RESEARCH ARTICLE

PINDEX: Private multi-linked index for encrypted document retrieval

A. John Prakash¹, B. Lydia Elizabeth^{2*}¹ Ramanujan Computing Centre, Anna University, Chennai, TN, India, ² Department of IT, Anna University, Chennai, TN, India* lydiajohn@annauniv.edu

Abstract

Cryptographic cloud storage is used to make optimal use of the cloud storage infrastructure to outsource sensitive and mission-critical data. The continuous growth of encrypted data outsourced to cloud storage requires continuous updating. Attacks like file-injection are reported to compromise confidentiality of the user as a consequence of information leakage during update. It is required that dynamic schemes provide forward privacy guarantees. Updates should not leak information to the untrusted server regarding the previously issued queries. Therefore, the challenge is to design an efficient searchable encryption scheme with dynamic updates and forward privacy guarantees. In this paper, a novel private multi-linked dynamic index for encrypted document retrieval namely PINDEX is proposed. The multi-linked dynamic index is constructed using probabilistic homomorphic encryption mechanism and secret orthogonal vectors. Full security proofs for correctness and forward privacy in the random oracle model is provided. Experiments on real world Enron dataset demonstrates that our construction is practical and efficient. The security and performance analysis of PINDEX shows that the dynamic multi-linked index guarantees forward privacy without significant loss of efficiency.

OPEN ACCESS

Citation: Prakash AJ, Elizabeth BL (2021) PINDEX: Private multi-linked index for encrypted document retrieval. PLoS ONE 16(8): e0256223. <https://doi.org/10.1371/journal.pone.0256223>

Editor: Anandakumar Haldorai, Sri Eshwar College of Engineering, INDIA

Received: March 11, 2021

Accepted: August 2, 2021

Published: August 20, 2021

Copyright: © 2021 Prakash, Elizabeth. This is an open access article distributed under the terms of the [Creative Commons Attribution License](https://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Data Availability Statement: All relevant data are within the paper.

Funding: The author(s) received no specific funding for this work.

Competing interests: The authors have declared that no competing interests exist.

1 Introduction

Cloud computing has revolutionized data storage by offering effortless data storage for personal, enterprises, governments and institutions. Data owners have flexible, on-demand access to data with simplified data management and significant cost benefits with increased performance. Data owners outsource sensitive data such as the Electronic Health Records (EHR), citizens' personal information, emails, credit cards data, government information and critical business data to the public cloud [1]. Despite the numerous benefits of cloud storage fuelled by high speed networking technologies and although the cloud storage providers (CSPs) claim to adopt strong security measures governments, organizations and businesses are slow to fully embrace the public cloud storage due to privacy and security concerns. When data owners outsource sensitive data to the cloud; they lose control over their data leading to a number of security issues [2].

Data owners encrypt their sensitive information to protect privacy of data stored with the honest but curious cloud storage and to defend against unauthorized access. Encryption imposes restriction on searching which is the only way to access the data. Unless they can easily be indexed, shared, retrieved and utilized, encrypted data serves no purpose. Searchable encryption enables a user to perform efficient keyword search while preserving privacy. Indexes are used to improve the search performance by significantly reducing the search complexity. A number of searchable encryption schemes [3–6] have been proposed but most of the schemes provide a static mechanism which hinders its application in reality.

Dynamic searchable encryption allows clients to dynamically update files or keywords without rebuilding the encrypted index. For any searchable encryption scheme to be practical the scheme must be secure and should allow efficient updates with optimal search time. A few reported works [7–11] support dynamic index. Nevertheless, most of the solutions leaks critical information since the adversary can observe and learn more information from the interaction between data user and CSP. This leakage can be from search pattern, access pattern, update pattern, size pattern, file identifiers containing a specific keyword, trace and trapdoor linkability. As a result of the leakages, forward privacy was introduced by Stefanov et al. [12]. Zhang et al. [13] presented a file injection attack on dynamic constructions by adding files to the database. There is a trade-off between security and practicality. In addition to the data privacy, it is imperative that the computational correctness should be guaranteed by design. This is because the malicious server can return incorrect results due to hardware or software malfunction or to save resources. In order to fully utilize the services offered by the cloud server, there is a need for a provably secure dynamic searchable encryption scheme with efficient search, forward privacy and support for parallelism, which is a challenging problem.

From the above discussions, the significant research problems in designing protocols for privacy-preserving search over encrypted outsourced cloud data can be summarized as: a) an efficient and secure index construction to improve search without reconstructing the index. b) a privacy preserving search over encrypted data with allowable minimum leakage of information to the CSP. c) providing support for efficient dynamic updates to encrypted index with allowable minimum leakage and without requirement of trusted platform at the CSP. d) Support parallel execution of multi-keyword search and update operations. e) a construction with minimum communicational and computational overhead. Motivated by the challenges summarized above the main contributions and results of the proposed method, namely, P_{INDEX} are:

1. *Secure Dynamic Index*: A novel private multi-linked dynamic index construction using probabilistic homomorphic encryption and a secret orthogonal vector as building blocks. Support to add/delete keywords or documents without reconstructing the outsourced encrypted index using the hash table that contains the sum of inner product of the rows that are linked by orthogonal vectors.
2. *Forward Privacy*: The key used in construction of the index and trapdoor or token generation are indistinguishable from random functions. This is because P_{INDEX} uses probabilistic homomorphic encryption and secret orthogonal vectors. Thus, the server cannot learn information from search or access patterns from previous queries.
3. *Efficient and Parallelizable*: The server performs $\mathcal{O}(|F_q|)$ and $\mathcal{O}(|W_f|)$ functions evaluations for each search and update operation where $|F_q|$ is the number of documents that contain the query keyword w_q and $|W_f|$ is the number of keywords contained in a file f to be updated. The client and server storage overhead is reduced to $\mathcal{O}(n)$. The multi-linked index

structure allows multi-keywords search and update operations to run independently over p processors.

4. The results obtained from security analysis shows that the proposed design is secure in the real-ideal security model, efficient with sub-linear time complexity $\mathcal{O}(|F_q|)$, efficiently parallelized and exhibits relatively better performance in empirical analysis.

The rest of the paper is organized in the order of research methodology adopted which is similar to the standard works [5, 8, 9, 12, 14, 15]. First, we define the security requirements of the protocol under design in the form of definitions such that it captures thoroughly the adversarial environment and adversary. Such definitions stated as a security model for the protocol to serve as the basis for further design is described in section 2. Second, we describe in section 3 the design of the cryptographic protocol adhering to the security model devised in section 2. Third, we present in section 4, a thorough theoretical correctness and security analysis using standard provable security attack or adversary model to investigate vulnerabilities in the design. Fourth in section 5, we support the theoretical correctness and efficiency with results obtained from empirical simulations. Lastly, we present in section 6 the related works, followed by discussion in section 7 and conclusion in section 8.

2 Model

This section describes the preliminaries and security model which captures the requirements of the protocol under design. The model captures the notions of security and adversary through precise definitions to be used later as the basis of design.

2.1 Notations and system model

Let $D = \{d_1, \dots, d_n\}$ be the set of document collection, $C = \{C_1, \dots, C_n\}$ be the encrypted collection of documents and $W = \{w_1, \dots, w_m\}$ be the set of keywords. Let $F = \{f_1, \dots, f_n\}$ denote the file identifiers of documents given by $f_i = id(C_i)$. A collection of document D is to be outsourced to the Cloud Service Provider (CSP). The data owner encrypts the document set D to obtain an encrypted document collection C . To enable privacy preserving searching over the encrypted document collection C , a secure encrypted index denoted by γ is built using BUILDINDEX. The encrypted document collection C , the secure encrypted Index γ are outsourced to the CSP. An authorised Data User (DU) acquires a search token denoted by τ_q generated using SRCHTOKEN corresponding to a given query keyword w_q from the Data Owner (DO). The search token τ_q is then sent to the CSP using a standard communication protocol. The CSP searches for the matching documents for the query keyword using SEARCH over the encrypted index γ preserving privacy. The SEARCH algorithm outputs a set of file identifiers relevant to the query denoted by F_q that contain the query keyword. The DO updates the index using UPDATE whenever the document changes or files need to be added/deleted. The overall system working mechanism of the search is shown in Fig 1. We use the notation $=$ to denote a deterministic assignment, $\leftarrow$ a probabilistically computed assignment and $\leftarrow_\$$ a uniformly sampled assignment.

2.2 Security definitions

The construction of the PINDEX scheme aims to securely perform computations at the CSP on the encrypted data while preserving privacy and is defined as in [9]:

Definition 1 (PINDEX). PINDEX is a dynamic searchable encryption scheme consisting of a tuple $\{\text{KGen}, \text{Enc}_1, \text{Dec}_1, \text{Enc}_2, \text{Dec}_2, \text{BUILDINDEX}, \text{SRCHTOKEN}, \text{SEARCH}, \text{UPDTOKEN}, \text{UPDATE}\}$ of ten algorithms such that:

Fig 1. System model.

<https://doi.org/10.1371/journal.pone.0256223.g001>

1. $K \leftarrow \text{KGen}(1^\lambda)$ is a probabilistic polynomial time algorithm that outputs a secret key $K \leftarrow \{pk, sk, k\}$ when given as input a security parameter 1^λ .
2. $c \leftarrow \text{Enc}_1(k, f)$ is a CPA-secure deterministic polynomial time algorithm that outputs a sequence of encrypted ciphertexts c when given as input a secret key k and a file f .
3. $f = \text{Dec}_1(k, c)$ is a deterministic polynomial time algorithm that outputs a file f when given as input a secret key k and a ciphertext c .
4. $c \leftarrow \text{Enc}_2(pk, m, r)$ is a CPA-secure probabilistic polynomial time additive homomorphic algorithm that outputs an encrypted ciphertext c when given as input a secret key pk , a message m and randomness r .
5. $m = \text{Dec}_2(sk, c)$ is a deterministic polynomial time algorithm that outputs m when given as input a secret key sk and a ciphertext c .
6. $(\gamma, \Gamma) \leftarrow \text{BUILDINDEX}(K, \delta, V, D, \text{Enc}_2)$ is a probabilistic polynomial time algorithm that outputs a encrypted index γ and a parameter hash table Γ when given as input a secret K , vector V , primitive Enc_2 , document collection D and a index δ .
7. $\tau_q \leftarrow \text{SRCHTOKEN}(pk, w_q, \Gamma)$ is a probabilistic polynomial time algorithm that outputs a search token τ_q when given as input a secret key pk , query keyword w_q and a parameter hash table Γ .
8. $F_q \leftarrow \text{SEARCH}(\gamma, \tau_q)$ is a deterministic polynomial time algorithm that outputs a set of identifiers $F_q \subseteq D$ when given as input an encrypted index γ and a search token τ_q .
9. $\tau_u \leftarrow \text{UPDTOKEN}(pk, f_i, u, w_u)$ is a probabilistic polynomial time algorithm. Given a secret key K , update keyword w_u and a file f_i as input, the algorithm outputs an update token τ_u for the update type $u \in \{\text{add}, \text{delete}\}$.

10. $\gamma' \leftarrow \text{UPDATE}(\gamma, \tau_u)$ is a deterministic polynomial time algorithm that outputs an updated encrypted index γ' given an encrypted index γ and an update token τ_u .

A scheme as defined above is secure when it is designed based on the real/ideal security paradigm. The paradigm requires defining correctness and privacy. The correctness captures the notion that the specified polynomial time algorithms in Definition 1 outputs correctly and it is given as:

Definition 2 (Correctness). Let π be a dynamic searchable encryption scheme as given in Definition 1. Such π is said to be correct if $\forall k \in \mathbb{N}, \forall K: K \leftarrow \text{KGen}(1^\lambda), \forall (\delta, f), \forall \gamma: \gamma \leftarrow \text{BUILDINDEX}(K, \delta, V, D, \text{Enc}_2), \forall c: c \leftarrow \text{Enc}_1(k, f)$ and for all update operations of $\text{UPDATE}(\gamma, \tau_u)$, where τ_u is the update token obtained by $\forall (f, \tau_u): \tau_u \leftarrow \text{UPDTOKEN}(\text{pk}, f_b, w)$ and all $u \in \{\text{add}, \text{delete}\}, \forall (w, \tau_q: \tau_q \leftarrow \text{SRCHTOKEN}(\text{pk}, w, \Gamma), \forall F_q: F_q \leftarrow \text{SEARCH}(\gamma, \tau_q)$, the plaintext $f_w = \{\text{Dec}_1(k, c_i): i \in F_q\}$ are all plaintexts in f containing keyword w .

Most of the dynamic searchable encryption schemes leak information which could be exploited to launch attacks such as file injection attacks to recover the encryption key. For example, information leak such as file identifiers returned for a respective search or update query cannot be prevented. The leakage function captures the notion of allowable leakages in a dynamic searchable encryption defined as in [9]:

Definition 3 (Leakage). The leakage functions are defined as follows:

1. Search Pattern $\mathcal{P}(\delta, q, t)$: It is defined by a one dimensional binary matrix of length $m \times n$. Given a search keyword w at t , if the search at time $i \leq t$ is for the keyword w then it is marked as 1 at i and 0 otherwise.
2. Access Pattern $\Delta(\delta, f, w, t)$: It is defined as the set of identifiers f_w at time t given a search query for keyword w at time t .
3. $\mathcal{L}_1(\delta, f)$: The function outputs the number of keywords m , the number of files n , the identifiers i of the file and the size of each file when given the index δ and set of file identifiers.
4. $\mathcal{L}_2(\delta, f, w, t)$: The function outputs $\mathcal{P}(\delta, q, t)$ and $\Delta(\delta, f, w, t)$ for a search query at time t , given the index δ , set of files f and keyword w as input.

The definition of privacy or security captures the notion that the view of each party can be separately simulated given the leakage function and it is defined as:

Definition 4 (Security). Let π be a dynamic searchable encryption scheme as defined in Definition 1. Let $\mathcal{A}$ be a stateful adversary, Sim be a stateful simulator and $(\mathcal{L}_1, \mathcal{L}_2)$ are stateful leakage algorithms in the following experiments:

1. $\text{REAL}_{\mathcal{A}}(\lambda)$: The challenger $\mathcal{C}$ runs $K \leftarrow \text{KGen}$. The adversary outputs $(\delta, f) \leftarrow \mathcal{A}$ and receives $(\gamma, \Gamma) \leftarrow \text{BUILDINDEX}(K, \delta, V, D, \text{Enc}_1)$ from $\mathcal{C}$. Each time the adversary computes $q \in \{w, f_i\}$ and makes polynomial number of adaptive queries. $\mathcal{A}$ obtains from $\mathcal{C}$ a search token $\tau_q \leftarrow \text{SRCHTOKEN}(\text{pk}, w, \Gamma)$ whenever it wants to perform a search query $q = w$. If $\mathcal{A}$ wants to make a update query $q = f_i$ it receives a update token $\tau_u \leftarrow \text{UPDTOKEN}(\text{pk}, f_i, w_u)$ from $\mathcal{C}$. $\mathcal{A}$ returns an output bit b at the end of the experiment.
2. $\text{IDEAL}_{\mathcal{A}, S}(\lambda)$: Sim outputs (γ, c) to $\mathcal{A}$ when given (δ, f) and $\mathcal{L}_1(\delta, f)$. $\mathcal{A}$ then can make polynomial number of adaptive queries $q \in \{w, f_i\}$. $\mathcal{A}$ makes a search query $q = w$ then the simulator is given $\mathcal{L}_2(\delta, f, w, t)$ and when q is update query, the Sim is given a new $\mathcal{L}_2(\delta, f, w, t)$ including the adaptive query history by $\mathcal{A}$. Also, the Sim returns a token τ to $\mathcal{A}$ appropriately. $\mathcal{A}$ then returns an output bit b at the end of the experiment.

The scheme π is said to be $(\mathcal{L}_1, \mathcal{L}_2)$ -secure against adaptive dynamic chosen-keyword attacks if for all PPT adversaries $\mathcal{A}$, there exists a PPT simulator Sim such that

$$\Pr[\text{REAL}_{\mathcal{A}}(k) = 1] - \Pr[\text{IDEAL}_{\mathcal{A}, \text{Sim}}(k) = 1] \leq \text{negl}(\lambda) \quad (1)$$

A dynamic searchable scheme is considered secure if no information is revealed during execution of all its operations. However, designing such schemes which do not leak any information is a challenge. For example, when a search or update is performed for the keyword w , the CSP can correlate the file identifiers with its query and learn about the outsourced data. Therefore, there is a need for stronger security guarantees for dynamic searchable schemes.

Definition 5 (Forward private). A dynamic searchable encryption scheme which is $(\mathcal{L}_1, \mathcal{L}_2)$ -adaptively secure is said to be forward private if the following holds:

$$\mathcal{L}^{\text{upd}} \leftarrow \mathcal{L}'(\text{op}, \{f_i, \mu_i\}) \quad (2)$$

where $\mathcal{L}^{\text{upd}}$ is the update leakage function, $\{f_i, \mu_i\}$ is the set of document-number of keywords updated pair, f_i is the document updated, μ_i is the number of keywords updated on respective f_i and $\mathcal{L}'$ a stateless function.

The forward private definition 5 states that the server should be able to learn only about the file identifier(s), size of file(s) and number of keywords contained in file(s) after observing searches before and after a dynamic update operation $\text{op} \in \{\text{ADD}, \text{DELETE}\}$ over encrypted index is performed.

2.3 Homomorphic and orthogonality

The design of dynamic searchable schemes satisfying definitions 4 and 5 more often requires the use of encryption mechanisms called homomorphic cryptosystems as a building block.

Definition 6 (Homomorphic). A public key encryption scheme $E = (\text{KGen}, \text{Enc}, \text{Dec}, M, C)$ is said to be homomorphic if it holds that $\text{Dec}(\text{sk}, c_1 \diamond c_2) = m_1 \diamond m_2$ for all $m_1, m_2 \in M$ and $c_1, c_2 \in C$ with $m_1 = \text{Dec}(\text{sk}, c_1)$ and $m_2 = \text{Dec}(\text{sk}, c_2)$ such that for all c obtained from $\text{Enc}(\text{pk}, m)$, c belongs to C .

The proposed PINDEX uses Paillier cryptosystem to instantiate the encryption which exhibits homomorphic addition property. The group operation $\diamond$ for Paillier cryptosystem has homomorphic property such that $\text{Dec}(\text{sk}, c_1 \diamond c_2) = m_1 + m_2$ [16]. Let $V = v_1, v_2, \dots, v_{|W|}$ be a matrix of mutually orthogonal vectors where v_i is the i^{th} row vector and $|W|$ is the number of keywords in the keyword collection W . A square $n \times n$ matrix V with elements ± 1 that satisfies $V \times V^T = nI_n$ is called a Hadamard matrix of order n [17, 18]. Such matrices make cross correlation values to be zero and this property is used in the design of the proposed search and update mechanism of PINDEX.

3 Construction

The cloud data storage service involves three entities namely, the Data Owner (DO), Cloud Server or the Cloud Service Provider (CSP) and the Data User (DU). The index is constructed using a novel multi-linked hash map structure. The index construction based on the security model presented in section 2 is given below:

3.1 Multi-linked hash map construction

The two dimensional matrix M_{mn} whose elements are stored in a multi-linked hash map δ , where each element $\{m_{ij}; m_{ij} = [\text{next}, \text{prev}], m_{ij} \in M_{mn}, 1 \leq i \leq m, 1 \leq j \leq n\}$ has link to the next non-empty element in the respective row i and column j . The row links are implemented by

Fig 2. Multi-linked hash map δ .

<https://doi.org/10.1371/journal.pone.0256223.g002>

orthogonal vectors and respective column links using pointers as shown in Fig 2. The hash table value $\delta(i)$ is orthogonally linked row values for all $1 \leq i \leq m$ as given below:

$$\delta(i) = \left\{ \sum_{j=1}^n v_j \cdot m_{ij} : m_{ij} \neq \emptyset \right\} \quad (3)$$

Eq 3 shows that each value is a sum of inner product of the orthogonal vector v_j and a pointer to the next non-empty value in the column j . Each v_j represents a column and all v_j 's are mutually orthogonal 4.

$$\langle v_a, v_b \rangle = \begin{cases} 1, & \text{if } a = b \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

Hence, assuming y such that $y = j$,

$$\begin{aligned} \langle v_y, \delta(i) \rangle &= \left\langle \sum_{j=1}^n v_j \cdot m_{ij}, v_y \right\rangle \\ &= v_1 \cdot m_{i1} \cdot v_y + \dots + v_j \cdot m_{ij} \cdot v_y + \dots + v_n \cdot m_{in} \cdot v_y = m_{ij} \end{aligned} \quad (5)$$

where m_{ij} by definition is a pointer to the next non-empty element in the column j and $\langle \cdot, \cdot \rangle$ denotes vector inner product. Therefore, all the non-zero elements in the column j can be retrieved using the recursive function:

$$x_{i+1} = \begin{cases} \langle v_y, \delta(x_i) \rangle & \text{if } x_i \neq \perp \\ \perp, & \text{otherwise} \end{cases} \quad (6)$$

The hash table value δ can be used as an index for retrieving documents containing a given keyword. Let the rows and columns in matrix M represent documents and keywords respectively. Each element in a column j is a pointer to next document containing the keyword w_j . Each row i is encoded as an orthogonal sum of product of keyword vector v_j and the next document f_i containing keyword w_j and stored in a keyword multi-linked hash map as $\delta(f_i)$. Each element $m_{ij} = [f_{next}, f_{prev}]$ represents the next and previous file of f_i containing the keyword w_j

and an empty value $m_{ij} = \emptyset$ represents that $w_j \notin f_i$. Therefore, using Eq 6 set of all documents F_q containing query keyword w_q can be retrieved in $F_q \subseteq F$ in sub-linear time as for all practical queries $|F_q| \ll |F|$. The multi-linked hash map data structure is designed as described above such that it supports addition of new documents which includes encoding a row, adding encoded value and updating either the last or first row pointers accordingly. The multi-linked hash map data structure design improves search to sub-linear time and supports parallel execution of search and updates.

3.2 Algorithm $K \leftarrow \text{KGen}(1^\lambda)$

It is a probabilistic polynomial time algorithm which generates the secret parameters used for building index and encryption. The PINDEX algorithm uses Paillier Cryptosystem for building the index and generation of search and update tokens. The key (pk , sk) generation setup is described in [18] where pk denotes the public key and sk denotes the private key of Paillier cryptosystem. The KGen also generates a secret key k to be used by Enc_1 for encrypting the document collection D . The design choice for Enc_1 is the CPA-secure AES while the design choice for Enc_2 is the CPA-secure Paillier cryptosystem which also exhibits additive homomorphic properties.

3.3 Algorithm $c \leftarrow \text{Enc}_2(K, m, r)$

It is a probabilistic algorithm, given a message $m \in P$ and a public key $\text{pk} = (n, g)$, $\text{Enc}_2(\text{pk}, m, r)$ returns the ciphertext $c = g^m r^n \bmod n^2$, where given r is a random value chosen such that $r \in \mathbb{Z}_n^* | \gcd(r, n) = 1$.

3.4 Algorithm $m = \text{Dec}_2(\text{sk}, c)$

It is a probabilistic algorithm, given a ciphertext $c \in C$ and a private key $\text{sk} = (p, q, \lambda)$, $\text{Dec}_2(\text{sk}, c)$ returns the message m .

$$m = \frac{L(c^\lambda \bmod n^2)}{L(g^\lambda \bmod n^2)} \bmod n \quad (7)$$

3.5 Algorithm $(\gamma, \Gamma) \leftarrow \text{BuildIndex}(K, \delta, V, D, \text{Enc}_2)$

BUILDINDEX algorithm constructed by the data owner takes as input five parameters namely the document set $F = \{f_1, f_2, \dots, f_{|D|}\}$, keyword set $W = \{w_1, w_2, \dots, w_{|W|}\}$, a pre-built multi-linked hash map δ as described in section 3.1 for F , mutually orthogonal vector V and the key K as input. The algorithm outputs multi-linked hash map index γ and keyword parameter hash table Γ . The algorithm computes:

1. For each row i or file f_i in δ :
 - For each column j or word w_j in f_i :
 - if $m_{ij} \neq \emptyset$ and $\Gamma(w_j) = \emptyset$ then compute $\Gamma(w_j) = (v \leftarrow_{\$} V_w, r \leftarrow_{\$} \mathbb{Z}_n)$
 - Compute $s \leftarrow \text{Enc}_2(\text{pk}, w_j, r)$
 - Compute $\gamma(0) = \gamma(0) + v.s.f$ and $\gamma(f_i) = \gamma(f_i) + v.s.\perp$
 - if $m_{ij} \neq \emptyset$ and $\Gamma(w_j) \neq \emptyset$ then retrieve $(v, r) = \Gamma(w_j)$
 - Compute $s \leftarrow \text{Enc}_2(\text{pk}, w_j, r)$
 - if $f_{\text{next}} \neq \emptyset$ then compute $\gamma(f_i) = \gamma(f_i) + v.s.f_{\text{next}}$ otherwise $\gamma(f_i) = \gamma(f_i) + v.s.\perp$
 - Compute $v_r \leftarrow_{\$} V_r$ and $r' \leftarrow_{\$} \mathbb{Z}_m$
 - Compute $\gamma(f_i) = \gamma(f_i) + v_r.r'$

Fig 3. Index structure.

<https://doi.org/10.1371/journal.pone.0256223.g003>

2. Return (γ, Γ)
3. Send γ to CSP and Γ is kept secret at the DO

The pictorial representation of index structure is given in Fig 3. The encrypted index (γ, Γ) can be observed to be designed such that the link can be traversed vertical through the pointers and traversed horizontal through the orthogonal property without leaking critical information.

3.6 Algorithm $\tau_s \leftarrow \text{SrchToken}(w_q, \text{pk}, V)$

The SRCHToken algorithm is computed by the Data Owner. It takes a query keyword $w_q \in W$, key pk , set of mutually orthogonal vectors V as input and outputs a search token τ_q . The algorithm is given below:

1. Get the keyword secret $(v, r) \leftarrow \Gamma(w_q)$ from the keyword parameter hash table Γ and compute $r^{-1} \in \mathbb{Z}_n$ such that $rr^{-1} \equiv 1 \pmod{n^2}$
2. Compute $s^{-1} \leftarrow \text{Enc}_2(\text{pk}, -w_q, r^{-1})$ and returns to DU the search token τ_q by computing:

$$\tau_q \leftarrow v.s^{-1} + v'.r' \quad (8)$$

where $r' \leftarrow \mathbb{Z}_n$ and $v' \leftarrow V_r$

The trapdoor τ_q is designed to contain the encrypted keyword parameters embedded with the corresponding orthogonal vector v and random $v'.r'$ are added at each instance to make the token appear indistinguishable even when the same query is repeated.

3.7 Algorithm $F_q \leftarrow \text{Search}(\gamma, \tau_q)$

The SEARCH algorithm is computed at CSP and it takes as input the encrypted index γ , search token τ_q , key pk and outputs $F_q = \{f_1, f_2, \dots, f_n\}$. F_q is the set of document identifiers such that w_q satisfies $w_q \in W$ and $\perp$ otherwise, where $\perp$ denotes null. The search algorithm is given below:

1. Compute $\langle \gamma(0), \tau_q \rangle$ to obtain the first file pointer $f_q = f_1$ of document containing the keyword w_q .
2. If $f_q = \emptyset$ then there are no documents matching the query keyword w_q or the search token is invalid. The algorithm terminates returning $\perp$.

3. Otherwise, obtain all f_q computing recursively as given below with base condition $f_q = f_1$ until $f_q = \perp$

$$f_{q+1} = \begin{cases} \langle \tau_q, \gamma(f_q) \rangle & \text{if } f_q \neq \perp \\ \perp, & \text{otherwise} \end{cases} \quad (9)$$

4. Return all file identifiers $\{f_1, \dots, f_q, \dots, |F_q|\}$ obtained.

Thus, it can be observed that the design of the search algorithm is simple and dependent on the search token design. Since, the search token is designed to be created with the orthogonally embedded encrypted keyword parameters (v, s^{-1}) and the index contains (v, s) , a inner product of the two will result in retrieving the corresponding document identifiers. The proof of correctness for which is presented in section 4.

3.8 Algorithm $\tau_q \leftarrow \text{UpdToken}(\text{pk}, f, u, w)$

The UPDTOKEN probabilistic algorithm is computed by the DO. The algorithm returns an update token to the CSP for securely updating the index. We define a header node to be the node pointed by $\gamma(0)$ or the row when $i = 1$ in δ , a terminal node to be the last file containing $\perp$ for a w in δ and any other node as intermediate nodes containing a keyword w . The algorithm computes:

1. When the update operation is DELETEKEYWORD $w \in W$ from document f then compute:
 - a. Get $(v, r) \leftarrow \Gamma(w)$, compute $s \leftarrow \text{Enc}_2(\text{pk}, w, r)$
 - b. When f is the header node, compute $\tau_1 \leftarrow v.s.f_{next}$, $\tau_2 \leftarrow v.s.f$ and return $\tau \leftarrow (\tau_1, \tau_2, f_1 = 0, f_2 = f)$
 - c. When f is the terminal node, compute $\tau_1 \leftarrow v.s.\perp$, $\tau_2 \leftarrow v.s.f$ and return $\tau \leftarrow (\tau_1, \tau_2, f_1 = f_{prev}, f_2 = f)$
 - d. When f is an intermediate node, determine f_{next}, f_{prev} , compute $\tau_1 \leftarrow v.s.f_{next}$, $\tau_2 \leftarrow v.s.f$ and return $\tau_u \leftarrow (\tau_1, \tau_2, f_1 = f_{prev}, f_2 = f)$
2. When the update operation is DELETEFILE for a document f then compute:
 - a. Get $(v, r) \leftarrow \Gamma(w)$, compute $s \leftarrow \text{Enc}_2(\text{pk}, w, r)$,
 - b. When f is the header node, compute

$$\begin{aligned} \tau_1 &= \tau_1 \cup v_i.s_i.f_{next}, & 1 < i < |\forall w \in f| \\ f_1 &= f_1 \cup f_{next,i}, & 1 < i < |\forall w \in f| \\ \tau_2 &= \tau_2 \cup v_i.s_i.f, & 1 < i < |\forall w \in f| \end{aligned}$$

and return $\tau \leftarrow (\tau_1, \tau_2, f_1, f_2 = f)$

- c. When f is the terminal node, compute

$$\begin{aligned} \tau_1 &= \tau_1 \cup v_i.s_i.\perp, & 1 < i < |\forall w \in f| \\ f_1 &= f_1 \cup f_{prev,i}, & 1 < i < |\forall w \in f| \\ \tau_2 &= \tau_2 \cup v_i.s_i.f, & 1 < i < |\forall w \in f| \end{aligned}$$

and return $\tau \leftarrow (\tau_1, \tau_2, f_1, f_2 = f)$

d. When f is an intermediate node, determine f_{next}, f_{prev} , compute

$$\begin{aligned}\tau_1 &= \tau_1 \cup v_i \cdot s_i \cdot f_{next,i}, & 1 < i < |\forall w \in f| \\ f_1 &= f_1 \cup f_{prev,i}, & 1 < i < |\forall w \in f| \\ \tau_2 &= \tau_2 \cup v_i \cdot s_i \cdot f, & 1 < i < |\forall w \in f|\end{aligned}$$

and return $\tau_u \leftarrow (\tau_1, \tau_2, f_1, f_2 = f)$

3. When the update operation is **ADDKEYWORD** $w \in W$ to document f then compute:

- Get $v \leftarrow_{\S} V_w, r \leftarrow_{\S} \mathbb{Z}_n, v' \leftarrow_{\S} V_r$, compute $s \leftarrow \text{Enc}_2(\text{pk}, w, r)$
- When f is the header node, compute $\tau_1 \leftarrow v \cdot s \cdot f, \tau_2 \leftarrow v \cdot s \cdot f_{next}$ and return $\tau \leftarrow (\tau_1, \tau_2, f_1 = 0, f_2 = f)$
- When f is the terminal node, compute $\tau_1 \leftarrow v \cdot s \cdot f, \tau_2 \leftarrow v \cdot s \cdot \perp$ and return $\tau \leftarrow (\tau_1, \tau_2, f_1 = f_{prev}, f_2 = f)$
- When f is an intermediate node, determine f_{next}, f_{prev} in respect of f , compute $\tau_1 \leftarrow v \cdot s \cdot f, \tau_2 \leftarrow v \cdot s \cdot f_{next}$ and return $\tau_u \leftarrow (\tau_1, \tau_2, f_1 = f_{prev}, f_2 = f)$

4. When the update operation is **ADDFILE** for a document f then compute:

- Get $v \leftarrow_{\S} V_w, r \leftarrow_{\S} \mathbb{Z}_n, v' \leftarrow_{\S} V_r$
- Compute $s \leftarrow \text{Enc}_2(\text{pk}, w, r)$
- Add f as the header node by computing

$$\tau_1 = \sum_{i=1}^{|w \in f|} v_i \cdot s_i \cdot f, \quad \tau_2 = \sum_{i=1}^{|w \in f|} v_i \cdot s_i \cdot f_{next,i}$$

and return $\tau_u = (\tau_1, \tau_2, f_1 = 0, f_2 = f)$

3.9 Algorithm $\gamma' \leftarrow \text{Update}(\gamma, \tau_u, f_1, f_2)$

This algorithm takes an index γ , an update token $\tau_u = (\tau_1, \tau_2)$ and the public key pk as input and updates the encrypted index. The updated index γ' is computed by **UPDATE** algorithm is given below:

1. When the update operation $u \in \{\text{DELETEKEYWORD}, \text{DELETEFILE}\}$

$$\gamma' = \begin{cases} \gamma(f) = \gamma(f) + k - \tau_2, & \forall k \in \tau_1, \forall f \in f_1 \\ \gamma(f_2) = \gamma(f_2) - \tau_1, & \forall k \in \tau_1 \end{cases}$$

2. When the update operation $u \in \{\text{ADDKEYWORD}, \text{ADDFILE}\}$

$$\gamma' = \begin{cases} \gamma(f_1) = \gamma(f_1) + \tau_1 - \tau_2 \\ \gamma(f_2) = \gamma(f_2) + \tau_2 \end{cases}$$

The rationale behind the design of **UPDATE** described above and depicted in Fig 4 is based on exploiting the orthogonal property of V and homomorphic property of s used in creating

Fig 4. Update working.

<https://doi.org/10.1371/journal.pone.0256223.g004>

the index (γ, Γ) . This enables addition and deletion over encrypted index such that the tokens are indistinguishable guaranteed by CPA-security.

4 Security analysis

The privacy guarantees and correctness of the proposed design, PINDEX with respect to the security model given in section 2 and particularly in definition 4 is analysed in this section. The first formal security model given by Bellare et al. [19] is a game-based definition in which the adversary is allowed to interact with a set of oracles (encryption and decryption). The definition models communicating parties in a network where the adversary's goal is to distinguish between a correct shared secret or random value for the given challenge. Similar approach is used to prove the indistinguishability of the allowed information leakage of the protocol PINDEX . Firstly, the correctness of the proposed protocol is proved independently. Secondly, the privacy and correctness with respect to the security model is proved. The SEARCH can handle only two cases: $w_q \in W$ and $w_q \notin W$ during its operation. To prove the correctness in each case, the proof follows two lemmas:

Lemma 1. When given a search token τ_{w_q} for a keyword ($w_q \notin W$) and the index γ , the $\text{SEARCH}(\gamma, \tau_{w_q})$ returns $\perp$ such that

$$\Pr[F_q \leftarrow \text{SEARCH}(\gamma, \tau_{w_q \notin W}) : F_q = \{\perp\}_{w_q \notin W}] = 1$$

Proof. The concise representation of the SEARCH algorithm for the above case is given below:

$$\begin{aligned} f_l &= \gamma(0) \cdot \tau_{w_q} = \left(\sum_{i=1}^{|W|} v_i \cdot s_i \cdot f_{z_i} + v_r \cdot r \right) \cdot (v_q \cdot s_q^{-1} + v_r \cdot r) \\ &= \left(\sum_{i=1}^{|W|} v_i \cdot \text{Enc}_2(\text{pk}, w_i, r_i) \cdot f_{z_i} + v_r \cdot r \right) \cdot (v_q \cdot \text{Enc}_2(\text{pk}, -w_q, r_q^{-1}) + v_r \cdot r) \\ &= v_i \cdot v_q \cdot \text{Enc}_2(\text{pk}, w_i, r_i) \cdot \text{Enc}_2(\text{pk}, -w_q, r_q^{-1}) \cdot f_{z_i} = 0 \end{aligned}$$

When $i \neq q$, $v_i.v_q = 0$ and $\forall (v_i, v_q) \notin V$, $v_i.v_q = 0$. It can be observed from the SEARCH algorithm that $\perp$ is returned whenever $f_i = 0$. Therefore, whenever a query keyword $w_q \notin W$ is searched over the document set the SEARCH algorithm returns $\perp$ with probability 1.

Lemma 2. When given a search token τ_{w_q} for a keyword ($w_q \in W$) and the index γ , the SEARCH(γ, τ_{w_q}) returns $F_q = \{f_i\}_{w_q \in W}^{i=1, \dots, |w_q \in W|}$ such that

$$\Pr[F_q \leftarrow \text{SEARCH}(\gamma, \tau_{w_q \in W}) : F_q = \{f_i\}_{w_q \in W}^{i=1, \dots, |w_q \in W|}] = 1$$

Proof. Similar to lemma 1, the SEARCH algorithm can be generally represented for $w_q \in W$ as:

$$\begin{aligned} f_i &= \gamma(0) \cdot \tau_{w_q} = \left(\sum_{i=1}^{|W|} v_i \cdot s_i \cdot f_{z_i} + v_r \cdot r \right) \cdot (v_q \cdot s_q^{-1} + v_r \cdot r) \\ &= \left(\left(\sum_{i=1}^{|W|} v_i \cdot \text{Enc}_2(\text{pk}, w_i, r_i) \cdot f_{z_i} + v_r \cdot r \right) \cdot (v_q \cdot \text{Enc}_2(\text{pk}, -w_q, r_q^{-1}) + v_r \cdot r) \right) \\ &= v_i \cdot v_q \cdot g^{w_i} \cdot r_i^n \cdot g^{-w_q} \cdot (r_q^n)^{-1} \cdot f_i \bmod n^2 = f_i \end{aligned}$$

When $i = q$, $v_i.v_q = 1$ for all $(v_i, v_q) \in V$ and $g^{w_i-w_q} = g^0 = 1$ and $r_i^n \cdot (r_q^n)^{-1} = 1$. Thus, a file identifier f_i will be obtained by the SEARCH algorithm. Therefore, when the computation is recursively repeated until $f_i = \perp$, the algorithm returns the set of file identifiers ($F_q = \{f_i\}_{w_q \in W}^{i=1, \dots, |w_q \in W|} = \{f_1, f_n\}$ with probability 1.

Lemma 3. The algorithm UPDATE provides correct dynamic index update for a given update token τ_u .

Proof. The UPDATE algorithm updates the encrypted index using an update token generated for both the cases addition or deletion of files/keywords.

Case DELETEKEYWORD: The process of deleting a keyword is as follows:

1. When f is the header node file pointer, the UPDATETOKEN computes $\tau_1 \leftarrow v.s.f_{next}$, $\tau_2 \leftarrow v.s.f$ and returns $\tau_u \leftarrow (\tau_1, \tau_2, f_1 = 0, f_2 = \perp)$. The UPDATE is represented as:

$$\gamma(f_1) = \gamma(f_1) + \tau_1 - \tau_2 = \gamma(0) + v_{w_u} \cdot s_{w_u} \cdot f_{next} - v_{w_u} \cdot s_{w_u} \cdot f \quad (10)$$

2. When f is the terminal node, compute $\tau_1 = v.s.\perp$, $\tau_2 = v.s.f$ and return

$$\begin{aligned} \tau_u &= (\tau_1 = \{k\}_{i=1}^{|w \in f|}, \tau_2, f_1 = \{f_{prev,i}\}_{i=1}^{|w \in f|}, f_2 = \perp) \\ \gamma(f_1) &= \gamma(f_1) + \tau_1 - \tau_2 = \gamma(f_{prev}) + v_{w_u} \cdot s_{w_u} \cdot \perp - v_{w_u} \cdot s_{w_u} \cdot f \end{aligned} \quad (11)$$

3. When f is neither a header nor a terminal node pointer, determine f_{next}, f_{prev} , compute $\tau_1 \leftarrow v.s.f_{next}$, $\tau_2 \leftarrow v.s.f$ and return $\tau_u \leftarrow (\tau_1 = \{k\}_{i=1}^{|w \in f|}, \tau_2, f_1 = \{f_{prev,i}\}_{i=1}^{|w \in f|}, f_2 = \perp)$

$$\gamma(f_1) = \gamma(f_1) + \tau_1 - \tau_2 = \gamma(f_{prev}) + v_{w_u} \cdot s_{w_u} \cdot f_{next} - v_{w_u} \cdot s_{w_u} \cdot f \quad (12)$$

Eqs 10, 11 and 12 demonstrates how the file pointers are adjusted when the keyword w_u has to be added or removed. The old pointers are removed and new pointers are updated from/to previous or next appropriate file depending on the case.

Case ADDKEYWORD: The process of adding a keyword can be represented as follows:

1. When f is the first node, compute $\tau_1 \leftarrow v.s.f$, $\tau_2 \leftarrow v.s.f_{next}$ and return $\tau_u \leftarrow (\tau_1, \tau_2, f_1 = 0, f_2 = f)$

$$\begin{aligned}\gamma(f_1) &= \gamma(f_1) + \tau_1 - \tau_2 = \gamma(0) + v_{w_u}.s_{w_u}.f_{next} - v_{w_u}.s_{w_u}.f \\ \gamma(f_2) &= \gamma(f_2) + \tau_2 = \gamma(f) + v.s.f_{next}\end{aligned}\quad (13)$$

2. When f is the terminal node, compute $\tau_1 \leftarrow v.s.f$, $\tau_2 \leftarrow v.s.\perp$ and return $\tau_u \leftarrow (\tau_1, \tau_2, f_1 = f_{prev}, f_2 = \perp)$

$$\gamma(f_1) = \gamma(f_1) + \tau_1 - \tau_2 = \gamma(f_{prev}) + v_{w_u}.s_{w_u}.f - v_{w_u}.s_{w_u}.\perp \quad (14)$$

3. When f is a neither a header nor a terminal node pointer, determine f_{next}, f_{prev} , compute $\tau_1 \leftarrow v.s.f$, $\tau_2 \leftarrow v.s.f_{next}$ and return $\tau_u \leftarrow (\tau_1, \tau_2, f_1 = f_{prev}, f_2 = f)$

$$\begin{aligned}\gamma(f_1) &= \gamma(f_1) + \tau_1 - \tau_2 = \gamma(f_{prev}) + v_{w_u}.s_{w_u}.f - v_{w_u}.s_{w_u}.f_{next} \\ \gamma(f_2) &= \gamma(f_2) + \tau_2 = \gamma(f) + v.s.f_{next}\end{aligned}\quad (15)$$

Eqs 13, 14 and 15 shows how the file pointers are modified while adding the keyword w_u . File addition ADDFILE and deletion DELETEFILE follow similar modification of file pointers. Hence, the dynamic index update is performed correctly.

To prove the indistinguishability between the real game and the ideal game by any *PPT* distinguisher *Dist*, we construct a *PPT* simulator *Sim* which uses the leakage functions $\mathcal{L}_1$ and $\mathcal{L}_2$ to simulate functionality correctly of the algorithms described in Definition 1 indistinguishably.

Theorem 1. *Given the Definition(CKA-2 security) 4, the PINDEX scheme given above is $(\mathcal{L}_1, \mathcal{L}_2)$ -secure in the random oracle model, where $\mathcal{L}_1$ leaks the number of keywords, the number of documents, the identifiers of the documents and the size of each document; and $\mathcal{L}_2$ leaks the search pattern and the access pattern.*

Proof. Let *Sim* be the simulator which interacts with an adversary $\mathcal{A}$ in an execution of an $\text{IDEAL}_{\mathcal{A}, \text{Sim}}(k)$ as in Definition 4. We construct (γ, c) given the leakage $\mathcal{L}_1(\delta, f)$ as follows:

1. Given the number of files n and size of each file l through the leakage function, the *Sim* obtains $c = (c_1, \dots, c_n)$ by simulating the encryption of files using $\text{Sim}_{\text{Enc}_1}$ which exists by definition of CPA-Secure Enc_1 .
2. Given the file identifiers $F = (f_1, \dots, f_n)$ through the leakage function, the *Sim* simulates the index γ :
 - a. The *Sim* builds the multi-linked list δ by generating a order m hadamard matrix, V where m is the number of keywords in the universal keyword set. If no such order m exists then the least greatest order to m is chosen.
 - b. Construct $(n + 1) \times m$ state matrix S where each entry $S_{ij} \in \{0, 1\}$ is determined by flipping a coin.

- c. Compute $\gamma(0) \leftarrow \sum_{j=1}^m v_j \cdot \text{Enc}_2(\text{pk}, j, r) \cdot f_x$ such that $S_{ij} \neq 0$ and $x = \sup(X)$ where $X = \{x : S_{yj} \neq 0 \wedge 1 < y \leq (n+1)\}$, $v_j \leftarrow_{\$} V_w$, $v_r \leftarrow_{\$} V_r$ and r is obtained from random oracle H given j . Store $\delta(j) \leftarrow (v_j, r)$.
 - d. Compute $\gamma(f_i) \leftarrow \sum_{j=1}^m v_j \cdot \text{Enc}_2(\text{pk}, j, r) \cdot f_x$ for all $1 \leq i \leq n+1$ such that $S_{ij} \neq 0$ and if $\sup(X) \neq \emptyset$ then $x = \sup(X)$ otherwise $x = \perp$ where $X = \{x : S_{yj} \neq 0 \wedge i < y \leq (n+1)\}$, $(v_j, r) \leftarrow \delta(j)$.
3. Sim then gives adversary $\mathcal{A}$ the encrypted index γ .
 4. When the Sim receives an adaptive query q for keyword w_j , it computes $\tau_q \leftarrow v_j \cdot \text{Enc}_2(\text{pk}, -j, r^{-1}) + v_r \cdot r'$ and returns τ_q to the adversary $\mathcal{A}$. The $\mathcal{A}$ may use the SEARCH function for the obtained token τ_q and it will obtain exactly the same file identifiers as in the leakage function $\mathcal{L}_2$ every time correctly although the search token are different for same keyword w_j . It repeats only after making $|V_r|$ queries for the same keyword w_j .
 5. When the Sim receives an adaptive update query $q = u = f_i$ and if u is for add, the simulator uses $\text{Sim}_{\text{Enc}_1}$ to simulate c_i assuming that the leakage function $\mathcal{L}_1$ reveals the identifier and performs:
 - a. Updates the state by adding a row to the state matrix S where $q = n+2$ and each entry $S_{qj} \in \{0,1\}$ is determined by flipping a coin.
 - b. When f is first row and j is the keyword, compute $\tau_1 \leftarrow v \cdot \text{Enc}_2(\text{pk}, j, r) \cdot f$, $\tau_2 \leftarrow v \cdot \text{Enc}_2(\text{pk}, j, r) \cdot f_{\text{next}}$ where $\text{next} = \sup(X)$ and $X = \{\text{next} : S_{yj} \neq 0 \wedge 1 < y \leq (n+1)\}$ and $(v, r) \leftarrow \delta(j)$. Returns $\tau \leftarrow (\tau_1, \tau_2, f_1 = 0, f_2 = f)$.
 - c. When f is the terminal row and j is the keyword, compute $\tau_1 \leftarrow v \cdot \text{Enc}_2(\text{pk}, j, r) \cdot f$, $\tau_2 \leftarrow v \cdot \text{Enc}_2(\text{pk}, j, r) \cdot \perp$. Returns $\tau \leftarrow (\tau_1, \tau_2, f_1 = f_{\text{prev}}, f_2 = \perp)$ where $\text{prev} = \inf(X)$ and $X = \{\text{prev} : S_{yj} \neq 0 \wedge i > y > 0\}$ and $(v, r) \leftarrow \delta(j)$.
 - d. When f is a neither a header nor a terminal node pointer, determine $f_{\text{next}}, f_{\text{prev}}$, compute $\tau_1 \leftarrow v \cdot s \cdot f$, $\tau_2 \leftarrow v \cdot s \cdot f_{\text{next}}$ where $\text{prev} = \inf(X)$ and $X = \{\text{prev} : S_{yj} \neq 0 \wedge i > y > 0\}$ while $\text{next} = \sup(X)$ and $X = \{\text{next} : S_{yj} \neq 0 \wedge i < y \leq (n+1)\}$ and $(v, r) \leftarrow \delta(j)$. Returns $\tau_u \leftarrow (\tau_1, \tau_2, f_1 = f_{\text{prev}}, f_2 = f)$.
 - e. $\mathcal{A}$ can then use τ_u on UPDATE function and obtain the updated encrypted index γ' .
 6. Similarly, Sim can simulate operation ADD and DELETE for keywords and file.

The results returned to the $\mathcal{A}$'s random oracle queries by Sim are consistent. The keys used in the tokens and keys used in the construction of index γ are indistinguishable since $(\text{Enc}_1, \text{Enc}_2)$ used to encrypt is by definition indistinguishable from random. Hence, the CPA-Security of $(\text{Enc}_1, \text{Enc}_2)$ guarantee indistinguishability between real and simulated encryptions of the files and index by $\mathcal{A}$.

5 Performance analysis

This section presents the performance analysis of the proposed design for its efficiency. PINDEX is implemented in Java and the experimental results are evaluated on a Windows server with Intel Xeon Processor running at 2.30 GHz and 16 GB RAM. The experiment uses real world Enron email dataset [20]. The performance of PINDEX is evaluated and compared with MRSE [21]. The experimental results are shown in Fig 5.

Fig 5. Experimental results.

<https://doi.org/10.1371/journal.pone.0256223.g005>

5.1 Multi-link index

Building the multi-linked index is a one-time process carried out by the data owner. The multi-linked dynamic searchable index γ can be constructed using two steps. First, encrypting the keyword w using Paillier cryptosystem if it belongs to document f and secondly computing the encrypted index γ which is a multi-link between the document f and keywords $w_i \in W$. Therefore the time cost of index generation is proportional to the number of files and number of keywords contained in each file. It can be observed from Fig 5a that the time cost for building the index with varying number of documents of the proposed system performs relatively better than MRSE. This is due to use of relatively small matrix dimensions in the design and computationally less intensive operations like scalar multiplications ($|W|$ operations) and vector additions ($|W| + 1$ operations).

5.2 Search token generation

The search token τ_q is constructed for the keywords in the query. The process involves retrieving the pair (v, r) from the parameter hash table Γ and encrypting the query keyword w_q . The time cost for search token depends on the number of query keywords. Fig 5b shows the time cost for generating the search token in logarithmic scale for document size $n = |D| = 1000$ and varying the size of the query keyword. It is observed that PINDEX performance is better than MRSE. This is because our scheme use less computationally intensive operations and the number of operations involved in building a trapdoor is minimum. The operations include two scalar multiplications, one vector addition for indistinguishability and one lookup.

5.3 Search

Search operation executed by the cloud server consists of multiplying the search token τ_q with the encrypted index γ . If $w_q \in W$, the first pointer is computed and recursively the multi linked index is traversed until $f_q = \perp$. For a multi-keyword search with $|w_q|$ keywords in a query q and run on p processors parallel which returned $|F_q|$ files then time complexity would be $\mathcal{O}(|F_q|/p)$. In MRSE similarity scores for the document set is computed and therefore the time complexity is $\mathcal{O}(N)$. The time cost for varying the number of documents for $m = |W| = 1000$ is shown in Fig 5c and 5d. Search time of P_{INDEX} is relatively better than MRSE. Also, it can be observed that as the number of keywords per query is increased, the corresponding increase in time of query to be sub-linear for the proposed design. It is due to optimal number of operations used in the design of search operation which involves just one vector inner product.

5.4 Update

The update (add and delete) of both keywords and document involves adding the update token τ_1 and deleting τ_2 from f_{prev} . The worst case scenario for update delete file operation is when the number of f_{prev} equals the number of keywords contained in the file to be deleted. Updates are not compared as MRSE does not support dynamic index, the index has to be rebuilt for updates and the results are shown in Fig 5e and 5f. The time cost for update i.e to add/ delete a keyword is $\mathcal{O}(1)$ and to add/delete a file is $\mathcal{O}(k/p)$. The proposed method supports dynamic updates unlike MRSE and it can be observed that dynamic updates comprises of building update trapdoor and executing update algorithm. The time to build update trapdoor is significantly less due to the same reasoning as that of building trapdoor and execution of update algorithm involves only vector additions and deletions as given in section 3.

6 Related works

The confidentiality of sensitive data like Electronic Health Records (EHR) outsourced to CSP is protected from external and internal attacks by deploying a searchable encryption scheme specially designed to support operations like search and updates over encrypted data [22].

6.1 Searchable encryption

The searchable encryption schemes are primarily classified into Searchable Symmetric Encryption and Public Key Encryption with Keyword Search [2] depending upon the type of encryption primitive used and notion of provable security. The searchable encryption systems is further classified based on the type of search operation into keyword based and semantic based searchable encryption system [23]. The keyword based searchable encryption system uses either specially designed secure encrypted indexes to support search operations over encrypted data or sequential scanning of encrypted documents to support search [4]. The keyword based searchable encryption system are exact search keyword based while the semantic keyword search matches even closest query keywords. The proposed design is based on keyword based searchable encryption schemes using secure encrypted index. The first practical solution to symmetric searchable encryption was proposed by Song et al. [3]. Followed by a number of searchable encryption schemes especially with secure index construction, improved efficiency, security definitions and formalizations [4, 5, 8, 9, 24–26].

6.2 Secure index

To improve performance and decrease the search complexity, clients build keyword-based indexes and outsource it to the cloud. Searchable encryption schemes in literature use a

forward, inverted or tree-based index [4, 24, 27, 28]. However, the scalability of the index based schemes can be achieved by either rebuilding the index or by using expensive techniques [5]. The index contains information such as document identifiers, document length, number of keywords in the file. The adversary should not be able to determine the query keyword using statistical analysis. Therefore, the index must not leak information which could aid honest-but-curious CSP to link tokens, keywords and document identifiers. The indexes and the tokens or trapdoors are inherently linked. Deterministic trapdoors are known to leak information therefore probabilistic trapdoors are used. Goh [4] uses forward index using bloom filters per document. The search complexity is proportional to the number of files and bloom filters have an inherent problem of false positives. Chang and Mitzenmacher [24] use prebuilt dictionary of distinct keywords to build the forward index. The search complexity is proportional to the number of files. Inverted index or index per keyword was introduced by [5] and thus reduced the search time to a sub-linear scheme with optimal search time. [26] proposed a scheme based on inverted index using hash tables.

6.3 Dynamic searchable encryption

Dynamism is a main requirement for any searchable encryption scheme to be practical. The first dynamic searchable encryption was proposed by Kamara et al. [8] using inverted index and dynamic addition and deletion of documents. The update complexity is linear to the updated document/keyword pair but the update leaks some information about the documents. Kamara and Pamananthou [9] proposed a red-black tree index based on an encrypted linked-list multi-map. To insert a new keyword, they added as a new entry to the right list and deletion using a dual representation of the index but update leaks information. Cash et al. [10] proposed a dynamic index using a separate update database to add and delete tuples stored in the revocation list. However, the update is inefficient and leaks significant information. Naveed et al. [11] presented a dynamic searchable encryption with blind storage. Instead of encrypting the index, the stored blocks are scattered using hashing. This scheme leaks the information while adding keyword. Other dynamic searchable encryption schemes are [7, 14, 29–31].

6.4 Forward privacy

There is a trade-off between practicality and security as most of the dynamic searchable encryption schemes leak significant amount of information about the document. Attacker can leverage this leakage to reveal information on the client's queries. Deterministic encryption schemes leak information regarding size, search, access patterns from the repetition of the queried keyword. File injection attack on dynamic searchable encryption by Zhang et al. [13] triggered much attention on forward privacy schemes. By injecting few carefully selected files, the adversary can recover keywords queried in the past from the information leaked from the search token. Thus, the prior search token needs to be invalidated to achieve forward security. Therefore, a stronger property called forward privacy and an informal definition was introduced by Stefanov et al. [12] based on Oblivious RAM. However, the scheme leads to a search cost $\mathcal{O}(|F_q| \log^3 N)$ and $\mathcal{O}(k \log^2 N)$ update cost. Existing schemes that achieve forward privacy are generally based on trapdoor permutation or using pseudorandom functions. Bost proposed the first formal definition of forward security and a trapdoor permutation based scheme *Sophos* [32]. Later proposed *Diana* [15] a constrained PRF based scheme. A dual dictionary *Dual* [33] was proposed that simultaneously uses both forward with inverted index and achieves forward privacy using fresh keys. To realize forward security, a symmetric punctured encryption *Janus ++* [34] was proposed. Etemad et al. [35] achieve forward privacy by replacing the keys revealed to the server. Sun et al. *Aura* [36] use a non-interactive DSSE using bloom

Table 1. Complexity and feature comparison.

Scheme	Client storage	Server storage	Search Cost	Update Cost	Forward	Parallel
Kamara et al. [8]	$\mathcal{O}(1)$	$\mathcal{O}(N)$	$\mathcal{O}(F_q)$	$\mathcal{O}(k)$	No	No
Parallel SSE [9]	$\mathcal{O}(1)$	$\mathcal{O}(mn)$	$\mathcal{O}(F_q \log n)/p$	$\mathcal{O}(m \log n/p)$	No	Yes
Blind Storage [11]	$\mathcal{O}(1)$	$\mathcal{O}(N)$	$\mathcal{O}(F_q + n_d)/p$	$\mathcal{O}(k/p)$	No	Yes
Cash et al. [10]	$\mathcal{O}(m)$	$\mathcal{O}(N)$	$\mathcal{O}(F_q + n_{ad})/p$	$\mathcal{O}(k/p)$	No	Yes
Practical SSE [12]	$\mathcal{O}(\sqrt{N})$	$\mathcal{O}(N)$	$\mathcal{O}(F_q \log^3 N)$	$\mathcal{O}(k \log^2 N)$	Yes	No
Sophos [32]	$\mathcal{O}(m)$	$\mathcal{O}(N)$	$\mathcal{O}(F_q + n_{ad})$	$\mathcal{O}(k)$	Yes	No
Etemad et al. [35]	$\mathcal{O}(m + n)$	$\mathcal{O}(N)$	$\mathcal{O}(F_q + n_d)/p$	$\mathcal{O}(k/p)$	Yes	Yes
Ours	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(F_q /p)$	$\mathcal{O}(1)/\mathcal{O}(k/p)$	Yes	Yes

n —total number of files, m —total number of keywords, $|F_q|$ —number of files containing a multiple keywords, k —number of unique keywords in a file. p and N are number of processors and (keyword, file ID) mappings respectively, n_{ad} and n_d —number of times a keyword has been affected by file deletions since beginning and since the last search for the same keyword, respectively and ($n_{ad} \geq n_d$).

<https://doi.org/10.1371/journal.pone.0256223.t001>

filters and multi-puncturable PRF. Wei et al. [37] proposed an index structure with keyed block chain. *Khons* [38] uses inverted index and achieves weak forward security by exploitation hidden pointer technique.

7 Discussion

A comparison of our work with existing dynamic schemes is given in Table 1. In our work, there is a relative improvement in client and server storage. The client storage is $\mathcal{O}(n)$ as the client needs to store the δ , which is the sum of the inner product of orthogonal vectors for documents and Γ the keyword parameter hash table. The server storage is $\mathcal{O}(n)$ as the server stores the encrypted multi-linked index γ that contains the number of document and the keywords $w_i \in W$. Efficient updates are performed without rebuilding the index. The search and update tokens are indistinguishable from the previous queries. The proposed multi-linked index is parallelizable and provides relatively efficient search and update with $\mathcal{O}(|F_q|/p)$ and $\mathcal{O}(k/p)$ respectively. The efficient time complexity of search is achieved by exploiting the vertical pointers and orthogonal horizontal linking property of the proposed multi-linked encrypted index structure which enables retrieval of the encrypted keyword index with a vector inner product. It can be observed from the `SRCHToken` algorithm that for each search, the token generated consists of the orthogonal sum of (v_i, s_i^{-1}) and (v_r, r) in which (v_r, r) is chosen uniformly random for each query to make the token indistinguishable for even the same query. Similarly, the `UPDTOKEN` algorithm also generates each token which are indistinguishable. This token generation strategy makes the tokens, file identifiers and keywords unlinkable which eventually provides forward privacy as in Definition 5. Moreover, the search can be performed in parallel for multiple tokens simultaneously in parallel on a shared index. This makes the proposed scheme more efficient in exploiting the available parallel processors and improves the execution speed of search operations.

8 Conclusion

This paper proposes a novel private multi-linked dynamic index construction for efficient retrieval of encrypted documents with forward privacy guarantees. The multi-linked index is constructed using probabilistic homomorphic encryption and secret orthogonal vectors. Experimental evaluations on Enron dataset proves that our construction achieves optimal

search and update cost. PINDEX supports parallelism using the multi-link structure. Thus, the server can distribute the load to its available p processors with search and update cost as $\mathcal{O}(|F_q|/p)$ and $\mathcal{O}(k/p)$ respectively. Forward privacy guarantees is achieved using probabilistic algorithms for search and update token generation. Further, the client and server storage is reduced to $\mathcal{O}(n)$. Security analysis of PINDEX using random oracle models proves the privacy and correctness of search and update operations. Our work PINDEX can be used on critical cloud infrastructure for secure and efficient encrypted document retrieval with forward privacy guarantees. As a future work, PINDEX can be adapted to provide verifiability and backward privacy.

Author Contributions

Conceptualization: A. John Prakash, B. Lydia Elizabeth.

Data curation: A. John Prakash, B. Lydia Elizabeth.

Formal analysis: A. John Prakash, B. Lydia Elizabeth.

Investigation: A. John Prakash, B. Lydia Elizabeth.

Methodology: A. John Prakash, B. Lydia Elizabeth.

Project administration: A. John Prakash, B. Lydia Elizabeth.

Resources: A. John Prakash, B. Lydia Elizabeth.

Software: A. John Prakash, B. Lydia Elizabeth.

Supervision: A. John Prakash, B. Lydia Elizabeth.

Validation: A. John Prakash, B. Lydia Elizabeth.

Visualization: A. John Prakash, B. Lydia Elizabeth.

Writing – original draft: A. John Prakash, B. Lydia Elizabeth.

Writing – review & editing: A. John Prakash, B. Lydia Elizabeth.

References

1. Kamara S, Lauter K. Cryptographic Cloud Storage. In: Financial Cryptography and Data Security. Springer Berlin Heidelberg; 2010. p. 136–149. Available from: https://doi.org/10.1007%2F978-3-642-14992-4_13.
2. Bösch C, Hartel P, Jonker W, Peter A. A Survey of Provably Secure Searchable Encryption. ACM Computing Surveys. 2015; 47(2):1–51.
3. Song DX, Wagner D, Perrig A. Practical techniques for searches on encrypted data. In: Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000. IEEE Comput. Soc.; Available from: <https://doi.org/10.1109%2Fsecpri.2000.848445>.
4. Goh EJ, et al. Secure indexes. IACR Cryptol ePrint Arch. 2003; 2003:216.
5. Curtmola R, Garay J, Kamara S, Ostrovsky R. Searchable symmetric encryption: Improved definitions and efficient constructions. Journal of Computer Security. 2011; 19(5):895–934. <https://doi.org/10.3233/JCS-2011-0426>
6. Boneh D, Waters B. Conjunctive, Subset, and Range Queries on Encrypted Data. In: Theory of Cryptography. Springer Berlin Heidelberg; p. 535–554. Available from: https://doi.org/10.1007%2F978-3-540-70936-7_29.
7. Hahn F, Kerschbaum F. Searchable Encryption with Secure and Efficient Updates. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. ACM; 2014. Available from: <https://doi.org/10.1145%2F2660267.2660297>.
8. Kamara S, Papamanthou C, Roeder T. Dynamic searchable symmetric encryption. In: Proceedings of the 2012 ACM conference on Computer and communications security—CCS'12. ACM Press; 2012. Available from: <https://doi.org/10.1145%2F2382196.2382298>.

9. Kamara S, Papamanthou C. Parallel and Dynamic Searchable Symmetric Encryption. In: Financial Cryptography and Data Security. Springer Berlin Heidelberg; 2013. p. 258–274. Available from: https://doi.org/10.1007%2F978-3-642-39884-1_22.
10. Cash D, Jaeger J, Jarecki S, Jutla C, Krawczyk H, Roşu MC, et al. Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation. In: Proceedings 2014 Network and Distributed System Security Symposium. Internet Society; 2014. Available from: <https://doi.org/10.14722%2Fndss.2014.23264>.
11. Naveed M, Prabhakaran M, Gunter CA. Dynamic Searchable Encryption via Blind Storage. In: 2014 IEEE Symposium on Security and Privacy. IEEE; 2014. Available from: <https://doi.org/10.1109%2Fsp.2014.47>.
12. Stefanov E, Papamanthou C, Shi E. Practical Dynamic Searchable Encryption with Small Leakage. In: Proceedings 2014 Network and Distributed System Security Symposium. Internet Society; 2014. Available from: <https://doi.org/10.14722%2Fndss.2014.23298>.
13. Zhang Y, Katz J, Papamanthou C. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In: 25th {USENIX} Security Symposium ({USENIX} Security 16); 2016. p. 707–720.
14. Elizabeth BL, Prakash AJ. Verifiable top-k searchable encryption for cloud data. *Sādhanā*. 2019; 45(1).
15. Bost R, Minaud B, Ohrimenko O. Forward and Backward Private Searchable Encryption from Constrained Cryptographic Primitives. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM; 2017. Available from: <https://doi.org/10.1145%2F3133956.3133980>.
16. Paillier P. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In: Advances in Cryptology—EUROCRYPT'99. Springer Berlin Heidelberg; . p. 223–238. Available from: https://doi.org/10.1007%2F3-540-48910-x_16.
17. Agaian SS. Construction of generalized Hadamard matrices. In: Lecture Notes in Mathematics. Springer Berlin Heidelberg; 1985. p. 103–133. Available from: <https://doi.org/10.1007%2Fbfb0101076>.
18. Casazza PG. THE ART OF FRAME THEORY. *Taiwanese Journal of Mathematics*. 2000; 4(2). <https://doi.org/10.11650/twjm/1500407227>
19. Bellare M, Rogaway P. Random oracles are practical. In: Proceedings of the 1st ACM conference on Computer and communications security—CCS'93. ACM Press; 1993. Available from: <https://doi.org/10.1145%2F168588.168596>.
20. Cohen WW. Enron email dataset. 2009.
21. Cao N, Wang C, Li M, Ren K, Lou W. Privacy-Preserving Multi-Keyword Ranked Search over Encrypted Cloud Data. 2014; 25(1):222–233.
22. Pham H, Woodworth J, Salehi MA. Survey on secure search over encrypted data on the cloud. *Concurrency and Computation: Practice and Experience*. 2019; p. e5284. <https://doi.org/10.1002/cpe.5284>
23. Liu G, Yang G, Bai S, Zhou Q, Dai H. FSSE: An Effective Fuzzy Semantic Searchable Encryption Scheme Over Encrypted Cloud Data. *IEEE Access*. 2020; 8:71893–71906. <https://doi.org/10.1109/ACCESS.2020.2966367>
24. Chang YC, Mitzenmacher M. Privacy Preserving Keyword Searches on Remote Encrypted Data. In: Applied Cryptography and Network Security. Springer Berlin Heidelberg; 2005. p. 442–455. Available from: https://doi.org/10.1007%2F11496137_30.
25. Amanatidis G, Boldyreva A, O'Neill A. Provably-Secure Schemes for Basic Query Support in Outsourced Databases. In: Data and Applications Security XXI. Springer Berlin Heidelberg; 2007. p. 14–30. Available from: https://doi.org/10.1007%2F978-3-540-73538-0_2.
26. Chase M, Kamara S. Structured Encryption and Controlled Disclosure. In: Advances in Cryptology—ASIACRYPT 2010. Springer Berlin Heidelberg; 2010. p. 577–594. Available from: https://doi.org/10.1007%2F978-3-642-17373-8_33.
27. Golle P, Staddon J, Waters B. Secure Conjunctive Keyword Search over Encrypted Data. In: Applied Cryptography and Network Security. Springer Berlin Heidelberg; 2004. p. 31–45. Available from: https://doi.org/10.1007%2F978-3-540-24852-1_3.
28. Liu Q, Wang G, Wu J. An Efficient Privacy Preserving Keyword Search Scheme in Cloud Computing. In: 2009 International Conference on Computational Science and Engineering. IEEE; 2009. Available from: <https://doi.org/10.1109%2Ficse.2009.66>.
29. Li H, Yang Y, Dai Y, Yu S, Xiang Y. Achieving Secure and Efficient Dynamic Searchable Symmetric Encryption over Medical Cloud Data. *IEEE Transactions on Cloud Computing*. 2020; 8(2):484–494. <https://doi.org/10.1109/TCC.2017.2769645>
30. Zuo C, Sun SF, Liu JK, Shao J, Pieprzyk J. Dynamic Searchable Symmetric Encryption Schemes Supporting Range Queries with Forward (and Backward) Security. In: Computer Security. Springer

International Publishing; 2018. p. 228–246. Available from: https://doi.org/10.1007%2F978-3-319-98989-1_12.

31. Xu L, Xu C, Liu JK, Zuo C, Zhang P. Building a dynamic searchable encrypted medical database for multi-client. *Information Sciences*. 2020; 527:394–405. <https://doi.org/10.1016/j.ins.2019.05.056>
32. Bost R. Forward Secure Searchable Encryption. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM; 2016. Available from: <https://doi.org/10.1145%2F2976749.2978303>.
33. Kim KS, Kim M, Lee D, Park JH, Kim WH. Forward Secure Dynamic Searchable Symmetric Encryption with Efficient Updates. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM; 2017. Available from: <https://doi.org/10.1145%2F3133956.3133970>.
34. Sun SF, Yuan X, Liu JK, Steinfeld R, Sakzad A, Vo V, et al. Practical Backward-Secure Searchable Encryption from Symmetric Puncturable Encryption; 2018. Available from: <https://doi.org/10.1145%2F3243734.3243782>.
35. Etemad M, Küpçü A, Papamanthou C, Evans D. Efficient Dynamic Searchable Encryption with Forward Privacy. *Proceedings on Privacy Enhancing Technologies*. 2018; 2018(1):5–20. <https://doi.org/10.1515/popets-2018-0002>
36. Sun SF, Steinfeld R, Lai S, Yuan X, Sakzad A, Liu J, et al. Practical Non-Interactive Searchable Encryption with Forward and Backward Privacy. In: *Proceedings 2021 Network and Distributed System Security Symposium*. Internet Society; 2021. Available from: <https://doi.org/10.14722%2Fndss.2021.24162>.
37. Wei Y, Lv S, Guo X, Liu Z, Huang Y, Li B. FSSE: Forward secure searchable encryption with keyed-block chains. *Information Sciences*. 2019; 500:113–126. <https://doi.org/10.1016/j.ins.2019.05.059>
38. Li J, Huang Y, Wei Y, Lv S, Liu Z, Dong C, et al. Searchable Symmetric Encryption with Forward Search Privacy. *IEEE Transactions on Dependable and Secure Computing*. 2021; 18(1):460–474. <https://doi.org/10.1109/TDSC.2019.2894411>