University of Kent

Kent Academic Repository

Abdelraheem, Mohamed Ahmed, Bhattacherjee, Sanjay, Henault, Théo and Majumder, Avishek (2026) *Conjunctive dynamic searchable symmetric encryption*. International Journal of Information Security, 25 (5). ISSN 1615-5270.

Downloaded from

https://kar.kent.ac.uk/116158/ The University of Kent's Academic Repository KAR

The version of record is available from

https://doi.org/10.1007/s10207-026-01322-1

This document version

Publisher pdf

DOI for this version

Licence for this version

CC BY (Attribution)

Additional information

Versions of research works

Versions of Record

If this version is the version of record, it is the same as the published version available on the publisher's web site. Cite as the published version.

Author Accepted Manuscripts

If this document is identified as the Author Accepted Manuscript it is the version after peer review but before type setting, copy editing or publisher branding. Cite as Surname, Initial. (Year) 'Title of article'. To be published in ***Title of Journal*** , Volume and issue numbers [peer-reviewed accepted version]. Available at: DOI or URL (Accessed: date).

Enquiries

If you have questions about this document contact ResearchSupport@kent.ac.uk. Please include the URL of the record in KAR. If you believe that your, or a third party's rights have been compromised through this document please see our Take Down policy (available from https://www.kent.ac.uk/guides/kar-the-kent-academic-repository#policies).

International Journal of Information Security (2026) 25:152
https://doi.org/10.1007/s10207-026-01322-1

REGULAR CONTRIBUTION

Conjunctive dynamic searchable symmetric encryption

Mohamed Ahmed Abdelraheem[1] · Sanjay Bhattacherjee[2] · Théo Henault[3] · Avishek Majumder[4]

Received: 21 December 2025 / Accepted: 13 August 2026
© The Author(s) 2026

Abstract

A practical *searchable symmetric encryption* (SSE) should allow users to privately store documents on untrusted servers with the abilities to (1) search with multiple keywords, and (2) add or delete documents dynamically. The oblivious cross tag (OXT) construction from Crypto'13 is the most efficient SSE scheme that allows conjunctive searches, but only for static databases. In NDSS'20, OXT was extended to ODXT supporting *dynamic* updates. However, ODXT is not *forward private*. In this work, we identify a commonly used setting for SSE where a document with its associated keywords can be dynamically added to or deleted from the database as a whole, but the keyword set is not modified in between. We call it the *non-modifiable document* (NMD) setting. In this NMD setting, we propose a generic framework for designing *conjunctive dynamic SSE* (CD-SSE), supporting conjunctive queries that allow dynamic updates while being forward and backward private at the same time. Our construction uses a generic dynamic single keyword SSE and the OXT scheme as modular black-boxes. Our scheme generalises ODXT. We provide new security definitions of forward and backward privacy for the new NMD setting. Our generic construction (and hence ODXT) achieves forward and BPUP backward privacy in the new setting, even when the underlying single keyword SSE scheme is only WBP backward private. We analyse the precise leakages of our scheme to the adversarial server in the NMD setting. We have instantiated our generic construction with three different single keyword schemes. Experiments show that our schemes are very efficient and practical. Our code is publicly available.

Keywords Searchable symmetric encryption · SSE · Conjunctive queries · Non-modifiable documents · Generic framework · Privacy

1 Introduction

With the advent of cheap and fault-tolerant third-party cloud service providers, storage of application data is commonly outsourced to untrusted servers. Data protection laws like GDPR require that online service provi-ders using third-party cloud services for storing data, must ensure user privacy "by design and by default"[1]. Hence the data stored on untrusted servers must be encrypted. One should also be able to search the data for use. This is commonly achieved through an *inverted index* that stores mappings of document identifiers id with keywords w as (id, w), to record the presence of keywords in the documents for search [39].

✉ Avishek Majumder
avishek.majumder1991@gmail.com

1 Entrust, Cambridge, UK

2 University of Kent, Canterbury, UK

3 Orange Cyberdefense, Rennes, France

4 Krea University, Sri City, India

Searchable Symmetric Encryption (SSE) allows storing the inverted index in an encrypted form (also known as *encrypted multi-map* or EMM [19]), so that user queries are hidden from the server, thus preserving user privacy. Each document in the database is assigned a unique identifier, and the task of SSE is to fetch those identifiers which point to documents containing the queried keyword [12, 15, 20, 21, 33]. SSEs are of two types – *static* and *dynamic*. In a static SSE, all documents are fixed at the beginning. Whereas, *dynamic SSE* (D-SSE) [18, 33] allows documents to be added or deleted dynamically to or from the encrypted inverted index.

Despite using SSE, user queries do leak some information to the server. Such leakage has led to several kinds of attacks on user privacy [14, 29, 36]. For example, in the famous "file-injection attack" [14, 55] the server may inject only a few documents (through the client) into the database, to be able to then recover a large fraction of the keywords searched by the client. This is possible due to the leakage from update queries that can be linked to previous search queries. The leakage of an SSE scheme determines its type of security.

Published online: 03 September 2026

Two key security notions for D-SSE were proposed in [15] – *forward privacy* and *backward privacy*. Intuitively, forward private SSE schemes ensure that the server cannot link an update operation for (id, w) with previous search operations performed on the inverted index with w or containing id in its result. This protects queries made to the inverted index with newly injected files by the adversary [55]. Backward privacy ensures that after a pair (id, w) has been deleted, the identifier id will not be leaked in future searches for w.

Most SSE schemes to date [12, 13, 15, 18, 20, 21, 26, 33, 46, 48, 49] support single keyword searches per query. Complex searches may involve multiple keywords combined via conjunctions or general boolean expressions. A naïve approach to achieve this is by performing individual single keyword searches and then computing set operations on the results. However, this method is inefficient, increases computation and communication on the client and server. This also reveals the number of matching documents for each keyword to the server. So the challenge of constructing SSE schemes allowing conjunctive or more general boolean queries is to optimise between (1) improving efficiency (compared to the naïve solutions), and (2) minimising the leakage to the server.

1.1 A common practical setting for searchable encryption

Previous (single and conjunctive) dynamic SSE schemes [12, 13, 15, 20, 26, 33, 41, 46, 48, 54, 56–58] allow users to add/delete encrypted document-keyword pairs (id, w) to/from the inverted index dynamically at any point of time. After a dynamic scheme has been initialised, adding a document to the inverted index means adding some pairs (id, w_i) where id is the document identifier and w_i are the keywords in the document. Additionally, more keywords may further be added corresponding to that document id already in the inverted index, or existing keywords may be deleted from it. When all keywords have been deleted for a document id, the document is considered to be deleted from the inverted index.

We define *"modifiability"* of a document as: *the ability to add/delete keywords to/from a document at any point.* We call a document "*non-modifiable*" if its *keyword set is fixed*. The implications on the inverted index are:

- While adding a *non-modifiable document* (NMD) to the database, all its identifier-keyword pairs (id, w) are added to the inverted index at the same time.
- While deleting an NMD from the database, all its identifier-keyword pairs (id, w) are deleted from the inverted index at the same time.
- The phrase "at the same time" in the above two statements implies that there is no search query involving the document id or its keywords w in between the addition queries or in between the deletion queries.

(Note that there may be other portions of the actual document – except its keyword set – that may change between its addition and deletion, but such changes do not have any effect on the inverted index used for searches.) In many real-world applications, the requirement is to add/delete NMDs dynamically (at any point after initialisation) to/from the database. It is important to note that in this setting *modifiability can be achieved through the "delete-then-add" technique: by deleting the document and then adding the updated version with a different identifier* as is common in SSE literature [13, 15, 27]. Below are some examples of applications handling non-modifiable documents.

1. *Biometric Data* like fingerprint, voice, iris scan, facial image, etc. are typically non-modifiable. It is crucial to secure them for privacy and data protection [34], especially in large databases like Aadhaar [4, 6], for voice data collected in bulk [37], etc.
2. *Media Subscription Services* like Netflix, Spotify, etc., the documents are the DRM-protected media files [5, 10, 38] that once uploaded, do not change.
3. *Digital Libraries and Archives* like the IEEE Xplore Digital Library store NMDs encrypted for access control [51]. They contain sensitive information requiring encryption [42, 43].
4. *Legal Documents* are non-modifiable, containing sensitive information (personal details, financial arrangements, strategic plans, etc.) and should hence be encryp ted [7].
5. *Medical Records* records are NMDs filled with personally identifiable information (PII), which must be protected under privacy laws,[1] especially for access control and data sharing for collaborative research [11, 44].
6. *Financial Records* contain sensitive information abo-ut a company's financial status and strategy. Such historical data is non-modifiable while searchability is essential for auditing and making financial decisions. Encryption is necessary to prevent unauthorised access that can lead to financial loss, reputational damage, or legal issues.

In each of these scenarios, protecting the privacy of the user "by design and by default" is a key responsibility of the service provider according to regulations like GDPR. So while the service providers may use third-party (cloud) storage services for the inverted indices, they must be encrypted to ensure user privacy. In applications like the above, modifiability is either not required (like for biometric data, digital

[1] Like HIPAA in the United States: https://www.ncbi.nlm.nih.gov/books/NBK500019/

media archives) or should be strictly prohibited (like for legal documents, medical and financial records). The existing definitions of SSE schemes and their security models do not consider this subtlety. That leaves scope for new definitions and scheme designs. As we do away with *modifiability*, several computations and storage requirements from previous SSE schemes are rendered redundant without compromising on the security. Hence, we arrive at new efficient schemes that would otherwise be insecure in the general SSE setting for modifiable documents.

1.2 Our contributions

There are several SSE schemes that support conjunctive queries (called C-SSE schemes) – [3, 16, 25, 32, 35, 40, 41, 53, 54, 56, 58]. The most efficient of these is the oblivious cross-tag (OXT) system from Crypto'13 [16], with search complexity $\mathcal{O}(n \cdot |\mathsf{DB}(w_1)|)$, where n is the number of keywords in the conjunction, w_1 is the least frequent keyword in the conjunction and $\mathsf{DB}(w_1)$ is the set of ids of documents in the database containing w_1. However, OXT is a static scheme.

Our focus is on dynamic SSE (D-SSE) schemes supporting conjunctive queries. We call such a scheme *conjunctive dynamic*-SSE or CD-SSE. The *oblivious dynamic cross tag* (ODXT) [41] scheme, proposed in 2020 as the first CD-SSE scheme with claimed forward and backward privacy, until the attack in [56] showed that the scheme is not forward private. A few other CD-SSE schemes [54, 56, 58] have followed, achieving forward privacy, and different flavours of backward privacy with efficiency trade-offs. However, these schemes are far from achieving the efficiency of the OXT scheme. Table 1 contains a detailed comparison between these CD-SSE schemes. These attempts indicate that constructing an efficient and secure (forward and backward private) CD-SSE is quite challenging. Our goal is the following.

> *Construct an adaptively secure forward and backward private CD-SSE scheme with computational efficiency comparable with* OXT *under standard and well-studied security definitions.*

We have identified the common SSE setting for NMDs described in Sect. 1.1, and achi-eved our goal in this new setting. We have extended and amended the definitions of forward and backward privacy for CD-SSE for the new NMD setting. We have constructed a generic CD-SSE for the NMD setting, that combines a generic forward and backward private D-SSE and OXT as components in a modular black-box fashion. Our generic CD-SSE is a generalisation of ODXT [41]. We have proved that our generic construction is forward and BPUP backward private in the new NMD setting, even when the component D-SSE is only WBP backward private. We have introduced a new method for analysing the

Table 1 Comparison of our construction with previous schemes

Scheme	Search Computation	Communication	RT	Update Add	Edit	Delete	Storage Client	Server	FP	BP	KPRP
VBTree [53]	$\mathcal{O}(u\|\mathsf{DB}(w_1)\|\log\|D\|)$	$\mathcal{O}(n+\|\mathsf{DB}(\omega)\|)$	1	$\mathcal{O}(W_d)$	$\mathcal{O}(v)$	$\mathcal{O}(1)$	$\mathcal{O}(\|W\|\log\|D\|)$	$\mathcal{O}(\|W\|h)$	Yes	No	No
BDXT [41]	$\mathcal{O}(u_1+n\|\mathsf{DB}(w_1)\|)$	$\mathcal{O}(u_1+n\|\mathsf{DB}(w_1)\|)$	2	$\mathcal{O}(1)$	-	$\mathcal{O}(1)$	$\mathcal{O}(\|W\|\log\|D\|)$	$\mathcal{O}(N)$	No	Type-II	No
ODXT [41]	$\mathcal{O}(nu_1)$	$\mathcal{O}(nu_1)$	1	$\mathcal{O}(1)$	-	$\mathcal{O}(1)$	$\mathcal{O}(\|W\|\log\|D\|)$	$\mathcal{O}(N)$	No	Type-II	No
FBSSE-CQ [58]	$\mathcal{O}(u\|D\|)$	$\mathcal{O}(n+\|D\|)$	1	$\mathcal{O}(\|D\|)$	$\mathcal{O}(\|D\|)$	$\mathcal{O}(\|D\|)$	$\mathcal{O}(\|W\|\log\|D\|+\lambda)$	$\mathcal{O}(N\|D\|)$	Yes	Type-II	Yes
HDXT [54]	$\mathcal{O}(u_1+n\|\mathsf{DB}(w_1)\|)$	$\mathcal{O}(u_1+n\|\mathsf{DB}(w_1)\|)$	2	$\mathcal{O}(\|W\|/W_d)$	$\mathcal{O}(\|D\|)$	$\mathcal{O}(1)$	$\mathcal{O}(\|W\|\log\|D\|+\lambda)$	$\mathcal{O}(\|W\|\|D\|)$	Yes	Type-II	Yes
Our construction (Sec 4)	$\mathcal{O}(nu_1)$	$\mathcal{O}(nu_1)$	1	$\mathcal{O}(1)$	$\mathcal{O}(W_d)$	$\mathcal{O}(1)$	$\mathcal{O}(\|W\|\log\|D\|)$	$\mathcal{O}(N)$	NMD	Type-II	No

Notation. W: set of keywords. D: Set of documents. W_d: Number of keywords in the document to be updated. N: Number of identifier-keyword pairs in the database. n: Number of keywords in a conjunction. u_i: Updates performed for keyword w_i. $u = \sum_{i=1}^{n} u_i$: Total updates for a conjunction of n keywords. v: Keywords involved in the edit operation. h: Average number of documents matched by any two keywords. $\mathsf{DB}(\omega)$: Set of matching document identifiers for query $\omega = w_1 \wedge \cdots \wedge w_n$, where $n \geq 1$. RT: Round-Trip. FP: Forward Private. BP: Backward Private. KPRP: Keyword-Pair Result Pattern Hiding. NMD: Non-Modifiable Documents. The asymptotic efficiencies are all in the NMD setting. The security levels for the previous schemes are in the general setting, while ours is in the NMD setting

actual leakage of any SSE through all possible combinations of query sequences.

The modular nature of our construction allows agile efficiency improvements. As more efficient D-SSEs are developed in the future, one may only substitute the D-SSE in our construction and get a more efficient CD-SSE scheme in the NMD setting. This will give a more efficient CD-SSE for the NMD setting, rather than developing a new one from the scratch.

Even though our scheme has been proved secure in our new NMD setting, for systems where modifiability is essential, Π can be used securely with some additional overhead costs, using the commonly used *"delete-then-add"* technique [13, 15, 27].

In Table 1, we compare our generic construction with VBTree [53], BDXT [41], ODXT [41], FBSSE-CQ [58], and HDXT [54]. Our generic CD-SSE instantiated with the D-SSE of Mitra [26], is essentially the ODXT scheme [41], in the NMD setting. We note that the asymptotic efficiencies of the other schemes in Table 1 are in the general setting allowing modification of documents. However, it is easy to see that *these schemes will have the same asymptotic performances in the NMD setting, as in the general setting*. This is because, when these schemes are used in the NMD setting, there will be no queries for modifying documents, while the insert and delete queries will continue functioning as in the general setting. Whether the security and/or efficiency of these schemes can be improved in the NMD setting, requires rigorous investigations beyond the scope of this work.

Our CD-SSE has a search complexity of $\mathcal{O}(nu_1)$, where u_1 is the number of updates related to the least frequent keyword (say w_1) in the conjunction. Table 1 shows that *our construction is more efficient than all previous secure CD-SSE schemes in the NMD setting*.

We have implemented instantiations of our generic construction with three D-SSEs – FAST [46], Mitra [26] and Π_{BP} [20] – and experimentally compared their performances with the Enron Email database[24], showing that all three schemes are practical and scalable. Our code is available at [8].

1.3 Related work

SSE was first introduced in [45], while the formal notions of security were first provided in [21]. However, the scheme in [21] was only *static*. The first truly dynamic solution for SSE with search complexity $\mathcal{O}(\mathsf{DB}(w))$ was proposed in [33]. Since then, several D-SSE schemes have been proposed. Attacks on SSE have typically exploited the inherent leakages of the scheme without active manipulation of queries by the adversary [14, 29, 36]. In [55], the adversary was allowed to inject some files with its choice of keywords into a dynamic database to demonstrate a novel attack.

To mitigate such attacks, *forward privacy* [47] has become essential for D-SSE schemes. The first formal definition of forward privacy and a scheme that achieves it was introduced in [12]. The first formal definition of backward privacy and relevant constructions were provided in [13]. Three types of backward privacy were defined [13] – BPIP/Type I, BPUP/Type II and WBP/Type III in increasing order of leakage to the server (decreasing order of security). A bitmap index with homomorphic addition was used in [57] to achieve Type I^- security whose leakage is even lesser than a Type I secure scheme. In [23], these works were extended to make the client robust, which can handle duplicate add or delete queries and a delete query issued for non-existent entries. A more efficient construction of a Type III backward private SSE scheme was proposed in [48] that uses symmetric puncturable encryption. The definition of Type III backward privacy of [13] was in a restricted setting. In [20], the restriction was lifted to provide a backward privacy definition in a more general setting. Since then, several forward and backward private SSE schemes [17, 20, 26, 57] have been proposed, improving their security and efficiency.

While all the above works only support single keyword queries, there are several constructions [9, 16, 25, 31, 35, 40] that support more complicated queries for static databases. There have been a few constructions [28, 32, 52, 53] that are dynamic and support conjunctive queries, but are not backward private. So, a forward and backward private CD-SSE scheme has been a long-standing open problem until the attempt of [41]. As mentioned earlier, the forward privacy of [41] was attacked in [56]. A fix was proposed [56] along with two new notions of backward privacy for C-SSE – Type O and Type O^-, without any comparison with existing notions of backward privacy. The communication and computation costs are high as well. There have been a few other CD-SSE schemes which are both forward and backward private [54, 56, 58]. The construction of [58] is secure, but has high search and storage overheads. The first forward and backward private CD-SSE with sub-linear search was proposed in [54]. However, their scheme requires two rounds of communication between the client and the server for the search operation, and the storage and update costs are high as well as shown in Table 1. So our new setting and the generic forward and backward private CD-SSE scheme is a significant leap in terms of functionality-efficiency-security tradeoffs.

Recently, a novel chain structure was used in [50] to construct a CD-SSE, which reduced the client's communication cost compared to ODXT [41], while increasing the client-side storage and the server search time. Also, their scheme [50] performed worse than ODXT [41] in all experimental search comparisons. Moreover, their scheme only allows to add or delete an entire document as a whole. Even though they do not point out this implicit assumption in their paper, their

scheme is in fact the first to be proved secure in the non-modifiable setting. We prove that ODXT and its generalisation is also secure in the non-modifiable setting. We provide further details and comparison with [50] in Sect. 7.

2 Preliminaries

All symmetric key algorithms use a λ-bit key as input from the set $\{0,1\}^\lambda$. Let $\mathcal{X}$ be any set, $x \xleftarrow{\$} \mathcal{X}$ means x is sampled randomly from the set $\mathcal{X}$. For any two non-negative integers a, b where $a < b$, $[a, b]$ denotes the set $\{a, a+1, \ldots, b\}$; whereas, for $a > 0$, $[a] = \{1, 2, \ldots, a\}$. A function $f : \mathbb{N} \to \mathbb{R}$ is negligible (denoted by $\mathsf{negl}(\cdot)$) iff for all $c > 0$, there exists a $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$, $f(n) < n^{-c}$. For a protocol P between a client and a server, their respective inputs are separated by a semicolon $\mathsf{P}(\ldots;\ldots)$.

2.1 Cryptographic assumptions

Pseudo- Random Function (PRF). A pseudo-random function $F : \mathcal{M} \times \mathcal{K} \to \mathcal{R}$ is a keyed two input function associated with $(\mathcal{M}, \mathcal{K}, \mathcal{R})$, where $\mathcal{K}$ is the key space, $\mathcal{M}$, $\mathcal{R}$ are the input and output space is defined as follows.

Definition 1 (PRF) Let $F : \{0,1\}^n \times \{0,1\}^\lambda \to \{0,1\}^m$ be an efficiently computable keyed function. F is said to be pseudo-random if for all PPT adversary $\mathcal{A}$ the following holds.

$$\mathsf{Adv}^{\mathsf{prf}}_{F,\mathcal{A}} = \left|\Pr[\mathcal{A}^{F_k(\cdot)}(1^\lambda)] - \Pr[\mathcal{A}^{f(\cdot)}(1^\lambda)]\right| \leq \mathsf{negl}(\lambda),$$

where probabilities are taken over random choices of $k \xleftarrow{\$} \{0,1\}^\lambda$ and f is chosen uniformly at random from the set of functions mapping n bit to m bits.

Decisional Diffie- Hellman assumption (ddh). Let $\mathcal{G}$ be a PPT algorithm which on input λ outputs a description of a cyclic group G of prime order p (st. $|p| = \mathsf{poly}(\lambda)$), and a generator $g \in G$. Then the *decisional Diffie-Hellman assumption* is the following.

Definition 2 (ddh-Assumption) We say that the ddh problem is hard relative to $\mathcal{G}$ if for all PPT adversaries $\mathcal{A}$,

$$\begin{aligned}&\mathsf{Adv}^{\mathsf{ddh}}_{G,\mathcal{A}}(\lambda)\\ &= \left|\Pr\left[\mathcal{A}\left(g, g^a, g^b, g^{ab}\right) = 1\right] - \Pr\left[\mathcal{A}\left(g, g^a, g^b, g^c\right) = 1\right]\right|\\ &\leq \mathsf{negl}(\lambda)\end{aligned}$$

where the probabilities are take over the randomness of $\mathcal{A}$ and random choices of $a, b, c \in \mathbb{Z}_p^*$.

Extended Decisional Diffie- Hellman Assumption (eddh-Assumption) Let $\mathcal{G}$ be a PPT algorithm which on input λ outputs a description of a cyclic group G of prime order p (st. $|p| = \mathsf{poly}(\lambda)$), and a generator $g \in G$. Then the *extended decisional Diffie-Hellman assumption* is the following.

Definition 3 (eddh-Assumption [41]) We say that the eddh problem is hard relative to $\mathcal{G}$ if for all PPT adversaries $\mathcal{A}$,

$$\begin{aligned}\mathsf{Adv}^{\mathsf{eddh}}_{\mathcal{A}} &= \left|\Pr\left[\mathcal{A}(g, M) = 1\right] - \Pr\left[\mathcal{A}\left(g, \widehat{M}\right) = 1\right]\right|\\ &\leq \mathsf{negl}(\lambda),\end{aligned}$$

where

$$\begin{bmatrix} g^{\alpha_1\beta_1} & g^{\alpha_1\beta_2} & \ldots & g^{\alpha_1\beta_n} \\ g^{\alpha_2\beta_1} & g^{\alpha_2\beta_2} & \ldots & g^{\alpha_2\beta_n} \\ \vdots & \vdots & \ddots & \vdots \\ g^{\alpha_m\beta_1} & g^{\alpha_m\beta_2} & \ldots & g^{\alpha_m\beta_n} \end{bmatrix}, \begin{bmatrix} g^{\gamma_{1,1}} & g^{\gamma_{1,2}} & \ldots & g^{\gamma_{1,n}} \\ g^{\gamma_{2,1}} & g^{\gamma_{2,2}} & \ldots & g^{\gamma_{2,1}} \\ \vdots & \vdots & \ddots & \vdots \\ g^{\gamma_{m,1}} & g^{\gamma_{m,2}} & \ldots & g^{\gamma_{m,n}} \end{bmatrix}$$

the probabilities are take over the randomness of $\mathcal{A}$ and for all $i \in [m]$, $j \in [n]$, $\alpha_i, \beta_j, \gamma_{i,j} \xleftarrow{\$} \mathbb{Z}_p^*$.

2.2 Conjunctive search

We assume a database to be a collection of documents and a set of keywords associated with those documents. A keyword w is an element of the set $W \subseteq \{0,1\}^*$. An SSE scheme may allow searches using multiple keywords and their conjunctions. Let $L = \{w_1, w_2, \ldots, w_n\} \subseteq W$ be a subset of keywords and $\omega(L) = w_1 \wedge w_2 \wedge \cdots \wedge w_n$ be a conjunction of the elements of L. For the conjunction $\omega(L)$, we assume the existence of a least frequent keyword, w.l.o.g say w_1. We denote, w_1 as the *s-term* and the set of *x-terms* as $\widetilde{L} = L \setminus \{w_1\}$. For convenience we denote $\omega(L)$ as ω.

A document to be stored using an SSE scheme has a unique identifier from the set $\mathcal{ID} \subseteq \{0,1\}^\lambda$, where λ is the security parameter. Also, a document is a collection of a set of keywords $\mathsf{w} \subseteq W$, and the number of keywords in the set $|\mathsf{w}|$ is W_d. In the formal definition of an SSE scheme, a document is completely specified by a pair $(\mathsf{id}, \mathsf{w})$ where $\mathsf{id} \in \mathcal{ID}$ is an identifier for the document and $\mathsf{w} \subseteq W$ is the set of all keywords in it. Similarly, let $\mathsf{DB}(w)$ denote the set of all ids for a keyword w. Formally, $\mathsf{DB}(w) = \{\mathsf{id}_i : w \in \mathsf{w}_i\}$, where w_i is the set of keywords in id_i. Now for a conjunctive query $\omega = w_1 \wedge w_2 \wedge \cdots \wedge w_n$, the set of identifiers matching the search query ω is denoted by $\mathsf{DB}(\omega)$. Formally, for a conjunctive query ω, we define $\mathsf{DB}(\omega) = \{\mathsf{id}_j : w_i \in \mathsf{w}_j\}_{i \in [n]}$.

In a CD-SSE Π for non-modifiable documents, we assume that all the keyword-identifier pairs corresponding to a single id are added to or deleted from the inverted index at once. Re-insertions of the same (id, w) pair after deletion are not allowed. However, a document $(\mathsf{id}, \mathsf{w})$ can be deleted

and then added back with a different id' and possibly with a modified keyword set w' as is common in the literature [13, 14, 33].

2.3 General dynamic searchable symmetric encryption

Definition 4 A *general dynamic searchable symmetric encryption scheme* $\Lambda = (\mathsf{Stp}, \mathsf{Udt}, \mathsf{Srch})$ is a collection of three protocols between the client and the server, each comprised of two algorithms.

$\mathsf{Stp}\left(1^\lambda, \mathsf{DB}\right)$

- $(k, \sigma, \mathsf{EDB}) \leftarrow \mathsf{Stp}_{\mathcal{C}}\left(1^\lambda, \mathsf{DB}\right)$: Inputs: security parameter 1^λ and an empty database DB. Outputs: secret key k, the client's initial state σ and the encrypted inverted index EDB generated from of DB.
- $\perp \leftarrow \mathsf{Stp}_{\mathcal{S}}(\mathsf{EDB})$: Takes as input the encrypted inverted index EDB and stores it.

$\mathsf{Udt}\,(k, \sigma, \mathsf{op}, (\mathsf{id}, w); \mathsf{utk}, \mathsf{EDB})$

- $(\sigma, \mathsf{utk}) \leftarrow \mathsf{Udt}_{\mathcal{C}}\,(k, \sigma, \mathsf{op}, (\mathsf{id}, w))$: Inputs: the secret key k, the current state σ of the client $\mathcal{C}$, the identifier id for a document, a keyword w, and a query $\mathsf{op} \in \{\mathsf{add}, \mathsf{del}\}$. Outputs: an update token utk and the updated state σ of the client.
- $(\mathsf{EDB}) \leftarrow \mathsf{Udt}_{\mathcal{S}}\,(\mathsf{utk}, \mathsf{EDB})$: Inputs: an update token utk and the current encrypted inverted index EDB. Output: an updated encrypted inverted index EDB.

$\mathsf{Srch}\,(k, \sigma, q; \mathsf{stk}, \mathsf{EDB})$

- $(\sigma, \mathsf{stk}) \leftarrow \mathsf{Srch}_{\mathcal{C}}\,(k, \sigma, q)$: Inputs: the secret key k, a search query q, and the current state σ of $\mathcal{C}$. Outputs: a search token stk and the updated state σ.
- $(\mathsf{res}, \mathsf{EDB}) \leftarrow \mathsf{Srch}_{\mathcal{S}}\,(\mathsf{stk}, \mathsf{EDB})$: Inputs: the search token stk and the encrypted inverted index EDB. Outputs: the search result res and the encrypted inverted index EDB. The search result is sent to the client.

Single- Keyword and Conjunctive. Definition 4 is for a general dynamic SSE Λ. Herein, $q = w$ in the input of $\mathsf{Srch}_{\mathcal{C}}$ gives a D-SSE Σ that only supports single keyword searches and not conjunctive queries. A CD-SSE Π has $q = \omega$.

Correctness. The correctness of Λ ensures that, after $(\ell - 1)$ operations, when the encrypted inverted index of the server is $\mathsf{EDB}_{\ell-1}$ and the client's state is $\sigma_{\ell-1}$, then for a search query q_ℓ, the client issues a search token stk_ℓ and the result $\mathsf{DB}(\omega_\ell)$ is retrieved from res_ℓ, except with negligible probability. Here, $(\mathsf{res}_\ell, \mathsf{EDB}_\ell) \leftarrow \mathsf{Srch}_{\mathcal{S}}\,(\mathsf{stk}_\ell, \mathsf{EDB}_{\ell-1})$ and $(\sigma_\ell, \mathsf{stk}_\ell) \leftarrow \mathsf{Srch}_{\mathcal{C}}\,(k, \sigma_{\ell-1}, \omega_\ell)$.

2.4 Security

The security of Λ is determined by the leakage function $\mathcal{L}_\Lambda$. The stateful leakage function $\mathcal{L}_\Lambda = \left(\mathcal{L}_{\Lambda,\mathsf{Stp}}, \mathcal{L}_{\Lambda,\mathsf{Srch}}, \mathcal{L}_{\Lambda,\mathsf{Udt}}\right)$ denotes the information that the adversary learns from the setup process and each execution of the search and update protocols. The leakage functions $\mathcal{L}_{\Lambda,\mathsf{Stp}}$ denotes the leakage due to $\mathsf{Stp}_{\mathcal{C}}\left(1^\lambda, \mathsf{DB}_0\right)$. $\mathcal{L}_{\Lambda,\mathsf{Srch}}$ denotes the leakage due to all the searches. $\mathcal{L}_{\Lambda,\mathsf{Udt}}$ denotes the leakage due to all updates.

2.4.1 Security game

The security of Λ is defined in a real-ideal simulation paradigm [15, 21, 33]. A leakage function $\mathcal{L}_\Lambda$ is defined to capture all the leakages (due to the setup, update and search protocols). The simulator is given access to the leakage function to simulate the real world execution of the scheme. Thus, the security definition is parameterised by the leakage function. The adversary we consider here is semi-honest, but curious.

Definition 5 (Adaptive Security) Let $\Lambda = (\mathsf{Stp}, \mathsf{Udt}, \mathsf{Srch})$ be an SSE scheme and $\mathcal{L}_\Lambda$ be its stateful leakage function. For any PPT adversary $\mathcal{A}$ and simulator $\mathcal{S}$, Λ is $\mathcal{L}_\Lambda$-adaptively secure if the advantage $\mathsf{Adv}^{\Lambda,\mathsf{SSE}}_{\mathcal{A},\mathcal{S},\mathcal{L}_\Lambda}$ of $\mathcal{A}$ is negligible:

$$\begin{aligned}&\mathsf{Adv}^{\Lambda,\mathsf{SSE}}_{\mathcal{A},\mathcal{S},\mathcal{L}_\Lambda}\\&= \left|\Pr\left[\mathrm{SSEREAL}^{\Lambda}_{\mathcal{A}}(\lambda) = 1\right] - \Pr\left[\mathrm{SSEIDEAL}^{\Lambda}_{\mathcal{A},\mathcal{S}}(\lambda) = 1\right]\right|\\&\leq \mathsf{negl}(\lambda).\end{aligned}$$

Here, the real and ideal worlds work as follows.

$\mathbf{SSEREAL}^{\Lambda}_{\mathcal{A}}(\lambda)$: The adversary $\mathcal{A}\left(1^\lambda\right)$ chooses a DB and the challenger runs $(k, \sigma, \mathsf{EDB}) \leftarrow \mathsf{Stp}_{\mathcal{C}}\left(1^\lambda, \mathsf{DB}\right)$ and returns EDB to $\mathcal{A}$. Then for subsequent search or update queries, the challenger runs $(\sigma, \mathsf{stk}) \leftarrow \mathsf{Srch}_{\mathcal{C}}\,(k, \sigma, q)$ or $(\sigma, \mathsf{utk}) \leftarrow \mathsf{Udt}_{\mathcal{C}}\,(k, \sigma, \mathsf{op}, (\mathsf{id}, w))$ respectively and provides the adversary with stk or utk. The adversary $\mathcal{A}$ stops by outputting a bit, which is the output of the experiment.

$\mathbf{SSEIDEAL}^{\Lambda}_{\mathcal{A},\mathcal{S}}(\lambda)$: The adversary $\mathcal{A}\left(1^\lambda\right)$ chooses a DB and the challenger runs $\mathsf{EDB} \leftarrow \mathcal{S}\left(\mathcal{L}_{\Lambda,\mathsf{Stp}}\right)$ and returns EDB to $\mathcal{A}$. Then for subsequent search or update queries, the challenger runs $\mathsf{stk} \leftarrow \mathcal{S}\left(\mathcal{L}_{\Lambda,\mathsf{Srch}}\right)$ or $\mathsf{utk} \leftarrow \mathcal{S}\left(\mathcal{L}_{\Lambda,\mathsf{Udt}}\right)$ respectively and provides the adversary with stk or utk. The adversary $\mathcal{A}$ stops by outputting a bit, which is the output of the experiment.

2.5 Forward and backward privacy of D-SSE

The two most essential security requirements (other than adaptive security) for a D-SSE Σ (allowing only single-keyword searches) are *forward and backward privacy*. Depending upon the information they leak during search and update operations, various flavours of backward privacy have been defined for Σ in decreasing order of security in the literature [12, 13]: (1) *backward privacy with insertion pattern* (BPIP), (2) *backward privacy with update pattern* (BPUP), and (3) *weak backward privacy* (WBP). As argued in [20], the definition of backward privacy in [13] missed some leakage components. Our leakage definitions include those from [13, 20]. A new notion of *backward privacy with link pattern* (BPLP) was also proposed in [20], which for our restricted setting of non-modifiable documents, boils down to WBP.

Leakages To define the common leakage functions, we define a query sequence QS as an ℓ-tuple, where ℓ is the total number of queries. So $\mathsf{QS} = \{(Q_1, \ldots, Q_i, \ldots, Q_\ell)\}$ is the list of all queries made by the client. Every query $Q_i = (u_i, \mathsf{op}_i, \mathsf{in}_i)$ is a three tuple, where u_i is the time-stamp of the query, $\mathsf{op}_i \in \{\mathsf{add}, \mathsf{srch}, \mathsf{del}\}$ is the operation of the query, and in_i is the input of the query depending upon the operation op_i. If $\mathsf{op}_i = \mathsf{srch}$ is a search query, then in_i is w or ω; if $\mathsf{op}_i \in \{\mathsf{add}, \mathsf{del}\}$ is an update query, then $\mathsf{in}_i = (\mathsf{id}_i, w_i)$. Below are some common leakage functions for D-SSE.

$\mathsf{sp}(w)$: is the set of timestamps of all search queries, also called the *search pattern*.

$$\mathsf{sp}(w) = \{u : (u, \mathsf{srch}, w) \in \mathsf{QS}\}.$$

$\mathsf{TimeDB}(w)$: is the timestamped list of documents currently matching w.

$$\mathsf{TimeDB}(w) = \{(u, \mathsf{id}) : ((u, \mathsf{add}, (\mathsf{id}, w)) \in \mathsf{QS}) \wedge \left(\forall u' > u, \left(u', \mathsf{del}, (\mathsf{id}, w) \notin \mathsf{QS}\right)\right)\}.$$

$\mathsf{Udts}(w)$: is the set of timestamps of all update queries for a given w.

$$\mathsf{Udts}(w) = \{u : (u, \mathsf{op}, (\mathsf{id}, w)) \in \mathsf{QS}, \text{ and } \mathsf{op} \in \{\mathsf{add}, \mathsf{del}\}\}.$$

$\mathsf{delHist}(w)$: is the set of all pairs of timestamps of addition and deletion for a given w.

$$\mathsf{delHist}(w) = \left\{\left(u^{\mathsf{add}}, u^{\mathsf{del}}\right) : \left(\left(u^{\mathsf{add}}, \mathsf{add}, (\mathsf{id}, w)\right) \in \mathsf{QS}\right) \wedge \left(\left(u^{\mathsf{del}}, \mathsf{del}, (\mathsf{id}, w)\right) \in \mathsf{QS}\right)\right\}.$$

With the above definitions in place, we recall the definition of forward and backward privacy of a D-SSE Σ as in [13, 20].

Definition 6 (Forward Privacy of D-SSE) An $\mathcal{L}_\Sigma$-adaptive secure D-SSE scheme Σ is forward private if its leakage due to the Udt protocol can be written as

$$\mathcal{L}^{\mathsf{FP}}_{\Sigma,\mathsf{Udt}}(\mathsf{op}, \mathsf{in}) = \widetilde{\mathcal{L}}_u(\mathsf{op}, \{\mathsf{id}, \mu_i\})$$

where the set $\{(\mathsf{id}_i, \mu_i)\}$ captures all updated documents as the number of keywords μ_i modified in document id_i, and $\widetilde{\mathcal{L}}_u$ is stateless.

Definition 7 (Backward Privacy of D-SSE) An $\mathcal{L}_\Sigma$-adaptive secure D-SSE scheme Σ is backward private as per the notions BPIP, BPUP and WBP if its leakage due to the Udt and Srch protocols can be written as

$$\begin{aligned}
\mathcal{L}^{\mathsf{BPIP}}_{\Sigma,\mathsf{Udt}}(\mathsf{op}, w, \mathsf{id}) &= \widetilde{\mathcal{L}}_u(\mathsf{op}) \\
\mathcal{L}^{\mathsf{BPIP}}_{\Sigma,\mathsf{Srch}}(w) &= \widetilde{\mathcal{L}}_s(\mathsf{sp}(w), \mathsf{TimeDB}(w), |\mathsf{Udts}(w)|), \\
\mathcal{L}^{\mathsf{BPUP}}_{\Sigma,\mathsf{Udt}}(\mathsf{op}, w, \mathsf{id}) &= \widetilde{\mathcal{L}}_u(\mathsf{op}, w) \\
\mathcal{L}^{\mathsf{BPUP}}_{\Sigma,\mathsf{Srch}}(w) &= \widetilde{\mathcal{L}}_s(\mathsf{sp}(w), \mathsf{TimeDB}(w), \mathsf{Udts}(w)), \\
\mathcal{L}^{\mathsf{WBP}}_{\Sigma,\mathsf{Udt}}(\mathsf{op}, w, \mathsf{id}) &= \widetilde{\mathcal{L}}_u(\mathsf{op}, w) \\
\mathcal{L}^{\mathsf{WBP}}_{\Sigma,\mathsf{Srch}}(w) &= \widetilde{\mathcal{L}}_s(\mathsf{sp}(w), \mathsf{TimeDB}(w), \mathsf{Udts}(w), \mathsf{delHist}(w)),
\end{aligned}$$

where, $\widetilde{\mathcal{L}}_u$ and $\widetilde{\mathcal{L}}_s$ are stateless.

3 Forward and backward privacy in CD-SSE for non-modifiable documents

In this section, we provide security definitions for CD-SSE schemes with non-modifiable documents. In [41], the definitions of some common leakage functions, forward privacy and backward privacy for a CD-SSE were developed by extending the definitions of a D-SSE Σ. However, the extension in [41] was for BPUP backward privacy only. We extend the definition for all three notions of backward privacy – BPIP, BPUP and WBP [13]. The definition of Type III backward privacy [13, 41] was generalised in [20], that is also relevant in the context of CD-SSE and hence is incorporated in our definition.

Consider the query sequence $\mathsf{QS} = \{Q_1, \ldots, Q_\ell\}$, a conjunctive query $\omega = w_1 \wedge \cdots \wedge w_n$ and its set of included keywords $L = \{w_1, \ldots, w_n\}$. We define the following sets of leakages for ω as follows.

$\mathsf{sp}(\omega)$: for a pair of keywords w_i and w_j in the conjunction ω, the search pattern is defined as $\mathsf{sp}(w_i, w_j) = \{u : (u, \mathsf{srch}, \omega) \in \mathsf{QS}; w_i, w_j \in L\}$. Hence,

$$\mathsf{sp}(\omega) = \mathsf{sp}(w_1) \cup \left(\cup_{i=2}^{n} \mathsf{sp}(w_1, w_i)\right).$$

$\mathsf{TimeDB}(\omega)$: is the set of all timestamped ids containing keywords in L that have been added to but are yet to be deleted from the database. As in [41],

$$\mathsf{TimeDB}(\omega) = \Big\{\big(\{u_i\}_{i\in[n]}, \mathsf{id}\big) : \forall w_i \in L, (u_i, \mathsf{add}, (\mathsf{id}, w_i)) \in \mathsf{QS} \wedge \big(\forall u' > u_i, \big(u', \mathsf{del}, (\mathsf{id}, w_i)\big) \notin \mathsf{QS}\big)\Big\}.$$

$\underline{\mathsf{Udts}(\omega)}$: For a keyword pair (w_i, w_j) in the conjunction ω, we define the set of pairs of timestamps of update operations $\mathsf{op} \in \{\mathsf{add}, \mathsf{del}\}$ on the database as

$$\mathsf{Udts}(w_i, w_j) = \big\{(u, u') : ((u, \mathsf{op}, (\mathsf{id}, w_i)) \in \mathsf{QS}) \wedge \big((u', \mathsf{op}, (\mathsf{id}, w_j)) \in \mathsf{QS} \wedge \mathsf{op} \in \{\mathsf{add}, \mathsf{del}\}\big)\big\}.$$

Using this definition, the update history for the conjunction ω is defined as

$$\mathsf{Udts}(\omega) = \mathsf{Udts}(w_1) \cup \big(\cup_{i=2}^{n} \mathsf{Udts}(w_1, w_i)\big).$$

$\underline{\mathsf{delHist}(\omega)}$: We define $\mathsf{delHist}(\omega)$ as the set of all pairs of timestamps of addition and deletion for conjunctive queries ω.

$$\mathsf{delHist}(\omega) = \Big\{\{(u_i^{\mathsf{add}}, u_i^{\mathsf{del}})\}_{i\in[n]} : w_i \in L, \Big(u_i^{\mathsf{add}}, \mathsf{add}, (\mathsf{id}, w_i)\Big) \in \mathsf{QS} \wedge \Big(u_i^{\mathsf{del}}, \mathsf{del}, (\mathsf{id}, w_i)\Big) \in \mathsf{QS}\Big\}.$$

Finally, we define forward and backward privacy for a CD-SSE scheme Π as follows.

Definition 8 (Forward Privacy of CD-SSE) An $\mathcal{L}_\Pi$-adaptive secure CD-SSE scheme Π is forward private if its leakages due to the Udt protocol can be written as the following.

$$\mathcal{L}^{\mathsf{FP}}_{\Pi,\mathsf{Udt}}(\mathsf{op}, \mathsf{in}) = \mathcal{L}^{\mathsf{FP}}_{\Sigma,\mathsf{Udt}}.$$

Definition 9 (Backward Privacy of CD-SSE) An $\mathcal{L}_\Pi$-adaptive secure CD-SSE scheme Π is backward private as per the notions BPIP, BPUP and WBP if its leakages due to the Udt and Srch protocols can be written as the following.

$$\begin{aligned}
\mathcal{L}^{\mathsf{BPIP}}_{\Pi,\mathsf{Udt}}(\mathsf{op}, w, \mathsf{id}) &= \widetilde{\mathcal{L}}_u(\mathsf{op})\\
\mathcal{L}^{\mathsf{BPIP}}_{\Pi,\mathsf{Srch}}(\omega) &= \widetilde{\mathcal{L}}_s(\mathsf{sp}(\omega), \mathsf{TimeDB}(\omega), |\mathsf{Udts}(\omega)|),\\
\mathcal{L}^{\mathsf{BPUP}}_{\Pi,\mathsf{Udt}}(\mathsf{op}, w, \mathsf{id}) &= \widetilde{\mathcal{L}}_u(\mathsf{op}, w)\\
\mathcal{L}^{\mathsf{BPUP}}_{\Pi,\mathsf{Srch}}(\omega) &= \widetilde{\mathcal{L}}_s(\mathsf{sp}(\omega), \mathsf{TimeDB}(\omega), \mathsf{Udts}(\omega)),\\
\mathcal{L}^{\mathsf{WBP}}_{\Pi,\mathsf{Udt}}(\mathsf{op}, w, \mathsf{id}) &= \widetilde{\mathcal{L}}_u(\mathsf{op}, w)\\
\mathcal{L}^{\mathsf{WBP}}_{\Pi,\mathsf{Srch}}(\omega) &= \widetilde{\mathcal{L}}_s(\mathsf{sp}(\omega), \mathsf{TimeDB}(\omega), \mathsf{Udts}(\omega), \mathsf{delHist}(\omega)),
\end{aligned}$$

where, $\widetilde{\mathcal{L}}_u$ and $\widetilde{\mathcal{L}}_s$ are stateless.

$\underline{\Sigma.\mathsf{Stp}\left(1^\lambda, \mathsf{DS}\right)}$:
– $(\mathsf{K}_s, \sigma, \mathsf{EDS}) \leftarrow \Sigma.\mathsf{Stp}_{\mathcal{C}}(1^\lambda, \mathsf{DS})$
– $\perp \leftarrow \Sigma.\mathsf{Stp}_{\mathcal{S}}(\mathsf{EDS})$
$\underline{\Sigma.\mathsf{Udt}\,(k, \sigma, \mathsf{op}, (\mathsf{id}, w); \mathsf{utk}, \mathsf{EDS})}$:
– $(\sigma, \Sigma.\mathsf{utk}) \leftarrow \Sigma.\mathsf{Udt}_{\mathcal{C}}\,(\mathsf{K}_s, \sigma, \mathsf{op}, (\mathsf{id}, w))$
– $\mathsf{EDS} \leftarrow \Sigma.\mathsf{Udt}_{\mathcal{S}}\,(\Sigma.\mathsf{utk}, \mathsf{EDS})$
$\underline{\Sigma.\mathsf{Srch}\,(k, \sigma, w; \mathsf{stk}, \mathsf{EDS})}$:
– $(\sigma, \Sigma.\mathsf{stk}) \leftarrow \Sigma.\mathsf{Srch}_{\mathcal{C}}\,(\mathsf{K}_s, \sigma, w)$
– $(\mathsf{V}_w, \mathsf{EDS}) \leftarrow \Sigma.\mathsf{Srch}_{\mathcal{S}}\,(\Sigma.\mathsf{stk}, \mathsf{EDS})$

Fig. 1 A generic dynamic single-keyword SSE (D-SSE) construction Σ

4 Our construction

Our generic CD-SSE construction for *non-modifiable documents* is denoted by Π. The definition of Π assumes the existence of a forward and WBP backward private D-SSE Σ with support for the add operation only, and the static OXT scheme Γ of [16]. The original description of the static OXT scheme has been presented in a more modular fashion – which we call Γ – so that it may be combined with Σ as black boxes. However, the functionality of OXT remains the same and hence we persist with its original name. Γ assumes that the client knows the *least frequent keyword* w_1 in a conjunction $\omega = w_1 \wedge \cdots \wedge w_n$. During a search operation in Γ, the server first conducts a single keyword search using the T-Set to find all ids containing the *s-term* w_1. For all such ids containing w_1, the X-Set is used to check if the *x-terms* w_i, $i \geq 2$ are also in id. In Π, Σ is used to search for the *s-term* and Γ to search for the conjunction.

A Generic D-SSE Scheme A generic D-SSE scheme $\Sigma = (\Sigma.\mathsf{Stp}, \Sigma.\mathsf{Udt}, \Sigma.\mathsf{Srch})$ is described in Fig. 1.

The Oblivious Cross Tag Scheme A modular description of the OXT scheme $\Gamma = (\Gamma.\mathsf{Stp}, \Gamma.\mathsf{Add}, \Gamma.\mathsf{Srch}_{\mathcal{C}}, \mathsf{Srch}_{\mathcal{S}})$ is in Fig. 2. It assumes the existence of three PRF families $F_p^{(I)} : \mathcal{K} \times \mathcal{ID} \times \{0,1\} \to \mathbb{Z}_p^*$, $F_p^{(X)} : \mathcal{K} \times W \to \mathbb{Z}_p^*$ and $F_p^{(Z)} : \mathcal{K} \times W \times \mathbb{N} \to \mathbb{Z}_p^*$. It omits the SK-SSE part of the original OXT scheme [16], to be substituted by the functionalities of the D-SSE Σ. The original OXT scheme [16] only supports static databases where all (id, w) pairs are added during setup. To adapt it to our dynamic setting, we have split the setup of [16] into $\Gamma.\mathsf{Stp}$ and $\Gamma.\mathsf{Add}$ protocols, without changing their functionalities. $\Gamma.\mathsf{Srch}_{\mathcal{S}}$ also assumes an input V_w which has $\mathsf{V}_w[y_j] = y_j$ for $j \in [\mathsf{cnt}[w]]$ generated within Γ for a keyword w.

The res however accumulates $\mathsf{V}_w[\mathsf{val}_j]$ from Σ. This step integrates Σ and Γ in a modular fashion within our CD-SSE Π.

Our Generic CD-SSE Scheme Our generic CD-SSE sch-eme $\Pi = (\Pi.\mathsf{Stp}, \Pi.\mathsf{Udt}, \Pi.\mathsf{Srch})$ is described using Σ and Γ in Fig. 3. Note the arguments passed to $\Gamma.\mathsf{Srch}_{\mathcal{S}}$ in step 3

Fig. 2 A modular description of Γ without the SK-SSE part of [16]

$\left(\mathsf{K}_x, F_p^{(I)}, F_p^{(X)}, F_p^{(Z)}, \mathsf{cnt}, \mathsf{XDB}\right) \leftarrow \Gamma.\mathsf{Stp}\left(1^\lambda, \mathsf{DB}\right)$:

1. Sample $\mathsf{K}_x = (k_I, k_X, k_Z) \in \mathcal{K}^3$, corresponding $F_p^{(I)}$, $F_p^{(X)}$ and $F_p^{(Z)}$,
2. Initialise an empty map $\mathsf{cnt} : W \to \mathbb{Z}$
3. Set an empty set XDB.
4. Send XDB to the server.

$(\mathsf{cnt}, \Gamma.\mathsf{addtk}) \leftarrow \Gamma.\mathsf{Add}\left(\mathsf{K}_x, \mathsf{cnt}, (\mathsf{id}, w)\right)$:

1. If $\mathsf{cnt}[w] = \perp$, then $\mathsf{cnt}[w] \leftarrow 1$; else, $\mathsf{cnt}[w] \leftarrow \mathsf{cnt}[w] + 1$
2. Compute $\mathsf{xid} \leftarrow F_p^{(I)}(k_I, \mathsf{id})$; $x_w \leftarrow F_p^{(X)}(k_X, w)$; $z_{\mathsf{cnt}[w]} \leftarrow F_p^{(Z)}(k_Z, w \| \mathsf{cnt}[w])$
3. Set $y_{\mathsf{cnt}[w]} = \left(\mathsf{xid} \cdot z_{\mathsf{cnt}[w]}^{-1}\right)$; $\mathsf{xtag} \leftarrow g^{x_w \cdot \mathsf{xid}}$
4. Output $\Gamma.\mathsf{addtk} = \left(y_{\mathsf{cnt}[w]}, \mathsf{xtag}\right)$ to be sent to the server;

$(\mathsf{cnt}, \Gamma.\mathsf{xtk}) \leftarrow \Gamma.\mathsf{Srch}_{\mathcal{C}}\left(\mathsf{K}_x, \mathsf{cnt}, \omega\right)$:

1. For $(j = 1, \ldots, \mathsf{cnt}[w_1])$
   For $(i = 2, \ldots, n)$
   $\mathsf{xtk}[j, i] \leftarrow g^{F_p^{(Z)}(k_Z, w_1 \| j) \cdot F_p^{(X)}(k_X, w_i)}$
   End For
   Randomly shuffle $\mathsf{xtk}[j, i]$.
   End For
2. Output $\Gamma.\mathsf{xtk} = \{\mathsf{xtk}[j, 2], \ldots, \mathsf{xtk}[j, n]\}$ to be sent to the server.

$\mathsf{res} \leftarrow \Gamma.\mathsf{Srch}_{\mathcal{S}}\left(\mathsf{cnt}, \Gamma.\mathsf{xtk}, \mathsf{V}_w, \mathsf{XDB}\right)$:

1. $\mathsf{res} \leftarrow \emptyset$.
2. For $(j = 1, \ldots, \mathsf{cnt})$
   A. $\mathsf{flag} \leftarrow \mathsf{true}$
   B. For $(i = 2, \ldots, n)$
      i. If $(\Pi.\mathsf{xtk}[j, i]^{\mathsf{V}_w[y_j]} \notin \mathsf{XDB})$,
         a. $\mathsf{flag} \leftarrow \mathsf{false}$
   C. If $(\mathsf{flag} == \mathsf{true})$,
      i. $\mathsf{res} \leftarrow \mathsf{res} \cup \{\mathsf{V}_w[\mathsf{val}_j]\}$
3. Return res

of $\Pi.\mathsf{Srch}_{\mathcal{S}}$ Fig. 3. The input cnt to $\Gamma.\mathsf{Srch}_{\mathcal{S}}$ is $\mathsf{st.cnt}[w_1]$ from $\Pi.\mathsf{Srch}_{\mathcal{S}}$. The input $\Pi.\mathsf{V}_{w_1}$ is from the single keyword search $\Sigma.\mathsf{Srch}_{\mathcal{S}}$ on the *s-term* w_1, containing pairs (val_j, y_j) for $j \in [\mathsf{st.cnt}[w_1]]$.

Note that on receiving the result res from the server in step 4 of $\Pi.\mathsf{Srch}_{\mathcal{S}}$, the client finds the search result for the conjunction. The identifier in the search result is of the form $\mathsf{id}\|\mathsf{op}$. To get the final search result $\mathsf{DB}(\omega)$, the client does the following. Initialise an empty set $\mathcal{I}_\omega \leftarrow \emptyset$. For each $r \in \mathsf{res}$, parse $r = \mathsf{id}\|\mathsf{op}$. Now if $(\mathsf{op} = \mathsf{add})$, $\mathcal{I}_\omega \leftarrow \mathcal{I}_\omega \cup \{\mathsf{id}\}$, and if $(\mathsf{op} = \mathsf{del})$. $\mathcal{I}_\omega \leftarrow \mathcal{I}_\omega \setminus \{\mathsf{id}\}$. Finally output, $\mathcal{I}_\omega$.

The ODXT scheme of [41] is an instantiation of our generic construction Π. Even though ODXT is not forward private in the general setting (as shown in [56]), it is secure in the newly identified non-modifiable document setting, as argued below.

> *For systems where modifiability is essential,* Π *can be used securely with some additional overhead costs (using a common technique* [13, 15, 27]).

4.1 Security of Π

The security of our generic construction is argued based on well-defined leakage functions that have been studied extensively. We analyse our scheme's security, using the de facto standard of assuming the maximum possible leakage of each of the components. This would allow future improvements in efficiency or adaptability within the context of non-modifiable documents and their associated leakages. *Our generic construction achieves*BPUP *backward privacy for CD-SSE, regardless of the type of backward privacy*(WBP or stronger) *achieved by the component D-SSE used.*

We recollect that Σ is an $\mathcal{L}_\Sigma$-adaptively secure forward and backward private D-SSE scheme as per Definitions 5, 6 and 7. With this assumption, we prove our generic CD-SSE scheme Π to be $\mathcal{L}_\Pi^{\mathsf{BPUP}}$-adaptively secure. The leakage functions in Definitions 8 and 9 subsume the actual leakages of our generic scheme Π. The leakage for WBP is a superset of the leakage of the other two notions of backward privacy –

Fig. 3 Our generic CD-SSE construction Π

$\Pi.\mathsf{Stp}\left(1^\lambda, \mathsf{DB}\right)$:

Client:

$(\mathsf{K}, \mathsf{st}, \Pi.\mathsf{EDB}) \leftarrow \Pi.\mathsf{Stp}_{\mathcal{C}}\left(1^\lambda, \mathsf{DB}\right)$:

$(\mathsf{K}_s, \sigma, \mathsf{EDS}) \leftarrow \Sigma.\mathsf{Stp}_{\mathcal{C}}(\mathsf{DB}, 1^\lambda)$; assign $\mathsf{st}.\sigma \leftarrow \sigma$ and $\Pi.\mathsf{EDS} \leftarrow \mathsf{EDS}$

$(\mathsf{K}_x, F_p^{(I)}, F_p^{(X)}, F_p^{(Z)}, \mathsf{cnt}, \mathsf{XDB}) \leftarrow \Gamma.\mathsf{Stp}(\mathsf{DB}, 1^\lambda)$

Assign $\mathsf{st}.\mathsf{cnt} \leftarrow \mathsf{cnt}$ and $\Pi.\mathsf{XDB} \leftarrow \mathsf{XDB}$

Set $\mathsf{K} \leftarrow (\mathsf{K}_s, \mathsf{K}_x)$, and $\mathsf{st} \leftarrow (\mathsf{st}.\sigma, \mathsf{st}.\mathsf{cnt})$

Send, $\Pi.\mathsf{EDB} \leftarrow (\Pi.\mathsf{EDS}, \Pi.\mathsf{XDB})$ to the server

Server:

$\perp \leftarrow \Pi.\mathsf{Stp}_{\mathcal{S}}(\Pi.\mathsf{EDB})$: Server stores $\Pi.\mathsf{EDB}$

$\Pi.\mathsf{Udt}(k, \mathsf{st}, \mathsf{op}, (\mathsf{id}, \mathsf{w}); \Pi.\mathsf{utk}, \Pi.\mathsf{EDB})$:

Client:

$(\mathsf{st}, \Pi.\mathsf{utk}) \leftarrow \Pi.\mathsf{Udt}_{\mathcal{C}}(\mathsf{K}, \mathsf{st}, \mathsf{op}, (\mathsf{id}, \mathsf{w}))$:

1. For all $w \in \mathsf{w}$
   a. Call $\Sigma.\mathsf{Udt}_{\mathcal{C}}(\mathsf{K}_s, \mathsf{st}.\sigma, \mathsf{add}, (w, (\mathsf{id}\|\mathsf{op})))$ to get $\Sigma.\mathsf{utk}$
   b. Call $\Gamma.\mathsf{add}(\mathsf{K}_x, \mathsf{st}.\mathsf{cnt}, w, \mathsf{id}\|\mathsf{op})$ to get $\Gamma.\mathsf{addtk} = \left(y_{\mathsf{cnt}[w]}, \mathsf{xtag}\right)$
   c. Set $\Pi.\mathsf{utk}_s \leftarrow (\Sigma.\mathsf{utk}, y_{\mathsf{cnt}[w]})$ and $\Pi.\mathsf{utk}_x \leftarrow \{\mathsf{xtag}\}$;
2. Send $\Pi.\mathsf{utk} \leftarrow (\Pi.\mathsf{utk}_s, \Pi.\mathsf{utk}_x)$ to the server

Server:

$\Pi.\mathsf{EDB} \leftarrow \Pi.\mathsf{Udt}_{\mathcal{S}}(\Pi.\mathsf{utk}, \Pi.\mathsf{EDB})$:

1. Call $\Sigma.\mathsf{Udt}_{\mathcal{S}}(\Pi.\mathsf{utk}_s, \Pi.\mathsf{EDS})$ and to get the output EDS as following
   a. Parse $(\Sigma.\mathsf{utk}, y_{\mathsf{cnt}[w]}) \leftarrow \Pi.\mathsf{utk}_s$
   b. Further parse $(\mathsf{loc}, \mathsf{val}) \leftarrow \Sigma.\mathsf{utk}$
   c. Update $\Pi.\mathsf{EDS}[\mathsf{loc}] \leftarrow (\mathsf{val}, y_{\mathsf{cnt}[w]})$
2. Set $\Pi.\mathsf{XDB}$ as $\Pi.\mathsf{XDB} \leftarrow \Pi.\mathsf{XDB} \cup \Pi.\mathsf{utk}_x$
3. Output $\Pi.\mathsf{EDB} \leftarrow (\Pi.\mathsf{EDS}, \Pi.\mathsf{XDB})$

$\Pi.\mathsf{Srch}(k, \mathsf{st}, \omega; \Pi.\mathsf{stk}, \Pi.\mathsf{EDB})$:

Client:

$(\Pi.\mathsf{stk}, \Pi.\mathsf{xtk}) \leftarrow \Pi.\mathsf{Srch}_{\mathcal{C}}(\mathsf{K}, \mathsf{st}, \omega)$:

1. Call $\Sigma.\mathsf{Srch}_{\mathcal{C}}(\mathsf{K}_s, \mathsf{st}.\sigma, w_1)$ to get $\Sigma.\mathsf{stk}$; assign $\Pi.\mathsf{stk} \leftarrow \Sigma.\mathsf{stk}$
2. Call $\Gamma.\mathsf{Srch}(\mathsf{K}_x, \mathsf{st}.\mathsf{cnt}, \omega)$ to get $\Gamma.\mathsf{xtk}$; assign $\Pi.\mathsf{xtk} \leftarrow \Gamma.\mathsf{xtk}$
3. Send $(\Pi.\mathsf{stk}, \Pi.\mathsf{xtk})$ to the server

Server:

$\mathsf{res} \leftarrow \Pi.\mathsf{Srch}_{\mathcal{S}}((\Pi.\mathsf{stk}, \Pi.\mathsf{xtk}), \Pi.\mathsf{EDB})$:

1. $\mathsf{res} \leftarrow \phi$.
2. Call $\Sigma.\mathsf{Srch}_{\mathcal{S}}(\Pi.\mathsf{stk}, \Pi.\mathsf{EDS})$ to get $\Pi.\mathsf{V}_{w_1} = \{(\mathsf{val}_j, y_j) : j \in [\mathsf{st}.\mathsf{cnt}[w_1]]\}$
3. $\mathsf{res} \leftarrow \Gamma.\mathsf{Srch}_{\mathcal{S}}(\mathsf{st}.\mathsf{cnt}, \Pi.\mathsf{xtk}, \Pi.\mathsf{V}_{w_1}, \Pi.\mathsf{XDB})$
4. Send res to the client

BPIP and BPUP. We prove Π to be BPUP backward private, assuming Σ to be WBP backward private. So Π would still be at least BPUP backward private even when Σ is instantiated with a BPIP or a BPUP backward private D-SSE scheme. We assume the leakage $\mathcal{L}_\Pi^{\mathsf{BPUP}} = (\mathcal{L}_{\Pi,\mathsf{Stp}}, \mathcal{L}_{\Pi,\mathsf{Udt}}, \mathcal{L}_{\Pi,\mathsf{Srch}})$ to be

$$\mathcal{L}_{\Pi,\mathsf{Stp}} = \perp, \qquad \mathcal{L}_{\Pi,\mathsf{Udt}}(\mathsf{op}, \mathsf{id}, w) \preceq \{\mathsf{op}\},$$
$$\mathcal{L}_{\Pi,\mathsf{Srch}}(\omega) \preceq \{\mathsf{sp}(\omega), \mathsf{TimeDB}(\omega), \mathsf{Udts}(\omega)\}$$

following the common practice of defining and subsequently proving the security of SSE, assuming standard leakage functions that are supersets of the actual leakage of our scheme Π. We also find the actual leakage of Π. It is essential to point out that the leakage $\mathcal{L}_\Pi$ becomes exactly $\mathcal{L}_\Sigma$ if $\omega = w$.

Theorem 1 *Assuming that (1) Σ is an $\mathcal{L}_\Sigma$-adaptively secure forward and backward private D-SSE, (2) F_p is a secure PRF, and (3) the* eddh *problem is hard in the group output by $\mathcal{G}$, our scheme Π is an $\mathcal{L}_\Pi^{\mathsf{BPUP}}$-adaptively secure CD-SSE.*

Proof Idea. The cryptographic primitives in Π are replaced with ideal primitives through six games $\mathsf{G}_0, \ldots, \mathsf{G}_6$, each showing that no PPT adversary can detect the change in any of them. Finally, we show that there exists a simulator that takes the leakages $\mathcal{L}_\Pi = \left(\mathcal{L}_{\Pi,\mathsf{Stp}}, \mathcal{L}_{\Pi,\mathsf{Udt}}, \mathcal{L}_{\Pi,\mathsf{Srch}}\right)$ as input and simulates the protocol's transcript that is indistinguishable

from the actual execution of the protocol with ideal primitives.

Proof (Theorem 1)

GAME G_0. The experiment starts with the real execution of the protocol Π. So

$$\Pr\left[\mathsf{SSEReal}^{\Pi}_{\mathcal{A}}(1^\lambda, q) = 1\right] = \Pr[\mathsf{G}_0 = 1].$$

Since Σ is adaptively secure, we do not explicitly argue about the simulation of the components generated from Σ.

GAME G_1. We replace all cryptographic primitives used in the construction of $\Sigma.\mathsf{utk}$, similar to the security proof of Σ. We can do so as Σ is $\mathcal{L}_\Sigma$-adaptive secure. Thus the advantage of any adversary distinguishing games G_0 and G_1 is bounded by the SSE advantage of Σ. So

$$|\Pr[\mathsf{G}_0 = 1] - \Pr[\mathsf{G}_1 = 1]| \leq \mathsf{Adv}^{\Sigma,\mathsf{SSE}}_{\mathcal{A},\mathcal{S},\mathcal{L}_\Sigma} \leq \mathsf{negl}(\lambda).$$

GAME G_2. Next the challenger replaces all executions of $F_p^{(I)}$, $F_p^{(X)}$ and $F_p^{(Z)}$ one by one with randomly sampled strings from $\mathbb{Z}_p^*$. The experiment maintains three lists I, X and Z to store the values sampled for $F_p^{(I)}$, $F_p^{(X)}$ and $F_p^{(Z)}$, thus can be reused later if needed. The entries of I, X and Z are indexed by $(\mathsf{id}, \mathsf{op})$,

w and $(w, \mathsf{cnt}[w])$. For an update operation of $(\mathsf{op}, (\mathsf{id}, w))$, if $I[\mathsf{op}, \mathsf{id}] = \perp$ then the challenger picks a random value form $\mathbb{Z}_p^*$ and uses it for $F_p^{(I)}(k_I, \mathsf{id}\|\mathsf{op})$, and also stores the random value in $I[\mathsf{op}, \mathsf{id}]$. If $I[\mathsf{op}, \mathsf{id}] \neq \perp$, then the challenger uses the value stored in $I[\mathsf{op}, \mathsf{id}]$. The other two PRFs $F_p^{(X)}$ and $F_p^{(Z)}$ are simulated similarly. Now if there exist PPT adversaries $\mathcal{A}_I, \mathcal{A}_X$ and $\mathcal{A}_Z$ who can distinguish between the games G_1 and G_2 based on the use of $F_p^{(I)}$, $F_p^{(X)}$ and $F_p^{(Z)}$ respectively, we can use these adversaries to break the security of F_p. Thus,

$$|\Pr[\mathsf{G}_1 = 1] - \Pr[\mathsf{G}_2 = 1]| \leq \mathsf{Adv}^{\mathsf{prf}}_{F_p^{(I)},\mathcal{A}_I}(1^\lambda) + \mathsf{Adv}^{\mathsf{prf}}_{F_p^{(X)},\mathcal{A}_X}(1^\lambda) + \mathsf{Adv}^{\mathsf{prf}}_{F_p^{(Z)},\mathcal{A}_Z}(1^\lambda) \leq \mathsf{negl}(\lambda).$$

GAME G_3. For a search query $\omega = w_1 \wedge w_2 \wedge \cdots \wedge w_n$ in step 1 of $\Pi.\mathsf{Srch}_{\mathcal{C}}$ the client generates $\Pi.\mathsf{stk}$ using $\Sigma.\mathsf{stk}$. So the challenger proceeds exactly as the security proof of Σ to show the generation of $\Sigma.\mathsf{stk}$ is indistinguishable from its simulated form. The same argument will hold in this proof as well.

Since the difference between G_2 and G_3 is only in the generation of $\Sigma.\mathsf{stk}$ (using ideal primitives) and the advantage of any such adversary is at most $\mathsf{Adv}^{\mathsf{sse}}_{\Sigma,\mathcal{A},\mathcal{S}}$, hence we have

$$|\Pr[\mathsf{G}_2 = 1] - \Pr[\mathsf{G}_3 = 1]| \leq \mathsf{Adv}^{\Sigma,\mathsf{SSE}}_{\mathcal{A},\mathcal{S},\mathcal{L}_\Sigma} \leq \mathsf{negl}(\lambda).$$

GAME G_4. For a search query $\omega = w_1 \wedge w_2 \wedge \cdots \wedge w_n$, the challenger next changes the manner in which $\mathsf{xtk}_{i,j}$ is generated. Recall,

$$\mathsf{xtk}_{i,j} = g^{F_p^{(Z)}(k_Z, w_1\|j)\cdot F_p^{(X)}(k_X, w_i)},$$

where $j \in [\mathsf{cnt}[w]]$ and $i \in [2, n]$. The challenger knows all ids that were updated corresponding to the *s-term* w_1. The challenger computes the xtag as, $\mathsf{xid} \leftarrow F_p^{(I)}(k_I, \mathsf{id}\|\mathsf{op})$, $x_w \leftarrow F_p^{(X)}(k_X, w)$. Then it computes

$$\mathsf{xtag} \leftarrow g^{x_w \cdot \mathsf{xid}}$$

for all values corresponding to all the ids that were updated for w_1 and all the keywords in the *x-term*s. Notice that the challenger can do this using its lists I and X. Then the challenger computes the values for $y_{i,j} = \mathsf{xid} \cdot z^{-1}_{\mathsf{cnt}[w]}$, where $z_{\mathsf{cnt}[w]} \leftarrow F_p^{(Z)}(k_Z, w\|\mathsf{cnt}[w])$.

Finally the challenger computes

$$\mathsf{xtk}_{i,j} = \mathsf{xtag}_{i,j}^{y^{-1}_{i,j}} = (g^{x_w \cdot \mathsf{xid}})^{\mathsf{xid}^{-1} \cdot z_{\mathsf{cnt}[w]}} = g^{x_w \cdot z_{\mathsf{cnt}[w]}}.$$

Now, the view of the adversary in games G_3 and G_4 is identical, and no adversary has any advantage of distinguishing these two games. Thus,

$$\Pr[\mathsf{G}_3 = 1] = \Pr[\mathsf{G}_4 = 1].$$

GAME G_5. Next, the challenger samples random values $y \xleftarrow{\$} \mathbb{Z}_p^*$ and uses them in place of $y_{\mathsf{cnt}[w]}$'s in the update operations. We recollect that $y_{\mathsf{cnt}[w]} = \mathsf{xid} \cdot z^{-1}_{\mathsf{cnt}[w]} \in \mathbb{Z}_p^*$. Now if $a, b, c \xleftarrow{\$} \mathbb{Z}_p^*$, then c is statistically indistinguishable from $(a \cdot b) \in \mathbb{Z}_p^*$ for any adversary. The secure PRF F_p is used to generate xid and $z_{\mathsf{cnt}[w]}$. So xtag and $z^{-1}_{\mathsf{cnt}[w]}$ are also indistinguishable for any random element from $\mathbb{Z}_p^*$. In the Games G_2, we have already replaced all uses of F_p with randomly sampled values from $\mathbb{Z}_p^*$. Replacing the value of $y_{\mathsf{cnt}[w]}$ with a randomly sampled element from $\mathbb{Z}_p^*$ does not change the advantage of any adversary. Thus,

$$\Pr[\mathsf{G}_4 = 1] = \Pr[\mathsf{G}_5 = 1].$$

GAME G_6. In game G_6, for each update operation, the challenger samples a random value $\gamma \xleftarrow{\$} \mathbb{Z}_p^*$ and replaces the value of xtag in every update operation with the value g^γ. Recall that $\mathsf{xtag} = g^{F_p^{(I)}(k_I, \mathsf{id})\cdot F_p^{(X)}(k_X, w)}$ where F_p is a secure PRF. We have already replaced the values of every execution of F_p with random values from $\mathbb{Z}_p^*$. In particular, we already have $\mathsf{xtag} = g^{\alpha\cdot\beta}$ for random values $\alpha, \beta \in \mathbb{Z}_p^*$. Upon replacing the values of xtag with g^γ if any adversary $\mathcal{A}$

can detect the change between the games G_5 and G_6, we can construct another adversary $\mathcal{B}$ which uses $\mathcal{A}$ as a subroutine to break the eddh problem. Due to [41], breaking the eddh problem is equivalent to breaking the ddh assumption if the number of queries is bounded by $q = \mathsf{poly}(\lambda)$. Thus,

$$|\Pr[\mathsf{G}_5 = 1] - \Pr[\mathsf{G}_6 = 1]| \leq \mathsf{Adv}^{\mathsf{eddh}}_{\mathcal{B}} \leq \mathsf{negl}(\lambda).$$

Simulation. Finally, we demonstrate a simulator that perfectly simulates a transcript, which would be indistinguishable from the transcript in game G_6. The simulator does not have the actual queries in Π. Instead, the simulator only has access to the leakage functions $\mathcal{L}_\Pi$. The simulation has two parts, one is to simulate the components generated in the execution of Σ, and another is to simulate the components of Γ, which together construct Π. Now the components generated using Σ can be simulated perfectly as Σ is $\mathcal{L}_\Sigma$-adaptively secure SSE. We note that $\mathcal{L}_\Sigma \subseteq \mathcal{L}_\Pi$.

We demonstrate how to simulate the components generated in the execution of Γ. To simulate update queries, the simulator has access to the leakage function $\mathcal{L}^{\mathsf{BPUP}}_{\Pi,\mathsf{Udt}}(\mathsf{op}, w, \mathsf{id}) = \mathsf{op}$. The simulator knows all the time stamps of update queries. For every update query, the simulator simulates $y_{\mathsf{cnt}[w]}$ and xtag by sampling two uniform random values α, β from $\mathbb{Z}^*_p$ and provides α and g^β in place of $y_{\mathsf{cnt}[w]}$ and xtag (similar to G_6). So the simulation is indistinguishable. The simulator also stores the values of $y_{\mathsf{cnt}[w]}$ and xtag sampled for each update in a map Map indexed by the timestamps of the queries.

Now to simulate a conjunctive search query $\omega = w_1 \wedge w_2 \wedge \cdots \wedge w_n$, the simulator simulates $\mathsf{xtk}_{i,j}$ as follows. From the leakage functions $\mathsf{Udts}(w)$ and $\mathsf{TimeDB}(w)$, the simulator knows all timestamps of update queries for the *s-term* and their corresponding ids. From $\mathsf{TimeDB}(\omega)$ and $\mathsf{Udts}(\omega)$, the simulator knows the timestamps where a keyword in the *x-term* matches these ids. The simulator uses these timestamps to determine the corresponding $\mathsf{xtag}_{i,j}$ and $y_{i,j}$ values to construct $\mathsf{xtk}_{i,j} = \mathsf{xtag}_{i,j}^{y^{-1}_{\mathsf{cnt}_{i,j}}}$ (similar to G_6). So the simulation is indistinguishable.

Lastly, if two search queries ω_1 and ω_2 have different *s-terms* but the same *x-term* and share some common ids, then the simulator should simulate the xtk such that the same xtag is generated for those queries. The simulator knows this by considering $\mathsf{sp}(\omega_1)$, $\mathsf{Udts}(\omega_1)$, $\mathsf{sp}(\omega_2)$ and $\mathsf{Udts}(\omega_2)$. Hence this simulation is also indistinguishable, and we have

$$\Pr\left[\mathsf{SSEIdeal}^{\Pi}_{\mathcal{A},\mathcal{S},\mathcal{L}_\Pi}(1^\lambda, q) = 1\right] = \Pr[\mathsf{G}_6 = 1].$$

Thus,

$$\left|\Pr\left[\mathsf{SSEReal}^{\Pi}_{\mathcal{A}}(1^\lambda, q) = 1\right] - \Pr\left[\mathsf{SSEIdeal}^{\Pi}_{\mathcal{A},\mathcal{S},\mathcal{L}_\Pi}(1^\lambda, q) = 1\right]\right| \leq \mathsf{negl}(\lambda)$$

□

Corollary 1 *Assuming Σ is only $\mathcal{L}^{\mathsf{FP}}_\Sigma$ forward private, Π is only $\mathcal{L}^{\mathsf{FP}}_\Pi$ forward private.*

Proof The proof of Theorem 1 showed that $\mathcal{L}^{\mathsf{FP}}_{\Pi,\mathsf{Udt}}(\mathsf{op}, \mathsf{in}) = \mathcal{L}^{\mathsf{FP}}_{\Sigma,\mathsf{Udt}}$. So from Definition 8, we get the result. □

Corollary 2 *Assuming Σ is only $\mathcal{L}^{\mathsf{WBP}}_\Sigma$ secure backward private, Π is $\mathcal{L}^{\mathsf{BPUP}}_\Pi$ backward private.*

Proof In Fig 3, the Π.Udt protocol never uses $\mathsf{op} = \mathsf{del}$. Thus, the delete history for every keyword remains empty, i.e., $\mathsf{delHist}(w) = \emptyset$. Since $\mathsf{delHist}(w) = \emptyset$, the backward privacy of Π naturally upgrades to BPUP security from WBP. □

Corollary 3 *Assuming Σ is $\mathcal{L}^{\mathsf{BPIP}}_\Sigma$ secure backward private, Π is only $\mathcal{L}^{\mathsf{BPUP}}_\Pi$ backward private.*

Proof The xtags of OXT (Fig 2) are stored in a set membership query data structure. With every search for an xtag in the Γ.Srch protocol, an xtag found in the XSet also reveals the time when the element was added to the set (i.e. time of update is leaked), thereby degrading the whole security to BPUP. □

We note that constructing a forward and BPIP backward private CD-SSE is still an open problem.

Asymptotic Performance. Π is significantly more efficient compared to the most efficient CD-SSE of the "modifiable" setting [54, 58]. With our generalization, ODXT [41] is now an instantiation of our scheme. The scheme of [58] stores xtag in a separate D-SSE and fetches them during search to maintain forward privacy. Thus, as reported in [58], their scheme performs slower than [41], hence our scheme Π. Our scheme Π has both computation and communication complexities of $\mathcal{O}(nu_1)$ for the search operation, and $\mathcal{O}(1)$ for the update operation. In [54], the computation and communication complexities of the search operation are both $\mathcal{O}(u_1 + n \cdot \mathsf{DB}(w_1))$ and involves two rounds, where u_1 is the number of updates related to w_1; the complexities of the update operation are $\mathcal{O}(|W|/|W_d|)$ for adding a document, $\mathcal{O}(|D|)$ for editing and $\mathcal{O}(1)$ for deleting, where $|W|$ is the number of keywords in the database and $|W_d|$ is the number of keywords contained in the updated document. The storage requirement of [54] is $\mathcal{O}(|W||D|)$, as compared to $\mathcal{O}(N)$ (typically, $N < |W||D|$ in practice) for our scheme, where N is the number of identifier-keyword pairs in the database. This additional cost they pay for hiding the KPRP leakage, which is to know the results of all the sub-queries of the form $w_i \wedge w_j$ for all w_i, w_j in the conjunction $w_1 \wedge w_2 \wedge \cdots \wedge w_n$.

As mentioned in [54], for a file injection attack [55] to work properly, the adversary must know the KPRP leakage. Although our scheme does not prevent such leakage, the precise leakage of our scheme is $w_1 \wedge w_i$ for all w_i in the conjunction. Moreover, as shown by [41], random shuffling of xtks reduces the success probability of the file injection attack like [55] to almost 5% even when the amount of injected files is 60% of the entire database.

5 Actual leakage of our CD-SSE construction

Historically, SSE constructions without proper leakage analyses have been attacked by exploiting leakages that were missed in the original proposals. The leakage missed in most D-SSE constructions (except [18, 47]) was exploited by the file injection attack [55]. The update leakage of [41] was incorrectly assumed to be null that led to an attack on its forward privacy [58]. Thus, a thorough leakage analysis is essential for any SSE. We introduce the method of analysing the actual leakage through all possible combinations of query sequences. Such thorough analysis of the actual leakage of any SSE scheme gives us confidence in their security and avoids oversights. Note that our security arguments are still based on standard leakage functions from the literature, that are supersets of these actual leakages. In particular, the security proof of Π we assumed much higher leakage than the actual leakage of Π. In fact the actual leakage $\widehat{\mathcal{L}_\Pi} = \left(\widehat{\mathcal{L}_{\Pi,\mathsf{Stp}}}, \widehat{\mathcal{L}_{\Pi,\mathsf{Srch}}}, \widehat{\mathcal{L}_{\Pi,\mathsf{Udt}}}\right)$ of Π is much less than $\mathcal{L}_\Pi$ assumed in Theorem 1. Next, we provide a thorough leakage analysis of our scheme Π. The following are public at the beginning of Π: the security parameter λ, the description of Π, the set of all possible keywords W, the set $\mathcal{ID}$ from which each document is assigned a unique identifier, the key space $\mathcal{K}$, the descriptions of the data-structures $\Pi.\mathsf{EDS}$ and $\Pi.\mathsf{XDB}$ to be stored in the server, the descriptions of the PRF families $F_p^{(I)}(\cdot,\cdot)$, $F_p^{(X)}(\cdot,\cdot)$, $F_p^{(Z)}(\cdot,\cdot)$.

5.1 Setup leakage

During setup, we assume that the initial database DB that is input to $\Sigma.\mathsf{Stp}_\mathcal{C}$ is empty. (To work with an initial non-empty database, we would send an empty database in the setup phase and then update it using the update protocol.) Hence, the initial structures of $\Pi.\mathsf{EDS}$ and $\Pi.\mathsf{XDB}$ are already known to the adversary. The server receives as input $\Pi.\mathsf{EDB} = (\Pi.\mathsf{EDS}, \Pi.\mathsf{XDB})$ and stores it. Since $\Pi.\mathsf{EDB}$ does not have any data from $\mathcal{ID}$ or W, there is no leakage during the setup of Π. In other words, $\widehat{\mathcal{L}_{\Pi,\mathsf{Stp}}} = \emptyset$.

After the setup, the server receives additional information with each update or search query. To find the leakage due to a query $q_i \in \mathsf{QS}$, we should consider all information that the server had before q_i and those that it attains due to and after processing q_i.

5.1.1 Update leakage

Single Update Leakage Before an update query for $(\mathsf{op}, (\mathsf{id}, w))$, let the state of the client be $\mathsf{st} = (\mathsf{st}.\sigma, \mathsf{st}.\mathsf{cnt})$ while the server has the encrypted database $\Pi.\mathsf{EDB} = (\Pi.\mathsf{EDS}, \Pi.\mathsf{XDB})$. During an update operation, the client calls $\Pi.\mathsf{Udt}_\mathcal{C}$ and the output $\Pi.\mathsf{utk} = \left(\left(\Sigma.\mathsf{utk}, y_{\mathsf{cnt}[w]}\right), \mathsf{xtag}\right)$ is sent to the server. So the leakage due to the update operation must be from $\Pi.\mathsf{utk}$ only. Note that $\Sigma.\mathsf{utk}$ is an output of $\Sigma.\mathsf{Udt}_\mathcal{C}$ and $(y_{\mathsf{cnt}[w]}, \mathsf{xtag})$ is an output of $\Gamma.\mathsf{Add}$. We argue that these two outputs together do not leak any more than $\emptyset$.

1. $\Sigma.\mathsf{utk}$ *leaks* $\emptyset$. We first note that Π calls $\Sigma.\mathsf{Udt}_\mathcal{C}$ only with $\mathsf{op} = \mathsf{add}$ and never uses $\mathsf{op} = \mathsf{del}$. By Definitions 6 and 7, the update token $\Sigma.\mathsf{utk}$ output by $\Sigma.\mathsf{Udt}_\mathcal{C}(\mathsf{K}_s, \sigma, \mathsf{op}, (\mathsf{id}, w))$ leaks at most the operation op which is always add in Π. However, for both $\mathsf{op} = \mathsf{add}, \mathsf{del}$ the algorithm only issues an add token, so add and del operations are indistinguishable. So there is no leakage from $\Sigma.\mathsf{utk}$.
2. xtag *leaks* $\emptyset$. Recall that, $x_w = F_p^{(X)}(k_X, w)$ is fixed for every w, and $\mathsf{xid} = F_p^{(I)}(k_I, \mathsf{id}\|\mathsf{op})$ is fixed for every $(\mathsf{id}\|\mathsf{op})$. In our setting for non-modifiable documents, the tuple $(\mathsf{op}, w, \mathsf{id})$ is always unique for every update operation (as (id, w) is added or deleted only once). The pseudo-randomness of F_p ensures that x_w and xid for a single update are two random elements from the group G. Thus ddh hardness of G (Definition 2) ensures that no efficient adversary can distinguish the xtag which is $g^{x_w \cdot \mathsf{xid}}$ from a random element of G.
3. $y_{\mathsf{cnt}[w]}$ *leaks* $\emptyset$. The value $\mathsf{cnt}[w]$ changes (incremented by 1) for every update operation corresponding to every keyword w and never repeats. As a result, $(w\|\mathsf{cnt}[w])$ is always unique. The pseudo-randomness of F_p ensures that $z_{\mathsf{cnt}[w]}$ which is $F_p^{(Z)}(k_Z, w\|\mathsf{cnt}[w])$ and consequently $y_{\mathsf{cnt}[w]}$ which is $\mathsf{xid} \cdot z_{\mathsf{cnt}[w]}^{-1}$ are random elements from G (a formal argument is provided in Game G_5 of Theorem 1). Thus, $y_{\mathsf{cnt}[w]}$ leaks nothing.
4. *Together,* $((\Sigma.\mathsf{utk}, y_{\mathsf{cnt}[w]}), \mathsf{xtag})$ *leak* $\emptyset$. $\Sigma.\mathsf{utk}$ is generated from an independent protocol from $(y_{\mathsf{cnt}[w]}, \mathsf{xtag})$. Thus, $\Sigma.\mathsf{utk}$ together with $(y_{\mathsf{cnt}[w]}, \mathsf{xtag})$ leaks nothing. Additionally, $y_{\mathsf{cnt}[w]}$ and xtag are two random elements of the group G. So altogether there is no leakage.

Thus, the leakage of a single update operation is $\emptyset$.

Update-Update Leakage

1. $\Sigma.\mathsf{utk}$ *of many updates leaks*$\emptyset$. The leakage from the update of Σ is $\emptyset$ even for many updates. This ensures that $\Sigma.\mathsf{utk}$ of our scheme is also $\emptyset$.
2. $y_{\mathsf{cnt}[w]}$*leaks* $\emptyset$. For every update operation, the input $w\|\mathsf{cnt}[w]$ to $F_p^{(Z)}$ is always unique. The pseudo-random ness of F_p ensures that $y_{\mathsf{cnt}[w]}$ in several updates are indistinguishable from a random element of G.
3. xtag *leaks* $\emptyset$. The above discussion shows, that for any two update operations, both x_w and xid can not be the same. Moreover, the server never gets to see x_w or xid in clear in any operation. For many update operations, two updates could use the same x_w (when keyword is same) or the same xid (when id and op are same). However, both x_w and xid would never be the same. Hence, eddh hardness of G (Definition 3) ensures that no efficient adversary can distinguish the xtag which is $g^{x_w \cdot \mathsf{xid}}$ from a random element of G even when the server sees many xtags either with same x_w or with same xid (argument similar to Game G_6 of Theorem 1).
4. *Together,* $((\Sigma.\mathsf{utk}, y_{\mathsf{cnt}[w]}), \mathsf{xtag})$*leak* $\emptyset$. From the games G_1, G_5 and G_6 in Theorem 1, we know that each of these values can be simulated independently of each other. Since the adversary can not distinguish between any of these games (except with negligible probability), the server does not get any additional information from this tuple, other than what it does from each of its elements individually.

Search-Update Leakage Consider the following sequence of two queries. (1) The current update query q_β at time instance β is, $q_\beta = (\beta, \mathsf{op}_\beta, (\mathsf{id}_\beta, w_\beta))$, that is input to $\Pi.\mathsf{Udt}_{\mathcal{C}}$ and outputs $((\Sigma.\mathsf{utk}_\beta, y_\beta), \mathsf{xtag}_\beta)$. (2) For $\alpha < \beta$, a previous search query $q_\alpha = (\alpha, \mathsf{srch}_\alpha, \omega_\alpha)$, where $\omega_\alpha = w_1 \wedge \cdots \wedge w_n$, that is input to $\Pi.\mathsf{Srch}_{\mathcal{C}}$ and outputs $(\Sigma.\mathsf{stk}_\alpha, \Gamma.\mathsf{xtk}_\alpha)$.

We argue that the leakage due to the outputs of q_α and q_β in each case is no more when combined. Both $\Sigma.\mathsf{utk}_\beta$ and $\Sigma.\mathsf{stk}_\alpha$ are outputs to the server for the underlying single keyword search protocol Σ. Hence the leakage due to the combination of these two parameters is no more than has been accounted for in $\mathcal{L}_{\Sigma,\mathsf{Udt}}$. As we only invoke Σ with the add operation, there is no leakage.

In the proof of Theorem 1, we have argued in Game G_5 that the values $y_j \leftarrow F_p^{(I)}(k_I, \mathsf{id}\|\mathsf{op}) \cdot (F_p^{(Z)}(k_Z, w\|j))^{-1}$, are essentially random elements of G to the adversary. Due to our setting, the tuple $(\mathsf{op}, w, \mathsf{id})$ received in q_β has never appeared in any previous update. This ensures that y_β received in the current update operation when combined with xtk_α received during the search operations at time instance α, does not generate an $\mathsf{xtag} = \mathsf{xtk}_\alpha[i, j]^{y_\beta}$ that is present in XDB. Similarly, xtag_β received in q_β is also not present in XDB. Thus, xtag_β and y_β received in the current update combined with any previous search query q_α do not leak any more than the leakage after the current update. All of the above together show that the update leakage for our scheme is $\widehat{\mathcal{L}_{\Pi,\mathsf{Udt}}} = \emptyset$.

Remark 1 Even though we see that $\widehat{\mathcal{L}_{\Pi,\mathsf{Udt}}} = \emptyset$, denoting that there is no update leakage, this is only true when observed collectively for all update tokens. However, when a document and it's associated keywords are added or deleted at a time, for each such update operation, the server can learn the number of keywords present in the document by simply monitoring the number of update tokens. This issue can be easily avoided by "batching" multiple update tokens together – a common technique from the SSE literature [17]. The idea is to store the update tokens in a small client-side buffer. Once the buffer is full with tokens, all of them are sent to the server together for execution at one go. The adversary will not know if the update tokens are for a partial, whole, or multiple documents. This hides the number of keywords present in the individual documents, and also reduces the communication overhead.

5.1.2 Search leakage

Single Search Leakage For a conjunctive search query $\omega = w_1 \wedge w_2 \wedge \cdots \wedge w_n$, w_1 is the *s-term* and the search token is $(\Pi.\mathsf{stk}, \Pi.\mathsf{xtk})$.

1. From $\Pi.\mathsf{stk} = \Sigma.\mathsf{stk}$, the server will know $\mathsf{sp}(w_1)$, $\mathsf{TimeDB}(w_1)$ and $\mathsf{Udts}(w_1)$, as these are leakages of Σ (see Definition 7). Note that $\mathsf{delHist}(w_1)$ shall not be leaked as we call $\Sigma.\mathsf{Udt}_{\mathcal{C}}$ with $\mathsf{op} = \mathsf{add}$. The actual update operation is appended to the id. So even if the id in two updates are the same, $(\mathsf{id}\|\mathsf{op})$ is always different for the same keyword. Thus, if a keyword is added and deleted from the same identifier, the id in both cases would be different, and the server cannot link the add and delete operations.
2. Next, the server computes $\mathsf{xtk}[j, i]^{y_j} g^{F_p^{(Z)}(k_Z, w_1\|j) \cdot x_{w_i} \cdot \left(\mathsf{xid} \cdot F_p^{(Z)}(k_Z, w_1\|j)^{-1}\right)}$, for all $j \in [\mathsf{cnt}[w_1]]$ and $i \in [2, n]$, using $\Pi.\mathsf{xtk}$ output by $\Gamma.\mathsf{Srch}$, where $x_{w_i} = F_p^{(X)}(k_X, w_i)$. The server then checks if the generated $\mathsf{xtag}_{i,j} = \mathsf{xtk}[j, i]^{y_j}$ is in XDB or not.
   (a) If $\mathsf{xtag}_{i,j} \in \mathsf{XDB}$, the server knows that the $\mathsf{xid} = F_p(k_X, \mathsf{id}\|\mathsf{op})$ that was added for the *s-term* must have also been added for the *x-term*. Thus, the server comes to know those timestamps when the pair $(\mathsf{op}, \mathsf{id})$ in the *s-term* matches other keywords in the conjunction.
   (b) Otherwise, $\mathsf{xtag}_{i,j} \notin \mathsf{XDB}$. Our non-modifiability setting ensures that this event will not occur in future updates. So the $\mathsf{xtag}_{i,j}$ computed as $\mathsf{xtk}[j, i]^{y_j}$ which

is not in XDB, does not convey any information other than the result of the conjunction to the server.

Thus, the whole leakage is subsumed by the leakage through $\mathsf{Udts}(\omega)$ and $\mathsf{TimeDB}(\omega)$.

Here we note that $\mathsf{delHist}(\omega)$ is never leaked as we only add an (id, w) pair using Σ, even in case of deletion for the protocol Π. So Σ only works with $\mathsf{op} = \mathsf{add}$, and the actual operation remains hidden from Σ and hence from the server.

We note that $\mathsf{TimeDB}(\omega)$ is a huge overestimation of the actual leakage of Π. The actual leakage of $\mathsf{TimeDB}(\omega)$ in our scheme is the following.

$$\begin{aligned}\overline{\mathsf{TimeDB}}(\omega) =& \mathsf{TimeDB}(w_1) \cup \{(u, \mathsf{id}) : (i \geq 2) \wedge (\mathsf{id}, *) \\ & \in \mathsf{TimeDB}(w_1) \wedge (u, \mathsf{add}, (\mathsf{id}, w_i)) \in Q\}.\end{aligned}$$

In other words, the timestamps of those identifiers are leaked for *x-term*s which are only present in the *s-term*.

Search-Search Leakage For $\alpha < \beta$, consider two search queries $q_\alpha = (\alpha, \mathsf{srch}_\alpha, \omega_\alpha)$ and $q_\beta = (\beta, \mathsf{srch}_\beta, \omega_\beta)$. Any search token $\Pi.\mathsf{xtk}$ contains

$$\mathsf{xtk}[j, i] = g^{F_p^{(Z)}(k_Z, w_1 \| j) \cdot F_p^{(X)}(k_X, w_i)}$$

for $j \in [\mathsf{cnt}[w_1]]$ and $i \in [2, n]$. Now say, ω_α and ω_β have the same *s-term*s. So the xtks are the same whenever w_1 and w_i are repeated in the two searches. Hence, the server learns if a pair (w_1, w_i) has appeared in multiple conjunctive search queries through the equality of xtks. This is true for any $X = w_{i_1} \wedge w_{i_2} \wedge \cdots \wedge w_{i_\ell}$, where $w_{i_j} \in \widetilde{\mathcal{L}}_\omega$ for $j \in [\ell]$. That is, for the least frequent keyword w_1 and an arbitrary conjunction of set of keywords in $\widetilde{\mathcal{L}}(\omega)$, the server comes to know the joint-search pattern

$$\mathsf{sp}(w_1, X) = \{u | (u, \mathsf{srch}, \omega) \in \mathsf{QS}, \omega = w_1 \wedge \cdots \wedge X \wedge \cdots\}.$$

Hence, for a conjunctive query $\omega = w_1 \wedge \cdots \wedge w_n$, the total search pattern leakage of our scheme is

$$\mathsf{sp}(\omega) = \mathsf{sp}(w_1) \cup \left(\cup_{i=2}^{n} \mathsf{sp}(w_1, w_i)\right).$$

Update-Search Leakage Consider a search query $q_\beta = (\beta, \mathsf{srch}_\beta, \omega_\beta)$. For $\alpha < \beta$, consider a previous update query $q_\alpha = (\alpha, \mathsf{op}_\alpha, (\mathsf{id}_\alpha, w_\alpha))$, where $\mathsf{op}_\alpha \in \{\mathsf{add}, \mathsf{del}\}$ and $\omega_\beta = w_1 \wedge w_2 \wedge \cdots \wedge w_n$. If $w_\alpha \notin \mathcal{L}(\omega_\beta)$, that is, if a keyword that was updated previously is not present in the current search query, then such update and search queries are unrelated and hence there is no additional leakage. If $w_\alpha \in \mathcal{L}(\omega_\beta)$, the leakage is precisely that captured by the single search leakage described above, as an update operation does not leak anything.

All of the above together show that the search leakage for our scheme is subsumed by the following leakage we have assumed for the proof of Theorem 1.

$$\widehat{\mathcal{L}_{\Pi,\mathsf{Srch}}} = \left(\mathsf{sp}(\omega), \overline{\mathsf{TimeDB}}(\omega), \mathsf{Udts}(\omega)\right).$$

Leakage Profile of Π We finally summarise the overall leakage of our scheme, by combining the leakage of Σ and Γ. The actual leakage of Π is at most

$$\widehat{\mathcal{L}_{\Pi,\mathsf{Stp}}} = \emptyset, \quad \widehat{\mathcal{L}_{\Pi,\mathsf{Udt}}} = \emptyset,$$
$$\widehat{\mathcal{L}_{\Pi,\mathsf{Srch}}} = \left(\mathsf{sp}(\omega), \overline{\mathsf{TimeDB}}(\omega), \mathsf{Udts}(\omega)\right).$$

6 Implementation and experimental results

Implementation We instantiate Π using three D-SSE schemes Σ – FAST [46], Mitra [26], and Π_{BP} [20] – and experimentally compare their performances. Among them, FAST is only forward private, and thus FAST-OXT only achieves forward privacy in NMD setting. However, very efficient, with a constant size search token sent from the client to the server. In comparison, Mitra is forward and backward private but has a search token size of $\mathcal{O}(u)$, where u is the number of updates performed for the search keyword. Π_{BP} [20] extended FAST [46] to make it backward private. Our instantiations are called FAST-OXT, Mitra-OXT and Π_{BP}-OXT respectively. Mitra-OXT is the ODXT construction of [41], while the other two are new schemes. Π_{BP}-OXT has a shorter search token size than Mitra-OXT [41] due to the use of Π_{BP} as the D-SSE in place of Mitra. We have implemented our schemes in C++; our code is available at [8]. We conducted experiments on the Enron Email database[24] to observe that all three implementations are practical and scalable.

Experiments All our experiments were performed on a desktop computer with Intel Core i7-6700 CPU, 3.40 GHz with 8 Cores and 16GB RAM. The database was stored using RocksDB [22], and all cryptographic primitives were implemented using Crypto++[2]. We report our results in a similar fashion as in [16, 41, 56]. We have used the Enron Email Dataset[24] for our experiments. In [20, 46], when two consecutive queries have the same keyword, the server stores the previous search result to optimise the search. However, for a fair comparison of the time taken by the three schemes, we have not implemented this optimisation – all times reported are for the fresh search of the *s-term*s. We do not report our search results with different probabilities of search operation. However, the modification suggested in [20, 46] can also be applied for single keyword searches, which will improve the search performance of the corresponding implementations.

EDB Creation and Update Operation The EDB was created using the Enron Email Dataset. We wrote a C++ program to extract IDs and keywords from the dataset. We got 517, 401 IDs and 72, 252, 961 ID-keyword pairs. Table 2 shows the sizes of the encrypted databases and clients' state sizes for different schemes. We also report the total time and the time per pair, required to create the EDB. The EDB creation essentially calls the update process of our protocol (without the network delay), so the average update time per pair was calculated while generating the EDB, and is reported in Table 2. The update time for an id-keyword pair is almost constant in all three instantiations. We have performed an independent experiment by inserting 10^4 pairs and taking the average time. The experiment also matches the result reported in Table 2.

Search Operation To evaluate the performance of search operations, we follow the same setting as in [41], and [56]. We report the time for search queries that are conjunctions of two keywords ($w_1 \wedge w_2$). We have conducted two types of search operations,

1. In the first experiment, we keep the size of the number of updates performed on the second keyword to a constant $|\mathsf{Udts}(w_2)| = 10^5$ and vary $|\mathsf{Udts}(w_1)|$ from 100 to 10^5 (Fig. 4). The exact values are in Table 3
2. In the second case, we keep the size of the number of updates performed on the first keyword to a constant, that is, $|\mathsf{Udts}(w_1)| = 10$ and vary $|\mathsf{Udts}(w_2)|$ from 10 to 10^5 (Fig. 5).

In both cases, we report the search time. The time required to search two keywords both having order of 10^5 updates takes $\sim$ 80 sec for FAST-OXT, $\sim$ 66 sec for Mitra-OXT and $\sim$ 52 sec for Π_{BP}-OXT (Fig. 4). Our results concur with the results from [20], as Mitra-OXT outperforms Π_{BP}-OXT when the search output size is small and Π_{BP}-OXT outperforms Mitra-OXT for large search output sizes. The search time is high for FAST-OXT as decryption is done at the server.

We did not perform searches with more than $n = 2$ keywords in the conjunction as the search for elements on XSet can be *parallelised* and hence will essentially take similar time as a conjunctive query with 2 keywords. This scaling of the performance of Π with increasing number of keywords in the conjunction ($n > 2$) is explained in more detail later.

In [56], an SSE is used to store the entries of *s-term* and *x-term* as well. The implementation of [56] is not available. However, as reported in their paper, they did their experiment on a Ubuntu 20.04.3 LTS workstation with Intel @Xeon(R) W-2123 CPU 3.60GHz with 8 cores and 32GB RAM. Their setup uses more processing power with a double-sized RAM. As reported in their paper, their protocol has a higher running time when the number of updates for both w_1 and w_2 is high. This is understandably so, as for each *s-term* and *x-term* they have to access an SSE. However, their running time overshoots our scheme by a huge margin when the number of updates for the *s-term* is small but the number of updates for the *x-term* is high. This is because we only fetch very few *x-term*s (precisely the number of times the *s-term* has been updated) whereas they have to fetch all the *x-term*s related to the keyword. Similarly, as reported in [54], they only test their scheme for a database with 23,643 documents, 60,879 keywords, and 8,373,977 keyword-document pairs, which is almost a magnitude smaller than our test case. Their server machines were running Ubuntu 18.04 LTS and had 16 cores (Intel Core i9-9900 CPU 3.10 GHz), 31GB RAM, and 483 GB SSD disk space. However, their running time is also similar to our results when the matching search output size is small and their search time grows rapidly when the matching search output size increases, making their scheme much less suitable for large databases compared to ours.

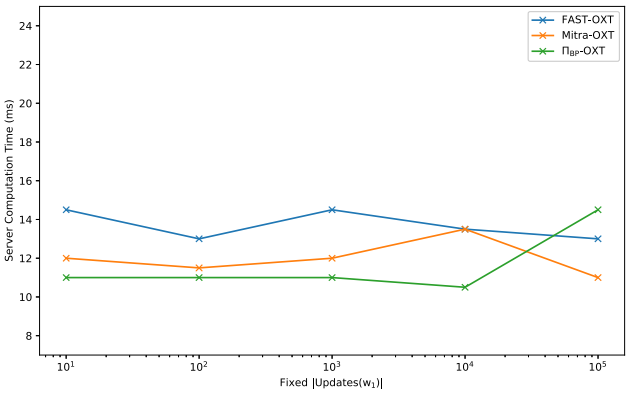

Fig. 4 Search time comparison between the three instantiations of our generic CD-SSE for fixed $|\text{Updates}(w_2)| = 10^5$, while varying $|\text{Updates}(w_1)|$ from 10^2 to 10^5

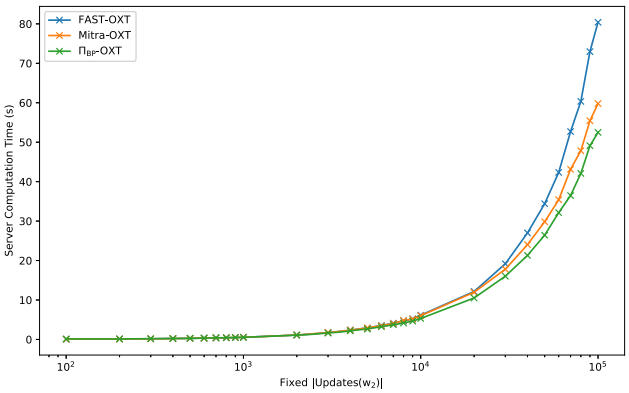

Fig. 5 Search time comparison between the three instantiations of our generic CD-SSE for fixed $|\text{Updates}(w_1)| = 10$, while varying $|\text{Updates}(w_2)|$ from 10 to 10^5

Choice of Σ. Not all D-SSEs are backward private. According to Corollary 1, if Σ is only forward private, then Π is also only forward private in the NMD setting. This makes Π (backward) compatible with existing/new forward private

Table 2 Storage cost and time required for each update

Schemes	Time (Hr.)	Time/Pair (ms)	Server Storage (GB)	Client Storage (MB)
FAST-OXT	12.6	0.63	15.5	20
Mitra-OXT	10.4	0.52	9.1	2
Π_{BP}-OXT	11.1	0.55	13.3	3.1

Table 3 Search time comparison for fixed $|\mathsf{Udts}(w_2)| = 10^5$ with varying $|\mathsf{Udts}(w_1)|$ from 10^2 to 10^5

$\lvert\mathsf{Udts}(w_1)\rvert$	Mitra-OXT (s)	FAST-OXT (s)	Π_{BP}-OXT (s)
10^2	0.06	0.06	0.13
10^3	0.57	0.54	0.53
10^4	6.01	6.15	5.31
10^5	59.79	80.41	52.49

only D-SSE schemes Σ. Since no practical attacks exploiting the absence of backward privacy in SSE are currently known, applications that do not explicitly require backward privacy can directly integrate into Π with their existing and already deployed forward private SSE schemes, thereby benefiting from our construction without requiring a complete redesign.

In our experiments, we demonstrate this flexibility and (backward) compatibility of Π to achieve efficiency at various security levels. We have chosen three D-SSE schemes FAST [46], Mitra [26], and Π_{BP} [20] as Σ in Π. Among them, FAST is only forward private and the other two are both forward and backward private with different efficiency tradeoffs. So FAST-OXT is only forward private and not backward private. While FAST [47] and Π_{BP} [20] achieve better client-side computation than Mitra [26], they differ in other performance aspects. Since FAST is a response-revealing scheme, the server immediately learns the search result and therefore does not require the additional round of communication needed by Mitra and Π_{BP}. In contrast, Π_{BP} achieves the lowest overall computation time among these schemes.

Scaling. Our framework utilises the OXT scheme as a modular black-box. For a search query, the computation and communication overhead of Π for larger conjunctions with $n > 2$ keywords, scale from the $n = 2$ case of the underlying OXT construction. For a search query with $n \geq 2$ keywords in the conjunction, every additional keyword in the conjunction would require sending an additional cross-token array $(\mathsf{xtk}[j, i])$ with $\mathcal{O}(u_1)$ elements from the client to the server (see line no 1. of $\Gamma.\mathsf{Srch}_{\mathcal{C}}$ in Fig 2), and the server has to check the $\mathcal{O}(u_1)$ xtags for those elements in the $\mathsf{xtk}[j, i]$ array. However, the generation and checking of those cross-tokens (i.e. loop 2.B of $\Gamma.\mathsf{Srch}_{\mathcal{S}}$ in Fig 2) can be performed in parallel. Thus, like the OXT protocol, the communication cost of Π grows linearly with the number of keywords (i.e., $\mathcal{O}(n.u_1)$, see Table 1) in the conjunction. However, the computation cost of Π can be reduced to the $n = 2$ keyword case using parallelisation.

The client-side computation (token generation) and the server-side computation of Π are identical operations, with an additional xtag checking at the server. All these operations can be parallelised at both ends. Thus, the end-to-end latency of Π for general $n \geq 2$ is the same as that of $n = 2$. Furthermore, our implementation keeps the number of updates for every keyword as a client's state (see Line No 1 of protocol $\Gamma.\mathsf{Srch}_{\mathcal{C}}$ in Fig 2). This storage is essential for any forward private SSE scheme. However, due to this additional information, we were able to eliminate the need for an explicit server-side "stop" signal used in the original OXT protocol, thereby optimising the round-trip communication and minimizing end-to-end latency.

In summary, the results for general $n \geq 2$ may be easily extrapolated from the $n = 2$ results.

7 Conclusion

We have proposed a generic dynamic searchable symmetric encryption scheme Π that allows conjunctive queries while being both forward and backward private. Our scheme is specifically crafted to protect *non-modifiable* documents with a fixed set of keywords, while the document itself may be edited. This is a common setting that is relevant for a variety of applications, including biometric databases, digital archives, legal documents, medical records, etc. Our construction allows the keyword set associated with a document to be modified with some extra overhead. With efficiency closest to OXT (among known dynamic schemes) and by ensuring both forward and backward privacy, our proposed construction offers an ideal solution for preserving the privacy of dynamic datasets while facilitating conjunctive queries.

Our construction Π is modular and generic. It can be instantiated with a suitable D-SSE Σ, to get a new CD-SSE. Given the well known difficulty of constructing efficient and secure (forward and backward private) CD-SSEs, our generic method should be useful for many practical scenarios that fit the NMD model. It gives us Π_{BP}-OXT that is more efficient than ODXT or Mitra-OXT in every parameter.

Recent work [50] introduces a forward and backward private CD-SSE scheme that employs a novel chain structure to

address the forward privacy limitation of [41]. Their chaining structure also reduces the size of the search token, though at the cost of increased search time. Notably, all of this is achieved under a non-modifiable document setting, which they do not explicitly state. In that same setting, [41] is also secure. Furthermore, while their definition of security aligns with ours, it does not incorporate the recommendations of [20]. Finally, our rigorous leakage analysis is missing in most of the previous work.

Future Directions. The recent C-SSE scheme ConjFilter [40] is similar to [16]. However, [40] precomputes all possible 2-conjunctions to facilitate conjunctive search. This is difficult to achieve in a dynamic setting where the database is not known beforehand. Using [40] for Γ in Π instead of OXT may give us the first CD-SSE to use symmetric key primitives only.

The OXT framework has been extended to support multi-client settings in a modular black-box fashion [30]. Our construction also uses OXT in a black-box manner, without altering its functionality. Therefore, our generic scheme may be extended to support multi-clients. We leave this as another future work.

Acknowledgements Authors like to thank Dr Neil Glackin and his team at Intelligent Voice for the helpful discussions.

Author Contributions Dr. Abdelraheem and Dr. Majumder contributed to the design of the protocol. Dr. Bhattacherjee and Dr. Majumder were responsible for defining the framework, formalizing the security notations, and developing the mathematical proofs of the protocol. Mr. Henault handled the implementation aspects of the work.
All authors reviewed the manuscript and contributed to the finalization of the paper.

Funding No funding for this work.

Data Availability No datasets were generated or analysed during the current study.

Declarations

Conflict of interest The authors declare no conflict of interest.

Ethical approval The authors declare full compliance with ethical standards. This article does not contain any studies involving humans or animals performed by any of the authors.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Article 25 of GDPR as on 2026/08/18 06:06:03. https://gdpr-info.eu/art-25-gdpr/. Accessed 2026/08/18 06:06:03
2. The crypto++ library. https://www.cryptopp.com/
3. Abdelraheem, M. A., Gehrmann, C., Lindström, M., Nordahl, C.: Executing boolean queries on an encrypted bitmap index. In *Proceedings of the 2016 ACM on Cloud Computing Security Workshop*, CCSW '16, page 11–22. Association for Computing Machinery, (2016)
4. Agrawal, S., Banerjee, S., Sharma, S.: Privacy and security of aadhaar: A computer science perspective. Econ. Pol. Wkly **52**(37), 93–102 (2017)
5. Agrawal, S., Bhattacherjee, S., Phan, D. H., Stehlé, D., Yamada S.: Efficient public trace and revoke from standard assumptions: Extended abstract. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 2277–2293. Association for Computing Machinery, (2017)
6. Anand, N.: New principles for governing aadhaar: Improving access and inclusion, privacy, security, and identity management. Journal of Science Policy & Governance **18**(01), 1–14 (2021)
7. Anderson, J.C.: Transmitting legal documents over the internet: How to protect your client and yourself. Rutgers Computer & Tech. LJ **27**, 1 (2001)
8. Anonymous. Our implementations. https://github.com/anonCod/CD-SSE
9. Bag, A., Talapatra, D., Rastogi, A., Patranabis, S., Mukhopadhyay, D.: Two-in-one-sse: Fast, scalable and storage-efficient searchable symmetric encryption for conjunctive and disjunctive boolean queries. Proc. Priv. Enhancing Technol. **2023**(1), 115–139 (2023)
10. Bhattacherjee, S., Sarkar, P.: Complete tree subset difference broadcast encryption scheme and its analysis. Des. Codes Cryptogr. **66**(1–3), 335–362 (2013)
11. Joppe, W., Bos, K., Lauter, M.: Naehrig: Private predictive analysis on encrypted medical data. Special Issue on Informatics Methods in Medical Privacy **50**, 234–243 (2014). (**Journal of Biomedical Informatics**)
12. Bost, R.: $\sum o\varphi o\varsigma$: Forward secure searchable encryption. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 1143–1154. Association for Computing Machinery, (2016)
13. 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*, CCS '17, page 1465–1482. Association for Computing Machinery, (2017)
14. Cash, D., Grubbs, P., Perry, J., Ristenpart, T.: Leakage-abuse attacks against searchable encryption. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 668–679. Association for Computing Machinery, (2015)
15. Cash, D., Jaeger, J., Jarecki, S., Jutla, C.S., Krawczyk, H., Rosu, M.-C., Steiner, M.: Dynamic searchable encryption in very-large databases: Data structures and implementation. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*. The Internet Society, (2014)
16. Cash, D., Jarecki, S., Jutla, C., Krawczyk, H., Roşu, M.-C., Steiner, M.: Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology – CRYPTO 2013*, pages 353–373. Springer Berlin Heidelberg, (2013)
17. Chakraborty, D., Majumder, A., Samajder, S.: Making searchable symmetric encryption schemes smaller and faster. Int. J. Inf. Secur. **24**(1), 10 (2025)

18. Chang, Y.-C., Mitzenmacher, M.: Privacy preserving keyword searches on remote encrypted data. In John Ioannidis, Angelos D. Keromytis, and Moti Yung, editors, *Applied Cryptography and Network Security, Third International Conference, ACNS 2005, New York, NY, USA, June 7-10, 2005, Proceedings*, volume 3531 of *Lecture Notes in Computer Science*, pages 442–455, (2005)
19. Chase, M., Kamara, S.: Structured encryption and controlled disclosure. In *Advances in Cryptology - ASIACRYPT 2010*, pages 577–594, Berlin, Heidelberg, (2010). Springer Berlin Heidelberg
20. Chatterjee, S., Puria, S.K.P., Shah, A.: Efficient backward private searchable encryption. J. Comput. Secur. **28**(2), 229–267 (2020)
21. Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: Improved definitions and efficient constructions. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, page 79–88, New York, NY, USA, (2006). Association for Computing Machinery
22. Dong, S., Kryczka, A., Jin, Y.: and Michael Stumm. Evolution of development priorities in a key-value store serving large-scale applications, Rocksdb (2021)
23. Dou, H., Dan, Z., Peng, X., Wang, W., Shuning, X., Chen, T., Jin, H.: Dynamic searchable symmetric encryption with strong security and robustness. IEEE Trans. Inf. Forensics Secur. **19**, 2370–2384 (2024)
24. Corp, E., Cohen, W.W.: Enron email dataset. United States Federal Energy Regulatory Commissioniler, (2015) comp
25. Faber, S., Jarecki, S., Krawczyk, H., Nguyen, Q., Rosu, M.-C., Steiner, M.: Rich queries on encrypted data: Beyond exact matches. In *Computer Security - ESORICS 2015 - 20th European Symposium on Research in Computer Security, Vienna, Austria, September 21-25, 2015, Proceedings, Part II*, volume 9327 of *Lecture Notes in Computer Science*, pages 123–145. Springer, (2015)
26. Ghareh Chamani, J., Papadopoulos, D., Papamanthou, C., Jalili, R.: New constructions for forward and backward private symmetric searchable encryption. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 1038–1055. Association for Computing Machinery, (2018)
27. Guo, C., Li, W., Tang, X.: Kim-Kwang Raymond Choo, and Yining Liu. Forward private verifiable dynamic searchable symmetric encryption with efficient conjunctive query, IEEE Transactions on Dependable and Secure Computing (2023)
28. Chengyu, H., Song, X., Liu, P., Xin, Y., Yuqin, X., Duan, Y., Hao, R.: Forward secure conjunctive-keyword searchable encryption. IEEE Access **7**, 35035–35048 (2019)
29. Islam, M.S., Kuzu, M., Kantarcioglu, M.: Access pattern disclosure on searchable encryption: ramification, attack and mitigation. In NDSS **20**, 12 (2012)
30. Jarecki, S., Jutla, C., Krawczyk, H., Rosu, M., Steiner, M.: Outsourced symmetric private information retrieval. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, page 875–888, New York, NY, USA, (2013). Association for Computing Machinery
31. Jutla, C.S. Patranabis, S.: Efficient searchable symmetric encryption for join queries. In *Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part III*, volume 13793 of *Lecture Notes in Computer Science*, pages 304–333. Springer, (2022)
32. Kamara, S., Moataz, T.: Boolean searchable symmetric encryption with worst-case sub-linear complexity. In *Advances in Cryptology – EUROCRYPT 2017*, pages 94–124. Springer International Publishing, (2017)
33. Kamara, S., Papamanthou, C., Roeder, T.: Dynamic searchable symmetric encryption. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, page 965–976, New York, NY, USA, (2012). Association for Computing Machinery
34. Kindt, E. J.: *Privacy and data protection issues of biometric applications*, volume 1. Springer, (2016)
35. Lai, S., Patranabis, S., Sakzad, A., Liu, J. K., Mukhopadhyay, D., Steinfeld, R., Sun, S.-F., Liu, D., Zuo, C.: Result pattern hiding searchable encryption for conjunctive queries. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 745–762. Association for Computing Machinery, (2018)
36. Liu, C., Zhu, L., Wang, M., Tan, Y.A.: Search pattern leakage in searchable encryption Attacks and new construction. Inf. Sci. **265**, 176–188 (2014)
37. Judith, A.: Markowitz: Voice biometrics. Commun. ACM **43**(9), 66–73 (2000)
38. Naor, D., Naor, M., Lotspiech, J.: Revocation and tracing schemes for stateless receivers. In *Advances in Cryptology — CRYPTO 2001*, pages 41–62, Berlin, Heidelberg, (2001). Springer Berlin Heidelberg
39. Ntoulas, A., Cho, J.: Pruning policies for two-tiered inverted index with correctness guarantee. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, page 191–198, New York, NY, USA, (2007). Association for Computing Machinery
40. Patel, S., Persiano, G., Seo, J. Y., Yeo, K.: Efficient boolean search over encrypted data with reduced leakage. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part III*, volume 13092 of *Lecture Notes in Computer Science*, pages 577–607. Springer, (2021)
41. Patranabis, S., Mukhopadhyay, D.: Forward and backward private conjunctive searchable symmetric encryption. In *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*. The Internet Society, (2021)
42. Sabry, M., Samavi, R.: ArchiveSafe LT: Secure long-term archiving system. In *Proceedings of the 38th Annual Computer Security Applications Conference*, ACSAC '22, page 936–948, New York, NY, USA, (2022). Association for Computing Machinery
43. Sabry, M., Samavi, R., Stebila, D.: Archivesafe: Mass-leakage-resistant storage from proof-of-work. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 89–107, Cham, (2020). Springer International Publishing
44. Scheibner, J., Raisaro, J.L., Troncoso-Pastoriza, J.R., Ienca, M., Fellay, J., Vayena, E., Hubaux, J.-P.: Revolutionizing medical data sharing using advanced privacy-enhancing technologies: Technical, legal, and ethical synthesis. J. Med. Internet Res. **23**(2), e25120 (2021)
45. Song, D. X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In *Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000*, pages 44–55, (2000)
46. Song, X., Dong, C., Yuan, D., Qiuliang, X., Zhao, M.: Forward private searchable symmetric encryption with optimized I/O efficiency. IEEE Trans. Dependable Secure Comput. **17**(5), 912–927 (2020)
47. Stefanov, E., Papamanthou, C., Shi, E.: 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*. Internet Society, (2014)
48. Sun, S.-F., Yuan, X., Liu, J K., Steinfeld, R., Sakzad, A., Vo, V., Nepal, S.: Practical backward-secure searchable encryption from symmetric puncturable encryption. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 763–780. Association for Computing Machinery, (2018)
49. Sun, S., Steinfeld, R., Lai, S., Yuan, X., Sakzad, A., Liu, J.K., Nepal, S., Gu, D.: Practical non-interactive searchable encryption with forward and backward privacy. In *28th Annual Network*

and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021. The Internet Society, (2021)
50. Wang, B., Li, Y., Chen, J., He, K., Jia, M., Du, R.: Forward and backward private conjunctive dynamic searchable symmetric encryption with refined leakage function and low communication. IEEE Trans. Inf. Forensics Secur. **20**, 8224–8236 (2025)
51. Wang, N., Junsong, F., Bhargava, B.K., Zeng, J.: Efficient retrieval over documents encrypted by attributes in cloud computing. IEEE Trans. Inf. Forensics Secur. **13**(10), 2653–2667 (2018)
52. Wang, Y., Wang, J., Sun, S., Miao, M., Chen, X.: Toward forward secure sse supporting conjunctive keyword search. IEEE Access **7**, 142762–142772 (2019)
53. Zhiqiang, W., Li, K.: Vbtree: forward secure conjunctive queries over encrypted data for cloud computing. VLDB J. **28**(1), 25–46 (2019)
54. Yuan, D., Zuo, C., Cui, S., Russello, G.: Result-pattern-hiding conjunctive searchable symmetric encryption with forward and backward privacy. Proc. Priv. Enhancing Technol. **2023**(2), 40–58 (2023)
55. Zhang, Y., Katz, J., Papamanthou, C.: All your queries are belong to us: The power of file-injection attacks on searchable encryption. In *Proceedings of the 25th USENIX Conference on Security Symposium*, SEC'16, page 707–720. USENIX Association, (2016)
56. Zuo, C., Lai, S., Yuan, X., Liu, J.K., Shao, J., Wang, H.: Searchable encryption for conjunctive queries with extended forward and backward privacy. *IACR Cryptol. ePrint Arch.*, page 1585, (2021)
57. Zuo, C., Sun, S.-F., Liu, J. K., Shao, J., Pieprzyk, J.: Dynamic searchable symmetric encryption with forward and stronger backward privacy. In *Computer Security – ESORICS 2019*, pages 283–303. Springer International Publishing, (2019)
58. Zuo, C., Sun, S., Liu, J. K., Shao, J., Pieprzyk, J., Wei, G.: Forward and backward private dynamic searchable symmetric encryption for conjunctive queries. *IACR Cryptol. ePrint Arch.*, page 1357, (2020)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.