathsf{S}, \mathsf{NIKE.sk}_{\mathsf{Sender}}, \mathsf{R}, \mathsf{NIKE.pk}_{\mathsf{Receiver}})$. We say that NIKE is correct if $\Pr[k_{\mathsf{shared}} = k'_{\mathsf{shared}}] \geq 1 - \mathsf{negl}(\lambda)$ (here the probability is over the random coins of $\mathsf{NIKE.Setup}$ and $\mathsf{NIKE.KeyGen}$).

Freire, Hofheinz, Kiltz, and Paterson [24] introduced several security definitions (based on the CKS (Cash-Kiltz-Shoup) model [12]). In the searchable encryption context, Li and Boyen [38] employed the DKR-CKS model where DKR stands for dishonest key registration which is defined as follows.

Definition 1 (DKR-CKS)**.** *We define the following experiment. We remark that the bit b is chosen in the* Chall *oracle.*

$$
\begin{aligned}
&\mathsf{Exp}^{\mathsf{DKR\text{-}CKS}}_{\mathsf{NIKE},\mathcal{A}}(\lambda):\\
&\quad \mathsf{params} \leftarrow \mathsf{NIKE.Setup}(1^\lambda);\ \mathcal{L} \leftarrow \emptyset\\
&\quad b' \leftarrow \mathcal{A}^{\mathsf{RegHon}(\cdot),\mathsf{RegCorr}(\cdot),\mathsf{Reavel}(\cdot),\mathsf{Chall}(\cdot)}(\mathsf{params})\\
&\quad \textit{If } b = b' \textit{ then output } 1 \textit{ and } 0 \textit{ otherwise.}
\end{aligned}
$$

- RegHon*: The honest key registration oracle takes an identity* ID *as input. If* $(corr, \mathsf{ID}, \perp, \cdot) \in \mathcal{L}$*, then return* $\perp$*. Otherwise, run* $(\mathsf{NIKE.pk}, \mathsf{NIKE.sk}) \leftarrow \mathsf{NIKE.KeyGen}(\mathsf{params}, \mathsf{ID})$*, update* $\mathcal{L} \leftarrow (honest, \mathsf{ID}, \mathsf{NIKE.sk}, \mathsf{NIKE.pk}) \cup \mathcal{L}$*, and return* $\mathsf{NIKE.pk}$*.*
- $\mathsf{RegCorr}$*: The dishonest key registration oracle takes an identity* ID *and a public key* $\mathsf{NIKE.pk}$ *as input. If* $(corr, \mathsf{ID}, \perp, \cdot) \in \mathcal{L}$*, then set* $(corr, \mathsf{ID}, \perp, \cdot) = (corr, \mathsf{ID}, \perp, \mathsf{NIKE.pk})$*. Otherwise, update* $\mathcal{L} \leftarrow (corr, \mathsf{ID}, \perp, \mathsf{NIKE.pk}) \cup \mathcal{L}$*.*

- Reavel*: The shared key revealing oracle takes two identities* ID_1 *and* ID_2 *as input with the restriction that at least one of the two identities was registered as honest. If* $(honest, \mathsf{ID}_1, \mathsf{NIKE.sk}_1, \mathsf{NIKE.pk}_1) \in \mathcal{L}$ *and* $(corr, \mathsf{ID}_2, \perp, \mathsf{NIKE.pk}_2) \in \mathcal{L}$*, then return* $\mathsf{SharedKey}(\mathsf{ID}_1, \mathsf{NIKE.sk}_1, \mathsf{ID}_2, \mathsf{NIKE.pk}_2)$*. If* $(corr, \mathsf{ID}_1, \perp, \mathsf{NIKE.pk}_1) \in \mathcal{L}$ *and* $(honest, \mathsf{ID}_2, \mathsf{NIKE.sk}_2, \mathsf{NIKE.pk}_2) \in \mathcal{L}$*, then return* $\mathsf{SharedKey}(\mathsf{ID}_2, \mathsf{NIKE.sk}_2, \mathsf{ID}_1, \mathsf{NIKE.pk}_1)$*. If* $(honest, \mathsf{ID}_1, \mathsf{NIKE.sk}_1, \mathsf{NIKE.pk}_1) \in \mathcal{L}$ *and* $(honest, \mathsf{ID}_2, \mathsf{NIKE.sk}_2, \mathsf{NIKE.pk}_2) \in \mathcal{L}$*, then return* $\mathsf{SharedKey}(\mathsf{ID}_1, \mathsf{NIKE.sk}_1, \mathsf{ID}_2, \mathsf{NIKE.pk}_2)$*.*
- Chall*: The challenge oracle takes two identities* ID_1 *and* ID_2 *as input. If* $\mathsf{ID}_1 = \mathsf{ID}_2$ *or* $(corr, \mathsf{ID}_1, \cdot, \cdot) \in \mathcal{L}$ *or* $(corr, \mathsf{ID}_2, \cdot, \cdot) \in \mathcal{L}$*, then output* $\perp$*. Let* $(honest, \mathsf{ID}_1, \mathsf{NIKE.sk}_1, \mathsf{NIKE.pk}_1) \in \mathcal{L}$ *and* $(honest, \mathsf{ID}_2, \mathsf{NIKE.sk}_2, \mathsf{NIKE.pk}_2) \in \mathcal{L}$*. Choose* $b \xleftarrow{\$} \{0,1\}$*. If* $b = 0$*, then return* $k^*_{\mathsf{shared}} \xleftarrow{\$} \{0,1\}^\lambda$*. If* $b = 1$*, then return* $k^*_{\mathsf{shared}} \leftarrow \mathsf{SharedKey}(\mathsf{ID}_1, \mathsf{NIKE.sk}_1, \mathsf{ID}_2, \mathsf{NIKE.pk}_2)$*.*

We say that NIKE *is DKR-CKS secure if*

$$\mathsf{Adv}^{\mathsf{DKR\text{-}CKS}}_{\mathsf{NIKE},\mathcal{A}}(\lambda) := |\Pr[\mathsf{Exp}^{\mathsf{DKR\text{-}CKS}}_{\mathsf{NIKE},\mathcal{A}}(\lambda) = 1] - 1/2|$$

is negligible in the security parameter for all PPT adversaries $\mathcal{A}$*.*

We assume that the Chall oracle is queried only once, since the resulting security notion is polynomially equivalent to the one allowing multiple challenge queries [24].

When RegCorr and Reveal are removed, the model is called HKR-CKS where HKR stands for honest key registration. Then, we say that NIKE is HKR-CKS secure. In our construction, NIKE is required to be NKR-CKS secure as in Li-Boyen. Because a NIKE scheme secure in DKR-CKS model is also secure in the HKR-CKS model, we can employ DKR-CKS secure NIKEs as building blocks of our generic construction.

2.2 IND-PCPA Secure Symmetric Key Encryption (SKE)

Let $\mathsf{SKE} = (\mathsf{SKE.KG}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ be a SKE scheme. The key generation algorithm SKE.KG takes a security parameter $\lambda \in \mathbb{N}$ as input, and outputs a secret key $k \in \mathsf{KeySpace}$, where KeySpace is a key space. The encryption algorithm SKE.Enc takes k and a plaintext $M \in \mathsf{MSpace}$ as input, and outputs a ciphertext $C_{\mathsf{SKE}} \in \mathsf{CSpace}$, where MSpace and CSpace are a plaintext space and a ciphertext space, respectively. The decryption algorithm SKE.Dec takes k and C_{SKE} as input, and outputs M. The correctness is required to be held where for all $\lambda \in \mathbb{N}$, $k \leftarrow \mathsf{SKE.KG}(1^\lambda)$, and $M \in \mathsf{MSpace}$, $\mathsf{SKE.Dec}(k, \mathsf{SKE.Enc}(k, M)) = M$ holds.

Next, we define IND-PCPA security [16]. Intuitively, it guarantees that a ciphertext and a random value chosen from a ciphertext space is indistinguishable. Let state be state information that $\mathcal{A}$ can preserve any information, and state is used for transferring state information to the other stage.

Definition 2 (IND-PCPA)**.** *We define the following experiment.*

$$\begin{aligned}
&\mathsf{Exp}^{\mathsf{IND\text{-}PCPA}}_{\mathsf{SKE},\mathcal{A}}(\lambda): \\
&\quad k^* \leftarrow \mathsf{SKE.KG}(1^\lambda) \\
&\quad (M^*, \mathsf{state}) \leftarrow \mathcal{A}^{\mathsf{SKE.Enc}(k^*,\cdot)}(1^\lambda) \ s.t.\ M^* \in \mathsf{MSpace} \\
&\quad b \xleftarrow{\$} \{0,1\};\ C^{(0)}_{\mathsf{SKE}} \leftarrow \mathsf{SKE.Enc}(k^*, M^*);\ C^{(1)}_{\mathsf{SKE}} \xleftarrow{\$} \mathsf{CSpace} \\
&\quad b' \leftarrow \mathcal{A}(C^{(b)}_{\mathsf{SKE}}, \mathsf{state}) \\
&\quad \textit{If } b = b' \textit{ then output } 1 \textit{ and } 0 \textit{ otherwise.}
\end{aligned}$$

We say that SKE *is IND-PCPA secure if*

$$\mathsf{Adv}^{\mathsf{IND\text{-}PCPA}}_{\mathsf{SKE},\mathcal{A}}(\lambda) := |\Pr[\mathsf{Exp}^{\mathsf{IND\text{-}PCPA}}_{\mathsf{SKE},\mathcal{A}}(\lambda) = 1] - 1/2|$$

is negligible in the security parameter for all PPT adversaries $\mathcal{A}$*.*

2.3 Pseudorandom Functions (PRFs)

Let m and ℓ be polynomial and λ be a security parameter. A pseudorandom function (PRF) is a family of functions $\mathsf{PRF} = \{F_K : \{0,1\}^{m(\lambda)} \to \{0,1\}^{\ell(\lambda)}\}$ where $K \in \{0,1\}^\lambda$.

Definition 3 (Pseudo-randomness)**.** *We define the following experiment. Here,* $\mathcal{RF}$ *is a set of random functions mapping* $m(\lambda)$ *bits to* $\ell(\lambda)$ *bits.*

$\mathsf{Exp}^{\mathsf{pseudo\text{-}random}}_{\mathsf{PRF},\mathcal{A}}(\lambda)$:

$b \xleftarrow{\$} \{0,1\}$

If $b = 0$: $K \xleftarrow{\$} \{0,1\}^\lambda$; *Set* $\mathcal{O}(\cdot) = F_K(\cdot)$

If $b = 1$: $R \xleftarrow{\$} \mathcal{RF}$; *Set* $\mathcal{O}(\cdot) = R(\cdot)$

$b' \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(1^\lambda)$

If $b = b'$ *then output* 1 *and* 0 *otherwise.*

We say that PRF *is pseudo-random if*

$$\mathsf{Adv}^{\mathsf{pseudo\text{-}random}}_{\mathsf{PRF},\mathcal{A}}(\lambda) := |\Pr[\mathsf{Exp}^{\mathsf{pseudo\text{-}random}}_{\mathsf{PRF},\mathcal{A}}(\lambda) = 1] - 1/2|$$

is negligible in the security parameter for all PPT adversaries $\mathcal{A}$*.*

3 Definition of SPCHS

Let $\mathsf{SPCHS} = (\mathsf{SPCHS.Setup}, \mathsf{Structure}, \mathsf{SPCHS.Enc}, \mathsf{Trapdoor}, \mathsf{Search})$ be a SPCHS scheme defined as follows. Since $\mathsf{sk_S}$ contains a list to store hidden structures HS and HS is updated for each encryption, it is a stateful scheme.

SPCHS.Setup: The system setup algorithm takes a security parameter $\lambda \in \mathbb{N}$ as input, and outputs a pair of master public and secret keys $(\mathsf{pk_R}, \mathsf{sk_R})$. Assume that $\mathsf{pk_R}$ contains a keyword space $\mathcal{KS}$ and a ciphertext space $\mathcal{CS}$.

Structure: The hidden structure initialization algorithm takes $\mathsf{pk_R}$ as input, and outputs a pair of public and secret keys $(\mathsf{pk_S}, \mathsf{sk_S})$.

SPCHS.Enc: The encryption algorithm takes $\mathsf{pk_R}$, $\mathsf{sk_S}$, and a keyword $kw \in \mathcal{KS}$ as input, and outputs a ciphertext $C \in \mathcal{CS}$ and updates $\mathsf{sk_S}$.

Trapdoor: The trapdoor generation algorithm takes $\mathsf{sk_R}$, $\mathsf{pk_S}$, and a keyword $kw \in \mathcal{KS}$, and outputs a trapdoor T_{kw}.

Search: The search algorithm takes $\mathsf{pk_R}$, $\mathsf{pk_S}$, a set of ciphertexts $\mathsf{CSet} \subset \mathcal{CS}$, and T_{kw}, and outputs a set of ciphertexts $\mathsf{CSet}' \subset \mathsf{CSet}$.

Following the typical usage of public-key searchable encryption, we assume that the ciphertext of a keyword and the ciphertext of a document containing that keyword are stored simultaneously. Furthermore, the document ciphertext is encrypted using the recipient's public key. Upon search, if the ciphertext of a certain keyword is matched, the corresponding document ciphertext is returned. That is, CSet', returned by the Search algorithm, contains the corresponding ciphertexts of documents. We omit this procedure for readability.

Correctness guarantees that a ciphertext of a keyword kw is contained in the result of the Search algorithm with the trapdoor for kw. Formally, we say that SPCHS provides correctness if for all $\lambda \in \mathbb{N}$, $(\mathsf{pk_R}, \mathsf{sk_R}) \leftarrow \mathsf{SPCHS.Setup}(1^\lambda)$, $(\mathsf{pk_S}, \mathsf{sk_S}) \leftarrow \mathsf{Structure}(\mathsf{pk_R})$, $kw \in \mathcal{KS}$, $(C, \mathsf{sk_S}) \leftarrow \mathsf{SPCHS.Enc}(\mathsf{pk_R}, \mathsf{sk_S}, kw)$, and $T_{kw} \leftarrow \mathsf{Trapdoor}(\mathsf{sk_R}, \mathsf{pk_S}, kw)$, let $C \in \mathsf{CSet}$. Then, for $\mathsf{CSet}' \leftarrow \mathsf{Search}(\mathsf{pk_R}, \mathsf{pk_S}, \mathsf{CSet}, T_{kw})$, $C \in \mathsf{CSet}'$ holds with probability $1 - \mathsf{negl}(\lambda)$.

Consistency guarantees that a ciphertext of a keyword kw is not contained in the result of the Search algorithm with the trapdoor for a different keyword $kw' \neq kw$, which is formally defined as follows.

Definition 4 (Consistency)**.** *We define the following experiment.*

$$
\begin{aligned}
&\mathsf{Exp}^{\mathsf{Consist}}_{\mathsf{SPCHS},\mathcal{A}}(\lambda): \\
&\quad (\mathsf{pk_R}, \mathsf{sk_R}) \leftarrow \mathsf{SPCHS.Setup}(1^\lambda);\ (\mathsf{pk_S}, \mathsf{sk_S}) \leftarrow \mathsf{Structure}(\mathsf{pk_R}) \\
&\quad (kw, kw') \leftarrow \mathcal{A}(\mathsf{pk_R}, \mathsf{pk_S})\ \textit{s.t.}\ kw, kw' \in \mathcal{KS}\ \textit{and}\ kw \neq kw' \\
&\quad (C, \mathsf{sk_S}) \leftarrow \mathsf{SPCHS.Enc}(\mathsf{pk_R}, \mathsf{sk_S}, kw) \\
&\quad T_{kw'} \leftarrow \mathsf{Trapdoor}(\mathsf{sk_R}, \mathsf{pk_S}, kw') \\
&\quad \textit{Set}\ \mathsf{CSet} = \{C\};\ \mathsf{CSet}' \leftarrow \mathsf{Search}(\mathsf{pk_R}, \mathsf{pk_S}, \mathsf{CSet}, T_{kw'}) \\
&\quad \textit{If}\ C \in \mathsf{CSet}'\ \textit{then output}\ 1\ \textit{and}\ 0\ \textit{otherwise.}
\end{aligned}
$$

We say that SPCHS *provides consistency if*

$$\mathsf{Adv}^{\mathsf{Consist}}_{\mathsf{SPCHS},\mathcal{A}}(\lambda) := \Pr[\mathsf{Exp}^{\mathsf{Consist}}_{\mathsf{SPCHS},\mathcal{A}}(\lambda) = 1]$$

is negligible in the security parameter for all PPT adversaries $\mathcal{A}$*.*

Next, we define indistinguishability against chosen keyword and structure attacks (IND-CKSA) to guarantee that no information of keyword is revealed from ciphertexts. As in Full Cipher-Keyword Indistinguishability (Full CI) [38] (which is a standard security model in PAEKS), an adversary is allowed to issue encryption queries for the challenge keywords kw_0^* and kw_1^*. Moreover, if a ciphertext (generated by $\mathsf{sk_S}$) leaks information of $\mathsf{pk_S}$, then it reveals the hidden structure associated to the ciphertext. Thus, as in the model defined by Xu et al. [52], it is required that no adversary can distinguish whether the challenge ciphertext is generated by $(kw_0^*, \mathsf{pk_S}^{(0)})$ or $(kw_1^*, \mathsf{pk_S}^{(1)})$.

Definition 5 (IND-CKSA)**.** *We define the following experiment.*

$\mathsf{Exp}^{\mathsf{IND\text{-}CKSA}}_{\mathsf{SPCHS},\mathcal{A}}(\lambda)$:

$(\mathsf{pk_R}, \mathsf{sk_R}) \leftarrow \mathsf{SPCHS.Setup}(1^\lambda)$; $\mathsf{KSet} := \emptyset$; $\mathsf{CKSet} := \emptyset$

$(kw_0^*, kw_1^*, \mathsf{pk_S}^{*(0)}, \mathsf{pk_S}^{*(1)})$

$\leftarrow \mathcal{A}^{\mathsf{Structure}(\mathsf{pk_R}),\mathsf{Reveal}(\cdot),\mathsf{SPCHS.Enc}(\mathsf{pk_R},\cdot,\cdot),\mathsf{Trapdoor}(\mathsf{pk_R},\cdot,\cdot)}(\mathsf{pk_R})$

$s.t.\ kw_0^*, kw_1^* \in \mathcal{KS} \wedge kw_0^* \neq kw_1^* \wedge (\mathsf{pk_S}^{*(0)}, \cdot), (\mathsf{pk_S}^{*(1)}, \cdot) \in \mathsf{KSet} \setminus \mathsf{CKSet}$

$\wedge\ \mathcal{A}$ *did not query* $\mathsf{Trapdoor}(\mathsf{pk_R}, \mathsf{pk_S}^{*(0)}, kw_0^*)$ *or* $\mathsf{Trapdoor}(\mathsf{pk_R}, \mathsf{pk_S}^{*(1)}, kw_1^*)$

Let $(\mathsf{pk_S}^{*(0)}, \mathsf{sk_S}^{*(0)}), (\mathsf{pk_S}^{*(1)}, \mathsf{sk_S}^{*(1)}) \in \mathsf{KSet}$

$b \xleftarrow{\$} \{0,1\}$; $(C^*, \mathsf{sk'_S}^{(b)}) \leftarrow \mathsf{SPCHS.Enc}(\mathsf{pk_R}, \mathsf{sk_S}^{*(b)}, kw_b^*)$

$b' \leftarrow \mathcal{A}^{\mathsf{Structure}(\mathsf{pk_R}),\mathsf{Reveal}(\cdot),\mathsf{SPCHS.Enc}(\mathsf{pk_R},\cdot,\cdot),\mathsf{Trapdoor}(\mathsf{pk_R},\cdot,\cdot)}(C^*)$

If $b = b'$ *then output* 1 *and* 0 *otherwise.*

- $\mathsf{Structure}$*: The hidden structure initialization oracle takes* $\mathsf{pk_R}$ *as input, runs* $(\mathsf{pk_S}, \mathsf{sk_S}) \leftarrow \mathsf{Structure}(\mathsf{pk_R})$*, updates* $\mathsf{KSet} \leftarrow \mathsf{KSet} \cup \{(\mathsf{pk_S}\ \mathsf{sk_S})\}$*, and returns* $\mathsf{pk_S}$*.*
- Reveal*: The secret key reveal oracle takes* $\mathsf{pk_S}$ *as input (after the challenge phase, it is required that* $\mathsf{pk_S} \notin \{\mathsf{pk_S}^{*(0)}, \mathsf{pk_S}^{*(1)}\}$*). If* $(\mathsf{pk_S}, \cdot) \notin \mathsf{KSet}$*, then return* $\perp$*. Otherwise, let* $(\mathsf{pk_S}\ \mathsf{sk_S}) \in \mathsf{KSet}$*. Return* $\mathsf{sk_S}$ *and update* $\mathsf{CKSet} \leftarrow \mathsf{CKSet} \cup \{(\mathsf{pk_S}\ \mathsf{sk_S})\}$ *where* CKSet *stands for the set of corrupted keys.*
- $\mathsf{SPCHS.Enc}$*: The encryption oracle takes* $\mathsf{pk_R}$*,* $\mathsf{pk_S}$*, and* kw *as input. If* $(\mathsf{pk_S}, \cdot) \notin \mathsf{KSet}$*, then return* $\perp$*. Otherwise, let* $(\mathsf{pk_S}, \mathsf{sk_S}) \in \mathsf{KSet}$*. Run* $(C, \mathsf{sk'_S}) \leftarrow \mathsf{SPCHS.Enc}(\mathsf{pk_R}, \mathsf{sk_S}, kw)$*, update* $\mathsf{KSet} \leftarrow \mathsf{KSet} \setminus \{(\mathsf{pk_S}, \mathsf{sk_S})\}$ *and* $\mathsf{KSet} \leftarrow \mathsf{KSet} \cup \{(\mathsf{pk_S}, \mathsf{sk'_S})\}$*, and return* C*.*
- $\mathsf{Trapdoor}$*: The trapdoor oracle takes* $\mathsf{pk_S}$ *and* kw *as input (after the challenge phase, it is required that* $(\mathsf{pk_S}, kw) \notin \{(\mathsf{pk_S}^{*(0)}, kw_0^*), (\mathsf{pk_S}^{*(1)}, kw_1^*)\}$*). If* $(\mathsf{pk_S}, \cdot) \notin \mathsf{KSet}$*, then return* $\perp$*. Otherwise, run* $T_{kw} \leftarrow \mathsf{Trapdoor}(\mathsf{sk_R}, \mathsf{pk_S}, kw)$ *and return* T_{kw}*.*

We say that SPCHS *is IND-CKSA secure if*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CKSA}}_{\mathsf{SPCHS},\mathcal{A}}(\lambda) := |\Pr[\mathsf{Exp}^{\mathsf{IND\text{-}CKSA}}_{\mathsf{SPCHS},\mathcal{A}}(\lambda) = 1] - 1/2|$$

is negligible in the security parameter for all PPT adversaries $\mathcal{A}$*.*

4 Proposed SPCHS Construction

4.1 Proposed Generic Construction

In this section, we construct $\mathsf{SPCHS} = (\mathsf{SPCHS.Setup}, \mathsf{Structure}, \mathsf{SPCHS.Enc}, \mathsf{Trapdoor}, \mathsf{Search})$ from $\mathsf{NIKE} = (\mathsf{NIKE.Setup}, \mathsf{NIKE.KeyGen}, \mathsf{SharedKey})$, $\mathsf{SKE} = (\mathsf{SKE.KG}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$, $\mathsf{PRF} = \{F_K : \mathcal{KS} \to \{0,1\}^{2\lambda}\}$, and $\mathsf{PRF}' = \{F'_{K'} : \{0,1\}^\lambda \to \{0,1\}^\lambda\}$ as follows.

$\mathsf{SPCHS.Setup}(1^\lambda)$**:** Let R be the identity of the receiver who runs the algorithm. Run $\mathsf{params} \leftarrow \mathsf{NIKE.Setup}(1^\lambda)$ and $(\mathsf{NIKE.pk_{Receiver}}, \mathsf{NIKE.sk_{Receiver}}) \leftarrow \mathsf{NIKE.KeyGen}(\mathsf{params}, \mathsf{R})$, and output $(\mathsf{pk_R}, \mathsf{sk_R}) = ((\mathsf{params}, \mathsf{R}, \mathsf{NIKE.pk_{Receiver}}), (\mathsf{R}, \mathsf{NIKE.sk_{Receiver}}))$.

$\mathsf{Structure}(\mathsf{pk}_\mathsf{R})$**:** Let S be the identity of the sender who runs the algorithm. Parse $\mathsf{pk}_\mathsf{R} = (\mathsf{params}, \mathsf{R}, \mathsf{NIKE.pk}_\mathsf{Receiver})$. Run $(\mathsf{NIKE.pk}_\mathsf{Sender}, \mathsf{NIKE.sk}_\mathsf{Sender}) \leftarrow \mathsf{NIKE.KeyGen}(\mathsf{params}, \mathsf{S})$, choose $\mathsf{Head} = R_0 \xleftarrow{\$} \{0,1\}^\lambda$, and output $(\mathsf{pk}_\mathsf{S}, \mathsf{sk}_\mathsf{S}) = ((\mathsf{S}, \mathsf{Head}, \mathsf{NIKE.pk}_\mathsf{Sender}), (\mathsf{S}, \mathsf{Head}, \mathsf{NIKE.sk}_\mathsf{Sender}, \mathsf{HS}))$ where HS is a list to store hidden structures (with the form $\{(kw, R)\}$) and is initialized as $\mathsf{HS} := \emptyset$.

$\mathsf{SPCHS.Enc}(\mathsf{pk}_\mathsf{R}, \mathsf{sk}_\mathsf{S}, kw)$**:** Parse $\mathsf{pk}_\mathsf{R} = (\mathsf{params}, \mathsf{R}, \mathsf{NIKE.pk}_\mathsf{Receiver})$ and $\mathsf{sk}_\mathsf{S} = (\mathsf{S}, \mathsf{Head}, \mathsf{NIKE.sk}_\mathsf{Sender}, \mathsf{HS})$. Run $k_\mathsf{shared} \leftarrow \mathsf{SharedKey}(\mathsf{S}, \mathsf{NIKE.sk}_\mathsf{Sender}, \mathsf{R}, \mathsf{NIKE.pk}_\mathsf{Receiver})$. Compute $T_{kw} := K_{kw} || k_{kw} = F_{k_\mathsf{shared}}(kw)$ where $K_{kw}, k_{kw} \in \{0,1\}^\lambda$. Search $(kw, \cdot) \in \mathsf{HS}$.

- If there is no such the entry, compute $R_1 \leftarrow F'_{K_{kw}}(R_0))$ where $\mathsf{Head} = R_0$, randomly choose $R_2 \xleftarrow{\$} \{0,1\}^\lambda$, update $\mathsf{HS} := \mathsf{HS} \cup \{(kw, R_2)\}$, and output $C = (R_1, \mathsf{SKE.Enc}(k_{kw}, R_2))$.
- Let $(kw, R) \in \mathsf{HS}$. Randomly choose $R' \xleftarrow{\$} \{0,1\}^\lambda$, update $\mathsf{HS} := \mathsf{HS} \setminus \{(kw, R)\}$ and $\mathsf{HS} := \mathsf{HS} \cup \{(kw, R')\}$, and output $C = (R, \mathsf{SKE.Enc}(k_{kw}, R'))$.

$\mathsf{Trapdoor}(\mathsf{sk}_\mathsf{R}, \mathsf{pk}_\mathsf{S}, kw)$**:** Parse $\mathsf{sk}_\mathsf{R} = (\mathsf{R}, \mathsf{NIKE.sk}_\mathsf{Receiver})$ and $\mathsf{pk}_\mathsf{S} = (\mathsf{S}, \mathsf{Head}, \mathsf{NIKE.pk}_\mathsf{Sender})$. Run $k_\mathsf{shared} = \mathsf{SharedKey}(\mathsf{S}, \mathsf{NIKE.pk}_\mathsf{Sender}, \mathsf{R}, \mathsf{NIKE.sk}_\mathsf{Receiver})$, compute $T_{kw} = F_{k_\mathsf{shared}}(kw)$, and output T_{kw}.

$\mathsf{Search}(\mathsf{pk}_\mathsf{R}, \mathsf{pk}_\mathsf{S}, \mathsf{CSet}, T_{kw})$**:** Parse $\mathsf{pk}_\mathsf{R} = (\mathsf{params}, \mathsf{R}, \mathsf{NIKE.pk}_\mathsf{Receiver})$, $\mathsf{pk}_\mathsf{S} = (\mathsf{S}, \mathsf{Head}, \mathsf{NIKE.pk}_\mathsf{Sender})$, and $T_{kw} := K_{kw} || k_{kw}$. Let $C \in \mathsf{CSet}$ be a ciphertext and the ciphertext can be parsed as $(C[1], C[2]) = (R, C_\mathsf{SKE})$. Initialize $\mathsf{CSet}' := \emptyset$.

1. Compute $R \leftarrow F'_{K_{kw}}(R_0)$ where $\mathsf{Head} = R_0$.
2. From CSet, seek a ciphertext having $C[1] = R$ (this part can be done without any cryptographic operation and efficient search algorithms such as hash table can be employed).
   - If it exists, add C to CSet' and remove C from CSet.
   - If no matching ciphertext is found, output CSet'.
3. Compute $R \leftarrow \mathsf{SKE.Dec}(k_{kw}, C[2])$ and go to Step 2.

Obviously, correctness holds if NIKE is correct (that guarantees $\mathsf{SharedKey}(\mathsf{S}, \mathsf{NIKE.pk}_\mathsf{Sender}, \mathsf{R}, \mathsf{NIKE.sk}_\mathsf{Receiver}) = \mathsf{SharedKey}(\mathsf{S}, \mathsf{NIKE.sk}_\mathsf{Sender}, \mathsf{R}, \mathsf{NIKE.pk}_\mathsf{Receiver})$) and SKE is correct (that guarantees $\mathsf{SKE.Dec}(k_{kw}, C_\mathsf{SKE})$ correctly outputs R where $C_\mathsf{SKE} = \mathsf{SKE.Enc}(T_{kw}, R)$).

4.2 Lightweight Instantiation from DH NIKE

We explicitly describe an efficient instantiation. Let $\mathbb{G}$ be a group with prime order p (i.e., Curve 25519 [5]), $g \in \mathbb{G}$ be a generator, and $H : \{0,1\}^* \to \mathbb{G}$ be a hash function which is modeled as a random oracle. Here, a DH key is $\mathsf{NIKE.pk}_\mathsf{Receiver}^{\mathsf{NIKE.sk}_\mathsf{Sender}} = \mathsf{NIKE.pk}_\mathsf{Sender}^{\mathsf{NIKE.sk}_\mathsf{Receiver}} = g^{\mathsf{NIKE.sk}_\mathsf{Sender} \cdot \mathsf{NIKE.sk}_\mathsf{Receiver}}$. Note that the DH key is computed once for encryption and trapdoor generation respectively, and the search algorithm depends solely on symmetric-key primitives and is independent of any public-key primitives. F and F' are PRFs instantiated by SHA256-based HMAC, and SKE is AES (in counter mode).

- $\mathsf{SPCHS.Setup}(1^\lambda)$: Let R be the identity of the receiver who runs the algorithm. Run $(\mathbb{G}, g, p, H) \leftarrow \mathsf{NIKE.Setup}(1^\lambda)$, choose $\mathsf{NIKE.sk}_\mathsf{Receiver} := x_\mathsf{R} \xleftarrow{\$} \mathbb{Z}_p$, and compute $\mathsf{NIKE.pk}_\mathsf{Receiver} = g^{x_\mathsf{R}}$. Output $(\mathsf{pk}_\mathsf{R}, \mathsf{sk}_\mathsf{R}) = ((\mathsf{params}, \mathsf{R}, \mathsf{NIKE.pk}_\mathsf{Receiver}), (\mathsf{R}, \mathsf{NIKE.sk}_\mathsf{Receiver}))$.

- $\mathsf{Structure}(\mathsf{pk_R})$: Let S be the identity of the sender who runs the algorithm. Parse $\mathsf{pk_R} = (\mathsf{params}, \mathsf{R}, \mathsf{NIKE.pk_{Receiver}})$. Choose $\mathsf{NIKE.sk_{Sender}} := x_\mathsf{S} \xleftarrow{\$} \mathbb{Z}_p$ and compute $\mathsf{NIKE.pk_{Sender}} = g^{x_\mathsf{S}}$. Choose $\mathsf{Head} = R_0 \xleftarrow{\$} \{0,1\}^\lambda$, and output $(\mathsf{pk_S}, \mathsf{sk_S}) = ((\mathsf{S}, \mathsf{Head}, \mathsf{NIKE.pk_{Sender}}), (\mathsf{S}, \mathsf{Head}, \mathsf{NIKE.sk_{Sender}}, \mathsf{HS}))$ where HS is a list to store hidden structures (with the form $\{(kw, R) \in \mathcal{KS} \times \{0,1\}^\lambda\}$) and is initialized as $\mathsf{HS} := \emptyset$.

- $\mathsf{SPCHS.Enc}(\mathsf{pk_R}, \mathsf{sk_S}, kw)$: Parse $\mathsf{pk_R} = (\mathsf{params}, \mathsf{R}, \mathsf{NIKE.pk_{Receiver}})$ and $\mathsf{sk_S} = (\mathsf{S}, \mathsf{Head}, \mathsf{NIKE.sk_{Sender}}, \mathsf{HS})$ where $\mathsf{NIKE.sk_{Sender}} = x_\mathsf{S}$. Compute $k_\mathsf{shared} = H(\mathsf{S}, \mathsf{R}, \mathsf{NIKE.pk}_\mathsf{Receiver}^{x_\mathsf{S}})$. Compute $T_{kw} := K_{kw}||k_{kw} = F_{k_\mathsf{shared}}(kw)$ where $K_{kw}, k_{kw} \in \{0,1\}^\lambda$. Search $(kw, \cdot) \in \mathsf{HS}$.
  - If there is no such the entry, compute $R_1 \leftarrow F'_{K_{kw}}(R_0))$ where $\mathsf{Head} = R_0$, randomly choose $R_2 \xleftarrow{\$} \{0,1\}^\lambda$, update $\mathsf{HS} := \mathsf{HS} \cup \{(kw, R_2)\}$, and output $C = (R_1, \mathsf{SKE.Enc}(k_{kw}, R_2))$.
  - Let $(kw, R) \in \mathsf{HS}$. Randomly choose $R' \xleftarrow{\$} \{0,1\}^\lambda$, update $\mathsf{HS} := \mathsf{HS} \setminus \{(kw, R)\}$ and $\mathsf{HS} := \mathsf{HS} \cup \{(kw, R')\}$, and output $C = (R, \mathsf{SKE.Enc}(k_{kw}, R'))$.

- $\mathsf{Trapdoor}(\mathsf{sk_R}, \mathsf{pk_S}, kw)$: Parse $\mathsf{sk_R} = (\mathsf{R}, \mathsf{NIKE.sk_{Receiver}})$ and $\mathsf{pk_S} = (\mathsf{S}, \mathsf{Head}, \mathsf{NIKE.pk_{Sender}})$ where $\mathsf{NIKE.sk_{Receiver}} = x_\mathsf{R}$. Compute $k_\mathsf{shared} = H(\mathsf{S}, \mathsf{R}, \mathsf{NIKE.pk}_\mathsf{Sender}^{x_\mathsf{R}})$ and $T_{kw} = F_{k_\mathsf{shared}}(kw)$, and output T_{kw}.

- $\mathsf{Search}(\mathsf{pk_R}, \mathsf{pk_S}, \mathsf{CSet}, T_{kw})$: Parse $\mathsf{pk_R} = (\mathsf{params}, \mathsf{R}, \mathsf{NIKE.pk_{Receiver}})$, $\mathsf{pk_S} = (\mathsf{S}, \mathsf{Head}, \mathsf{NIKE.pk_{Sender}})$, and $T_{kw} := K_{kw}||k_{kw}$. Let $C \in \mathsf{CSet}$ be a ciphertext and the ciphertext can be parsed as $(C[1], C[2]) = (R, C_\mathsf{SKE})$. Initialize $\mathsf{CSet}' := \emptyset$.
  1. Compute $R \leftarrow F'_{K_{kw}}(R_0)$ where $\mathsf{Head} = R_0$.
  2. From CSet, seek a ciphertext having $C[1] = R$.
     - If it exists, add C to CSet' and remove C from CSet.
     - If no matching ciphertext is found, output CSet'.
  3. Compute $R \leftarrow \mathsf{SKE.Dec}(k_{kw}, C[2])$ and go to Step 2.

5 Security Proofs

Theorem 1. *The proposed SPCHS construction is consistent if* NIKE *is HKR-CKS secure and* PRF *and* PRF′ *are pseudo-random.*

Proof Overview. The condition $C \in \mathsf{CSet}'$ happens if, for $C = (R_1, C_\mathsf{SKE})$ and $\mathsf{Head} = R_0$ where $R_1 = F'_{K_{kw}}(R_0)$, $R_1 = F'_{K_{kw'}}(R_0)$ holds where $T_{kw'} := K_{kw'}||\cdot$. This happens with negligible probability due to the pseudo-randomness of PRF′. To reduce the pseudo-randomness, the PRF keys, K_{kw} and $K_{kw'}$ need to be randomly chosen. Since they are generated by PRF by $T_{kw} := K_{kw}||k_{kw} = F_{k_\mathsf{shared}}(kw)$ and $T_{kw'} := K_{kw'}||k_{kw'} = F_{k_\mathsf{shared}}(kw')$, we also need to switch k_shared to a random value due to the security of NIKE. We remark that the Search algorithm adds a ciphertext C' to CSet' if for $R \leftarrow \mathsf{SKE.Dec}(k_{kw}, C'[2])$, $C'[1] = R$ holds. This indicates that a security of SKE may be related to provide consistency. However, our proof shows that the first connection, compute $R \leftarrow F'_{K_{kw'}}(R_0)$ where $\mathsf{Head} = R_0$ and seek a ciphertext having $C[1] = R$, fails since $R \neq R_1$. Thus, no security of SKE is required to provide consistency. We prove the theorem via sequences of games $\mathsf{Game}_0, \ldots, \mathsf{Game}_3$. Let $\mathcal{A}$ be an adversary of consistency and E_k denote an event that $\mathcal{A}$ wins in Game_k $(k \in \{0,1,2,3\})$.

Game_0**:** This game is the same as the original computational consistency game. Due to the definition, $\mathsf{Adv}^{\mathsf{Consist}}_{\mathsf{SPCHS},\mathcal{A}}(\lambda) = \Pr[E_0]$.

Game_1**:** This game is the same as Game_0 except that $k_{\mathsf{shared}} \xleftarrow{\$} \{0,1\}^\lambda$.

Lemma 1. *There exists an algorithm* $\mathcal{B}_1$ *where* $|\Pr[E_0] - \Pr[E_1]| \leq \mathsf{Adv}^{\mathsf{HKR\text{-}CKS}}_{\mathsf{NIKE},\mathcal{B}_1}(\lambda)$ *holds.*

Proof. Let $\mathcal{C}$ be the challenger of NIKE. $\mathcal{C}$ runs $\mathsf{params} \leftarrow \mathsf{NIKE.Setup}(1^\lambda)$ and sends params to $\mathcal{B}_1$. $\mathcal{B}_1$ sends R to the RegHon oracle. $\mathcal{C}$ runs $(\mathsf{NIKE.pk}_{\mathsf{Receiver}}, \mathsf{NIKE.sk}_{\mathsf{Receiver}}) \leftarrow \mathsf{NIKE.KeyGen}(\mathsf{params}, \mathsf{R})$ and returns $\mathsf{NIKE.pk}_{\mathsf{Receiver}}$ to $\mathcal{B}_1$. $\mathcal{B}_1$ also sends S to the RegHon oracle. $\mathcal{C}$ runs $(\mathsf{NIKE.pk}_{\mathsf{Sender}}, \mathsf{NIKE.sk}_{\mathsf{Sender}}) \leftarrow \mathsf{NIKE.KeyGen}(\mathsf{params}, \mathsf{S})$ and returns $\mathsf{NIKE.pk}_{\mathsf{Sender}}$ to $\mathcal{B}_1$. $\mathcal{B}_1$ randomly chooses $\mathsf{Head} = R_0 \xleftarrow{\$} \{0,1\}^\lambda$. $\mathcal{B}_1$ sends $(\mathsf{pk}_{\mathsf{R}}, \mathsf{pk}_{\mathsf{S}}) = ((\mathsf{params}, \mathsf{R}, \mathsf{NIKE.pk}_{\mathsf{Receiver}}), (\mathsf{S}, \mathsf{Head}, \mathsf{NIKE.pk}_{\mathsf{Sender}}))$ to $\mathcal{A}$. $\mathcal{A}$ declares kw and kw'. $\mathcal{B}_1$ sends (S, R) to the Chall oracle. $\mathcal{C}$ flips $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, then $\mathcal{C}$ returns $k_{\mathsf{shared}} \xleftarrow{\$} \{0,1\}^\lambda$ to $\mathcal{B}_1$ and if $b = 1$, then $\mathcal{C}$ returns $k_{\mathsf{shared}} \leftarrow \mathsf{SharedKey}(\mathsf{S}, \mathsf{NIKE.sk}_{\mathsf{Sender}}, \mathsf{R}, \mathsf{NIKE.pk}_{\mathsf{Receiver}})$ to $\mathcal{B}_1$. If $b = 0$, then $\mathcal{B}_1$ simulates Game_0 and if $b = 1$, then $\mathcal{B}_1$ simulates Game_1. Thus, $|\Pr[E_0] - \Pr[E_1]| \leq \mathsf{Adv}^{\mathsf{HKR\text{-}CKS}}_{\mathsf{NIKE},\mathcal{B}_1}(\lambda)$ holds. □

Game_2**:** This game is the same as Game_1 except that $T_{kw}, T_{kw'} \xleftarrow{\$} \{0,1\}^{2\lambda}$.

Lemma 2. *There exists an algorithm* $\mathcal{B}_2$ *where* $|\Pr[E_1] - \Pr[E_2]| \leq \mathsf{Adv}^{\mathsf{pseudo\text{-}random}}_{\mathsf{PRF},\mathcal{B}_2}(\lambda)$ *holds.*

Proof. Let $\mathcal{C}$ be the challenger of PRF. $\mathcal{C}$ flips $b \xleftarrow{\$} \{0,1\}$ and $K \xleftarrow{\$} \{0,1\}^\lambda$. $\mathcal{B}_2$ implicitly sets $K = k^*_{\mathsf{shared}}$ (this is possible since $k^*_{\mathsf{shared}} \xleftarrow{\$} \{0,1\}^\lambda$ in this game). $\mathcal{B}_2$ runs $\mathsf{params} \leftarrow \mathsf{NIKE.Setup}(1^\lambda)$, $(\mathsf{NIKE.pk}_{\mathsf{Receiver}}, \mathsf{NIKE.sk}_{\mathsf{Receiver}}) \leftarrow \mathsf{NIKE.KeyGen}(\mathsf{params}, \mathsf{R})$, and $(\mathsf{NIKE.pk}_{\mathsf{Sender}}, \mathsf{NIKE.sk}_{\mathsf{Sender}}) \leftarrow \mathsf{NIKE.KeyGen}(\mathsf{params}, \mathsf{S})$. $\mathcal{B}_2$ randomly chooses $\mathsf{Head} = R_0 \xleftarrow{\$} \{0,1\}^\lambda$. $\mathcal{B}_2$ sends $(\mathsf{pk}_{\mathsf{R}}, \mathsf{pk}_{\mathsf{S}}) = ((\mathsf{params}, \mathsf{R}, \mathsf{NIKE.pk}_{\mathsf{Receiver}}), (\mathsf{S}, \mathsf{Head}, \mathsf{NIKE.pk}_{\mathsf{Sender}}))$ to $\mathcal{A}$. $\mathcal{A}$ declares kw and kw'. $\mathcal{B}_2$ sends kw to $\mathcal{C}$. If $b = 0$, then $\mathcal{C}$ returns $F_K(kw)$ to $\mathcal{B}_2$ and if $b = 1$, then $\mathcal{C}$ returns a 2λ-bit random value. $\mathcal{B}_2$ sets the returned value as T_{kw}. Similarly, $\mathcal{B}_2$ sends kw' to $\mathcal{C}$. If $b = 0$, then $\mathcal{C}$ returns $F_K(kw')$ to $\mathcal{B}_2$ and if $b = 1$, then $\mathcal{C}$ returns a 2λ-bit random value. $\mathcal{B}_2$ sets the returned value as $T_{kw'}$. If $b = 0$, then $\mathcal{B}_1$ simulates Game_1 and if $b = 1$, then $\mathcal{B}_1$ simulates Game_2. Thus, $|\Pr[E_1] - \Pr[E_2]| \leq \mathsf{Adv}^{\mathsf{pseudo\text{-}random}}_{\mathsf{PRF},\mathcal{B}_2}(\lambda)$ holds. □

Game_3**:** This game is the same as Game_2 except that $R_1 \xleftarrow{\$} \{0,1\}^\lambda$.

Lemma 3. *There exists an algorithm* $\mathcal{B}_3$ *where* $|\Pr[E_2] - \Pr[E_3]| \leq \mathsf{Adv}^{\mathsf{pseudo\text{-}random}}_{\mathsf{PRF}',\mathcal{B}_3}(\lambda)$ *holds.*

Proof. Let $\mathcal{C}$ be the challenger of PRF′. $\mathcal{C}$ flips $b \xleftarrow{\$} \{0,1\}$ and $K \xleftarrow{\$} \{0,1\}^\lambda$. $\mathcal{B}_3$ implicitly sets $K = k^*_{\mathsf{shared}}$. $\mathcal{B}_3$ runs $\mathsf{params} \leftarrow \mathsf{NIKE.Setup}(1^\lambda)$, $(\mathsf{NIKE.pk}_{\mathsf{Receiver}}, \mathsf{NIKE.sk}_{\mathsf{Receiver}}) \leftarrow \mathsf{NIKE.KeyGen}(\mathsf{params}, \mathsf{R})$, and $(\mathsf{NIKE.pk}_{\mathsf{Sender}}, \mathsf{NIKE.sk}_{\mathsf{Sender}}) \leftarrow \mathsf{NIKE.KeyGen}(\mathsf{params}, \mathsf{S})$. $\mathcal{B}_3$ randomly chooses $\mathsf{Head} = R_0 \xleftarrow{\$} \{0,1\}^\lambda$. $\mathcal{B}_3$ sends $(\mathsf{pk}_{\mathsf{R}}, \mathsf{pk}_{\mathsf{S}}) = ((\mathsf{params}, \mathsf{R}, \mathsf{NIKE.pk}_{\mathsf{Receiver}}), (\mathsf{S}, \mathsf{Head}, \mathsf{NIKE.pk}_{\mathsf{Sender}}))$ to $\mathcal{A}$. $\mathcal{A}$ declares kw and kw'. $\mathcal{B}_3$ implicitly sets $K' = K_{kw}$ (this is possible since $T_{kw} := K_{kw}||k_{kw} \xleftarrow{\$} \{0,1\}^{2\lambda}$ in this game). $\mathcal{B}_3$ randomly chooses $k_{kw} \xleftarrow{\$} \{0,1\}^\lambda$ and $T_{kw'} \xleftarrow{\$} \{0,1\}^{2\lambda}$. $\mathcal{B}_3$ sends R_0 to $\mathcal{C}$. If $b = 0$, then $\mathcal{C}$ returns $F'_{K'}(R_0)$ to $\mathcal{B}_3$ and if $b = 1$, then $\mathcal{C}$ returns a λ-bit random value. $\mathcal{B}_3$ sets the returned value as R_1. $\mathcal{B}_3$ computes

$C_{\mathsf{SKE}} \leftarrow \mathsf{SKE.Enc}(k_{kw}, R_1)$ and sets $C = (R_0, C_{\mathsf{SKE}})$. If $b = 0$, then $\mathcal{B}_3$ simulates Game_2 and if $b = 1$, then $\mathcal{B}_3$ simulates Game_3. Thus, $|\Pr[E_2] - \Pr[E_3]| \leq \mathsf{Adv}^{\mathsf{pseudo\text{-}random}}_{\mathsf{PRF}',\mathcal{B}_3}(\lambda)$ holds. □

Finally, we evaluate $\Pr[E_3]$. In this game, $\mathsf{Head} = R_0 \xleftarrow{\$} \{0,1\}^\lambda$, $R_1 \xleftarrow{\$} \{0,1\}^\lambda$, and $T_{kw'} := K'_{kw}||k'_{kw} \xleftarrow{\$} \{0,1\}^{2\lambda}$. $C \in \mathsf{CSet}'$ if $R_1 = F'_{K_{kw'}}(R_0)$ holds. We remark that $F'_{K_{kw'}}(R_0) \xleftarrow{\$} \{0,1\}^\lambda$ in this game. Thus, $\Pr[E_3] = 2^{-\lambda}$.

Putting it all together, we have

$$\begin{aligned}
&\mathsf{Adv}^{\mathsf{Consist}}_{\mathsf{SPCHS},\mathcal{A}}(\lambda) \\
&= \Pr[E_0] \\
&= \Pr[E_0] - \Pr[E_1] + \Pr[E_1] - \Pr[E_2] + \Pr[E_2] - \Pr[E_3] + \Pr[E_3] \\
&\leq |\Pr[E_0] - \Pr[E_1]| + |\Pr[E_1] - \Pr[E_2]| + |\Pr[E_2] - \Pr[E_3]| + \Pr[E_3] \\
&\leq \mathsf{Adv}^{\mathsf{HKR\text{-}CKS}}_{\mathsf{NIKE},\mathcal{B}_1}(\lambda) + \mathsf{Adv}^{\mathsf{pseudo\text{-}random}}_{\mathsf{PRF},\mathcal{B}_2}(\lambda) + \mathsf{Adv}^{\mathsf{pseudo\text{-}random}}_{\mathsf{PRF}',\mathcal{B}_3}(\lambda) + 2^{-\lambda}
\end{aligned}$$

This concludes the proof of Theorem 1. □

Theorem 2. *The proposed SPCHS construction is IND-CKSA secure if* NIKE *is HKR-CKS secure and* PRF, PRF′ *are pseudo-random, and* SKE *is IND-PCPA secure.*

Proof Overview. The ultimate objective is to utilize IND-PCPA security in order to replace C_{SKE} with a random value. Then, since $C = (R, C_{\mathsf{SKE}})$ is randomly distributed over $\{0,1\}^\lambda \times \mathsf{CSpace}$ and independent of the keyword, no keyword information is revealed from the ciphertexts. We remark that, for the challenge ciphertext $C^* = (R, C_{\mathsf{SKE}})$, the last ciphertext of the hidden-structure chain for kw^*_b together with the challenge sender's public key contains an SKE ciphertext of R linking the chain to C^*. Therefore, we also need to replace the SKE ciphertext with a random value, since otherwise it would reveal information about the hidden-structure chain. To employ the IND-PCPA security, we need to guarantee that the secret key k_{kw} is randomly chosen. Thus, as in the proof of Theorem 1, we employ the pseudo-randomness of PRF and PRF′. Before utilizing the pseudo-randomness, we need to guarantee that k_{shared} is random. This replacement can be done by HKR-CKS security of NIKE. Since an adversary may issue Reveal queries, the simulator needs to respond $\mathsf{NIKE.sk}_{\mathsf{Receiver}}$. Thus, the simulator first guesses when $\mathsf{pk}^{*(b)}_{\mathsf{S}}$ is generated and then the simulator uses the RegHon oracle. Let $\mathcal{A}$ be an adversary of IND-CKSA security and E_k denote an event that $\mathcal{A}$ wins in Game_k ($k \in \{0, \ldots, 6\}$).

Game_0**:** This game is the same as the original IND-CKSA game. Due to the definition, $\mathsf{Adv}^{\mathsf{IND\text{-}CKSA}}_{\mathsf{SPCHS},\mathcal{A}}(\lambda) = |\Pr[E_0] - 1/2|$.

Game_1**:** Let $q_s = \mathsf{poly}(\lambda)$ be the number of Structure queries and $i \xleftarrow{\$} [q_s]$. This game is the same as Game_0 except that the game aborts if the i-th Structure query does not return $\mathsf{pk}^{*(b)}_{\mathsf{S}}$. The guessing is correct with probability $1/q_s$ (which is non-negligible). Since Game_0 and Game_1 are identical if the guessing is correct, we have $|\Pr[E_0] - 1/2| = q_s|\Pr[E_1] - 1/2|$.

Game_2**:** This game is the same as Game_1 except that $k_{\mathsf{shared}} \xleftarrow{\$} \{0,1\}^\lambda$ where k_{shared} is used for generating the challenge ciphertext C^*.

Lemma 4. *There exists an algorithm* $\mathcal{B}_2$ *where* $|\Pr[E_1] - \Pr[E_2]| \leq \mathsf{Adv}^{\mathsf{HKR\text{-}CKS}}_{\mathsf{NIKE},\mathcal{B}_2}(\lambda)$ *holds.*

Proof. Let $\mathcal{C}$ be the challenger of NIKE. $\mathcal{C}$ runs $\mathsf{params} \leftarrow \mathsf{NIKE.Setup}(1^\lambda)$ and sends params to $\mathcal{B}_2$. $\mathcal{B}_2$ sends R to the RegHon oracle. $\mathcal{C}$ runs $(\mathsf{NIKE.pk}_{\mathsf{Receiver}}, \mathsf{NIKE.sk}_{\mathsf{Receiver}}) \leftarrow \mathsf{NIKE.KeyGen}(\mathsf{params}, \mathsf{R})$ and returns $\mathsf{NIKE.pk}_{\mathsf{Receiver}}$ to $\mathcal{B}_2$. $\mathcal{B}_2$ initializes $\mathsf{KSet} := \emptyset$ and $\mathsf{CKSet} := \emptyset$ and sends $\mathsf{pk}_{\mathsf{R}} = (\mathsf{params}, \mathsf{R}, \mathsf{NIKE.pk}_{\mathsf{Receiver}})$ to $\mathcal{A}$. $\mathcal{B}_2$ flips $b \xleftarrow{\$} \{0,1\}$ and simulates each oracle as follows.

- Structure: $\mathcal{A}$ sends $\mathsf{pk}_{\mathsf{R}} = (\mathsf{params}, \mathsf{R}, \mathsf{NIKE.pk}_{\mathsf{Receiver}})$.
  - If this is the i-th query, $\mathcal{B}_2$ sends S_i to the RegHon oracle. $\mathcal{C}$ runs $(\mathsf{NIKE.pk}_{\mathsf{Sender}}, \mathsf{NIKE.sk}_{\mathsf{Sender}}) \leftarrow \mathsf{NIKE.KeyGen}(\mathsf{params}, \mathsf{S}_i)$ and returns $\mathsf{NIKE.pk}_{\mathsf{Sender}}$ to $\mathcal{B}_2$. $\mathcal{B}_2$ randomly chooses $\mathsf{Head}_i \xleftarrow{\$} \{0,1\}^\lambda$, and sets $\mathsf{pk}_{\mathsf{S}} = (\mathsf{S}_i, \mathsf{Head}_i, \mathsf{NIKE.pk}^{(i)}_{\mathsf{Sender}})$. $\mathcal{B}_2$ updates $\mathsf{KSet} \leftarrow \mathsf{KSet} \cup \{(\mathsf{pk}_{\mathsf{S}}\,(\mathsf{S}_i, \mathsf{Head}, -, \mathsf{HS}_i))\}$ where $\mathsf{HS}_i := \emptyset$, and returns $\mathsf{pk}^{*(b)}_{\mathsf{S}} = \mathsf{pk}^{(i)}_{\mathsf{S}}$ to $\mathcal{A}$.
  - Otherwise, in the j-th query (i.e., $j \neq i$), $\mathcal{B}_2$ runs $(\mathsf{NIKE.pk}^{(j)}_{\mathsf{Sender}}, \mathsf{NIKE.sk}^{(j)}_{\mathsf{Sender}}) \leftarrow \mathsf{NIKE.KeyGen}(\mathsf{params}, \mathsf{S}_j)$, chooses $\mathsf{Head}_j \xleftarrow{\$} \{0,1\}^\lambda$, sets $(\mathsf{pk}^{(j)}_{\mathsf{S}}, \mathsf{sk}^{(j)}_{\mathsf{S}}) = ((\mathsf{S}_j, \mathsf{Head}_j, \mathsf{NIKE.pk}^{(j)}_{\mathsf{Sender}}), (\mathsf{S}_j, \mathsf{Head}_j, \mathsf{NIKE.sk}^{(j)}_{\mathsf{Sender}}, \mathsf{HS}_j))$ where $\mathsf{HS}_j := \emptyset$, updates $\mathsf{KSet} \leftarrow \mathsf{KSet} \cup \{(\mathsf{pk}^{(j)}_{\mathsf{S}}, \mathsf{sk}^{(j)}_{\mathsf{S}})\}$, and returns $\mathsf{pk}^{(j)}_{\mathsf{S}}$ to $\mathcal{A}$.
- Reveal: $\mathcal{A}$ sends pk_{S}. If $(\mathsf{pk}_{\mathsf{S}}, \cdot) \notin \mathsf{KSet}$, then $\mathcal{B}_2$ returns $\perp$. Otherwise, let $(\mathsf{pk}^{(j)}_{\mathsf{S}}, \mathsf{sk}^{(j)}_{\mathsf{S}}) \in \mathsf{KSet}$ such that $\mathsf{pk}_{\mathsf{S}} = \mathsf{pk}^{(j)}_{\mathsf{S}}$ (since the guessing of i is correct in this game, $\mathsf{sk}^{(j)}_{\mathsf{S}}$ is stored in KSet). $\mathcal{B}_2$ returns $\mathsf{sk}^{(j)}_{\mathsf{S}}$ and updates $\mathsf{CKSet} \leftarrow \mathsf{CKSet} \cup \{(\mathsf{pk}^{(j)}_{\mathsf{S}}\, \mathsf{sk}^{(j)}_{\mathsf{S}})\}$.
- SPCHS.Enc: $\mathcal{A}$ sends pk_{R}, pk_{S}, and kw. If $(\mathsf{pk}_{\mathsf{S}}, \cdot) \notin \mathsf{KSet}$, then $\mathcal{B}_2$ returns $\perp$.
  - If $\mathsf{pk}_{\mathsf{S}} = \mathsf{pk}^{(i)}_{\mathsf{S}}$, then $\mathcal{B}_2$ sends $(\mathsf{S}_i, \mathsf{R})$ to the Chall oracle. $\mathcal{C}$ returns either $k_{\mathsf{shared}} \leftarrow \mathsf{SharedKey}(\mathsf{S}_i, \mathsf{NIKE.sk}^{(i)}_{\mathsf{Sender}}, \mathsf{R}, \mathsf{NIKE.pk}_{\mathsf{Receiver}})$ or $k_{\mathsf{shared}} \xleftarrow{\$} \{0,1\}^\lambda$.
  - Otherwise, let $\mathsf{pk}_{\mathsf{S}} = \mathsf{pk}^{(j)}_{\mathsf{S}}$ and retrieve $(\mathsf{pk}^{(j)}_{\mathsf{S}}, \mathsf{sk}^{(j)}_{\mathsf{S}}) \in \mathsf{KSet}$. $\mathcal{B}_2$ runs $k_{\mathsf{shared}} \leftarrow \mathsf{SharedKey}(\mathsf{S}_j, \mathsf{NIKE.sk}^{(j)}_{\mathsf{Sender}}, \mathsf{R}, \mathsf{NIKE.pk}_{\mathsf{Receiver}})$.

  $\mathcal{B}_2$ computes $T_{kw} := K_{kw} || k_{kw} = F_{k_{\mathsf{shared}}}(kw)$. The remaining part is the same as the SPCHS.Enc algorithm.
- Trapdoor: $\mathcal{A}$ sends pk_{S} and kw. If $(\mathsf{pk}_{\mathsf{S}}, \cdot) \notin \mathsf{KSet}$, then return $\perp$. Otherwise, $\mathcal{B}_2$ computes $T_{kw} := K_{kw} || k_{kw} = F_{k_{\mathsf{shared}}}(kw)$ as in the SPCHS.Enc query and returns T_{kw}.

$\mathcal{A}$ declares $(kw^*_0, kw^*_1, \mathsf{pk}^{*(0)}_{\mathsf{S}}, \mathsf{pk}^{*(1)}_{\mathsf{S}})$. $\mathcal{B}_1$ sends $(\mathsf{S}_i, \mathsf{R})$ to the Chall oracle (if $\mathcal{B}_2$ did not send the query), and $\mathcal{B}_1$ computes $T_{kw^*_b} := K_{kw^*_b} || k_{kw^*_b} = F_{k_{\mathsf{shared}}}(kw^*_b)$. If $\mathcal{C}$ returns $k_{\mathsf{shared}} \leftarrow \mathsf{SharedKey}(\mathsf{S}_i, \mathsf{NIKE.sk}^{(i)}_{\mathsf{Sender}}, \mathsf{R}, \mathsf{NIKE.pk}_{\mathsf{Receiver}})$, then $\mathcal{B}_2$ simulates Game_1 and if $k_{\mathsf{shared}} \xleftarrow{\$} \{0,1\}^\lambda$, then $\mathcal{B}_2$ simulates Game_2. Thus, $|\Pr[E_1] - \Pr[E_2]| \leq \mathsf{Adv}^{\mathsf{HKR\text{-}CKS}}_{\mathsf{NIKE},\mathcal{B}_1}(\lambda)$ holds. □

Game_3**:** This game is the same as Game_2 except that $T_{kw}, T_{kw'} \xleftarrow{\$} \{0,1\}^{2\lambda}$.

Lemma 5. *There exists an algorithm $\mathcal{B}_3$ where $|\Pr[E_2] - \Pr[E_3]| \leq \mathsf{Adv}^{\mathsf{pseudo\text{-}random}}_{\mathsf{PRF},\mathcal{B}_3}(\lambda)$ holds.*

Proof. Let $\mathcal{C}$ be the challenger of PRF. $\mathcal{C}$ chooses $K \xleftarrow{\$} \{0,1\}^\lambda$. $\mathcal{B}_3$ runs $\mathsf{params} \leftarrow \mathsf{NIKE.Setup}(1^\lambda)$ and $(\mathsf{NIKE.pk}_{\mathsf{Receiver}}, \mathsf{NIKE.sk}_{\mathsf{Receiver}}) \leftarrow \mathsf{NIKE.KeyGen}(\mathsf{params}, \mathsf{R})$. $\mathcal{B}_3$ initializes $\mathsf{KSet} := \emptyset$ and $\mathsf{CKSet} := \emptyset$ and sends $\mathsf{pk}_{\mathsf{R}} = (\mathsf{params}, \mathsf{R}, \mathsf{NIKE.pk}_{\mathsf{Receiver}})$ to $\mathcal{A}$. $\mathcal{B}_3$ flips $b \xleftarrow{\$} \{0,1\}$ and implicitly sets $K = k^*_{\mathsf{shared}}$ (this is possible since $k_{\mathsf{shared}} \xleftarrow{\$} \{0,1\}^\lambda$ in this

game). For all Structure queries, $\mathcal{B}_3$ runs the NIKE.KeyGen algorithm that allows $\mathcal{B}_3$ to compute T_{kw}, except the following case. If $\mathcal{A}$ sends $(\mathsf{pk}_S = \mathsf{pk}_S^{*(b)}, kw)$ as SPCHS.Enc or Trapdoor queries, $\mathcal{B}_3$ sends kw to $\mathcal{C}$. $\mathcal{C}$ returns either $F_K(kw)$ or a 2λ-bit random value. $\mathcal{B}_2$ sets the returned value as $T_{kw} := K_{kw}||k_{kw}$. In the challenge phase, $\mathcal{B}_3$ sends kw_b^* to $\mathcal{C}$ and obtains $T_{kw_b^*} := K_{kw_b^*}||k_{kw_b^*}$ and uses $T_{kw_b^*}$ to compute the challenge ciphertext. If $\mathcal{C}$ returns $F_K(kw)$, then $\mathcal{B}_3$ simulates Game_2 and if $\mathcal{C}$ returns a 2λ-bit random value, then $\mathcal{B}_3$ simulates Game_3. Thus, $|\Pr[E_2] - \Pr[E_3]| \leq \mathsf{Adv}^{\text{pseudo-random}}_{\mathsf{PRF},\mathcal{B}_3}(\lambda)$ holds. □

Game_4: This game is the same as Game_3 except that $R_1 \xleftarrow{\$} \{0,1\}^\lambda$ that is originally computed by $F'_{K_{kw_b^*}}(R_0)$.

Lemma 6. *There exists an algorithm $\mathcal{B}_4$ where $|\Pr[E_3] - \Pr[E_4]| \leq \mathsf{Adv}^{\text{pseudo-random}}_{\mathsf{PRF}',\mathcal{B}_4}(\lambda)$ holds.*

Proof. Basically, it is the same as the proof of Lemma 5 and is omitted. □

Due to the replacement in Game_4, the SKE key becomes independent of the encrypted keyword, and consequently an SPCHS ciphertext becomes independent of the keyword. Thus, what we need to prove is that the linkage information provided by the hidden-structure chain remains hidden from the adversary.

Game_5: This game is the same as Game_4 except that C_{SKE} contained in $(C^*, \cdot) \leftarrow \mathsf{SPCHS.Enc}(\mathsf{pk}_R, \mathsf{sk}_S^{*(b)}, kw_b^*)$ is randomly chosen from CSpace.

Lemma 7. *There exists an algorithm $\mathcal{B}_5$ where $|\Pr[E_4] - \Pr[E_5]| \leq \mathsf{Adv}^{\text{IND-PCPA}}_{\mathsf{SKE},\mathcal{B}_5}(\lambda)$ holds.*

Proof Sketch. Let $\mathcal{C}$ be the challenger of SKE. $\mathcal{C}$ runs $k^* \leftarrow \mathsf{SKE.KG}(1^\lambda)$. $\mathcal{B}_5$ runs $\mathsf{params} \leftarrow \mathsf{NIKE.Setup}(1^\lambda)$ and $(\mathsf{NIKE.pk}_{\mathsf{Receiver}}, \mathsf{NIKE.sk}_{\mathsf{Receiver}}) \leftarrow \mathsf{NIKE.KeyGen}(\mathsf{params}, \mathsf{R})$. $\mathcal{B}_5$ initializes $\mathsf{KSet} := \emptyset$ and $\mathsf{CKSet} := \emptyset$ and sends $\mathsf{pk}_R = (\mathsf{params}, \mathsf{R}, \mathsf{NIKE.pk}_{\mathsf{Receiver}})$ to $\mathcal{A}$. $\mathcal{B}_5$ implicitly sets $k^* = k_{kw_b^*}$ (this is possible since $T_{kw_b^*} := K_{kw_b^*}||k_{kw_b^*} \xleftarrow{\$} \{0,1\}^{2\lambda}$ in this game). When $\mathcal{A}$ submits the challenge keyword kw_0^* or kw_1^* together with the challenge sender's public key to the SPCHS.Enc oracle, $\mathcal{B}_5$ retrieves (kw_0^*, R) or (kw_1^*, R) from HS_i, where i was fixed in Game_1. $\mathcal{B}_5$ then randomly chooses R' and queries the SKE encryption oracle to obtain a ciphertext C'_{SKE} computed by k^*. $\mathcal{B}_5$ stores (kw_0^*, R') or (kw_1^*, R') in HS_i and returns (R, C'_{SKE}) to $\mathcal{A}$. In the challenge phase, $\mathcal{B}_5$ randomly chooses $R^* \xleftarrow{\$} \mathsf{MSpace}$ and sends it to $\mathcal{C}$. $\mathcal{C}$ returns either $C_{\mathsf{SKE}} \leftarrow \mathsf{SKE.Enc}(k^*, R^*)$ or $C_{\mathsf{SKE}} \xleftarrow{\$} \mathsf{CSpace}$. $\mathcal{B}_5$ returns $C^* = (R, C_{\mathsf{SKE}})$ to $\mathcal{A}$ where R is stored in $(kw_b^*, R) \in \mathsf{HS}_i$ or R_1 that is randomly chosen (since $R_1 \xleftarrow{\$} \{0,1\}^\lambda$ in Game_4). For a post-challenge query on kw_0^* or kw_1^* together with the challenge sender's public key to the SPCHS.Enc oracle, $\mathcal{B}_5$ randomly chooses $R \xleftarrow{\$} \mathsf{MSpace}$ and sends it to $\mathcal{C}$ as an SKE encryption query that is sufficient to maintain the hidden-structure chain. If $\mathcal{C}$ returns $C_{\mathsf{SKE}} \leftarrow \mathsf{SKE.Enc}(k^*, R^*)$, then $\mathcal{B}_5$ simulates Game_4 and if $\mathcal{C}$ returns $C_{\mathsf{SKE}} \xleftarrow{\$} \mathsf{CSpace}$, then $\mathcal{B}_5$ simulates Game_5. Thus, $|\Pr[E_4] - \Pr[E_5]| \leq \mathsf{Adv}^{\text{IND-PCPA}}_{\mathsf{SKE},\mathcal{B}_5}(\lambda)$ holds. □

Game_6: This game is the same as Game_5 except that, for the challenge cipheretxt $C^* = (R, C_{\mathsf{SKE}})$, the SKE ciphertext of R is randomly chosen from CSpace.

Lemma 8. *There exists an algorithm $\mathcal{B}_6$ where $|\Pr[E_5] - \Pr[E_6]| \leq 2q_{\mathsf{enc}}\mathsf{Adv}^{\text{IND-PCPA}}_{\mathsf{SKE},\mathcal{B}_6}(\lambda)$ holds, where q_{enc} is the number of encryption queries.*

Table 2: Average Execution Time over 10,000 Runs using PBC and mcl libraries

Operation	**PBC (Type-1, a128.param)**	**mcl (Type-3, bls12_381.hpp)**
Scalar Multiplication over $\mathbb{G}_1$	3,696.99 µs	57.28 µs
Scalar Multiplication over $\mathbb{G}_2$	3,696.99 µs	95.45 µs
Exponentiation over $\mathbb{G}_T$	483.88 µs	163.95 µs
Multiplication over $\mathbb{G}_T$	1.97 µs	1.63 µs
Pairing	5,512.49 µs	480.82 µs

The proof is basically the same as that of Lemma 7, except that we need to guess the encryption query corresponding to the last ciphertext in the hidden-structure chain that links to the challenge ciphertext C^*. More precisely, for $C^* = (R, C_{\mathsf{SKE}})$, the last ciphertext of the hidden-structure chain for kw_b^* contains an SKE ciphertext of R linking the chain to C^*. The success probability of guessing is al least $1/2q_{\mathsf{enc}}$ where the factor 2 comes from the random choice of b. Note that $\mathcal{A}$ may obtain ciphertexts corresponding to kw_0^* and kw_1^* through encryption queries and thus learn the corresponding hidden-structure chains. However, in Game_6, the SKE ciphertext in the challenge ciphertext C^* and the SKE ciphertext in the last ciphertext of the hidden-structure chain that would otherwise link the chain to the challenge ciphertext are both replaced with random ciphertexts. Therefore, $\mathcal{A}$ cannot determine whether the challenge ciphertext belongs to the hidden-structure chain of kw_0^* or that of kw_1^*. Hence, $\mathcal{A}$ has no advantage in Game_6, and $|\Pr[E_6] - 1/2| = 0$.

Putting it all together, we have

$$
\begin{aligned}
&\mathsf{Adv}^{\mathsf{IND\text{-}CKSA}}_{\mathsf{SPCHS},\mathcal{A}}(\lambda) = |\Pr[E_0] - 1/2| \\
&= q_s|(\sum_{i=2}^{6} \Pr[E_{i-1}] - \Pr[E_i]) + \Pr[E_6] - 1/2| \\
&\leq q_s((\sum_{i=2}^{6} |\Pr[E_{i-1}] - \Pr[E_i]|) + |\Pr[E_6] - 1/2|) \\
&\leq q_s(\mathsf{Adv}^{\mathsf{HKR\text{-}CKS}}_{\mathsf{NIKE},\mathcal{B}_2}(\lambda) + \mathsf{Adv}^{\mathsf{pseudo\text{-}random}}_{\mathsf{PRF},\mathcal{B}_3}(\lambda) + \mathsf{Adv}^{\mathsf{pseudo\text{-}random}}_{\mathsf{PRF}',\mathcal{B}_4}(\lambda) + \mathsf{Adv}^{\mathsf{IND\text{-}PCPA}}_{\mathsf{SKE},\mathcal{B}_5}(\lambda) + 2q_{\mathsf{enc}}\mathsf{Adv}^{\mathsf{IND\text{-}PCPA}}_{\mathsf{SKE},\mathcal{B}_6}(\lambda))
\end{aligned}
$$

This concludes the proof of Theorem 2. □

6 Efficiency Evaluation

In this section, we compare the DH-based instantiation of the proposed generic construction and the (asymmetric pairing-based) Wang et al. SPCHS scheme [49] (given in Appendix A). All experiments were conducted on a system running Ubuntu 24.04.2 LTS, featuring an AMD Ryzen™ 5800X 8-core processor, 128.0 GiB of RAM, and 9.0 TB of disk storage. Asymmetric pairing (Type-3) was implemented using the mcl library[1] with the BLS12-381 curve whose parameter is specified by bls12_381.hpp.

6.1 Selection of the Comparison Scheme

We have mentioned that Wang et al. [49] employed asymmetric pairings to improve the efficiency in terms of implementation, since asymmetric pairings are suitable for providing an efficient imple-

[1]mcl: library for pairing-based cryptography (`https://github.com/herumi/mcl`)

Table 3: Number of Cryptographic Operations in Trapdoor, Encryption, and Search

Algorithm	The Wang et al. scheme [49]	Our Construction
Trapdoor	$\mathsf{mul}_{\mathbb{G}_1}$	SharedKey+PRF
SPCHS.Enc	$2n_{kw}(\mathsf{Pairing} + \mathsf{mul}_{\mathbb{G}_1} + \mathsf{mul}_{\mathbb{G}_2} + \mathsf{exp}_{\mathbb{G}_T}) + \mathsf{exp}_{\mathbb{G}_T}$	SharedKey+PRF+n_{kw}SKE.Enc
Search	n_{kw}Pairing	PRF+$(n_{kw} - 1)$SKE.Dec

mentation [27] due to the difference of embedded degree. Note that Zhang et al. [56] employed the form $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. They used the elliptic curve: $y^2 = x^3 + x$, which provides a symmetric pairing. Here, we present the computation times for both symmetric and asymmetric pairings in Table 2, and demonstrate that the Wang et al. scheme is a valid subject for comparison. For the implementation, symmetric pairing (Type-1) was performed using the PBC library[2]. Since PBC library does not provide parameters for sufficient level security, we employ parameters given in [20] (Type A, defined on a 3,072-bit prime field and the order is 256 bits). For completeness and self-containment, the parameters introduced in [22] are provided in the Appendix B. We describe basic operations, Scalar Multiplication over $\mathbb{G}_1$ (i.e., for given $g_1 \in \mathbb{G}_1$ and $x \in \mathbb{Z}_p$, compute g_1^x), Scalar Multiplication over $\mathbb{G}_2$ (i.e., for given $g_2 \in \mathbb{G}_2$ and $x \in \mathbb{Z}_p$, compute g_2^x), Exponentiation over $\mathbb{G}_T$ (i.e., for given $e(g_1, g_2)$ and $x \in \mathbb{Z}_p$, compute $e(g_1, g_2)^x$), Multiplication over $\mathbb{G}_T$ (i.e., for given $R_1, R_2 \in \mathbb{G}_T$, compute $R_1 \cdot R_2$), and Pairing (i.e., for given $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$, compute $e(g_1, g_2)$). Note that $\mathbb{G} = \mathbb{G}_1 = \mathbb{G}_2$ for symmetric pairings. Pairing in symmetric settings is over ten times slower than in asymmetric ones, and scalar multiplication is 40-60 times slower. Thus, we conclude that selecting the asymmetric pairing-based Wang et al. SPCHS scheme [49] as the target for comparison is reasonable.

6.2 Theoretical Evaluation

We evaluate the efficiency of the proposed construction in terms of the number of cryptographic operations in Table 3 For the Wang et al. SPCHS scheme [49], we count the number of cryptographic operations as described in their paper, without any modifications. Let n_C be the number of total ciphertexts to be searched, n_{kw} be the number of ciphertexts of kw, and L be the number of keywords to be currently encrypted (it is not the same as $|\mathcal{KS}|$), i.e., $n_C = L \times n_{kw}$. The SPCHS.Enc algorithm can be decomposed into two phases: (1) computing R from Head and (2) encrypting R. The most computationally intensive operation in our construction is the SharedKey algorithm, which computes a shared key k_{shared}. This operation is required only once during the SPCHS.Enc algorithm to derive a PRF key K_{kw} and a SKE key k_{kw}. In our construction, the Search algorithm employs PRF and SKE only. Especially, n_{kw} depends on the number of SKE.Dec. We note that the SharedKey operation in the Trapdoor algorithm is independent of the keyword, and thus these operations can be run offline, and when a trapdoor for a keyword is generated, then one PRF operation is sufficient.

Next, we evaluate the size of each value in Table 4. The sizes of the groups when the BLS12-381 curve is employed (where BLS stands for Barreto-Lynn-Scott [4]) are as follows: $|\mathbb{G}_1|$ is 381 bits, $|\mathbb{G}_2|$ is 762 bits, and $|\mathbb{G}_T|$ is $4,572$ bits, respectively. Then, the ciphertext size of the Wang et al. scheme is 5,334 bits and that of the DH-based instantiation is 384 bits ($\lambda = 128$ and AES128 (an 128-bit AES ciphertext together with a 128-bit nonce/IV) is employed as the underlying SKE). Especially, we can reduce the ciphertext size by a factor of approximately 10.

[2]PBC: Pairing-Based Cryptography (`https://crypto.stanford.edu/pbc/`)

Table 4: Size of Each Value

Value	The Wang et al. scheme [49]	Our Construction
Ciphertext	$\lvert\mathbb{G}_2\rvert + \lvert\mathbb{G}_T\rvert$ (5,334 bits)	$\lambda + \lvert C_{\mathsf{SKE}}\rvert$ (384 bits)
Trapdoor	$\lvert\mathbb{G}_1\rvert$ (381 bits)	2λ (256 bits)

Table 5: Comparison of Search Time

Scheme	**Find 100 Ciphertexts**	**Average Time to Find a Single Ciphertext**
Wang et al. [49]	$50,287.77$ µs	502.88 µs
DH-based Instantiation	179.65 µs	1.80 µs

$*$ Set the number of keywords is 1,000, the number of ciphertexts for the same keyword is 100, and the total number of ciphertexts is 100,000.

Table 6: Search Algorithm Execution Time Details

Initial Computation of R	**Hash Table Lookup**	**AES Decryption**	**Other**
7.13%	15.73%	44.53%	32.61%

6.3 Implementation

Next, we compare the DH-based instantiation and the Wang et al. SPCHS scheme [49]. We employ SHA256 as HMAC and AES128 (counter mode) in the instantiation. Precisely, for $T_{kw} := K_{kw}||k_{kw} = F_{k_{\mathsf{shared}}}(kw)$, we set $K_{kw}, k_{kw} \in \{0,1\}^{128}$. We set the number of keywords is 1,000, the number of ciphertexts for the same keyword is 100, and the total number of ciphertexts is 100,000. For both schemes, we employ hash tables[3] for seeking a ciphertext having $C[1] = R$ from 100,000 ciphertexts. We have mentioned that a trapdoor (search query) depends on a sender unlike to the original syntax of SPCHS and this sender-dependent trapdoor may affect a negative impact on search efficiency. Thus, first we give the comparison when we fix the sender in Table 5. For reference, Boneh et al.'s symmetric-pairing-based PEKS scheme [6] requires one pairing per search; searching 100,000 ciphertexts takes 551.249 seconds (about nine minutes), according to the benchmark in Table 2. By Wang et al.'s scheme, a roughly four-order-of-magnitude speedup had been achieved, highlighting the strong efficiency of hidden structures. As shown in the table, the DH-based instantiation further improves search efficiency by more than a factor of 250 over the scheme of Wang et al.

The breakdown of the search time (179.65 µs in Table 5) is shown in Table 6. **Other** consists of procedures that add C to CSet' and remove C from CSet. We emphasize that, although we used a simple hash table, any search algorithm could be employed to find $C[1] = R$, and this part is not our contribution. In fact, this step is not dominant because it is faster than AES decryption. Therefore, we chose not to adopt a more sophisticated search algorithm here.

Next, we explore the case that the search algorithm terminates when the sender is mismatch, i.e., the case that a sender specified in the encryption algorithm and a sender specified in the trapdoor generation algorithm are different (Figure 3). The first R computation ($R \leftarrow F'_{K_{kw}}(R_0)$ where $\mathsf{Head} = R_0$) requires 14.27 µs, the first lookup (i.e., a ciphertext having $C[1] = R$ is not found) requires 0.24 µs, and the procedure requires 14.71 µs in total. Let the number of senders be n_s. When a receiver would like to search a keyword kw, the receiver needs to prepare n_s trapdoors

[3]We used C++ std::unordered_map for hash-based indexing, reducing search complexity from $O(N)$ to $O(1)$.

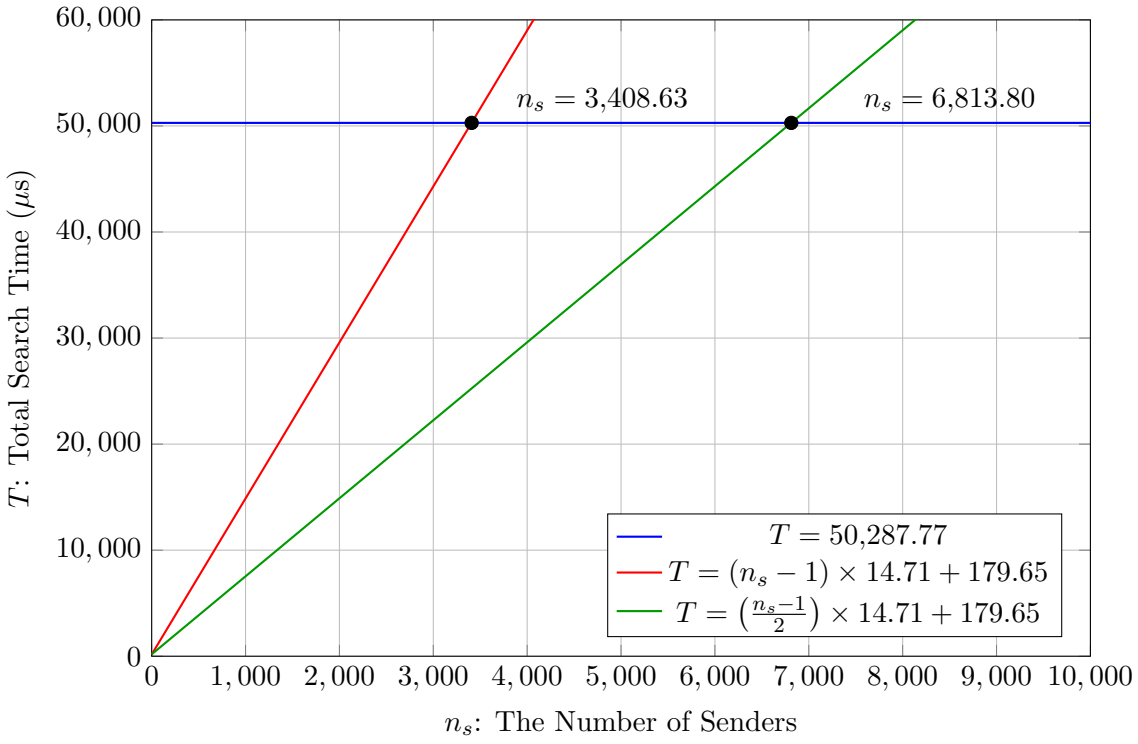


Figure 3: Comparison in the Sender Mismatch Case

and, in the worst case, the search algorithm fails to find a ciphertext with $n_s - 1$ trapdoors. That is, the total search time to find 100 ciphertexts is $(n_s - 1) \times 14.71 + 179.65$. In the average case, the search algorithm fails to find a ciphertext with $(n_s - 1)/2$ trapdoors. That is, the total search time to find 100 ciphertexts is $((n_s - 1)/2) \times 14.71 + 179.65$. When $n_s < 3408.63$ (resp. $n_s < 6813.80$), then $(n_s - 1) \times 14.71 + 179.65 < 50,287.77$ (resp. $((n_s - 1)/2) \times 14.71 + 179.65 < 50,287.77$). In the current setting, our DH-based instantiation outperforms the Wang et al. scheme in scenarios involving thousands of senders. Next, we argue that having a few thousand senders is sufficient in the scenario considered in the original PEKS paper [6] as follows (Section 1. Introduction in [6]):

> *Suppose user Alice wishes to read her email on a number of devices: laptop, desktop, pager, etc. Alice's mail gateway is supposed to route email to the appropriate device based on the keywords in the email. Now, suppose Bob sends encrypted email to Alice using Alice's public key. Both the contents of the email and the keywords are encrypted. In this case the mail gateway cannot see the keywords and hence cannot make routing decisions. As a result, the mobile people project is unable to process secure email without violating user privacy. Our goal is to enable Alice to give the gateway the ability to test whether "urgent" is a keyword in the email, but the gateway should learn nothing else about the email.*

In this scenario, where several thousand senders are assumed to send emails to one recipient, we conclude that the DH-based instantiation achieves faster search than the Wang et al. SPCHS scheme for personal use, provided that the number of senders is sufficiently large.

Finally, we discuss the number of senders for which the total search time (i.e., find 100 ciphertexts) becomes less than 1 millisecond in the DH-based instantiation (Figure 4). In the average

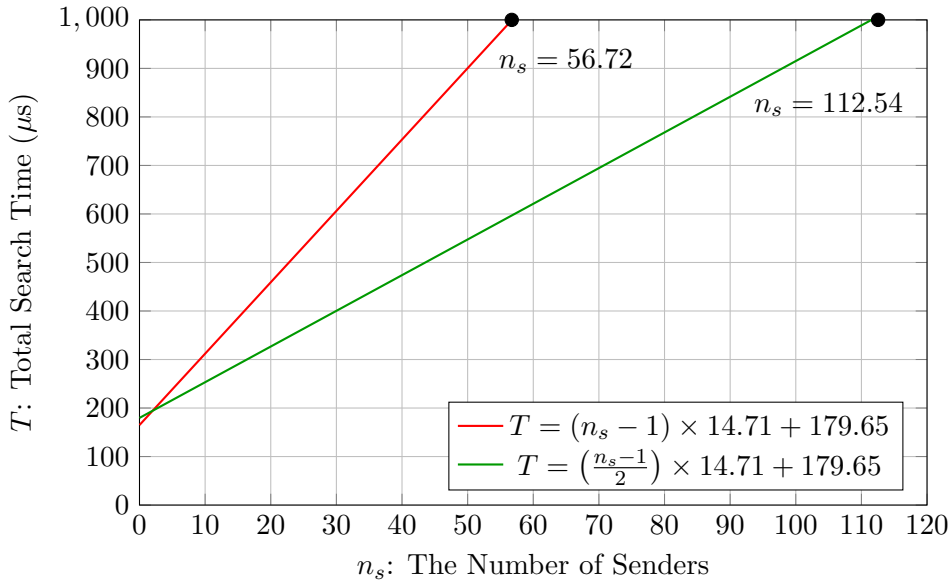


Figure 4: Total Search Time in terms of the Number of Senders

case, the total search time falls below 1 millisecond with approximately 112 senders, and even in the worst case, it remains under 1 millisecond with about 56 senders. Based on these results, we conclude that the proposed approach achieves practical search efficiency.

7 Conclusion

In this paper, we propose a generic construction of SPCHS from NIKE, SKE, and PRF. As a practical contribution, our evaluation demonstrates that the search time of the DH-based instantiation is over 250 times faster than that of the asymmetric pairing-based scheme by Wang et al. Moreover, we reduce the ciphertext size by a factor of approximately 10. As a theoretical contribution, the proposed generic construction can instantiate the first set of post-quantum SPCHS schemes based on lattices, isogenies, and codes, and a SPCHS scheme in the standard model.

As discussed, we highlight two main open problems: achieving forward privacy and enabling sender-independent trapdoors. For PAEKS, a generic compiler that adds forward privacy to any PAEKS scheme has been proposed [21]. Adapting the compiler for SPCHS remains an interesting research direction. PAEKS schemes with sender-independent search complexity have been proposed [15, 36, 37, 48, 53, 55] by introducing additional servers. A similar setting might be applicable to SPCHS. Further investigation is left for future work.

Acknowledgment: This work was supported by JSPS KAKENHI Grant Numbers JP26K02908, JP25H01106, and JP25K15119. This paper was edited for grammar using Copilot. Figure 3 and Figure 4 are generated using TikZ in LaTeX.

References

[1] Michel Abdalla, Mihir Bellare, Dario Catalano, Eike Kiltz, Tadayoshi Kohno, Tanja Lange, John Malone-Lee, Gregory Neven, Pascal Paillier, and Haixia Shi. Searchable encryption

revisited: Consistency properties, relation to anonymous ibe, and extensions. *Journal of Cryptology,*, 21(3):350–391, 2008.

[2] Michel Abdalla, Dario Catalano, and Dario Fiore. Verifiable random functions: Relations to identity-based key encapsulation and new constructions. *Journal of Cryptology*, 27(3):544–593, 2014.

[3] Erik Aronesty, David Cash, Yevgeniy Dodis, Daniel H. Gallancy, Christopher Higley, Harish Karthikeyan, and Oren Tysor. Encapsulated search index: Public-key, sub-linear, distributed, and delegatable. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *Public-Key Cryptography - PKC 2022 - 25th IACR International Conference on Practice and Theory of Public-Key Cryptography, Virtual Event, March 8-11, 2022, Proceedings, Part II*, volume 13178 of *Lecture Notes in Computer Science*, pages 256–285. Springer, 2022.

[4] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. Constructing elliptic curves with prescribed embedding degrees. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *Security in Communication Networks, Third International Conference, SCN 2002, Amalfi, Italy, September 11-13, 2002. Revised Papers*, volume 2576 of *Lecture Notes in Computer Science*, pages 257–267. Springer, 2002.

[5] Daniel J. Bernstein. Curve25519: New diffie-hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography, New York, NY, USA, April 24-26, 2006, Proceedings*, volume 3958 of *Lecture Notes in Computer Science*, pages 207–228. Springer, 2006.

[6] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522. Springer, 2004.

[7] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer, 2001.

[8] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic prfs and their applications. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 410–428. Springer, 2013.

[9] Dan Boneh, Periklis A. Papakonstantinou, Charles Rackoff, Yevgeniy Vahlis, and Brent Waters. On the impossibility of basing identity based encryption on trapdoor permutations. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, Philadelphia, PA, USA, October 25-28, 2008*, pages 283–292. IEEE Computer Society, 2008.

[10] Raphael Bost. $\sum o\varphi o\varsigma$: Forward secure searchable encryption. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1143–1154. ACM, 2016.

[11] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *Journal of the ACM*, 51(4):557–594, 2004.

[12] David Cash, Eike Kiltz, and Victor Shoup. The twin Diffie-Hellman problem and applications. In Nigel P. Smart, editor, *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, volume 4965 of *Lecture Notes in Computer Science*, pages 127–145. Springer, 2008.

[13] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: an efficient post-quantum commutative group action. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part III*, volume 11274 of *Lecture Notes in Computer Science*, pages 395–427. Springer, 2018.

[14] Biwen Chen, Libing Wu, Neeraj Kumar, Kim-Kwang Raymond Choo, and Debiao He. Lightweight searchable public-key encryption with forward privacy over IIoT outsourced data. *IEEE Transactions on Emerging Topics in Computing*, 9(4):1753–1764, 2021.

[15] Leixiao Cheng and Fei Meng. Server-aided public key authenticated searchable encryption with constant ciphertext and constant trapdoor. *IEEE Transactions on Information Forensics and Security*, 19:1388–1400, 2024.

[16] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, October 30 - November 3, 2006*, pages 79–88. ACM, 2006.

[17] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[18] Nico Döttling and Sanjam Garg. From selective IBE to full IBE and selective HIBE. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 372–408. Springer, 2017.

[19] Nico Döttling and Sanjam Garg. Identity-based encryption from the Diffie-Hellman assumption. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 537–569. Springer, 2017.

[20] Shuntaro Ema, Yuta Sato, Hinata Nishino, Keita Emura, and Toshihiro Ohigashi. Implementation and evaluation of an identity-based encryption with security against the KGC. *Journal of Information Processing*, 33:185–196, 2025.

[21] Keita Emura. Generic construction of forward secure public key authenticated encryption with keyword search. In Christina Pöpper and Lejla Batina, editors, *Applied Cryptography and Network Security - 22nd International Conference, ACNS 2024, Abu Dhabi, United Arab*

Emirates, March 5-8, 2024, Proceedings, Part I, volume 14583 of *Lecture Notes in Computer Science*, pages 237–256. Springer, 2024.

[22] Keita Emura, Katsuhiko Ito, and Toshihiro Ohigashi. Secure-channel free searchable encryption with multiple keywords: A generic construction, an instantiation, and its implementation. *Journal of Computer and System Sciences*, 114:107–125, 2020.

[23] Martín Farach-Colton, Andrew Krapivin, and William Kuszmaul. Optimal bounds for open addressing without reordering. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*, pages 594–605. IEEE, 2024.

[24] Eduarda S. V. Freire, Dennis Hofheinz, Eike Kiltz, and Kenneth G. Paterson. Non-interactive key exchange. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *Public-Key Cryptography - PKC 2013 - 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26 - March 1, 2013. Proceedings*, volume 7778 of *Lecture Notes in Computer Science*, pages 254–271. Springer, 2013.

[25] Eduarda S. V. Freire, Dennis Hofheinz, Kenneth G. Paterson, and Christoph Striecks. Programmable hash functions in the multilinear setting. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 513–530. Springer, 2013.

[26] Phillip Gajland, Bor de Kock, Miguel Quaresma, Giulio Malavolta, and Peter Schwabe. SWOOSH: efficient lattice-based non-interactive key exchange. In Davide Balzarotti and Wenyuan Xu, editors, *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*. USENIX Association, 2024.

[27] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

[28] Rishab Goyal, Venkata Koppula, and Mahesh Sreekumar Rajasree. A note on adaptive security in hierarchical identity-based encryption. In Yael Tauman Kalai and Seny F. Kamara, editors, *Advances in Cryptology - CRYPTO 2025 - 45th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2025, Proceedings, Part III*, volume 16002 of *Lecture Notes in Computer Science*, pages 101–136. Springer, 2025.

[29] Changhee Hahn. Towards secure and efficient wildcard search for cloud storage. *IEEE Transactions on Dependable and Secure Computing*, pages 1–14, 2025. Early Access.

[30] Min Han, Peng Xu, Willy Susilo, and Wei Wang. Dynamic searchable public-key encryption and its application. *Frontiers of Computer Science*, 20, 2026.

[31] Junichiro Hayata, Masahito Ishizaka, Yusuke Sakai, Goichiro Hanaoka, and Kanta Matsuura. Generic construction of adaptively secure anonymous key-policy attribute-based encryption from public-key searchable encryption. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 103-A(1):107–113, 2020.

[32] Javier Herranz. Attribute-based encryption implies identity-based encryption. *IET Information Security*, 11(6):332–337, 2017.

[33] Qiong Huang and Hongbo Li. An efficient public-key searchable encryption scheme secure against inside keyword guessing attacks. *Information Sciences*, 403:1–14, 2017.

[34] Russell W. F. Lai and Sherman S. M. Chow. Forward-secure searchable encryption on labeled bipartite graphs. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *Applied Cryptography and Network Security - 15th International Conference, ACNS 2017, Kanazawa, Japan, July 10-12, 2017, Proceedings*, volume 10355 of *Lecture Notes in Computer Science*, pages 478–497. Springer, 2017.

[35] Tung Le, Rouzbeh Behnia, Jorge Guajardo, and Thang Hoang. MUSES: efficient multi-user searchable encrypted database. In Davide Balzarotti and Wenyuan Xu, editors, *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*. USENIX Association, 2024.

[36] Hongbo Li, Qiong Huang, Jianye Huang, and Willy Susilo. Public-key authenticated encryption with keyword search supporting constant trapdoor generation and fast search. *IEEE Transactions on Information Forensics and Security*, 18:396–410, 2023.

[37] Hongbo Li, Willy Susilo, Jian Shen, Chen Wang, Leixiao Cheng, and Qiong Huang. Proxy-free public-key authenticated updatable and searchable encryption for cloud storage. *IEEE Transactions on Information Forensics and Security*, 23(1):236–249, 2026.

[38] Qinyi Li and Xavier Boyen. Public-key authenticated encryption with keyword search made easy. *IACR Communications in Cryptology*, 1(2):16, 2024.

[39] Qinyi Li and Xavier Boyen. Predicate-private asymmetric searchable encryption for conjunctions from lattices. In Vincent Nicomette, Abdelmalek Benzekri, Nora Boulahia-Cuppens, and Jaideep Vaidya, editors, *Computer Security - ESORICS 2025 - 30th European Symposium on Research in Computer Security, Toulouse, France, September 22-24, 2025, Proceedings, Part II*, volume 16054 of *Lecture Notes in Computer Science*, pages 262–281. Springer, 2025.

[40] Qin Liu, Yu Peng, Shuyu Pei, Jie Wu, Tao Peng, and Guojun Wang. Prime inner product encoding for effective wildcard-based multi-keyword fuzzy search. *IEEE Transactions on Services Computing*, 15(4):1799–1812, 2022.

[41] Fucai Luo, Xingfu Yan, Haining Yang, and Xiaofan Zheng. PAEWS: public-key authenticated encryption with wildcard search over outsourced encrypted data. *IEEE Transactions on Information Forensics and Security*, 20:2212–2223, 2025.

[42] Long Meng, Liqun Chen, Yangguang Tian, Mark Manulis, and Suhui Liu. FEASE: fast and expressive asymmetric searchable encryption. In Davide Balzarotti and Wenyuan Xu, editors, *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*. USENIX Association, 2024.

[43] Junling Pei and Fang-Wei Fu. MODRED: A code-based non-interactive key exchange protocol. *Theoretical Computer Science*, 1021:114943, 2024.

[44] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. Practical dynamic searchable encryption with small leakage. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*. The Internet Society, 2014.

[45] Jiafan Wang and Sherman S. M. Chow. Omnes pro uno: Practical multi-writer encrypted database. In Kevin R. B. Butler and Kurt Thomas, editors, *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, pages 2371–2388. USENIX Association, 2022.

[46] Jiafan Wang and Sherman S. M. Chow. Unus pro omnibus: Multi-client searchable encryption via access control. In *31st Annual Network and Distributed System Security Symposium, NDSS 2024, San Diego, California, USA, February 26 - March 1, 2024*. The Internet Society, 2024.

[47] Qing Wang, Donghui Hu, Meng Li, and Guomin Yang. Secure and flexible wildcard queries. *IEEE Transactions on Information Forensics and Security*, 19:7374–7388, 2024.

[48] Wei Wang, Dongli Liu, Zilin Zheng, Peng Xu, and Laurence Tianruo Yang. Identity-based group encryption with keyword search against keyword guessing attack. *IEEE Transactions on Information Forensics and Security*, 19:8023–8036, 2024.

[49] Wei Wang, Peng Xu, Dongli Liu, Laurence Tianruo Yang, and Zheng Yan. Lightweighted secure searching over public-key ciphertexts for edge-cloud-assisted industrial iot devices. *IEEE Transactions on Industrial Informatics*, 16(6):4221–4230, 2020.

[50] Peiming Xu, Shaohua Tang, Peng Xu, Qianhong Wu, Honggang Hu, and Willy Susilo. Practical multi-keyword and boolean search over encrypted e-mail in cloud server. *IEEE Transactions on Services Computing*, 14(6):1877–1889, 2021.

[51] Peng Xu, Shuanghong He, Wei Wang, Willy Susilo, and Hai Jin. Lightweight searchable public-key encryption for cloud-assisted wireless sensor networks. *IEEE Transactions on Industrial Informatics*, 14(8):3712–3723, 2018.

[52] Peng Xu, Qianhong Wu, Wei Wang, Willy Susilo, Josep Domingo-Ferrer, and Hai Jin. Generating searchable public-key ciphertexts with hidden structures for fast keyword search. *IEEE Transactions on Information Forensics and Security*, 10(9):1993–2006, 2015.

[53] Yongliang Xu, Hang Cheng, Jiguo Li, Ximeng Liu, Xinpeng Zhang, and Meiqing Wang. Lightweight multi-user public-key authenticated encryption with keyword search. *IEEE Transactions on Information Forensics and Security*, 20:3234–3246, 2025.

[54] Ming Zeng, Haifeng Qian, Jie Chen, and Kai Zhang. Forward secure public key encryption with keyword search for outsourced cloud storage. *IEEE Transactions on Cloud Computing*, 10(1):426–438, 2022.

[55] Shengjie Zhang, Jianchang Lai, Liquan Chen, Jinguang Han, and Ge Wu. Keyword-field-free conjunctive searchable encryption for multiuser in EMR system. *IEEE Internet of Things Journal*, 12(15):30850–30861, 2025.

[56] Shiwen Zhang, Yibin Yang, Jiayi He, Wei Liang, Kuanching Li, and Rubén González Crespo. FKSS: A fast keyword search scheme with access control for cloud-assisted edge computing. *International Journal of Information Security*, 25(1):21, 2026.

Appendix

A Wang et al. SPCHS scheme

In this Appendix, we introduce the Wang et al. SPCHS scheme [49] since this is the only scheme that employed asymmetric pairings. Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ be groups with prime order p, $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ be generators, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a pairing. Although they employed an edge-cloud architecture, where a pairing computation ($e(H(kw)^r, \mathsf{Head})$ below) is delegated to the edge

platform in the SPCHS.Enc algorithm, we omit this procedure to fit our syntax. Note that we omit $\mathsf{pk_S}$ from the input of the Trapdoor algorithm.

SPCHS.Setup(1^λ): Set $\mathsf{params} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, p, e)$. Let $H : \mathcal{KS} \to \mathbb{G}_1$ be a hash function which is modeled as a random oracle. Choose $s \xleftarrow{\$} \mathbb{Z}_p$ and compute $\mathsf{Head} = g_2^s$. Output $(\mathsf{pk_R}, \mathsf{sk_R}) = ((\mathsf{params}, H, \mathsf{Head}), (H, s))$.

Structure$(\mathsf{pk_R})$: Parse $\mathsf{pk_R} = (\mathsf{params}, H, \mathsf{Head})$. Choose $u, r_1 \xleftarrow{\$} \mathbb{Z}_p$ and compute g_2^u and g_2^t. Output $(\mathsf{pk_S}, \mathsf{sk_S}) = (g_2^u, (u, t, g_2^t, \mathsf{HS}))$ where HS is a list to store hidden structures (with the form $\{(kw, R) \in \mathcal{KS} \times \mathbb{G}_T\}$) and is initialized as $\mathsf{HS} := \emptyset$.

SPCHS.Enc$(\mathsf{pk_R}, kw)$: Parse $\mathsf{pk_R} = (\mathsf{params}, H, \mathsf{Head})$ and $\mathsf{sk_S} = (u, t, g_2^t, \mathsf{HS})$. Randomly choose $r \xleftarrow{\$} \mathbb{Z}_p$. Search $(kw, \cdot) \in \mathsf{HS}$.

- If there is no such the entry, compute $R_1 \leftarrow e(H(kw)^r, \mathsf{Head})^t$, update $\mathsf{HS} := \mathsf{HS} \cup \{(kw, R_1)\}$, and output $C = (e(H(kw)^r, \mathsf{Head})^{u/r}, (g_2^t)^r)$. Here, $e(H(kw)^r, \mathsf{Head})^{u/r} = e(H(kw)^s, g_2^u) = e(T_{kw}, \mathsf{pk_S})$ where T_{kw} is defined in the Trapdoor algorithm.
- Let $(kw, R) \in \mathsf{HS}$. Compute $H(kw)^r$ and $R' \leftarrow e(H(kw)^r, \mathsf{Head})^t$. Update $\mathsf{HS} := \mathsf{HS} \setminus \{(kw, R)\}$ and $\mathsf{HS} := \mathsf{HS} \cup \{(kw, R')\}$. Output $C = (R', (g_2^t)^r)$.

Trapdoor$(\mathsf{sk_R}, kw)$: Parse $\mathsf{sk_R} = (H, s)$. Output $T_{kw} = H(kw)^s$.

Search$(\mathsf{pk_R}, \mathsf{pk_S}, \mathsf{CSet}, T_{kw})$: Parse $\mathsf{pk_R} = (\mathsf{params}, H, \mathsf{Head})$, $\mathsf{pk_S} = g_2^u$, and $T_{kw} = H(kw)^s$. Let $C \in \mathsf{CSet}$ be a ciphertext and the ciphertext can be parsed as $(C[1], C[2]) \in \mathbb{G}_T \times \mathbb{G}_2$. Initialize $\mathsf{CSet}' := \emptyset$.

1. Compute $R \leftarrow e(T_{kw}, \mathsf{pk_S}) = e(H(kw)^s, g_2^u)$.
2. From CSet, seek a ciphertext having $C[1] = R$.
   - If it exists, add C to CSet' and remove C from CSet.
   - If no matching ciphertext is found, output CSet'.
3. Compute $R \leftarrow e(T_{kw}, C[2])$ and go to Step 2.

B PBC Parameters

We describe the parameters employed for presenting Table 2 in Table 7.

Table 7: Parameters for PBC Library (a128.param) [20]

Parameter	**Value**
Type	a
q	13782918213784191466093920316656277848107247286879921288373 60333737763894232758566008499657275579051453797871470115739 18838400696256791520969790954647234026134149836279179970069 91294170207718584689222874164514703754613783495801699344903 23687711177168008542310452451285148291313010481717176147391 96745940412209360282518205988243325127502858859823618043686 33686495627185042599777321960125642008227110912694341384713 26934527747330048566104052231617611044807535038087
r	57896044618658097711785492504343953926634992332820282019728 792006155588075521
h	23806320975064304888602247447209421656076606270975876064915 01669490467523842458294233853674422676606549634590188265566 42656137089040285666790582182002598333807307620189224986606 09790082315613645318317104917054336577361982953438656528379 18061641455996690236681218757201594259713810430291958752367 68247182750347222542569228103402257034633722433381878356381 95544071772040401323947245260352 8
exp1	41
exp2	255
sign0	1
sign1	1