Backward Private DSSE: Alternative Formulations of Information Leakage and Efficient Constructions

Sanjit Chatterjee, Shravan Kumar Parshuram Puria, and Akash Shah

Department of Computer Science and Automation,
Indian Institute of Science, Bangalore, India.
E-mail: {sanjit, shravan, shaha}@iisc.ac.in

Abstract

Dynamic Searchable Symmetric Encryption (DSSE), apart from providing support for search operation, allows a client to perform update operations on outsourced database efficiently. Two security properties, viz., forward privacy and backward privacy are desirable from a DSSE scheme. The former captures that the newly updated entries cannot be related to previous search queries and the latter ensures that search queries should not leak matching entries after they have been deleted. These security properties are formalized in terms of the information leakage that can be incurred by the respective constructions. Existing backward private constructions either have a non-optimal communication overhead or they make use of heavy cryptographic primitives. Our main contribution consists of three efficient backward private schemes that aim to achieve practical efficiency by using light weight symmetric cryptographic components only. In the process, we also revisit the existing definitions of information leakage for backward privacy [Bost et al. CCS'17] and propose alternative formulations. Our first construction Π_{BP}-prime achieves a stronger notion of backward privacy whereas our next two constructions Π_{BP} and Π_{WBP} achieve optimal communication complexity at the cost of some additional leakage. The prototype implementations of our schemes depict the practicability of the proposed constructions and indicate that the cost of achieving backward privacy over forward privacy is substantially small.

1 Introduction

Due to a variety of crucial benefits, enterprises outsource their data to cloud resident storage. If the outsourced data is stored as plaintext on remote servers then it may be intercepted by adversaries. Hence, data is stored in encrypted form on remote servers. However, if the client has to decrypt the data in order to get results for a search query, it defeats the purpose of outsourcing data. Generic tools such as fully homomorphic encryption [19] or oblivious RAM [22, 39] can be considered to construct protocols that leak almost no information to the server. But as of now, these tools are costly for large databases and hence, are impractical.

A practical solution to this problem is Searchable Symmetric Encryption (SSE) [36, 14, 13, 11] that trades efficiency for security. Dynamic Searchable Symmetric Encryption (DSSE) [27, 10, 33] adds a vital feature to static SSE schemes, i.e., the ability for the client to efficiently perform update operations remotely on the outsourced database with the guarantee that minimal information is leaked to the server in the process. These constructions of (D)SSE that aim to achieve an acceptable balance between security and performance, explicitly describe the leakage profile and formally prove that the information leaked from the scheme is bounded by the leakage profile.

Simultaneously with the works on constructing SSE schemes with improved efficiency, security and expressiveness of queries [11, 26, 16], there is another line of work that shows the real-world consequences of these leakages [9, 43, 2]. Zhang et al. [43] through *file injection attack* showed that it is possible to reveal the contents of past search queries of DSSE schemes with a few injection of documents and Abdelraheem et al. [2] showed that the consequences of this attack is even more devastating in the case of relational databases.

Because of the file injection attack, forward privacy has garnered significant interest in the research community. The notion of forward privacy was introduced in [38], while it was first formalized in [5]. In hindsight, the first DSSE scheme that satisfied the notion of forward privacy was proposed in 2005 [12]. Along with forward privacy, Stefanov et al. [38] asserted that backward privacy should also be satisfied by a DSSE scheme. Informally, backward privacy states that search queries should not leak matching entries after they have been deleted. The notion of backward privacy was first formalized by Bost at el. [7].

1.1 Related Work

Kamara et al. [27] proposed the first sublinear (in the size of the database) DSSE scheme. Forward private scheme $\Sigma o\phi o\zeta$ [5] achieves optimal communication complexity (linear in the size of result set) but it makes use of asymmetric cryptographic primitives and does not support parallel processing. A GGM-PRF [21] based forward private DSSE scheme Diana was proposed in [7]. Diana makes use of symmetric cryptographic primitives only and supports parallel processing but doesn't have optimal computational and communication complexity. Asymptotically optimal forward private DSSE schemes that make use of symmetric cryptographic primitives only and support parallelism were proposed in [29, 15, 37]. Further, FASTIO scheme [37] ensures a reasonable locality [10], a measure of I/O efficiency.

The notion of backward privacy was first formally described in [7], through three different security definitions, ordered from most to least secure called respectively BPIP, BPUP and WBP. Bost et al. [7] proposed a generic way to achieve backward privacy from any forward private DSSE scheme. However, the communication complexity of the derived backward private scheme isn't optimal. In [7], a backward private scheme $\mathsf{Diana}_{\mathsf{del}}$ based on constrained pseudo-random function (CPRF) [8, 4, 28] and a backward private Janus framework based on a puncturable encryption scheme with a particular incremental update property [25], were also proposed. The communication and computational complexity of search and update protocols of $\mathsf{Diana}_{\mathsf{del}}$ are not optimal. The search protocol in Janus is single-roundtrip and has an optimal communication complexity. However, the computational complexity of search protocol is $O(n_w \cdot d_w)$, where n_w and d_w respectively denote the number of documents matching keyword w and delete operations performed on keyword w. As acknowledged in [7], with just a few hundred deletions per keyword, Janus will not be practical because of both computational and storage overhead reasons. Further, [7] has imposed the following restriction on $\mathsf{Diana}_{\mathsf{del}}$ and Janus: reinsertion of document-keyword pairs that were previously deleted is not allowed. We refer to this constraint as *reinsertion restriction* in the rest of our paper.

While a previous version of this work was under submission in a conference, there appeared two works on backward private DSSE [20, 40]. Chamani et al. [20] proposed a practically efficient BPUP-secure scheme called Mitra. They also proposed two other backward private schemes Orion and Horus that achieve quasi-optimal (linear in n_w upto a logarithmic factor) search computation complexity but make use of Path ORAM [39] and as a result are impractical for large databases [32]. Sun et al. [40] proposed a symmetric puncturable encryption (SPE) scheme and instantiated the Janus framework with the proposed SPE scheme. We provide a comprehensive comparative analysis of the performance and security of these schemes vis à vis our proposed schemes in Section 4.5.

1.2 Our Contributions

We start with revisiting the notion of information leakage in the context of backward privacy. In our investigation, we identify that *Weak Backward Privacy* (WBP) notion proposed by Bost et al. [7] should only be used to argue backward privacy in reinsertion restriction setting. We propose two alternative formulations of information leakage, viz., BP-I and BP-II to capture the notion of backward privacy. BP-I is strong in the sense that, for a search query on keyword w, it allows only the timestamp of the update queries on w and the identifiers of the documents containing w to be leaked. On the other hand, BP-II relaxes the definition for backward privacy by allowing some additional leakage.

Our main contribution consists of three backward private schemes $\Pi_{\mathsf{BP}\text{-}\mathsf{prime}}$, Π_{BP} and Π_{WBP} that are BP-I, BP-II and WBP secure respectively. We start with a simple forward private scheme Π_{FP}, that serves as a building block for our backward private schemes. The construction $\Pi_{\mathsf{BP}\text{-}\mathsf{prime}}$ is similar to the approach of

applying the generic transformation to achieve backward privacy [7] from forward private scheme Π_{FP}. Due to its use of only light weight symmetric components and low leakage level, Π_{BP}-prime can be a suitable candidate for adoption in practice. However, one limitation of Π_{BP}-prime is that the communication complexity isn't optimal. We address this issue in our main construction Π_{BP}. Π_{BP} makes use of tags generated using a pseudo random permutation (PRP) which ensures that only the tags corresponding to the set of documents currently matching the keyword w are returned to client. Thus, Π_{BP} avoids unnecessary communication overhead, while at the same time ensuring that any information violating the notion of backward privacy is not leaked to the server. To the best of our knowledge, Π_{BP} is the first practical backward private scheme that has optimal update and search communication complexity and uses symmetric cryptographic primitives only. Further, Π_{BP} is easily parallelizable, provides reasonable locality and allows reinsertion of document-keyword pair. With a simple modification in Π_{BP}, we construct a single roundtrip weak backward private scheme Π_{WBP} that improves upon the concrete communication overhead in Search protocol by around 40%. All our constructions are forward private as well. A comparison of our schemes with some prior and concurrent works [7, 20, 40] is provided in Table 1.

Schemes	Computation		Communication			Backward Privacy
	Search	Update	Search	Update	# Rounds	
Fides [7]	$O(o'_w)$	$O(1)$	$O(o'_w)$	$O(1)$	2	BPUP
$\mathsf{Diana}_{\mathsf{del}}$ [7]	$O(a_w)$	$O(log(a_w))$	$O(n_w + d_w log(a_w))$	$O(1)$	2	WBP
Janus [7]	$O(n_w \cdot d_w)$	$O(1)$	$O(n_w)$	$O(1)$	1	WBP
Mitra [20]	$O(o'_w)$	$O(1)$	$O(o'_w)$	$O(1)$	2	BPUP
Orion [20]	$O(n_w log^2(N))$	$O(log^2(N))$	$O(n_w log^2(N))$	$O(log^2(N))$	$O(log(N))$	BPIP
Horus [20]	$O(n_w log(d_w) log(N))$	$O(log^2(N))$	$O(n_w log(d_w) log(N))$	$O(log^2(N))$	$O(log(d_w))$	WBP
Janus++ [40]	$O(n_w \cdot d)$	$O(d)$	$O(n_w)$	$O(1)$	1	WBP
Π_{BP}-prime (Our Work)	$O(o'_w)$	$O(1)$	$O(o'_w)$	$O(1)$	2	BP-I
Π_{BP} (Our Work)	$O(o'_w)$	$O(1)$	$O(n_w)$	$O(1)$	2	BP-II
Π_{WBP} (Our Work)	$O(o'_w)$	$O(1)$	$O(n_w)$	$O(1)$	1	WBP

All the constructions are also forward private. The client storage for all the constructions is $O(mlog(n))$ except Orion, where the corresponding complexity is $O(1)$. Refer Section 2.4 for the notations used. Generally, $o'_w < n_w \cdot d_w$, as the former term has an additive factor whereas the latter term has a multiplicative factor.

Table 1: Comparison of backward schemes Π_{BP}-prime, Π_{BP} and Π_{WBP} with some prior and concurrent works.

Π_{FP} is the most efficient forward private scheme in literature. Our implementation results show that the performance of Π_{BP} is comparable to Π_{FP}. For example, the time taken by the search protocol of Π_{BP} for a search that returned 150,000 results is around 0.68 seconds as compared to 0.54 seconds in Π_{FP}.

We also introduce a desirable property for DSSE schemes called *inverse backward privacy*. This property ensures that a search query should leak no information about the identifier of the document that had been deleted and re-inserted later.

2 Notations and Definitions

The security parameter is denoted by λ. All procedures in our construction implicitly take λ as input. By efficient, we mean probabilistic polynomial-time in λ. All the algorithms (including adversaries and simulators) are assumed to be efficient unless otherwise specified. A function f: $\mathbb{N} \to \mathbb{R}$ is said to be a negligible function iff for all $c > 0$, $\exists\ n_0 \in \mathbb{N}$ such that $\forall n \geq n_0$, $f(n) < n^{-c}$. The function $\mathsf{neg}(\lambda)$ denotes a negligible function in λ. For a finite set X, $x \xleftarrow{\$} X$ means that x is uniformly sampled from X and $|X|$ denotes the cardinality of set X. $x \leftarrow y$ denotes that variable x is assigned the value of variable y and operator $\|$ denotes concatenation. For a data structure DS, $|\mathsf{DS}|$ denotes the memory space occupied by the data structure in bits. $\mathsf{addr}(\mathsf{D})$ denotes the address in memory at which data structure instance D is stored. $\perp$ denotes null value. For sets $X_1, \ldots, X_n$ and Y, $\mathsf{Func}(X_1 \times \cdots \times X_n, Y)$ denotes the set of all functions

$\mathbf{Real}^{\mathsf{prf}}_{\mathcal{A}}(\lambda)$	$\mathbf{Ideal}^{\mathsf{prf}}_{\mathcal{A}}(\lambda)$
1. $K \xleftarrow{\$} \mathcal{K}$	1. $f \xleftarrow{\$} \mathsf{Func}(\mathcal{I}, \mathcal{O})$
2. $b \leftarrow \mathcal{A}^{\mathsf{F}(K,.)}$	2. $b \leftarrow \mathcal{A}^{f(.)}$
3. return b	3. return b

Figure 1: PRF Security Definition

$\mathbf{Real}^{\mathsf{prp}}_{\mathcal{A}}(\lambda)$	$\mathbf{Ideal}^{\mathsf{prp}}_{\mathcal{A}}(\lambda)$
1. $K \xleftarrow{\$} \mathcal{K}$	1. $f \xleftarrow{\$} \mathsf{Perm}(X)$
2. $b \leftarrow \mathcal{A}^{\mathsf{F}(K,.)}$	2. $b \leftarrow \mathcal{A}^{f(.)}$
3. return b	3. return b

Figure 2: PRP Security Definition

from $X_1 \times \cdots \times X_n$ to Y. For set X, $\mathsf{Perm}(X)$ denotes the set of all permutations on X. For sets X_1 and X_2, $\mathsf{Perm}_{X_1}(X_2)$ denotes the set of all functions from $X_1 \times X_2$ to X_2, where for every $x \in X_1$, we have a permutation on X_2.

We use pseudo random functions (PRF), pseudo random permutations (PRP) and RCPA secure symmetric key encryption schemes in our constructions.

2.1 Pseudorandom Function

Pseudorandom Function (PRF) F is polynomial-time computable in λ and is indistinguishable from a truly random function by any adversary $\mathcal{A}$.

Definition 2.1. *Let $\mathsf{F} \in \mathsf{Func}(\mathcal{K} \times \mathcal{I}, \mathcal{O})$ be an efficient, keyed function. For algorithms $\mathcal{A}$, we define the experiments $\mathbf{Real}^{\mathsf{prf}}_{\mathcal{A}}(\lambda)$ and $\mathbf{Ideal}^{\mathsf{prf}}_{\mathcal{A}}(\lambda)$ as shown in Figure 1.*

F is a pseudorandom function if for all probabilistic polynomial-time adversaries $\mathcal{A}$, $\mathbf{Adv}^{\mathsf{prf}}_{\mathsf{F},\mathcal{A}}(\lambda) =$

$\left|\Pr[\mathbf{Real}^{\mathsf{prf}}_{\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{Ideal}^{\mathsf{prf}}_{\mathcal{A}}(\lambda) = 1]\right| \leq \mathsf{neg}(\lambda)$.

2.2 Pseudorandom Permutation

Pseudorandom permutation (PRP) F is polynomial-time computable and invertible in λ and is indistinguishable from a truly random permutation by any adversary $\mathcal{A}$.

Definition 2.2. *Let $\mathsf{F} \in \mathsf{Perm}_{\mathcal{K}}(\mathcal{X})$ be an efficient, keyed function. For algorithms $\mathcal{A}$, we define the experiments $\mathbf{Real}^{\mathsf{prp}}_{\mathcal{A}}(\lambda)$ and $\mathbf{Ideal}^{\mathsf{prp}}_{\mathcal{A}}(\lambda)$ as shown in Figure 2.*

F is a pseudorandom permutation if for all probabilistic polynomial-time adversaries $\mathcal{A}$, $\mathbf{Adv}^{\mathsf{prp}}_{\mathsf{F},\mathcal{A}}(\lambda) =$

$\left|\Pr[\mathbf{Real}^{\mathsf{prp}}_{\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{Ideal}^{\mathsf{prp}}_{\mathcal{A}}(\lambda) = 1]\right| \leq \mathsf{neg}(\lambda)$.

2.3 Symmetric Key Encryption Scheme

A symmetric key encryption scheme $\mathcal{E}$ consists of three algorithms: $\mathsf{Gen}, \mathsf{Enc}$ and Dec.

- $\mathsf{Gen}()$: It outputs a key k.
- $\mathsf{Enc}(\mathsf{k}, \mathsf{m})$: It takes as input the key k, and a message $\mathsf{m} \in \mathcal{M}$, where $\mathcal{M}$ is the message space, and outputs a ciphertext e.

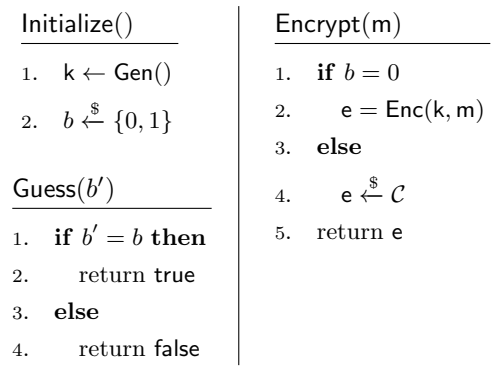


Figure 3: Security Game: $\text{RCPA}_{\mathcal{E}}$

- $\mathsf{Dec}(\mathsf{k}, \mathsf{e})$: It takes as input the key k and a ciphertext e and outputs a message $\mathsf{m} \in \mathcal{M}$ or $\perp$.

Correctness: For all $\mathsf{k} \leftarrow \mathsf{Gen}()$ and for all messages $\mathsf{m} \in \mathcal{M}$, it is required that $\mathsf{Dec}(\mathsf{k}, \mathsf{Enc}(\mathsf{k}, \mathsf{m})) = \mathsf{m}$.

RCPA Security Notion of Symmetric Key Encryption Scheme The pseudorandom ciphertexts under chosen plaintext attack (RCPA) security notion [10] for symmetric key encryption scheme $\mathcal{E}$=($\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}$) is captured in Figure 3. $\mathsf{Initialize}()$ algorithm generates the key k using Gen algorithm of $\mathcal{E}$ and picks a random challenge bit b. The adversary can then adaptively ask queries to $\mathsf{Encrypt}()$ oracle. The game returns true if the adversary's output b' equals the challenge bit b. In Figure 3, $\mathcal{C}$ denotes the ciphertext space.

Definition 2.3. *A symmetric key encryption scheme* $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is said to have pseudorandom ciphertexts under chosen plaintext attack* (RCPA) *if no adversary* $\mathcal{A}$ *can win the game shown in Figure 3, except with probability at most* $\frac{1}{2} + \mathsf{neg}(\lambda)$. *This probability is denoted by* $\Pr[\text{RCPA}_{\mathcal{E}} = 1]$.

The advantage of $\mathcal{A}$ *in* $\text{RCPA}_{\mathcal{E}}$ *game is defined as follows:* $\mathbf{Adv}^{RCPA}_{\mathcal{E},\mathcal{A}}(\lambda) = 2 \cdot \Pr[\text{RCPA}_{\mathcal{E}} = 1] - 1$.

2.4 Dynamic Searchable Symmetric Encryption (DSSE)

Our system consists of two parties: the client C (data owner) and the server S. C, who owns the database DB, encrypts the database and outsources it to S. The encrypted copy of the database created ensures that S responds to C's queries (search and update) in an efficient manner with the guarantee that minimal information apart from the intended output of the query operation is leaked to the server.

We primarily follow the formalization of Bost et al. [7] with certain additions. A keyword is denoted by w and a document is addressed by its document identifier ind. The database DB can be represented as: $\mathsf{DB} = \{(\mathsf{ind}_i, \mathsf{W}_i) : 1 \leq i \leq n\}$, where n denotes the number of documents in the database, $\mathsf{ind}_i \in \{0,1\}^\ell$ are distinct document indices and $\mathsf{W}_i \subseteq \{0,1\}^*$ is a set of keywords matching document ind_i, represented by binary strings of arbitrary length. Additionally, we consider the following notations:

$$
\begin{aligned}
\mathsf{W} &= \cup_{i=1}^{n} \mathsf{W}_i, \text{the set of keywords,} \\
m &= |\mathsf{W}|, \# \text{ keywords,} \\
N &= \sum_{i=1}^{n} |\mathsf{W}_i|, \# \text{ document-keyword pair,} \\
\mathsf{DB}(w) &= \{\mathsf{ind}_i : w \in \mathsf{W}_i\}, \text{ the set of documents containing } w, \\
n_w &= |\mathsf{DB}(w)|, \ \# \text{ documents containing } w \\
a_w, &\ \# \ \mathsf{add} \text{ operations performed on } w \\
d_w, &\ \# \ \mathsf{del} \text{ operations performed on } w
\end{aligned}
$$

o_w, # updates performed on w
n'_w, # documents containing w in previous[1] search operation
a'_w, # add operations performed on w after the previous search
d'_w, # del operations performed on w after the previous search
$o'_w = n'_w + a'_w + d'_w$.

A DSSE scheme Π comprises of the following [7]:

- $\mathsf{Setup}(\mathsf{DB})$ is a probabilistic algorithm that takes as input the initial database DB. It outputs $(\mathsf{st_C}, \mathsf{EDB})$, where the client's state $\mathsf{st_C}$ is given to C and the encrypted database EDB is given to S. Setup algorithm is executed by C.
- $\mathsf{Search}(\mathsf{q}, \mathsf{st_C};\ \mathsf{EDB}) = (\mathsf{Search_C}(\mathsf{q}, \mathsf{st_C}), \mathsf{Search_S}(\mathsf{EDB}))$ is a protocol (possibly probabilistic) between C and S. The input of C is the search query q and client's state $\mathsf{st_C}$. The input of S is the encrypted database EDB. The output to the client is the updated client's state $\mathsf{st_C}'$ and the set Res comprising of indices matching the search query q. The output to the server is the updated encrypted database EDB'.
- $\mathsf{Update}(\mathsf{q}, \mathsf{st_C};\ \mathsf{EDB}) = (\mathsf{Update_C}(\mathsf{q}, \mathsf{st_C}), \mathsf{Update_S}(\mathsf{EDB}))$ is a protocol (possibly probabilistic) between C and S. The input of C is the query $\mathsf{q}=(\mathsf{op}, \mathsf{in})$ comprising of update operation $\mathsf{op} \in \{\mathsf{add}, \mathsf{del}\}$ and the document-keyword pairs (w,ind) denoted by in, and client's state $\mathsf{st_C}$. The input of S is the encrypted database EDB. The output to the client is the updated client's state $\mathsf{st_C}'$. The output to the server is the updated encrypted database EDB'.

In this work, we consider the case of single keyword search, so, $\mathsf{q}=w$ and $\mathsf{Res}=\mathsf{DB}(w)$ in Search protocol. For simplicity, we consider $\mathsf{in}=(\mathsf{ind}, w)$, i.e., a single document-keyword pair in Update protocol. Bulk-Updates can be supported by calling the Update protocol repeatedly. Similar to [11], for a given query q, we consider the output of Search protocol to be the set of document identifiers satisfying q. This allows us to decouple the storage of documents from the storage of data structures used to realize the search operation, which is the focus of this work. SSE schemes are of two types: *response-revealing* and *response-hiding* [26]. The former reveals the query response in plaintext whereas the latter does not. We use this categorization in our paper.

Security We consider S to be honest-but-curious. The security definition follows the real/ideal simulation paradigm [14, 11]. The definition is parameterized by a leakage profile $\mathcal{L}$ which captures all the information that the adversary learns about the database and queries through its participation in the protocols. Hence, the view of the adversary in the real world can be simulated by $\mathcal{L}$.

$\mathcal{L} = \{\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Search}}, \mathcal{L}_{\mathsf{Update}}\}$, where $\mathcal{L}_{\mathsf{Setup}}$, $\mathcal{L}_{\mathsf{Search}}$ and $\mathcal{L}_{\mathsf{Update}}$ correspond to the information leaked in the Setup, Search and Update protocols respectively to the server.

Definition 2.4. *Let* $\Pi = (\mathsf{Setup}, \mathsf{Search}, \mathsf{Update})$ *be a* DSSE *scheme and let* $\mathcal{L} = \{\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Search}}, \mathcal{L}_{\mathsf{Update}}\}$ *be a stateful algorithm. For algorithms* $\mathcal{A}$ *and* Sim*, we define the experiments* $\mathbf{Real}^{\Pi}_{\mathcal{A}}(\lambda)$ *and* $\mathbf{Ideal}^{\Pi}_{\mathcal{A},\mathsf{Sim}}(\lambda)$ *as follows:*

$\mathbf{Real}^{\Pi}_{\mathcal{A}}(\lambda)$ $\mathcal{A}(1^\lambda)$ *chooses* DB. *The experiment then runs* $(\mathsf{st_C}, \mathsf{EDB}) \leftarrow \mathsf{Setup}(\mathsf{DB})$ *and gives* EDB *to* $\mathcal{A}$. *Then* $\mathcal{A}$ *makes polynomial number of adaptive queries. For each query* q*, if* q *is a search query (resp. update query), the game runs* $(\mathsf{Res}, \mathsf{st_C}, \mathsf{EDB}) \leftarrow \mathsf{Search}(\mathsf{q}, \mathsf{st_C}; \mathsf{EDB})$ *(resp.* $(\mathsf{st_C}, \mathsf{EDB}) \leftarrow \mathsf{Update}(\mathsf{q}, \mathsf{st_C}; \mathsf{EDB})$*) and gives the generated transcript to* $\mathcal{A}$. *Eventually* $\mathcal{A}$ *returns a bit that the game uses as its own output.*

$\mathbf{Ideal}^{\Pi}_{\mathcal{A},\mathsf{Sim}}(\lambda)$ *The game initializes a counter i=0 and an empty list* $\mathbf{q}$. $\mathcal{A}(1^\lambda)$ *chooses* DB. *The experiment then runs* $\mathsf{EDB} \leftarrow \mathsf{Sim}(\mathcal{L}_{\mathsf{Setup}}(\mathsf{DB}))$ *and gives* EDB *to* $\mathcal{A}$. *Then* $\mathcal{A}$ *makes polynomial number of adaptive queries. For each query* q*, the game records this as* $\mathbf{q}[i]$*, increments i and if* q *is a search query (resp. update*

[1]Throughout the paper by previous search on w, we mean the last search operation on w before the current search on w.

$\mathbf{DSSECor}^{\Pi}_{\mathcal{A}}(\lambda)$

1. $\mathsf{flag} = \mathsf{false}$
2. $(\mathsf{DB}, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}(1^{\lambda})$
3. $(\mathsf{st}_{\mathsf{C}0}, \mathsf{EDB}_0) \leftarrow \mathsf{Setup}(\mathsf{DB})$
4. **for** $1 \leq i \leq p(\lambda)$ **do**
5. $\quad(\mathsf{q}_i, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}(\mathsf{st}_{\mathcal{A}}, \mathsf{EDB}_0, \mathrm{T}_{i-1})$
6. $\quad$**if** q_i is a search query
7. $\quad\quad\mathsf{Res}_i, \mathsf{EDB}_i, \mathsf{st}_{\mathsf{C}i} \leftarrow \mathsf{Search}(\mathsf{q}_i, \mathsf{st}_{\mathsf{C}i-1}; \mathsf{EDB}_{i-1})$
8. $\quad\quad\tau_i \leftarrow \mathsf{Transcript}[\mathsf{Search}(\mathsf{q}_i, \mathsf{st}_{\mathsf{C}i-1}; \mathsf{EDB}_{i-1})]$
9. $\quad\quad$**if** $\mathsf{Res}_i \neq \mathsf{DB}(w_i)$ $\quad//\mathsf{q}_i = w_i$
10. $\quad\quad\quad\mathsf{flag} = \mathrm{true}$
11. $\quad$**else**
12. $\quad\quad\mathsf{EDB}_i, \mathsf{st}_{\mathsf{C}i} \leftarrow \mathsf{Update}(\mathsf{q}_i, \mathsf{st}_{\mathsf{C}i-1}; \mathsf{EDB}_{i-1})$
13. $\quad\quad\tau_i \leftarrow \mathsf{Transcript}[\mathsf{Update}(\mathsf{q}_i, \mathsf{st}_{\mathsf{C}i-1}; \mathsf{EDB}_{i-1})]$
14. return flag

Figure 4: DSSE Correctness

query), the game gives transcript generated by $\mathsf{Sim}(\mathcal{L}_{\mathsf{Search}}(\mathsf{DB}, \mathbf{q}))$ *(resp.* $\mathsf{Sim}(\mathcal{L}_{\mathsf{Update}}(\mathsf{DB}, \mathbf{q}))$*). Eventually* $\mathcal{A}$ *returns a bit that the game uses as its own output.*

We say that Π *is* $\mathcal{L}$*-semantically secure against adaptive attacks if for all adversaries* $\mathcal{A}$*, there exists an algorithm* Sim *such that* $\left|\Pr[\mathbf{Real}^{\Pi}_{\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{Ideal}^{\Pi}_{\mathcal{A},\mathsf{Sim}}(\lambda) = 1]\right| \leq \mathsf{neg}(\lambda)$.

Common Leakage We follow some of the notations of common leakages from [5, 7]. The leakage profile $\mathcal{L}$ keeps as state the query list $\mathbf{Q}$, i.e., the list of all queries issued so far along with their timestamp. The entries in $\mathbf{Q}$ are (u, w) for a search query on w, or $(u, \mathsf{op}, (\mathsf{ind}, w))$ for an update query $(\mathsf{op}, (\mathsf{ind}, w))$, where u denotes the timestamp of the query. Corresponding to the search queries, the search pattern $\mathsf{sp}(w)$ can be defined as $\mathsf{sp}(w) = \{\, u : (u, w) \in \mathbf{Q} \,\}$.

We also use the notation $\mathsf{Hist}(w)$ that denotes the list of all the modifications made to $\mathsf{DB}(w)$ over the time. It consists of $\mathsf{DB}_0(w)$, the set of document indices matching w at setup, and a list $\mathsf{UpHist}(w)$, comprising of the updates of documents matching w, called the *update history*. For example, consider two documents 1 and 2 matching w. Suppose, the update queries are $(\mathsf{add}, (1, w))$, $(\mathsf{add}, (2, w))$ and $(\mathsf{del}, (1, w))$ at timestamp 3, 12 and 20 respectively, then $\mathsf{UpHist}(w) = [\,(3, \mathsf{add}, 1),\ (12, \mathsf{add}, 2),\ (20, \mathsf{del}, 1)\,]$.

$\mathsf{Updates}(w)$ denotes the set of timestamps of updates on w. Formally,

$$\mathsf{Updates}(w) = \{u | (u, \mathsf{add}, (\mathsf{ind}, w)) \in \mathbf{Q} \text{ or } (u, \mathsf{del}, (\mathsf{ind}, w)) \in \mathbf{Q}\}.$$

$\mathsf{Updates}^{\mathsf{op}}(w)$ is exactly like $\mathsf{Updates}(w)$ except along with the timestamp it also stores op corresponding to the update query.

Correctness We say that a DSSE scheme is correct if the Search protocol returns the correct results for the keyword being searched (i.e., $\mathsf{DB}(w)$), except with negligible probability. We follow a similar formalization to [10].

In Figure 4, the adversary $\mathcal{A}$ makes $p(\lambda)$ many queries, for some polynomial p. $\mathsf{Transcript}[\mathsf{Protocol}]$ means the view of server in Protocol. $\mathrm{T}_0 = \emptyset$ and $\mathrm{T}_i = \{\tau_j : 1 \leq j \leq i\}$, $1 \leq i \leq p(\lambda)$. Here, τ_j denotes the transcript of the j^{th} query.

Definition 2.5. *Let* $\Pi = (\mathsf{Setup}, \mathsf{Search}, \mathsf{Update})$ *be a* DSSE *scheme. For algorithm* $\mathcal{A}$ *we define the experiment* $\mathbf{DSSECor}^{\Pi}_{\mathcal{A}}(\lambda)$ *as shown in Figure 4.*

We say that Π *is correct if for all adversaries* $\mathcal{A}$*,* $\Pr[\mathbf{DSSECor}^{\Pi}_{\mathcal{A}}(\lambda) = 1] \leq \mathsf{neg}(\lambda)$.

3 Security Notions in DSSE

In this section, we perform a critical analysis of the existing notions of information leakage in backward private DSSE followed by some alternative formulations. The question of what is the *right* definition of

security is a vexed one [30]. Even for a widely used cryptographic primitive like digital signature, it has been argued that the accepted standard definition of security [24, 23] does not take into account various issues that may crop up depending upon the application scenarios [31, 35, 42]. It is but natural that for a relatively new crypto/security protocol like DSSE, one needs to look at the definitional question from various perspectives. Apart from giving better insight about the real world security assurance of the protocol, this exercise also helps to choose among alternative definitions to achieve some sort of *optimal* balance between security and efficiency for the task at hand.

Several works in the context of searchable encryption [10, 11, 13, 14, 26, 27], argue security of SSE schemes by formulating a leakage profile $\mathcal{L}$ and proving that the leakage incurred in the proposed scheme is bounded by $\mathcal{L}$. However, works in the context of DSSE [5, 7] present the formulated information leakages corresponding to forward/backward privacy notion in the form of security definitions. Here, we follow the latter approach.

Bost at el. [7] made a seminal contribution in the area of DSSE by formalizing the notion of backward privacy through three different security definitions viz., BPIP, BPUP and WBP, ordered from most to least secure based on their respective information leakages. Naturally, like any other formalization of security this requires further investigation. In that vein, we first argue why *weak backward privacy* (WBP) cannot serve as a general definition for backward privacy and should be used to argue backward privacy in re-insertion restriction setting only. Second, we introduce additional definitions (Definition 3.5 and Definition 3.6) for backward privacy in terms of information leakage. Finally, we introduce a desirable property for DSSE scheme which we call *inverse backward privacy*.

For simplicity we will assume that, DB is initially empty. Thus, the Setup algorithm leaks no information. If DB is not initially empty, typically, $\mathcal{L}_{\mathsf{Setup}} = N$, where N denotes the number of document-keyword pairs.

3.1 Forward Privacy

We recall the strongest notion of forward privacy discussed in [7]. Informally, a DSSE scheme is forward private if the Update protocol leaks no information about the updated keywords. Definition 3.1 captures that an update operation doesn't leak more than the operation $\mathsf{op} = \{\mathsf{add}, \mathsf{del}\}$ of the update query q.

Definition 3.1. *(FP-I). An $\mathcal{L}$=$\{\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Search}}, \mathcal{L}_{\mathsf{Update}}\}$-semantically secure against adaptive attacks* DSSE *scheme* Π *is* FP-I *iff $\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Search}}, \mathcal{L}_{\mathsf{Update}}$ can be written as:*

$$\mathcal{L}_{\mathsf{Setup}}() = \emptyset. \quad \mathcal{L}_{\mathsf{Search}}(w) = \{\mathsf{sp}(w), \mathsf{Hist}(w)\}.$$
$$\mathcal{L}_{\mathsf{Update}}(\mathsf{op}, (\mathsf{ind}, w)) = \mathsf{op}.$$

3.2 Backward Privacy

The notion of backward privacy was informally introduced in [38] as "queries cannot be executed over the deleted documents". The follow up work [7] stated that a DSSE scheme is backward private if, whenever a document-keyword pair (ind, w) is added to the database and later deleted, subsequent searches on w should not reveal ind if it cannot be inferred from the search and access pattern. Based on the above notion, we now elaborate our interpretation of backward privacy. Let $\mathsf{q}_{\mathsf{prev}}$ be the previous search query on w and let $\mathsf{ind} \notin \mathsf{DB}(w)'$ (here, $\mathsf{DB}(w)'$ denotes the set of documents matching w in the previous search on w). If the last update operation on (ind, w) before the next search query $\mathsf{q}_{\mathsf{cur}}$ is del operation, then ind should not be revealed to S as it cannot be inferred from the results of current and previous search queries. Let the set of such inds be denoted by I_{BP}. For example, let $\mathsf{DB}(w)'$=$\{1, 2\}$ for search query $\mathsf{q}_{\mathsf{prev}}$ at timestamp 5. Let the update operations on w after query $\mathsf{q}_{\mathsf{prev}}$ and before the next search query $\mathsf{q}_{\mathsf{cur}}$ be $(6, \mathsf{add}, (3, w))$, $(12, \mathsf{del}, (3, w))$, then identifier 3 should not be revealed to S.

Backward privacy was formalized in [7], through three different security definitions, ordered from most to least secure called respectively BPIP, BPUP and WBP. Informally, these definitions are described below.

- *Backward Privacy with insertion pattern* (BPIP): During a search on some keyword w, BPIP schemes

leak the documents currently matching w, when they were inserted, and the total number of updates on w.

- *Backward Privacy with update pattern* (BPUP): During a search on w, BPUP schemes leak the documents currently matching w, when they were inserted, and when all the updates on w happened (but not their contents).
- *Weak backward privacy* (WBP): During a search on w, WBP schemes leak the documents currently matching w, when they were inserted, when all the updates on w happened, and which deletion update canceled which insertion update.

Let us demonstrate the differences between these notions with an example. Consider the following entries in query list $\mathbf{Q}_1$ corresponding to w: $(1, \mathsf{add}, (\mathsf{ind}_1, w))$, $(4, \mathsf{add}, (\mathsf{ind}_2, w))$, $(5, \mathsf{del}, (\mathsf{ind}_1, w))$, $(12, \mathsf{add}, (\mathsf{ind}_3, w))$. Let us consider the leakage for each definition after a search query on w at timestamp 15. The first notion reveals that ind_2 and ind_3 match keyword w and that this entries were added at time 4 and 12 respectively. It also reveals that there were a total of 4 updates for w. The second notion, additionally reveals that updates on w happened at time 1, 4, 5 and 12. Finally, the third definition also reveals that the index that was added for w at time 1 was deleted at time 5.

Let us recall the additional leakage functions from [7, 6] apart from the leakage functions described in Section 2.4, required to formally capture the notions of backward privacy mentioned above.

For a keyword w, $\mathsf{TimeDB}(w)$ is the list of all documents matching w, excluding the deleted ones together with the timestamp of when they were inserted in the database. Formally, $\mathsf{TimeDB}(w)$ can be constructed from the query list $\mathbf{Q}$ as follows:

$$\begin{aligned}\mathsf{TimeDB}(w) = \{(u, \mathsf{ind}) | (u, \mathsf{add}, (w, \mathsf{ind})) \in \mathbf{Q} \text{ and} \\ \forall u' > u, (u', \mathsf{del}, (w, \mathsf{ind})) \notin \mathbf{Q}\}. \end{aligned} \tag{1}$$

The deletion history $\mathsf{DelHist}(w)$ of w is the list of timestamps for all deletion operations, together with the timestamp of the inserted entry it removes. Formally, $\mathsf{DelHist}(w)$ is constructed as:

$$\begin{aligned}\mathsf{DelHist}(w) = \{(u^{\mathsf{add}}, u^{\mathsf{del}}) | \exists \mathsf{ind} \text{ s.t. } (u^{\mathsf{add}}, \mathsf{add}, (w, \mathsf{ind})) \in \mathbf{Q} \text{ and} \\ (u^{\mathsf{del}}, \mathsf{del}, (w, \mathsf{ind})) \in \mathbf{Q}\}. \end{aligned} \tag{2}$$

With these tools, we can now formally define these three notions of backward privacy formally.

Definition 3.2. *(BPIP). An $\mathcal{L}$=$\{\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Search}}, \mathcal{L}_{\mathsf{Update}}\}$-semantically secure against adaptive attacks* DSSE *scheme* Π *is* BPIP *iff $\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Search}}, \mathcal{L}_{\mathsf{Update}}$ can be written as:*

$$\begin{aligned}&\mathcal{L}_{\mathsf{Setup}}() = \emptyset. \quad \mathcal{L}_{\mathsf{Update}}(\mathsf{op}, (\mathsf{ind}, w)) = \{\mathsf{op}\}. \\ &\mathcal{L}_{\mathsf{Search}}(w) = \{\mathsf{TimeDB}(w), o_w\}.\end{aligned}$$

Definition 3.3. *(BPUP). An $\mathcal{L}$=$\{\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Search}}, \mathcal{L}_{\mathsf{Update}}\}$-semantically secure against adaptive attacks* DSSE *scheme* Π *is* BPUP *iff $\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Search}}, \mathcal{L}_{\mathsf{Update}}$ can be written as:*

$$\begin{aligned}&\mathcal{L}_{\mathsf{Setup}}() = \emptyset. \quad \mathcal{L}_{\mathsf{Update}}(\mathsf{op}, (\mathsf{ind}, w)) = \{\mathsf{op}, w\}. \\ &\mathcal{L}_{\mathsf{Search}}(w) = \{\mathsf{TimeDB}(w), \mathsf{Updates}(w)\}.\end{aligned}$$

Definition 3.4. *(WBP). An $\mathcal{L}$=$\{\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Search}}, \mathcal{L}_{\mathsf{Update}}\}$-semantically secure against adaptive attacks* DSSE *scheme* Π *is* WBP *iff $\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Search}}, \mathcal{L}_{\mathsf{Update}}$ can be written as:*

$$\begin{aligned}&\mathcal{L}_{\mathsf{Setup}}() = \emptyset. \quad \mathcal{L}_{\mathsf{Update}}(\mathsf{op}, (\mathsf{ind}, w)) = \{\mathsf{op}, w\}. \\ &\mathcal{L}_{\mathsf{Search}}(w) = \{\mathsf{TimeDB}(w), \mathsf{DelHist}(w)\}.\end{aligned}$$

The above definitions indicate that the notion of backward privacy is more involved than that of forward privacy. Forward privacy can be formalized by ensuring that the update query doesn't leak the keyword corresponding to which the update has been made. Whereas it is more subtle in the case of backward privacy. In formulating the leakage profile for backward privacy, one must ensure that no leakage is incurred w.r.t. document identifiers in I_{BP} described above. One approach is to formulate a strong leakage profile for the security notion in hand that allows very limited information leakage to be incurred by the constructions satisfying it. This approach was followed in formulating leakage profiles in definitions BPIP and BPUP. A possible shortcoming of this approach is that there could be candidate constructions that don't satisfy these strong definitions but may still satisfy the intuitive notion of backward privacy. The other extreme could be to allow 'as much information as one can think of' to be leaked that can be allowed by the security notion in hand. This seems to be the approach followed in WBP security notion. However, in this approach one must be careful not to violate the basic security notion of the corresponding task while allowing for more information leakage. We revisit the notion of WBP from this perspective.

Remark $\mathcal{L}_{\mathsf{Search}}$ in BPIP, BPUP and WBP should be augmented with leakage function $\mathsf{sp}(w)$. Moreover, $\mathcal{L}_{\mathsf{Search}}$ in WBP should also be augmented with leakage function $\mathsf{Updates}(w)$. The rationale behind the same is described in Section 3.4.

3.3 Revisiting Weak Backward Privacy

In this section, we scrutinize the notion of weak backward privacy. Let us consider the following entries in query list $\mathbf{Q}_1$ corresponding to w: $(1, \mathsf{add}, (\mathsf{ind}, w))$, $(3, w)$, $(5, \mathsf{del}, (\mathsf{ind}, w))$, $(6, w)$, $(12, \mathsf{add}, (\mathsf{ind}, w))$, $(14, \mathsf{del}, (\mathsf{ind}, w))$, $(18, w)$. Let us denote the search queries on w at timestamps 3, 6 and 18 by q_1, q_2 and q_3 respectively.

Leakage of the search query q_1:

$$\mathsf{TimeDB}(w) = \{(1, \mathsf{ind})\}. \quad \mathsf{DelHist}(w) = \emptyset.$$

Leakage of the search query q_2:

$$\mathsf{TimeDB}(w) = \emptyset. \quad \mathsf{DelHist}(w) = \{(1, 5)\}.$$

Note that, $\mathsf{DelHist}(w)$ leaks that the add operation at timestamp 1 is canceled by the del operation at timestamp 5. Through the content of $\mathsf{DB}(w)$ after queries q_1 and q_2 the adversary can infer that the del operation at timestamp 5 corresponds to document ind. Therefore, it adheres to the notion of backward privacy described at the beginning of this section.

Now, let us consider the search query q_3. After the search query q_2, (ind, w) was added at timestamp 12 and later deleted at timestamp 14. Through the same intuitive notion of backward privacy, based on the state of $\mathsf{DB}(w)$ after queries q_2 and q_3, the adversary should not infer which document does the update queries at timestamp 12 and 14 correspond to. However, the leakage of the search query q_3 is:

$$\mathsf{TimeDB}(w) = \emptyset. \quad \mathsf{DelHist}(w) = \{(1, 5),\ (1, 14),\ (12, 5),\ (12, 14)\}.$$

Hence, through the leakage profile the adversary can infer that updates at timestamp 12 and 14 correspond to document ind as it has already inferred which document the update queries at timestamp 1 and 5 correspond to. Clearly, this goes against the intuitive notion of backward privacy.

The following restriction is imposed on the constructions $\mathsf{Diana}_{\mathsf{del}}$ and Janus that are proven to be weak backward private in [7]: 'reinsertion of a document-keyword pair is not allowed after the deletion of the corresponding document-keyword pair'. The reinsertion restriction allows one to avoid scenarios such as above that violate the intuitive notion of backward privacy. Hence, WBP can be considered to argue backward privacy in reinsertion restriction setting only. However, WBP-constructions proposed in subsequent works [20, 40] do not explicitly mention that reinsertion of document-keyword pair is not allowed. Therefore, in order to avoid any ambiguity, we feel it should be clarified that WBP is applicable in such restricted scenarios only.

Remark The case of reinsertion of document-keyword pair, may not be a concern in certain use-cases of SSE schemes where a new document identifier can be assigned to the updated document, thereby, ensuring that the newly inserted document-keyword pairs cannot be related to older ones. But this trick may not always be applicable especially when the contents of file can change dynamically over time. Here, one needs to handle reinsertion of keyword in *existing* documents, i.e., the document identifier can't be changed. Therefore, in such scenarios one needs to support reinsertion of document-keyword pairs. As an example, consider the case of a hospital database where the patients' records are documents and the disease they are suspected to suffer from are keywords. One cannot rule out a scenario in which based on newer symptoms a patient is re-suspected to suffer from a disease, say *malignant brain tumor*, which she had been ruled out to suffer from earlier.

3.4 Suggested Modifications

Here, we point out subtle issues in Definitions 3.2, 3.3 and 3.4 and suggest modifications to address them. We first argue that $\mathsf{sp}(w)$ should be a part of $\mathcal{L}_{\mathsf{Search}}(w)$ in definitions of BPIP, BPUP and WBP [7, Definition 4.2]. The constructions satisfying the respective definitions leak $\mathsf{sp}(w)$ in Search protocol. This implies that $\mathsf{sp}(w)$ can be derived from the other leakage functions in the respective definitions. Now, consider the corner case where no updates corresponding to w has occurred so far and two search queries on w are executed at timestamps 5 and 8 respectively. For search query at timestamp 8, $\mathsf{sp}(w)$=$\{5\}$. However, the state of other leakage functions at timestamp 8 are: $\mathsf{TimeDB}(w)$=$\emptyset$, $\mathsf{Updates}(w)$=$\emptyset$ and $\mathsf{DelHist}(w)$=$\emptyset$. As can be observed, $\mathsf{sp}(w)$ cannot be derived from other leakage functions. Hence, $\mathsf{sp}(w)$ should be included in $\mathcal{L}_{\mathsf{Search}}(w)$ of BPIP, BPUP and WBP.

Next. we argue that $\mathcal{L}_{\mathsf{Search}}$ in WBP should also be augmented with leakage function $\mathsf{Updates}(w)$. Recall that the informal notion of WBP in [7] states that a search query on w should at most leak the documents currently matching w, when they were inserted, when all the updates on w happened and the total number of updates on w. While formalizing the leakage profile of WBP [7, Definition 4.2], the leakage in Search protocol is described as: $\mathcal{L}_{\mathsf{Search}}(w) = (\mathsf{TimeDB}(w), \mathsf{DelHist}(w))$. Note that, $\mathsf{Updates}(w)$ isn't explicitly a part of $\mathcal{L}_{\mathsf{Search}}(w)$. This implies that, given $\mathsf{TimeDB}(w)$ and $\mathsf{DelHist}(w)$ one can construct $\mathsf{Updates}(w)$. Now, consider the following entries in query list $\mathbf{Q}_1$ corresponding to w: $(1, \mathsf{add}, (1, w))$, $(2, \mathsf{del}, (2, w))$, $(4, \mathsf{add}, (3, w))$, $(6, \mathsf{del}, (1, w))$, $(8, w)$. Let us denote the search query on w at timestamps 8 by q_1.

Leakage functions at search query q_1:

$$\mathsf{TimeDB}(w) = \{(4, 3)\}. \quad \mathsf{DelHist}(w) = \{(1, 6)\}.$$
$$\mathsf{Updates}(w) = \{1, 2, 4, 6\}.$$

Note that, $\mathsf{TimeDB}(w)$ and $\mathsf{DelHist}(w)$ collectively do not capture any information about the update operation at timestamp 2. As a result, $\mathsf{Updates}(w)$ cannot be constructed given the leakage functions $\mathsf{TimeDB}(w)$ and $\mathsf{DelHist}(w)$. Hence, $\mathsf{Updates}(w)$ should be a part of $\mathcal{L}_{\mathsf{Search}}(w)$ of WBP. Further, the construction Janus also leaks $\mathsf{Updates}(w)$ in Search protocol. Hence, $\mathsf{Updates}(w)$ should be included in the leakage profile of Janus construction.

Thus, the search leakage in the respective security definitions become:

$$\mathcal{L}_{\mathtt{BPIP},\mathsf{Search}} : \{\mathsf{sp}(w), \mathsf{TimeDB}(w), o_w\}.$$
$$\mathcal{L}_{\mathtt{BPUP},\mathsf{Search}} : \{\mathsf{sp}(w), \mathsf{TimeDB}(w), \mathsf{Updates}(w)\}.$$
$$\mathcal{L}_{\mathtt{WBP},\mathsf{Search}} : \{\mathsf{sp}(w), \mathsf{TimeDB}(w), \mathsf{Updates}(w), \mathsf{DelHist}(w)\}.$$

3.5 Alternative Formulations of Leakage

We propose two alternative formulations of information leakage in the context of backward privacy (Definition 3.5 and 3.6). As mentioned earlier, similar to [11], for a given query q, we consider the output of Search protocol to be the set of document identifiers satisfying q. This allows us to decouple the storage of documents from the storage of data structures used to realize the search operation in all our constructions and provide security definitions accordingly.

Definition 3.5, i.e., BP-I captures that an update query should leak nothing and a search query should only reveal when the updates on w happened apart from the identifiers of the documents currently matching w. BP-I is similar to the definition of Backward privacy with update pattern (BPUP) in [7] except BP-I doesn't allow leakage of $\mathsf{TimeDB}(w)$. BP-I tightly captures the leakage profile of constructions Fides, Mitra[2] and Π_{BP}-prime. This exercise of tightly capturing the leakage avoids confusion regarding the immunity of constructions against attacks which result from leakages that are indeed not incurred by them, $\mathsf{TimeDB}(w)$ in this case.

Definition 3.5. *(BP-I) An $\mathcal{L}=\{\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Search}}, \mathcal{L}_{\mathsf{Update}}\}$-semantically secure against adaptive attacks* DSSE *scheme* Π *is* BP-I *iff $\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Search}}, \mathcal{L}_{\mathsf{Update}}$ can be written as:*

$$\mathcal{L}_{\mathsf{Setup}}() = \emptyset. \quad \mathcal{L}_{\mathsf{Update}}(\mathsf{op}, (\mathsf{ind}, w)) = \emptyset.$$
$$\mathcal{L}_{\mathsf{Search}}(w) = \{\mathsf{sp}(w), \mathsf{Updates}(w), \mathsf{DB}(w)\}.$$

Constructing backward private DSSE schemes with optimal communication complexity is an interesting question to explore. To achieve this goal it seems necessary for S to be able to identify which insertion entry is canceled by a particular delete entry. However, the stronger definitions BPIP, BPUP and BP-I do not allow such information to be leaked to S. Therefore, like WBP, one needs to craft a definition that allows some non-trivial relation among the update queries to be leaked. But unlike WBP it should be able to capture the notion of backward privacy in the general setting. We observe that there can be several alternatives to relax the formal notion of backward privacy. Next, we describe one such candidate definition, i.e., BP-II. Later, we show a natural construction (see Section 4.3), whose leakage profile is tightly captured by BP-II.

In order to describe the leakage profile of BP-II, let us introduce some new leakage functions. Let $\mathsf{DB}(w)$ and $\mathsf{DB}(w)'$ denote the set of documents matching w in the current and previous search respectively and let $\mathsf{LDB}(w)$ denote the list of documents matching w in the current search, in order of their insertion. An element in $\mathsf{LDB}(w)$ is of the form (i, ind), where i denotes the index and ind denotes the document identifier. Let the timestamp of the current and previous search be denoted as u_c and u_p respectively. We define the following leakage functions:

$$\begin{aligned}
\mathsf{PreUp}(w) = \{&(\mathsf{ind}, u) |\ \mathsf{ind} \in \mathsf{DB}(w)' \text{ and } (u, \mathsf{op}, (\mathsf{ind}, w)) \in \mathbf{Q} \\
&\text{and } u_p < u < u_c\}. \\
\mathsf{CurUp}(w) = \{&(\mathsf{ind}, u) |\ \mathsf{ind} \in \mathsf{DB}(w) \text{ and } (u, \mathsf{op}, (\mathsf{ind}, w)) \in \mathbf{Q} \\
&\text{and } u_p < u < u_c\}. \\
\mathsf{UpPair}(w) = \{&(u_1, u_2) | (u_1, \mathsf{op}, (\mathsf{ind}, w)) \in \mathbf{Q} \text{ and} \\
&(u_2, \mathsf{op}, (\mathsf{ind}, w)) \in \mathbf{Q} \text{ and } u_p < u_1 < u_2 < u_c\}.
\end{aligned}$$

$\mathsf{PreUp}(w)$ captures the relation between the identifiers obtained as a result in the previous search on w and the updates that happen after the previous search on w and before the current search on w. $\mathsf{CurUp}(w)$ captures the relation between the identifiers obtained as a result in the current search on w and the updates that happen after the previous search on w and before the current search on w. $\mathsf{UpPair}(w)$ captures the relation among the updates corresponding to some ind, that happen after the previous search on w and before the current search on w.

For example, let us consider the search query on w at timestamp 16. Let the timestamp of the previous search query on w be 4. Let $\mathsf{DB}(w)$ and $\mathsf{DB}(w)'$ be $\{1, 3\}$ and $\{1, 2\}$ respectively. Let $(5, \mathsf{del}, (1, w))$, $(7, \mathsf{del}, (2, w))$, $(12, \mathsf{add}, (1, w))$ and $(15, \mathsf{add}, (3, w))$ be the update queries on w that occur between these two searches. The respective leakage functions will be:

$$\mathsf{PreUp}(w) = \{(1, 5), (2, 7), (1, 12)\}. \quad \mathsf{UpPair}(w) = \{(5, 12)\}.$$
$$\mathsf{CurUp}(w) = \{(1, 5), (1, 12), (3, 15)\}.$$

Next, we describe the search leakage of BP-II. Though the leakage functions defined above may appear a bit complex, they are useful abstractions through which BP-II captures the notion of backward privacy as

[2] Mitra can be proven BP-I secure with a minor modification (sending document identifiers in search result $\mathsf{DB}(w)$ in random order).

shown below. In Search protocol, along with $\mathsf{sp}(w)$ and $\mathsf{Updates}^{\mathsf{op}}(w)$, leakage functions $\mathsf{PreUp}(w)$, $\mathsf{CurUp}(w)$, $\mathsf{UpPair}(w)$ and $\mathsf{LDB}(w)$ can be leaked to S. As already defined in the beginning of Section 3.2, recall that I_{BP} denotes the set of document identifiers relevant to the notion of backward privacy. Note that, $\forall \mathsf{ind} \in \mathrm{I}_{\mathsf{BP}}$, as $\mathsf{ind} \notin \mathsf{DB}(w)'$ and as the last update operation on (ind, w) is del operation, it follows that $\mathsf{ind} \notin \mathsf{DB}(w)$. Hence, no information about the inds in I_{BP} can be revealed through leakage functions $\mathsf{PreUp}(w)$, $\mathsf{CurUp}(w)$ and $\mathsf{LDB}(w)$. As the other leakage functions viz., $\mathsf{sp}(w)$ and $\mathsf{UpPair}(w)$, do not leak ind corresponding to an update, no information about identifiers in I_{BP} gets revealed.

Definition 3.6. *(*$\mathtt{BP\text{-}II}$*) An* $\mathcal{L}=\{\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Search}}, \mathcal{L}_{\mathsf{Update}}\}$*-semantically secure against adaptive attacks scheme* Π *is* $\mathtt{BP\text{-}II}$ *iff* $\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Search}}, \mathcal{L}_{\mathsf{Update}}$ *can be written as:*

$$\begin{aligned} \mathcal{L}_{\mathsf{Setup}}() =& \emptyset. \quad \mathcal{L}_{\mathsf{Update}}(\mathsf{op}, (\mathsf{ind}, w)) = \emptyset. \\ \mathcal{L}_{\mathsf{Search}}(w) =& \{\mathsf{sp}(w), \mathsf{Updates}^{\mathsf{op}}(w), \mathsf{PreUp}(w), \mathsf{CurUp}(w), \\ & \mathsf{UpPair}(w), \mathsf{LDB}(w)\} \end{aligned}$$

Note that Definitions 3.5 ($\mathtt{BP\text{-}I}$) and 3.6 ($\mathtt{BP\text{-}II}$) capture the notion of forward privacy as well. Therefore, constructions that are $\mathtt{BP\text{-}I}$ and $\mathtt{BP\text{-}II}$ secure are naturally forward private.

Inverse Backward Privacy We identify a new desirable property for a DSSE scheme called *inverse backward privacy* which captures the complementary situation of backward privacy. Analogous to backward privacy, a DSSE scheme is inverse backward private if whenever a document-keyword pair (ind, w) is deleted and later added, subsequent search queries on w won't reveal the fact that (ind, w) was deleted unless it can be inferred by the search and access pattern of the search query. For example, let $\mathsf{DB}(w)=\{1, 2\}$ for search query q at timestamp 5. Let the update operations on w after query q and before the next search query be $(6, \mathsf{del}, (1, w))$, $(12, \mathsf{add}, (1, w))$, then no information about the identifier 1 in update queries at timestamp 6 and 12 should be revealed to S on the next search query on w.

Inverse Backward Privacy property could be of relevance in various use-cases. For instance, consider the employee database where the employee records correspond to document and project teams she works in correspond to keywords. An employee E_1 maybe dropped and reincluded in a project team. We would like to hide the fact that employee E_1 was dropped briefly, if no search on that project team had been performed during that period. Definition 3.5 ($\mathtt{BP\text{-}I}$) is strong enough to capture our notion of inverse backward privacy. Therefore, all constructions that can be shown to satisfy Definition 3.5 are inverse backward private.

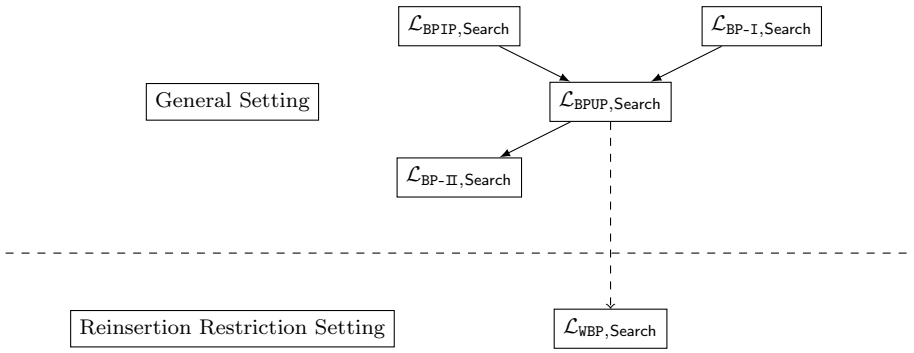
To summarize, we proposed alternative formulations of leakages in the form of two new security definitions. We argued that the two definitions capture the notion of backward privacy. An interesting open question is to analyze the real-world consequences of the leakages incurred by constructions satisfying the definitions of backward privacy, viz., $\mathtt{BPIP}$, $\mathtt{BPUP}$, $\mathtt{BP\text{-}I}$, $\mathtt{BP\text{-}II}$ or even $\mathtt{WBP}$ (in restricted setting).

3.6 Comparison among Definitions

Finally we discuss the relations among the definitions of backward privacy. Precisely, we give comparison among the leakages incurred in Search protocol in the definitions of backward privacy. As we remarked earlier, the notion of $\mathtt{WBP}$ is only suitable to argue backward privacy in a restricted setting. Hence, it would not be meaningful to compare the definitions of backward privacy which apply to the general setting with $\mathtt{WBP}$. Therefore, $\mathtt{WBP}$ is not considered for comparison in this section.

If $\mathcal{L}_1$ leaks less than $\mathcal{L}_2$, we denote it as $\mathcal{L}_1 \preceq \mathcal{L}_2$. By the proposition, "$\mathcal{L}_1$ leaks less than $\mathcal{L}_2$" we mean that $\mathcal{L}_1$ gives less information about the database and the queries to the simulator than $\mathcal{L}_2$, or, said otherwise, that every information given by $\mathcal{L}_1$ can be inferred from $\mathcal{L}_2$. If $\mathcal{L}_1$ leaks strictly less than $\mathcal{L}_2$, we denote it as $\mathcal{L}_1 \prec \mathcal{L}_2$. Here, we give an argument to derive the leakage functions in $\mathcal{L}_1$ through that of $\mathcal{L}_2$ in order to show that $\mathcal{L}_1 \prec \mathcal{L}_2$. And with the help of counter examples, we showcase that the leakage functions in $\mathcal{L}_1$ cannot be derived from that of $\mathcal{L}_2$.

Recall the leakage functions corresponding to Search protocol in the existing definitions [7] are related in



Here, $\mathcal{L}_1 \rightarrow \mathcal{L}_2$ denotes $\mathcal{L}_1 \prec \mathcal{L}_2$

Figure 5: Relations among Definitions of Backward Privacy

the following way:

$$\mathcal{L}_{\mathsf{BPIP},\mathsf{Search}} \prec \mathcal{L}_{\mathsf{BPUP},\mathsf{Search}} \prec \mathcal{L}_{\mathsf{WBP},\mathsf{Search}}. \tag{3}$$

We recall $\mathcal{L}_{\mathsf{Search}}$ of all the definitions below. See Section 2.4 (Common Leakages) and Section 3 for the description of the leakage functions used here.

$$\mathcal{L}_{\mathtt{BPIP},\mathsf{Search}} : \{\mathsf{sp}(w), \mathsf{TimeDB}(w), o_w\}. \tag{4}$$
$$\mathcal{L}_{\mathtt{BPUP},\mathsf{Search}} : \{\mathsf{sp}(w), \mathsf{TimeDB}(w), \mathsf{Updates}(w)\}. \tag{5}$$
$$\mathcal{L}_{\mathtt{BP\text{-}I},\mathsf{Search}} : \{\mathsf{sp}(w), \mathsf{Updates}(w), \mathsf{DB}(w)\}. \tag{6}$$
$$\mathcal{L}_{\mathtt{BP\text{-}II},\mathsf{Search}} : \{\mathsf{sp}(w), \mathsf{Updates}^{\mathsf{op}}(w), \mathsf{PreUp}(w), \mathsf{CurUp}(w) \; \mathsf{UpPair}(w), \mathsf{LDB}(w)\}. \tag{7}$$

As can be observed from (4) and (6), for BPIP, $\mathsf{TimeDB}(w)$ cannot be derived from leakage functions in $\mathcal{L}_{\mathtt{BP\text{-}I},\mathsf{Search}}$ (see Example 1, described below) and for BP-I, $\mathsf{Updates}(w)$ cannot be derived from leakage functions in $\mathcal{L}_{\mathtt{BPIP},\mathsf{Search}}$ (see Example 2, described below). Hence, $\mathcal{L}_{\mathtt{BPIP},\mathsf{Search}}$ and $\mathcal{L}_{\mathtt{BP\text{-}I},\mathsf{Search}}$ cannot be orderly related.

From (5) and (6) it is straightforward that all the leakage functions in $\mathcal{L}_{\mathtt{BP\text{-}I},\mathsf{Search}}$ can be derived from the leakage functions in $\mathcal{L}_{\mathtt{BPUP},\mathsf{Search}}$, as $\mathsf{DB}(w)$ can be easily obtained from $\mathsf{TimeDB}(w)$. Further, as described Example 1 (described below), $\mathsf{TimeDB}(w)$ cannot be derived from leakage functions in $\mathcal{L}_{\mathtt{BP\text{-}I},\mathsf{Search}}$. Hence, $\mathcal{L}_{\mathtt{BP\text{-}I},\mathsf{Search}} \prec \mathcal{L}_{\mathtt{BPUP},\mathsf{Search}}$.

From (5) and (7), one can conclude that all the leakage functions in $\mathcal{L}_{\mathtt{BPUP},\mathsf{Search}}$ can be derived from the leakage functions in $\mathcal{L}_{\mathtt{BP\text{-}II},\mathsf{Search}}$, as $\mathsf{TimeDB}(w)$ can be derived from $\mathsf{CurUp}(w)$, $\mathsf{Updates}^{\mathsf{op}}(w)$ and $\mathsf{DB}(w)$ (in particular $\mathsf{LDB}(w)$) of current and previous searches on w. Note that, $\mathsf{TimeDB}(w)$ comprises of information regarding the timestamp of the add updates corresponding to document identifiers in $\mathsf{DB}(w)$ that are not followed by their respective del update. This can be easily determined by keeping track of all the add updates corresponding to document identifiers in $\mathsf{DB}(w)$ that are not followed by the respective del update through $\mathsf{CurUp}(w)$ over all searches on w and $\mathsf{Updates}^{op}(w)$. Further, some leakage functions in $\mathcal{L}_{\mathtt{BP\text{-}II},\mathsf{Search}}$ say, $\mathsf{UpPair}(w)$, cannot be derived from $\mathcal{L}_{\mathtt{BPUP},\mathsf{Search}}$ (see Example 4, described below). Hence, $\mathcal{L}_{\mathtt{BPUP},\mathsf{Search}} \prec \mathcal{L}_{\mathtt{BP\text{-}II},\mathsf{Search}}$.

Examples

1. Consider the entries corresponding to w in query lists $\mathbf{Q}_1$ and $\mathbf{Q}_2$ as described below. Corresponding to w, $(1, \mathsf{add}, (1, w))$, $(2, \mathsf{add}, (2, w))$, $(3, \mathsf{del}, (1, w))$ and $(6, w)$ are present in $\mathbf{Q}_1$. Corresponding to w, $(1, \mathsf{add}, (1, w))$, $(2, \mathsf{del}, (1, w))$, $(3, \mathsf{add}, (2, w))$ and $(6, w)$ are present in $\mathbf{Q}_2$.

   Leakage functions corresponding to search query at timestamp 6 in $\mathbf{Q}_1$:

$$\begin{aligned} \mathsf{sp}(w) &= \{6\}. & o_w &= 3. \\ \mathsf{TimeDB}(w) &= \{(2,2)\}. & \mathsf{DB}(w) &= \{2\}. \\ \mathsf{Updates}(w) &= \{1,2,3\}. \end{aligned}$$

   Leakage functions corresponding to search query at timestamp 6 in $\mathbf{Q}_2$:

$$\begin{aligned} \mathsf{sp}(w) &= \{6\}. & o_w &= 3. \\ \mathsf{TimeDB}(w) &= \{(3,2)\}. & \mathsf{DB}(w) &= \{2\}. \\ \mathsf{Updates}(w) &= \{1,2,3\}. \end{aligned}$$

   As can be observed the leakage functions in $\mathcal{L}_{\mathsf{BP\text{-}I,Search}}$, i.e., $\mathsf{sp}(w)$, $\mathsf{Updates}(w)$ and $\mathsf{DB}(w)$ are the same for query lists $\mathbf{Q}_1$ and $\mathbf{Q}_2$. However, $\mathsf{TimeDB}(w)$ is different for query lists $\mathbf{Q}_1$ and $\mathbf{Q}_2$. Hence, $\mathsf{TimeDB}(w)$ cannot be uniquely determined from the leakage functions in $\mathcal{L}_{\mathsf{BP\text{-}I,Search}}$.

2. Consider the entries corresponding to w in query lists $\mathbf{Q}_1$ and $\mathbf{Q}_2$ as described below. Corresponding to w, $(1, \mathsf{add}, (1, w))$, $(2, \mathsf{add}, (2, w))$, $(3, \mathsf{del}, (1, w))$ and $(6, w)$ are present in $\mathbf{Q}_1$. Corresponding to w, $(1, \mathsf{add}, (1, w))$, $(2, \mathsf{add}, (2, w))$, $(5, \mathsf{del}, (1, w))$ and $(6, w)$ are present in $\mathbf{Q}_2$.

   Leakage functions corresponding to search query at timestamp 6 in $\mathbf{Q}_1$:

$$\begin{aligned} \mathsf{sp}(w) &= \{6\}. & o_w &= 3. \\ \mathsf{TimeDB}(w) &= \{(2,2)\}. & \mathsf{DB}(w) &= \{2\}. \\ \mathsf{Updates}(w) &= \{1,2,3\}. \end{aligned}$$

   Leakage functions corresponding to search query at timestamp 6 in $\mathbf{Q}_2$:

$$\begin{aligned} \mathsf{sp}(w) &= \{6\}. & o_w &= 3. \\ \mathsf{TimeDB}(w) &= \{(2,2)\}. & \mathsf{DB}(w) &= \{2\}. \\ \mathsf{Updates}(w) &= \{1,2,4\}. \end{aligned}$$

   As can be observed the leakage functions in $\mathcal{L}_{\mathsf{BPIP,Search}}$, i.e., $\mathsf{sp}(w)$, $\mathsf{TimeDB}(w)$ and o_w are the same for query lists $\mathbf{Q}_1$ and $\mathbf{Q}_2$. However, $\mathsf{Updates}(w)$ is different for query lists $\mathbf{Q}_1$ and $\mathbf{Q}_2$. Hence, $\mathsf{Updates}(w)$ cannot be uniquely determined from the leakage functions in $\mathcal{L}_{\mathsf{BP\text{-}I,Search}}$.

3. Consider the entries corresponding to w in query lists $\mathbf{Q}_1$ and $\mathbf{Q}_2$ as described below. Corresponding to w, $(1, \mathsf{add}, (1, w))$, $(2, \mathsf{add}, (2, w))$, $(3, \mathsf{del}, (1, w))$ and $(6, w)$ are present in $\mathbf{Q}_1$. Corresponding to w, $(1, \mathsf{add}, (1, w))$, $(2, \mathsf{del}, (1, w))$, $(3, \mathsf{add}, (2, w))$ and $(6, w)$ are present in $\mathbf{Q}_2$.

   Leakage functions corresponding to search query at timestamp 6 in $\mathbf{Q}_1$:

$$\begin{aligned} \mathsf{sp}(w) &= \{6\}. & \mathsf{Updates}(w) &= \{1,2,3\}. \\ \mathsf{DB}(w) &= \{2\}. & \mathsf{UpPair}(w) &= \{(1,3)\}. \end{aligned}$$

   Leakage functions corresponding to search query at timestamp 6 in $\mathbf{Q}_2$:

$$\begin{aligned} \mathsf{sp}(w) &= \{6\}. & \mathsf{Updates}(w) &= \{1,2,3\}. \\ \mathsf{DB}(w) &= \{2\}. & \mathsf{UpPair}(w) &= \{(1,2)\}. \end{aligned}$$

   As can be observed the leakage functions in $\mathcal{L}_{\mathsf{BP\text{-}I,Search}}$, i.e., $\mathsf{sp}(w)$, $\mathsf{Updates}(w)$ and $\mathsf{DB}(w)$ are the same for query lists $\mathbf{Q}_1$ and $\mathbf{Q}_2$. However, $\mathsf{UpPair}(w)$ is different for query lists $\mathbf{Q}_1$ and $\mathbf{Q}_2$. Hence, $\mathsf{UpPair}(w)$ cannot be uniquely determined from the leakage functions in $\mathcal{L}_{\mathsf{BP\text{-}I,Search}}$.

4. Consider the entries corresponding to w in query lists $\mathbf{Q}_1$ and $\mathbf{Q}_2$ as described below. Corresponding to w, $(1, \mathsf{add}, (1, w))$, $(2, \mathsf{add}, (2, w))$, $(3, \mathsf{del}, (1, w))$, $(4, \mathsf{del}, (2, w))$, $(5, \mathsf{add}, (3, w))$ and $(6, w)$ are present in $\mathbf{Q}_1$. Corresponding to w, $(1, \mathsf{add}, (1, w))$, $(2, \mathsf{add}, (2, w))$, $(3, \mathsf{del}, (2, w))$, $(4, \mathsf{del}, (1, w))$, $(5, \mathsf{add}, (3, w))$ and $(6, w)$ are present in $\mathbf{Q}_2$.

   Leakage functions corresponding to search query at timestamp 6 in $\mathbf{Q}_1$:

$$\mathsf{sp}(w) = \{6\}. \quad \mathsf{Updates}(w) = \{1, 2, 3, 4, 5, 6\}.$$
$$\mathsf{TimeDB}(w) = \{(5, 3)\}. \quad \mathsf{UpPair}(w) = \{(1, 3), (2, 4)\}.$$

   Leakage functions corresponding to search query at timestamp 6 in $\mathbf{Q}_2$:

$$\mathsf{sp}(w) = \{6\}. \quad \mathsf{Updates}(w) = \{1, 2, 3, 4, 5, 6\}.$$
$$\mathsf{TimeDB}(w) = \{(5, 3)\}. \quad \mathsf{UpPair}(w) = \{(1, 4), (2, 3)\}.$$

   As can be observed the leakage functions in $\mathcal{L}_{\mathtt{BPUP},\mathsf{Search}}$, i.e., $\mathsf{sp}(w)$, $\mathsf{Updates}(w)$ and $\mathsf{TimeDB}(w)$ are the same for query lists $\mathbf{Q}_1$ and $\mathbf{Q}_2$. However, $\mathsf{UpPair}(w)$ is different for query lists $\mathbf{Q}_1$ and $\mathbf{Q}_2$. Hence, $\mathsf{UpPair}(w)$ cannot be uniquely determined from the leakage functions in $\mathcal{L}_{\mathtt{BPUP},\mathsf{Search}}$.

4 Backward Private DSSE Constructions

In this section we propose three backward private schemes $\Pi_{\mathsf{BP}\text{-}\mathsf{prime}}$, Π_{BP} and Π_{WBP} that are respectively BP-I, BP-II and WBP secure. Our starting point is a forward private DSSE scheme Π_{FP} which is a modified version of the scheme proposed in [15].

4.1 Π_{FP}: A Warm-up Solution

The central idea in Π_{FP} (Figure 6) is to make updates using fresh keys. Hence, the keys disclosed in previous searches do not reveal anything about these new updates. Π_{FP} is described in Figure 6. The construction makes use of PRFs F_t, F_d: $\{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^\lambda$ and hash functions H_1: $\{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^{2\lambda}$ and H_2: $\{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^\mu$, where $\mu{=}\lambda{+}1$ and λ is security parameter.

The Setup algorithm generates secret keys k_t and k_d. C initiates three maps: $\mathbf{T}$, $\mathbf{D}$ and $\mathbf{W}$. The maps $\mathbf{T}$ and $\mathbf{D}$ are stored at S's end. Corresponding to w, $\mathbf{D}$ stores the pointer to PSet_w, the set of document identifiers in plaintext that were obtained as a result of the latest search operation on w and $\mathbf{T}$ stores encrypted entries inserted after the latest search operation on w. The map $\mathbf{W}$ is stored at C's end. In $\mathbf{W}$, corresponding to w, C stores the version ver_w (initialized to 0) and counter c_w (initialized to -1). ver_w ensures that the key k_w used in the Update protocol is unknown to S, c_w stores information about the number of entries added to $\mathbf{T}$ corresponding to w after the latest search operation.

Update: When C wants to $\mathsf{add}/\mathsf{del}$ a document-keyword pair (ind, w), it computes key k_w using keyword w and ver_w (see line 5) and increments c_w. Based on k_w and c_w, C computes the hash digests label and pad and sends $(\mathsf{label}, \mathsf{e}{=}\mathsf{pad} \oplus (\mathsf{b}||\mathsf{ind}))$ to S. S then adds $(\mathsf{label}, \mathsf{e})$ to $\mathbf{T}$. For add (resp. del) operation, $\mathsf{b}{=}0$ (resp. $\mathsf{b}{=}1$).

Note that, k_w used in processing update queries is computed using updated ver_w. Since, k_w is output of PRF F_t at $w||\mathsf{ver}_w$, it is indistinguishable from random for S. As label (resp. pad) is computed as $\mathsf{H}_1(\mathsf{k}_w||\mathsf{c}_w)$ (resp. $\mathsf{H}_2(\mathsf{k}_w||\mathsf{c}_w)$) for an update query, it is indistinguishable from random for S as H_1 (resp. H_2) is modeled as a random oracle. Hence, in the security proof, update queries can be simulated by generating random $(\mathsf{label}, \mathsf{e})$ pair.

Search: When C wants to perform a search query on w, it computes label_w (see line 5) and the key k_w is computed (see line 7) only if new entries are inserted in map $\mathbf{T}$. C sends label_w, k_w and c_w to S. k_w gets revealed to S only if a new entry was inserted to $\mathbf{T}$ after the previous search on w. Hence, C updates the version ver_w (see line 8). ver_w is not updated in a search query on w for which corresponding to w, no

$\underline{\mathsf{Setup}()}$

1. $\mathsf{k}_t \xleftarrow{\$} \{0,1\}^\lambda, \mathsf{k}_d \xleftarrow{\$} \{0,1\}^\lambda$
2. $\mathbf{W}, \mathbf{T}, \mathbf{D} \leftarrow$ empty map
3. return ($\mathsf{EDB} = (\mathbf{D}, \mathbf{T})$, $\mathsf{st_c} = (\mathsf{k}_t, \mathsf{k}_d, \mathbf{W})$) to (S, C)

$\underline{\mathsf{Update}(\mathsf{op}, w, \mathsf{ind}, \mathsf{st_C}; \mathsf{EDB})}$

$\mathsf{Update_C}(\mathsf{op}, w, \mathsf{ind}, \mathsf{st_C})$

1. $(\mathsf{ver}_w, \mathsf{c}_w) \leftarrow \mathbf{W}[w]$
2. **if** $(\mathsf{ver}_w, \mathsf{c}_w) = \perp$ **then**
3. $\quad \mathsf{ver}_w \leftarrow 0,\ \mathsf{c}_w \leftarrow -1$
4. $\mathsf{c}_w \leftarrow \mathsf{c}_w + 1$
5. $\mathsf{k}_w \leftarrow \mathsf{F}_t(\mathsf{k}_t, w \| \mathsf{ver}_w)$
6. $\mathsf{label} \leftarrow \mathsf{H}_1(\mathsf{k}_w \| \mathsf{c}_w)$
7. $\mathsf{pad} \leftarrow \mathsf{H}_2(\mathsf{k}_w \| \mathsf{c}_w)$
8. $\mathsf{b} \leftarrow 0$ (op=add)/ 1 (op=del)
9. $\mathsf{e} \leftarrow (\mathsf{b} \| \mathsf{ind}) \oplus \mathsf{pad}$
10. $\mathbf{W}[w] \leftarrow (\mathsf{ver}_w,\ \mathsf{c}_w)$
11. Send $(\mathsf{label}, \mathsf{e})$ to S

$\mathsf{Update_S}(\mathsf{EDB})$

12. Receive $(\mathsf{label}, \mathsf{e})$ from C
13. $\mathbf{T}[\mathsf{label}] \leftarrow \mathsf{e}$

$\underline{\mathsf{Search}(w, \mathsf{st_C}; \mathsf{EDB})}$

$\mathsf{Search_C}(w, \mathsf{st_C})$:

1. $(\mathsf{ver}_w, \mathsf{c}_w) \leftarrow \mathbf{W}[w]$
2. **if** $(\mathsf{ver}_w, \mathsf{c}_w) = \perp$ **then**
3. $\quad$ return $\emptyset$
4. **else**
5. $\quad \mathsf{label}_w \leftarrow \mathsf{F}_d(\mathsf{k}_d, w)$
6. $\quad$ **if** $\mathsf{c}_w \neq -1$ **then**
7. $\quad\quad \mathsf{k}_w \leftarrow \mathsf{F}_t(\mathsf{k}_t, w \| \mathsf{ver}_w)$
8. $\quad\quad \mathsf{ver}_w \leftarrow \mathsf{ver}_w + 1, \mathsf{c}_w \leftarrow -1$
9. $\quad\quad \mathbf{W}[w] \leftarrow (\mathsf{ver}_w,\ \mathsf{c}_w)$
10. $\quad$ **else**
11. $\quad\quad \mathsf{k}_w \leftarrow \perp$
12. $\quad$ Send $(\mathsf{label}_w, \mathsf{k}_w, \mathsf{c}_w)$ to S

$\mathsf{Search_S}(\mathsf{EDB})$:

13. Receive $(\mathsf{label}_w, \mathsf{k}_w, \mathsf{c}_w)$ from C
14. $\mathsf{addr}(\mathsf{PSet}_w) \leftarrow \mathbf{D}[\mathsf{label}_w]$
15. Retrieve PSet_w and set $\mathsf{AuxSet} \leftarrow \mathsf{PSet}_w$
16. **if** $\mathsf{k}_w \neq \perp$ **then**
17. $\quad \mathsf{c} \leftarrow 0$
18. $\quad$ **while** $\mathsf{c} \leq \mathsf{c}_w$ **do**
19. $\quad\quad \mathsf{label} \leftarrow \mathsf{H}_1(\mathsf{k}_w \| \mathsf{c}),\ \mathsf{e} \leftarrow \mathbf{T}[\mathsf{label}]$
20. $\quad\quad \mathsf{pad} \leftarrow \mathsf{H}_2(\mathsf{k}_w \| \mathsf{c}),\ (\mathsf{b} \| \mathsf{ind}) \leftarrow \mathsf{e} \oplus \mathsf{pad}$
21. $\quad\quad$ **if** $\mathsf{b} = 0$
22. $\quad\quad\quad \mathsf{AuxSet} \leftarrow \mathsf{AuxSet} \cup \{\mathsf{ind}\}$
23. $\quad\quad$ **else**
24. $\quad\quad\quad \mathsf{AuxSet} \leftarrow \mathsf{AuxSet} \setminus \{\mathsf{ind}\}$
25. $\quad\quad$ Remove $\mathbf{T}[\mathsf{label}]$
26. $\quad\quad \mathsf{c} \leftarrow \mathsf{c} + 1$
27. Store AuxSet in disk
28. Set $\mathbf{D}[\mathsf{label}_w] \leftarrow \mathsf{addr}(\mathsf{AuxSet})$
29. Send AuxSet to C

Figure 6: Scheme Π_{FP}

updates on map $\mathbf{T}$ were made after the previous search on w. Based on the information received from C, S computes the result set and updates $\mathbf{D}$ with the newly computed result set.

In Π_{FP}, ver_w ensures that S cannot relate later updates with previous search queries and c_w ensures that S cannot correlate update queries on w after the previous search operation on w.

Remark Essentially, Π_{FP} is same as the construction in [15] except the following: 1) The search counter corresponding to w (denoted as ver_w) is updated differently than in [15] to avoid unnecessary increments to the search counter, 2) After a search operation on w, the revealed document identifiers are stored together ($\mathsf{DB}(w)$) in plaintext (as suggested in [7]) to provide reasonable locality without incurring any additional leakage.

Example 1 Consider the following list of update queries: $(\mathsf{add}, (1, w_1))$, $(\mathsf{add}, (1, w_2))$, $(\mathsf{add}, (2, w_1))$, $(\mathsf{add}, (3, w_1))$, $(\mathsf{add}, (3, w_2))$, $(\mathsf{del}, (1, w_1))$. Figure 7.a shows the state of indexes at C and S after these updates are processed. Figure 7.b shows the state of indexes at C and S after search on w_1.

Correctness The scheme is correct as long as there are no repeated labels in maps $\mathbf{T}$ and $\mathbf{D}$. Since, F_d is a PRF, only with negligible probability label_w in $\mathbf{D}$ is same for two distinct keywords. The input to H_1 is repeated only with negligible probability as F_t is a PRF. If we consider H_1 to be a collision resistant hash function, only with negligible probability label in $\mathbf{T}$ is repeated.

In Appendix A, we provide complete proofs of correctness and FP-I-security (see Definition 3.1) of Π_{FP}.

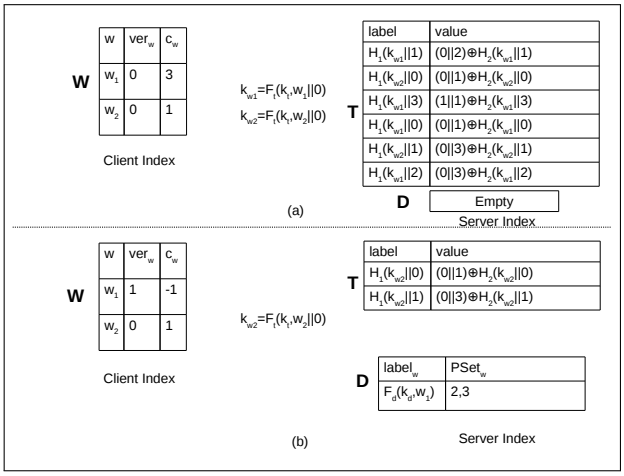


Figure 7: Example 1: Indexes at C and S before and after search on w_1 in construction Π_{FP}. $\mathbf{W}$ = Client Index, $\mathbf{D}$ = Index that stores search results of previous search query at S and $\mathbf{T}$ = Index that stores updates after the last search on w at S

4.2 Π_{BP}-prime : Realizing Strong Privacy

As Π_{FP} reveals the deleted entries, it isn't backward private. We propose backward private DSSE scheme Π_{BP}-prime that makes use of light weight symmetric primitives only. The scheme in Figure 8 is similar to applying the generic two-roundtrip backward-private scheme transformation [7] on Π_{FP} but with the following essential modification: the document identifiers revealed by search query are stored together in plaintext thus providing reasonable locality by avoiding re-encryption of revealed identifiers (as mentioned in Section 4.1). A high degree of locality ensures that the document identifiers are stored in contiguous memory locations which results in significant I/O efficiency. The construction makes use of PRFs F_t, F_d: $\{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^\lambda$, symmetric encryption scheme $\mathcal{E}$ and hash function H_1: $\{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^{2\lambda}$.

Unlike Π_{FP}, encryption of $\mathsf{b}\|\mathsf{ind}$ is stored in $\mathbf{T}$ at S in Update protocol (see line 8). Padding in Π_{FP} using H_2 was necessary to prevent S from learning anything about the update. However, in Π_{BP}-prime, encryption of the updated entry is stored which is random in the view of S, thus revealing no information about the updated entry.

The Search protocol in Π_{BP}-prime comprises of two rounds. Round 1 of Search is similar to the Search protocol of Π_{FP}. At the end of round 1 of Search, S sends PSet_w, the result set of previous search on w and AList, the list of ciphertexts corresponding to all the updates on w after the previous search on w, in order of their insertion. While preparing AList, S only learns the timestamps at which updates corresponding to w have happened because the entries in AList are encrypted. Based on the information received from S, C computes AuxSet (see line 28-35 of Search protocol). AuxSet consists of all the document identifiers currently matching keyword w in random order. In the second round of communication, AuxSet is forwarded to S, who stores AuxSet in $\mathbf{D}[\mathsf{label}_w]$.

Correctness As mentioned in correctness of Π_{FP}, the scheme is correct as long as there are no repeated labels in maps $\mathbf{T}$ and $\mathbf{D}$. Since, these labels are generated in the same manner as they were generated in Π_{FP}, correctness of Π_{BP}-prime immediately follows from that of Π_{FP}.

Asymptotic Complexity The communication complexity in round 1 of the Search protocol is $O(o'_w)$ and that of round 2 is $O(n_w)$. The computational complexity of the Search protocol is $O(o'_w)$. The communication and computation cost of the Update protocol is $O(1)$. Space complexity at the server's end is $O(N + D')$, where $D' = \sum_{\forall w} d'_w$ and at the client's end is $O(m\, log(n))$.

Setup()

1. $\mathsf{k}_t \xleftarrow{\$} \{0,1\}^\lambda$, $\mathsf{k}_d \xleftarrow{\$} \{0,1\}^\lambda$
2. $\mathsf{k}_e \leftarrow \mathcal{E}.\mathsf{Gen}()$
3. $\mathbf{W}, \mathbf{T}, \mathbf{D} \leftarrow$ empty map
4. return ($\mathsf{EDB} = (\mathbf{D}, \mathbf{T})$, $\mathsf{st_C} = (\mathsf{k}_e, \mathsf{k}_t, \mathsf{k}_d, \mathbf{W})$) to (S, C)

Update$(\mathsf{op}, w, \mathsf{ind}, \mathsf{st_C}; \mathsf{EDB})$

Update$_\mathsf{C}(\mathsf{op}, w, \mathsf{ind}, \mathsf{st_C})$

1. $(\mathsf{ver}_w, \mathsf{c}_w) \leftarrow \mathbf{W}[w]$
2. **if** $(\mathsf{ver}_w, \mathsf{c}_w) = \perp$ **then**
3. $\quad \mathsf{ver}_w \leftarrow 0,\ \mathsf{c}_w \leftarrow -1$
4. $\mathsf{c}_w \leftarrow \mathsf{c}_w + 1$
5. $\mathsf{k}_w \leftarrow \mathsf{F}_t(\mathsf{k}_t, w \| \mathsf{ver}_w)$
6. $\mathsf{label} \leftarrow \mathsf{H}_1(\mathsf{k}_w \| \mathsf{c}_w)$
7. $\mathsf{b} \leftarrow 0$ (op=add)/ 1 (op=del)
8. $\mathsf{e} \leftarrow \mathcal{E}.\mathsf{Enc}(\mathsf{k}_e, \mathsf{b} \| \mathsf{ind})$
9. $\mathbf{W}[w] \leftarrow (\mathsf{ver}_w,\ \mathsf{c}_w)$
10. Send $(\mathsf{label}, \mathsf{e})$ to S

Update$_\mathsf{S}$(EDB)

11. Receive $(\mathsf{label}, \mathsf{e})$ from C
12. $\mathbf{T}[\mathsf{label}] \leftarrow \mathsf{e}$

Search$(w, \mathsf{st_C}; \mathsf{EDB})$

Round 1

Search$_\mathsf{C}(w, \mathsf{st_C})$:

1. $(\mathsf{ver}_w, \mathsf{c}_w) \leftarrow \mathbf{W}[w]$
2. **if** $(\mathsf{ver}_w, \mathsf{c}_w) = \perp$ **then**
3. $\quad$ return $\emptyset$
4. **else**
5. $\quad \mathsf{label}_w \leftarrow \mathsf{F}_d(\mathsf{k}_d, w)$
6. $\quad$ **if** $\mathsf{c}_w \neq -1$ **then**
7. $\quad\quad \mathsf{k}_w \leftarrow \mathsf{F}_t(\mathsf{k}_t, w \| \mathsf{ver}_w)$
8. $\quad\quad \mathsf{ver}_w \leftarrow \mathsf{ver}_w + 1$
9. $\quad\quad \mathsf{c}_w \leftarrow -1$
10. $\quad\quad \mathbf{W}[w] \leftarrow (\mathsf{ver}_w,\ \mathsf{c}_w)$
11. $\quad$ **else**
12. $\quad\quad \mathsf{k}_w \leftarrow \perp$ //No need of round 2
13. $\quad$ Send $(\mathsf{label}_w, \mathsf{k}_w, \mathsf{c}_w)$ to S

Search$_\mathsf{S}$(EDB) :

14. Receive $(\mathsf{label}_w, \mathsf{k}_w, \mathsf{c}_w)$ from C
15. $\mathsf{addr}(\mathsf{PSet}_w) \leftarrow \mathbf{D}[\mathsf{label}_w]$
16. Retrieve PSet_w
17. $\mathsf{AList} \leftarrow \emptyset$
18. **if** $\mathsf{k}_w \neq \perp$ **then**
19. $\quad \mathsf{c} \leftarrow 0$
20. $\quad$ **while** $\mathsf{c} \leq \mathsf{c}_w$ **do**
21. $\quad\quad \mathsf{label} \leftarrow \mathsf{H}_1(\mathsf{k}_w \| \mathsf{c})$
22. $\quad\quad \mathsf{e} \leftarrow \mathbf{T}[\mathsf{label}]$
23. $\quad\quad$ Append e to AList
24. $\quad\quad$ Remove $\mathbf{T}[\mathsf{label}]$
25. $\quad\quad \mathsf{c} \leftarrow \mathsf{c} + 1$
26. Send $(\mathsf{PSet}_w, \mathsf{AList})$ to C

Round 2

Search$_\mathsf{C}(w, \mathsf{st_C})$:

27. Receive $(\mathsf{PSet}_w, \mathsf{AList})$ from S
28. $(\mathsf{ver}_w, \mathsf{c}_w) \leftarrow \mathbf{W}[w]$
29. $\mathsf{AuxSet} \leftarrow \mathsf{PSet}_w$
30. **for** $\mathsf{c} \leftarrow 0$ to $|\mathsf{AList}|$ **do**
31. $\quad (\mathsf{b} \| \mathsf{ind}) \leftarrow \mathcal{E}.\mathsf{Dec}(\mathsf{k}_e, \mathsf{AList}[\mathsf{c}])$
32. $\quad$ **if** $\mathsf{b} = 0$
33. $\quad\quad \mathsf{AuxSet} \leftarrow \mathsf{AuxSet} \cup \{\mathsf{ind}\}$
34. $\quad$ **else**
35. $\quad\quad \mathsf{AuxSet} \leftarrow \mathsf{AuxSet} \setminus \{\mathsf{ind}\}$
36. return AuxSet

Search$_\mathsf{S}$(EDB) :

37. Receive AuxSet from C
38. $\mathbf{D}[\mathsf{label}_w] \leftarrow \mathsf{AuxSet}$

Figure 8: Scheme Π_{BP}-prime

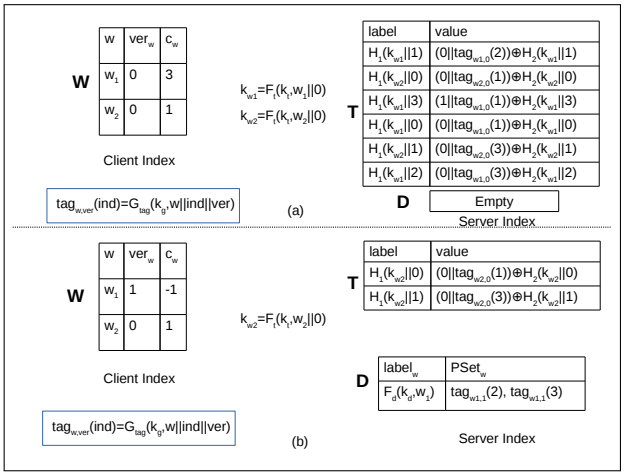


Figure 9: Example 2: Indexes at C and S before and after search on w_1 in Π_{BP}. $\mathbf{W}$ = Client Index, $\mathbf{D}$ = Index that stores tags of search results of previous search query at S and $\mathbf{T}$ = Index that stores updates after the last search on w at S

Security We prove Π_{BP}-prime is BP-I secure in the random oracle model. The proof relies on the pseudo randomness of F_d, F_t and RCPA security of encryption scheme $\mathcal{E}$. The complete proof appears in Appendix B.

In a concurrent work [20], a backward private construction Mitra was proposed independently. Mitra was further optimized to obtain Mitra* in [20]. Π_{BP}-prime is similar to Mitra* except the following minor difference: the revealed document identifiers in Π_{BP}-prime are stored in plaintext whereas in Mitra* they are re-encrypted. This property of Π_{BP}-prime results in better locality. The leakage profile of constructions Π_{BP}-prime and Mitra* are same.

4.3 Π_{BP}: Realizing Optimal Communication Complexity

As can be observed from the asymptotic analysis of Π_{BP}-prime, the communication complexity of Search protocol is $O(o'_w)$ which is not optimal i.e., $O(n_w)$. Further, as C has to process each ciphertext it receives from S, the computation complexity at C's end also becomes $O(o'_w)$ due to the above communication overhead. In order to obtain optimal communication complexity, S should send entries corresponding to only the set of documents currently matching w ($\mathsf{DB}(w)$) to C. One approach to satisfy the above requirement is to associate a *tag* corresponding to each update entry. Using these tags, S, while performing a search on w will be able to correlate the update queries on w corresponding to the same ind. As in $\mathsf{Diana}_{\mathsf{del}}$ and Janus, if the tags are generated deterministically using just ind and w, it leaks $\mathsf{DelHist}(w)$ and hence, will not satisfy BP-II. In Π_{BP}, we leverage the version ver_w to generate the tags in a clever yet simple manner to ensure that the leakage is bounded by Definition 3.6.

Scheme Π_{BP} is described in Figure 10. For a keyword w, $\mathbf{D}$ stores the pointer to PSet_w, the set of tags corresponding to document identifiers that were obtained as a result of the latest search operation on w. The construction makes use of PRFs F_t, F_d: $\{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^\lambda$, PRP $\mathsf{G}_{\mathsf{tag}}$: $\{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ and hash functions H_1: $\{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^{2\lambda}$ and H_2: $\{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^\mu$, where μ=λ+1.

For an update query $(\mathsf{op}, (\mathsf{ind}, w))$, a tag corresponding to (ind, w) is generated using the current version ver_w. Then, b‖tag is stored in $\mathbf{T}$ at S in Update protocol (see line 10).

Round 1 of Search protocol is similar to the Search protocol of Π_{FP}. At the end of round 1 of Search, S sends TS, the set of tags corresponding to the document identifiers currently matching w in order of their insertion. For every tag in TS, C computes $\mathsf{G}_{\mathsf{tag}}^{-1}$ to get the document identifier ind which it adds to AuxSet and re-computes tag using the updated version which it stores in PSet. AuxSet consists of all the document identifiers currently matching keyword w and PSet consists of the updated tags, in order of their insertion.

Setup()

1. $\mathsf{k}_t \xleftarrow{\$} \{0,1\}^\lambda, \mathsf{k}_d \xleftarrow{\$} \{0,1\}^\lambda,$ $\mathsf{k}_g \xleftarrow{\$} \{0,1\}^\lambda$
2. $\mathbf{W}, \mathbf{T}, \mathbf{D} \leftarrow$ empty map
3. return ($\mathsf{EDB} = (\mathbf{D}, \mathbf{T})$, $\mathsf{st_c} = (\mathsf{k}_t, \mathsf{k}_d, \mathsf{k}_g, \mathbf{W})$) to (S, C)

Update$(\mathsf{op}, w, \mathsf{ind}, \mathsf{st_C}; \mathsf{EDB})$

Update$_\mathsf{C}(\mathsf{op}, w, \mathsf{ind}, \mathsf{st_C})$

1. $(\mathsf{ver}_w, \mathsf{c}_w) \leftarrow \mathbf{W}[w]$
2. **if** $(\mathsf{ver}_w, \mathsf{c}_w) = \perp$ **then**
3. $\quad \mathsf{ver}_w \leftarrow 0,\ \mathsf{c}_w \leftarrow -1$
4. $\mathsf{c}_w \leftarrow \mathsf{c}_w + 1$
5. $\mathsf{k}_w \leftarrow \mathsf{F}_t(\mathsf{k}_t, w \| \mathsf{ver}_w)$
6. $\mathsf{label} \leftarrow \mathsf{H}_1(\mathsf{k}_w \| \mathsf{c}_w)$
7. $\mathsf{pad} \leftarrow \mathsf{H}_2(\mathsf{k}_w \| \mathsf{c}_w)$
8. $\mathsf{tag} \leftarrow \mathsf{G}_{\mathsf{tag}}(\mathsf{k}_g, w \| \mathsf{ind} \| \mathsf{ver}_w)$
9. $\mathsf{b} \leftarrow 0$ (op=add)/ 1 (op=del)
10. $\mathsf{e} \leftarrow (\mathsf{b} \| \mathsf{tag}) \oplus \mathsf{pad}$
11. $\mathbf{W}[w] \leftarrow (\mathsf{ver}_w,\ \mathsf{c}_w)$
12. Send $(\mathsf{label}, \mathsf{e})$ to S

Update$_\mathsf{S}$(EDB)

13. Receive $(\mathsf{label}, \mathsf{e})$ from C
14. $\mathbf{T}[\mathsf{label}] \leftarrow \mathsf{e}$

Search$(w, \mathsf{st_C}; \mathsf{EDB})$

Round 1

Search$_\mathsf{C}(w, \mathsf{st_C})$:

1. $(\mathsf{ver}_w, \mathsf{c}_w) \leftarrow \mathbf{W}[w]$
2. **if** $(\mathsf{ver}_w, \mathsf{c}_w) = \perp$ **then**
3. $\quad$ return $\emptyset$
4. **else**
5. $\quad \mathsf{label}_w \leftarrow \mathsf{F}_d(\mathsf{k}_d, w)$
6. $\quad$ **if** $\mathsf{c}_w \neq -1$ **then**
7. $\quad\quad \mathsf{k}_w \leftarrow \mathsf{F}_t(\mathsf{k}_t, w \| \mathsf{ver}_w)$
8. $\quad\quad \mathsf{ver}_w \leftarrow \mathsf{ver}_w + 1$
9. $\quad\quad \mathsf{c}_w \leftarrow -1$
10. $\quad\quad \mathbf{W}[w] \leftarrow (\mathsf{ver}_w,\ \mathsf{c}_w)$
11. $\quad$ **else**
12. $\quad\quad \mathsf{k}_w \leftarrow \perp$ //No need of round 2
13. $\quad$ Send $(\mathsf{label}_w, \mathsf{k}_w, \mathsf{c}_w)$ to S

Search$_\mathsf{S}$(EDB) :

14. Receive $(\mathsf{label}_w, \mathsf{k}_w, \mathsf{c}_w)$ from C
15. $\mathsf{addr}(\mathsf{PSet}_w) \leftarrow \mathbf{D}[\mathsf{label}_w]$
16. Retrieve PSet_w
17. $\mathsf{TS} \leftarrow \mathsf{PSet}_w$
18. **if** $\mathsf{k}_w \neq \perp$ **then**
19. $\quad \mathsf{c} \leftarrow 0$
20. $\quad$ **while** $\mathsf{c} \leq \mathsf{c}_w$ **do**
21. $\quad\quad \mathsf{label} \leftarrow \mathsf{H}_1(\mathsf{k}_w \| \mathsf{c})$
22. $\quad\quad \mathsf{e} \leftarrow \mathbf{T}[\mathsf{label}]$
23. $\quad\quad \mathsf{pad} \leftarrow \mathsf{H}_2(\mathsf{k}_w \| \mathsf{c})$
24. $\quad\quad (\mathsf{b} \| \mathsf{tag}) \leftarrow \mathsf{e} \oplus \mathsf{pad}$
25. $\quad\quad$ **if** $\mathsf{b} = 0$
26. $\quad\quad\quad \mathsf{TS} \leftarrow \mathsf{TS} \cup \{\mathsf{tag}\}$
27. $\quad\quad$ **else**
28. $\quad\quad\quad \mathsf{TS} \leftarrow \mathsf{TS} \setminus \{\mathsf{tag}\}$
29. $\quad\quad$ Remove $\mathbf{T}[\mathsf{label}]$
30. $\quad\quad \mathsf{c} \leftarrow \mathsf{c} + 1$
31. Send TS to C

Round 2

Search$_\mathsf{C}(w, \mathsf{st_C})$:

32. Receive TS from S
33. $(\mathsf{ver}_w, \mathsf{c}_w) \leftarrow \mathbf{W}[w]$
34. $\mathsf{PSet} \leftarrow \emptyset,\ \mathsf{AuxSet} \leftarrow \emptyset$
35. **for** all $\mathsf{tag} \in \mathsf{TS}$ **do**
36. $\quad (w \| \mathsf{ind} \| \mathsf{ver}') \leftarrow \mathsf{G}_{\mathsf{tag}}^{-1}(\mathsf{k}_g, \mathsf{tag})$
37. $\quad \mathsf{AuxSet} \leftarrow \mathsf{AuxSet} \cup \{\mathsf{ind}\}$
38. $\quad \mathsf{PSet} \leftarrow \mathsf{PSet} \cup \{\ \mathsf{G}_{\mathsf{tag}}(\mathsf{k}_g, w \| \mathsf{ind} \| \mathsf{ver}_w)\}$
39. return PSet, AuxSet

Search$_\mathsf{S}$(EDB) :

40. Receive PSet, AuxSet from C
41. $\mathbf{D}[\mathsf{label}_w] \leftarrow \mathsf{PSet}$

Figure 10: Scheme Π_{BP}

C sends PSet and AuxSet to S, who stores PSet at $\mathbf{D}[\mathsf{label}_w]$ and can use AuxSet to fetch the matching documents. Note that, recomputed tags enables S to consistently handle future searches and updates as the tags corresponding to subsequent updates on w are made using the same value of ver_w.

Let U be the set of update queries corresponding to w after the previous search and before the current search on w. The tags are generated using the same ver_w exclusively $\forall \mathsf{q_u} \in \mathsf{U}$ and $\forall\ \mathsf{ind} \in \mathsf{DB}(w)'$. Since, the tags are computed using PRP G_{tag} taking w, ind and ver_w as input, S can only learn the relation between the tags corresponding to the same ind among the update queries in U and $\mathsf{DB}(w)'$. As the inds in PSet are stored in order of insertion, S can link the update queries in U with these inds. The above leakage is precisely captured in BP-Π via leakage functions UpPair, PreUp and CurUp. Moreover, S cannot relate update queries in U with the update queries that are made before the previous search on w and after the current search on w as the tags are generated using different ver_w.

Example 2 Consider the following list of update queries: $(\mathsf{add}, (1, w_1))$, $(\mathsf{add}, (1, w_2))$, $(\mathsf{add}, (2, w_1))$, $(\mathsf{add}, (3, w_1))$, $(\mathsf{add}, (3, w_2))$, $(\mathsf{del}, (1, w_1))$. Figure 9.a shows the state of indexes at C and S after these updates are processed. Figure 9.b shows the state of indexes at C and S after search on w_1.

In conclusion, Π_{BP} achieves optimal communication complexity and uses symmetric primitives only. $\mathbf{D}$ provides reasonable locality as it stores the *tags* corresponding to previous search results together. Π_{BP} is easily parallelizable and doesn't impose reinsertion restriction.

Further, in order to obtain a response-hiding scheme, C sends only PSet to S in line 39 in second round

Setup()

$\mathbf{W}, \mathbf{T}, \mathbf{D} \leftarrow$ empty map
$\mathsf{bad} \leftarrow \mathsf{false}$
return $(\mathsf{EDB} = (\mathbf{D}, \mathbf{T}), \mathsf{st_c} = \mathbf{W})$ to (S, C)

Update$(\mathsf{op}, w, \mathsf{ind}, \mathsf{st_C}; \mathsf{EDB})$

Update$_\mathsf{C}(\mathsf{op}, w, \mathsf{ind}, \mathsf{st_C})$

$(\mathsf{ver}_w, \mathsf{c}_w) \leftarrow \mathbf{W}[w]$
if $(\mathsf{ver}_w, \mathsf{c}_w) = \perp$ **then**
  $\mathsf{ver}_w \leftarrow 0,\ \mathsf{c}_w \leftarrow -1$
$\mathsf{c}_w \leftarrow \mathsf{c}_w + 1$
$\mathsf{k}_w \leftarrow \mathsf{Key}_t[w, \mathsf{ver}_w]$
$\mathsf{val}_r \xleftarrow{\$} \{0,1\}^\lambda$
if $\mathbf{H}_1[\mathsf{k}_w, \mathsf{c}_w] \neq \perp$ **then**
  $\mathsf{bad} \leftarrow \mathsf{true}$, $\boxed{\mathsf{val}_r \leftarrow \mathsf{H}_1(\mathsf{k}_w \| \mathsf{c}_w)}$
$\mathsf{Hash}_1[w, \mathsf{ver}_w, \mathsf{c}_w] \leftarrow \mathsf{val}_r$
$\mathsf{pad} \leftarrow \mathsf{H}_2(\mathsf{k}_w \| \mathsf{c}_w)$
$\mathsf{tag} \leftarrow \mathsf{Key}_{\mathsf{tag}}[w, \mathsf{ind}, \mathsf{ver}_w]$
$\mathsf{b} \leftarrow 0$ (op=add)/ 1 (op=del)
$\mathsf{e} \leftarrow (\mathsf{b} \| \mathsf{tag}) \oplus \mathsf{pad}$
$\mathbf{W}[w] \leftarrow (\mathsf{ver}_w,\ \mathsf{c}_w)$
Send $(\mathsf{val}_r, \mathsf{e})$ to S

Search$(w, \mathsf{st_C}; \mathsf{EDB})$

Round 1

Search$_\mathsf{C}(w, \mathsf{st_C})$:

$(\mathsf{ver}_w, \mathsf{c}_w) \leftarrow \mathbf{W}[w]$
if $(\mathsf{ver}_w, \mathsf{c}_w) = \perp$ **then**
  return $\emptyset$
else
  $\mathsf{label}_w \leftarrow \mathsf{Key}_d[w]$
  if $\mathsf{c}_w \neq -1$ **then**
    $\mathsf{k}_w \leftarrow \mathsf{Key}_t[w, \mathsf{ver}_w]$
    for $i = 0$ to c_w **do**
      $\mathbf{H}_1[\mathsf{k}_w, i] \leftarrow \mathsf{Hash}_1[w, \mathsf{ver}_w, i]$
    $\mathsf{ver}_w \leftarrow \mathsf{ver}_w + 1$
    $\mathsf{c}_w \leftarrow -1$
    $\mathbf{W}[w] \leftarrow (\mathsf{ver}_w,\ \mathsf{c}_w)$
  else
    $\mathsf{k}_w \leftarrow \perp$ //No need of round 2
  Send $(\mathsf{label}_w, \mathsf{k}_w, \mathsf{c}_w)$ to S

Round 2

Search$_\mathsf{C}(w, \mathsf{st_C})$:

Receive TS from S
$(\mathsf{ver}_w, \mathsf{c}_w) \leftarrow \mathbf{W}[w]$
$\mathsf{PSet} \leftarrow \emptyset,\ \mathsf{AuxSet} \leftarrow \emptyset$
for all $\mathsf{tag} \in \mathsf{TS}$ **do**
  $(w, \mathsf{ind}, \mathsf{ver}') \leftarrow \mathsf{Key}_{\mathsf{tag}}^{-1}[\mathsf{tag}]$
  $\mathsf{AuxSet} \leftarrow \mathsf{AuxSet} \cup \{\mathsf{ind}\}$
  $\mathsf{tag}' \leftarrow \mathsf{Key}_{\mathsf{tag}}[w, \mathsf{ind}, \mathsf{ver}_w]$
  $\mathsf{PSet} \leftarrow \mathsf{PSet} \cup \{\mathsf{tag}'\}$
return PSet, AuxSet

$\mathsf{H}_1(\mathsf{k} \| \mathsf{c})$

$\mathsf{val} \leftarrow \mathbf{H}_1[\mathsf{k}, \mathsf{c}]$
if $\mathsf{val} = \perp$ **then**
  $\mathsf{val} \xleftarrow{\$} \{0,1\}^\lambda$
  if $\exists w, \mathsf{ver}$ s.t. $\mathsf{ver} = \mathbf{W}[w].\mathsf{ver}_w \wedge \mathsf{k} = \mathsf{Key}_t[w, \mathsf{ver}] \wedge \mathsf{c} \leq \mathbf{W}[w].\mathsf{c}_w$ **then**
    $\mathsf{bad} \leftarrow \mathsf{true}$, $\boxed{\mathsf{val} \leftarrow \mathsf{Hash}_1[w, \mathsf{ver}, \mathsf{c}]}$
  $\mathbf{H}_1[\mathsf{k}, \mathsf{c}] \leftarrow \mathsf{val}$
return val

Figure 11: Games G_3 and G'_3 (Theorem 4.1). G'_3 includes the box code and G_3 does not.

of search protocol. We can eliminate the second round of communication using standard piggybacking technique [15, 18] and upload the updated tag set PSet with the next search query, thus, achieving a *single roundtrip* response-hiding backward private DSSE protocol.

Correctness Like Π_{BP}-prime, correctness of Π_{BP} immediately follows from that of Π_{FP}.

Asymptotic Complexity The communication complexity of the Search protocol is $O(n_w)$. The computational complexity of the Search protocol is $O(o'_w)$. The communication and computational cost of the Update protocol is $O(1)$. Space complexity at the server's end is $O(N + D')$, where $D' = \sum_{\forall w} d'_w$ and at the client's end is $O(m\, log(n))$.

Security We prove Π_{BP} is BP-Π in the random oracle model. The proof relies on pseudo randomness of F_d, F_t and G_{tag}.

Theorem 4.1. *If* F_d, F_t *are secure* PRF*s,* $\mathsf{G}_{\mathsf{tag}}$ *is a secure* PRP *and* H_1, H_2 *are hash functions modeled as random oracles outputting* 2λ *and* μ *bits respectively then* Π_{BP} *is* BP-Π *secure (Definition 3.6).*

Proof. We structure our proof using a sequence of games G_0 to G_5. G_0 will compute a distribution identical to $\mathbf{Real}_{\mathcal{A}}^{\Pi_{\mathsf{BP}}}(\lambda)$ and G_5 will compute a distribution that can be simulated perfectly given the leakage profile $\mathcal{L}$, i.e., its distribution is identical to $\mathbf{Ideal}_{\mathcal{A},\mathsf{Sim}}^{\Pi_{\mathsf{BP}}}(\lambda)$.

Game G_0: G_0 is exactly identical to $\mathbf{Real}_{\mathcal{A}}^{\Pi_{\mathsf{BP}}}(\lambda)$.

$$\Pr[\mathbf{Real}_{\mathcal{A}}^{\Pi_{\mathsf{BP}}}(\lambda) = 1] = \Pr[\mathsf{G}_0 = 1]. \tag{8}$$

Game G_1: In G_1, every call to PRFs F_t and F_d are answered using tables Key_t and Key_d respectively. The entries in table Key_t are referred by (w, ver) and entries in table Key_d are referred by w. Conventionally, when an entry is being accessed for the first time, it is chosen at random and then used thereafter, which is followed in the rest of the paper unless mentioned explicitly. If there exists an adversary $\mathcal{A}$ that is able to distinguish between games G_0 and G_1, we can construct an adversary $\mathcal{B}_1$ that can distinguish F_t from a truly random function and/or an adversary $\mathcal{B}_2$ that can distinguish F_d from a truly random function. Formally, there exist adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$, such that

$$\begin{aligned} |\Pr[\mathsf{G}_0 = 1] - \Pr[\mathsf{G}_1 = 1]| &\leq \mathbf{Adv}^{\mathsf{prf}}_{\mathsf{F}_t, \mathcal{B}_1}(\lambda) + \mathbf{Adv}^{\mathsf{prf}}_{\mathsf{F}_d, \mathcal{B}_2}(\lambda) \\ &\leq \mathsf{neg}(\lambda). \end{aligned} \tag{9}$$

Game G_2: In G_2, every call to PRP $\mathsf{G}_{\mathsf{tag}}$ is answered using table $\mathsf{Key}_{\mathsf{tag}}$. The entries in table $\mathsf{Key}_{\mathsf{tag}}$ are referred by $(w, \mathsf{ind}, \mathsf{ver})$. If the randomly generated tag has been selected earlier in $\mathsf{Key}_{\mathsf{tag}}$, G_2 is aborted. Since, the number of queries to PRP $\mathsf{G}_{\mathsf{tag}}$, say q, is a polynomial in security parameter, by the birthday bound we can conclude that the probability that the two tags are equal is at most $\frac{q^2}{2^\lambda}$, i.e., $\mathsf{neg}(\lambda)$. Therefore, G_2 aborts with negligible probability.

Now, if there exists an adversary $\mathcal{A}$ that is able to distinguish between games G_1 and G_2, we can construct an adversary $\mathcal{B}$ that can distinguish $\mathsf{G}_{\mathsf{tag}}$ from a truly random permutation. Formally, there exists an adversary $\mathcal{B}$, such that

$$\begin{aligned} |\Pr[\mathsf{G}_1 = 1] - \Pr[\mathsf{G}_2 = 1]| &\leq \mathbf{Adv}^{\mathsf{prp}}_{\mathsf{G}_{\mathsf{tag}}, \mathcal{B}}(\lambda) + \frac{q^2}{2^\lambda} \\ &\leq \mathsf{neg}(\lambda). \end{aligned} \tag{10}$$

Game G_3: In G_3, instead of calling H_1 to generate label in the Update protocol, we pick random strings. Then, during the Search protocol, the random oracle H_1 is programmed accordingly, to ensure consistency.

Tables Hash_1 and $\mathbf{H}_1$ are used to simulate the random oracle H_1, the entries in the table Hash_1 are referred by $(w, \mathsf{ver}, \mathsf{c})$ and in the table $\mathbf{H}_1$ by (k, c).

Figure 11 formally describes G_3, and an intermediate game G'_3. In G'_3, H_1 is never programmed to two different values for the same input, thus, ensuring consistency. Instead of storing the randomly picked value in table Hash_1 at position $(w, \mathsf{ver}_w, \mathsf{c}_w)$, one first checks whether H_1 is already programmed at value (k_w, c_w) which can happen if there was a query to the random oracle H_1 with input $(k_w \| \mathsf{c}_w)$. If the check is true, the value $\mathsf{H}_1(\mathsf{k}_w \| \mathsf{c}_w)$ is stored in $\mathsf{Hash}_1[w, \mathsf{ver}_w, \mathsf{c}_w]$ else the randomly picked value is stored in $\mathsf{Hash}_1[w, \mathsf{ver}_w, \mathsf{c}_w]$. The random oracle when needed in the Search protocol in line 9 or by an adversary's query to random oracle H_1 in line 5 is lazily programmed in G_3, so that the outputs are consistent throughout.

The only difference between game G_2 and G'_3 is how we model the random oracle H_1. The outputs of H_1 is perfectly indistinguishable in both these games, therefore,

$$\Pr[\mathsf{G}'_3 = 1] = \Pr[\mathsf{G}_2 = 1]. \tag{11}$$

Let us denote the event 'the flag bad is set to true' in G'_3 by $\mathbb{E}_1$. The games G'_3 and G_3 are also perfectly identical unless the event $\mathbb{E}_1$ occurs, and we can apply *identical-until-bad* technique [3] to bound the distinguishing advantage between G'_3 and G_3.

$$|\Pr[\mathsf{G}'_3 = 1] - \Pr[\mathsf{G}_3 = 1]| \leq \Pr[\mathbb{E}_1]. \tag{12}$$

The event $\mathbb{E}_1$ occurs in line 8 of Update protocol and in line 5 of H_1 algorithm. The former captures the fact that adversary has already queried random oracle H_1 at input $(\mathsf{k}_w \| \mathsf{c}_w)$ and the latter captures the fact that the adversary queries random oracle H_1 on a valid input $(\mathsf{k} \| \mathsf{c})$ and the value k is not currently revealed to adversary. Since, the value k_w is picked uniformly at random, the probability with which event $\mathbb{E}_1$ occurs is negligible. Using (11) and (12), we can conclude:

$$|\Pr[\mathsf{G}_2 = 1] - \Pr[\mathsf{G}_3 = 1]| \leq \mathsf{neg}(\lambda). \tag{13}$$

Setup()

1. $\mathbf{W}, \mathbf{T}, \mathbf{D} \leftarrow$ empty map
2. $u = 0$
3. $\mathbf{Update}, \mathsf{STs} \leftarrow$ empty table
4. $\mathsf{STs}[w][0] \leftarrow -1, \quad \forall w \in \mathsf{W}$
5. return ($\mathsf{EDB} = (\mathbf{D}, \mathbf{T})$, $\mathsf{st_c} = \mathbf{W}$) to (S, C)

Update$(\mathsf{op}, w, \mathsf{ind}, \mathsf{st_C}; \mathsf{EDB})$

Update$_\mathsf{C}(\mathsf{op}, w, \mathsf{ind}, \mathsf{st_C})$

1. Append $(\mathsf{u}, \mathsf{op}, \mathsf{ind})$ to $\mathbf{Update}[w]$
2. $\mathsf{Hash}_1[\mathsf{u}] \xleftarrow{\$} \{0,1\}^{2\lambda}$
3. $\mathsf{Hash}_2[\mathsf{u}] \xleftarrow{\$} \{0,1\}^{\mu}$
4. Send $(\mathsf{Hash}_1[\mathsf{u}], \mathsf{Hash}_2[\mathsf{u}])$ to S
5. $\mathsf{u} \leftarrow \mathsf{u} + 1$

Search$(w, \mathsf{st_C}; \mathsf{EDB})$

Round 1

Search$_\mathsf{C}(w, \mathsf{st_C})$:

1. Append u to $\mathsf{STs}[w]$
2. $(\mathsf{empty}, \mathsf{ver}_w, \mathsf{c}_w, \{\mathsf{tag_c}, \mathbf{H}_{1,w,\mathsf{c}}, \mathbf{H}_{2,w,\mathsf{c}}\}_{0\le \mathsf{c} \le \mathsf{c}_w}) \leftarrow \mathsf{GetData}_{r1}($ $\mathbf{Update}, \mathsf{STs}, \mathsf{Hash}_1, \mathsf{Hash}_2, \mathsf{Tag})$
3. **if** empty = true **then**
4. return $\emptyset$
5. **else**
6. $\mathsf{label}_w \leftarrow \mathsf{Key}_d[w]$
7. **if** $\mathsf{c}_w \neq -1$ **then**
8. $\mathsf{k}_w \leftarrow \mathsf{Key}_t[w, \mathsf{ver}_w]$
9. **for** $i = 0$ to c_w **do**
10. $\mathsf{H}_1[\mathsf{k}_w, i] \leftarrow \mathbf{H}_{1,w,i}$
11. $\mathsf{H}_2[\mathsf{k}_w, i] \leftarrow \mathbf{H}_{2,w,i} \oplus \mathsf{tag}_i$
12. **else**
13. $\mathsf{k}_w \leftarrow \perp$
14. Send $(\mathsf{label}_w, \mathsf{k}_w, \mathsf{c}_w)$ to S

Round 2

Search$_\mathsf{C}(w, \mathsf{st_C})$:

15. Receive TS from S
16. $\mathsf{PSet}_w, \mathsf{AuxSet}_w \leftarrow \mathsf{GetData}_{r2}(\mathsf{Tag}, \mathsf{TS})$
17. return $\mathsf{PSet}_w, \mathsf{AuxSet}_w$
18. $\mathsf{u} \leftarrow \mathsf{u} + 1$

GetData$_{r2}$(Tag, TS)

1. $\mathsf{PSet} \leftarrow \emptyset$
2. $\mathsf{Tag}_{\mathsf{temp}} \leftarrow$ empty map
3. **for** $\forall$ tag $\in$ TS **do**
4. $(w, \mathsf{ind}) \leftarrow \mathsf{Tag}[w]^{-1}[\mathsf{tag}]$
5. $\mathsf{PSet} \leftarrow \mathsf{PSet} \cup \{\mathsf{Tag}_{\mathsf{temp}}[\mathsf{ind}]\}$
6. $\mathsf{AuxSet} \leftarrow \mathsf{AuxSet} \cup \{\mathsf{ind}\}$
7. $\mathsf{Tag}[w] \leftarrow \mathsf{Tag}_{\mathsf{temp}}$
8. return PSet, AuxSet

GetData$_{r1}$(**Update**, STs, Hash$_1$, Hash$_2$, Tag)

1. **if** $\mathbf{Update}[w] = \perp$
2. empty $\leftarrow$ true
3. **else**
4. empty $\leftarrow$ false
5. $\mathsf{ver}_w \leftarrow -1$
6. **for** $i = 1$ to $|\mathsf{STs}[w]| - 1$ **do**
7. **if** $\exists$ u, op, ind s.t. $(\mathsf{u}, \mathsf{op}, \mathsf{ind}) \in \mathbf{Update}[w] \wedge \mathsf{u} > \mathsf{STs}[w][i-1] \wedge \mathsf{u} < \mathsf{STs}[w][i]$ **then**
8. $\mathsf{ver}_w \leftarrow \mathsf{ver}_w + 1$
9. $\mathsf{c}_w \leftarrow -1$, $ul \leftarrow |\mathsf{STs}[w]| - 1$
10. **for** $\mathsf{u} = \mathsf{STs}[w][ul-1]$ to $\mathsf{STs}[w][ul]$ **do**
11. **if** $\exists$ op, ind s.t. $(\mathsf{u}, \mathsf{op}, \mathsf{ind}) \in \mathbf{Update}[w]$ **then**
12. $\mathsf{c}_w \leftarrow \mathsf{c}_w + 1$
13. $\mathsf{b} \leftarrow 0$ (op=add)/ 1 (op=del)
14. $\mathsf{tag}_{\mathsf{c}_w} \leftarrow \mathsf{b} \| \mathsf{Tag}[w, \mathsf{ind}]$
15. $\mathbf{H}_{1,w,\mathsf{c}_w} \leftarrow \mathsf{Hash}_1[\mathsf{u}]$
16. $\mathbf{H}_{2,w,\mathsf{c}_w} \leftarrow \mathsf{Hash}_2[\mathsf{u}]$
17. return $(\mathsf{empty}, \mathsf{ver}_w, \mathsf{c}_w, \{\mathsf{tag_c}, \mathbf{H}_{1,w,\mathsf{c}}, \mathbf{H}_{2,w,\mathsf{c}}\}_{0\le \mathsf{c} \le \mathsf{c}_w})$

Figure 12: Game G_5 (Theorem 4.1)

Game G_4: In G_4, what we did for H_1 in game G_3, we do for H_2. Using the same arguments, we can conclude:

$$|\Pr[\mathsf{G}_3 = 1] - \Pr[\mathsf{G}_4 = 1]| \le \mathsf{neg}(\lambda). \tag{14}$$

Game G_5: In G_5 (see Figure 12), we abstract out the information that needs to be simulated by the simulator in order to output transcripts identical to G_4. Using $\mathsf{GetData}_{r1}$ and $\mathsf{GetData}_{r2}$ algorithms in G_5, one keeps track of the randomly generated tag, label and pad differently than in G_4. In Search protocol, the random oracles are programmed identically to that in G_4. Queries to random oracles H_1 and H_2 can be simulated by outputting random values.

As we output fresh random strings in Update protocol, the transcripts of Update protocol is identical to that of Update protocol in G_4.

Next, we describe the Search protocol in G_5. Based on tables **Update**, STs (search timestamps) and Tag map, the value of the following components: empty_w, ver_w, c_w and $\{\mathsf{tag_c}, \mathbf{H}_{1,w,\mathsf{c}}, \mathbf{H}_{2,w,\mathsf{c}}\}_{0\le \mathsf{c} \le \mathsf{c}_w}$ are determined using $\mathsf{GetData}_{r1}$ algorithm. Flag empty is set to 1 if $\mathbf{Update}[w]$ is empty. ver_w is the count of searches for which there was an update on map $\mathbf{T}$ corresponding to keyword w after the previous search. The loop in line 6 of $\mathsf{GetData}_{r1}$ algorithm determines the number of times version number is updated, i.e., value of ver_w. Here, c_w denotes the count of updates on map $\mathbf{T}$ corresponding to keyword w after the previous search. tag is picked from the map Tag, therefore, if the indices are same, the tags are equal, thus ensuring the consistency. The values $\{\mathbf{H}_{1,w,\mathsf{c}}, \mathbf{H}_{2,w,\mathsf{c}}\}_{0\le \mathsf{c} \le \mathsf{c}_w}$ are used to simulate the random oracles consistently with the response given at the time of update queries. The loop in line 10 of $\mathsf{GetData}_{r1}$ algorithm

computes the values of c_w and $\{\mathsf{tag_c}, \mathbf{H}_{1,w,\mathsf{c}}, \mathbf{H}_{2,w,\mathsf{c}}\}_{0\leq \mathsf{c}\leq \mathsf{c}_w}$. $\mathsf{GetData}_{r2}$ algorithm outputs $\mathsf{LDB}(w)$ and new tags corresponding to the document identifiers currently present in $\mathsf{DB}(w)$. These tags are ordered based on the order of insertion of documents they correspond to. As the values of components are computed correctly and consistently w.r.t. previous queries, the transcripts of Search protocol is identical to that of Search protocol in G_4. Therefore, we conclude that:

$$\Pr[\mathsf{G}_5 = 1] = \Pr[\mathsf{G}_4 = 1]. \tag{15}$$

Simulator Sim: Finally, we construct a simulator that given the leakage profile $\mathcal{L}$ simulates game G_5 correctly. Sim can simulate Update protocol correctly as in G_5. Instead of using keyword w, Sim uses the counter $\overline{w}$=min $\mathsf{sp}(w)$ uniquely mapped from w using $\mathcal{L}_{\mathsf{Search}}$ in simulating the Search protocol (line 6 and line 8). In line 2 of Search protocol, Sim uses $\mathsf{sp}(w)$, $\mathsf{Updates}^{\mathsf{op}}(w)$, $\mathsf{PreUp}(w)$ and $\mathsf{UpPair}(w)$ instead of STs, **Update** and Tag as input to the $\mathsf{GetData}_{r1}$ algorithm. In $\mathsf{GetData}_{r1}$, we use the timestamps of search and update queries and make use of indices to generate tag. However, the indices are used just to identify when same tags need to be generated. This can be ensured using, $\mathsf{PreUp}(w)$ and $\mathsf{UpPair}(w)$.

Also, the output of $\mathsf{GetData}_{r2}$ is $\mathsf{LDB}(w)$ and an ordered list of freshly generated random tags which can be simulated easily using $\mathsf{CurUp}(w)$ and $\mathsf{LDB}(w)$. In $\mathsf{GetData}_{r2}$, the indices are used to just associate an order to the generated tags, which can be ensured by the components $\mathsf{CurUp}(w)$ and $\mathsf{LDB}(w)$ of the leakage profile. Thus, Sim is able to produce transcripts of output of Search and Update protocols identical to G_5. Hence, we conclude that:

$$\Pr[\mathbf{Ideal}^{\Pi_{\mathsf{BP}}}_{\mathcal{A},\mathsf{Sim}}(\lambda) = 1] = \Pr[\mathsf{G}_5]. \tag{16}$$

By connecting all the games, we conclude

$$\left|\Pr[\mathbf{Real}^{\Pi_{\mathsf{BP}}}_{\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{Ideal}^{\Pi_{\mathsf{BP}}}_{\mathcal{A},\mathsf{Sim}}(\lambda) = 1]\right| \leq \mathsf{neg}(\lambda). \tag{17}$$

□

4.4 Π_{WBP} : A Weak Backward Private Variant

As mentioned in Section 3.3, there are various use-cases such as storing a collection of text files in which reinsertion restriction is not a concern. Here, we propose a simple one roundtrip response-hiding backward private scheme Π_{WBP} in the *reinsertion restriction* setting. Our construction Π_{WBP} is essentially a simple modification to Π_{BP}. Further, it also inherits all the salient features of Π_{BP}. Π_{WBP} improves upon the concrete efficiency of Search protocol in Π_{BP}, by eliminating the computation and transmission of newly generated tags in the second round of communication. This shows that one can construct an efficient WBP scheme with optimal communication complexity using simple primitives only.

In Π_{BP}, we use ver_w in computation of tags to securely handle reinsertions. The tags corresponding to fresh updates after a search are computed using the updated ver_w. Hence, these tags cannot be related to the deleted entries before this search. This ensures backward privacy in cases where reinsertion is allowed. However, when we consider the reinsertion restriction setting, an add query is not allowed after a delete query corresponding to the same document-keyword pair. Hence, ver_w is not required in computation of tags in this setting. Therefore, for an update query, tag in line 8 of Update protocol in Π_{BP} (see Figure 10) is computed as $\mathsf{tag} \leftarrow \mathsf{G}_{\mathsf{tag}}(\mathsf{k}_g, w\|\mathsf{ind})$ in Π_{WBP}.

Search protocol in Π_{WBP} is identical to Round 1 of Search protocol in Π_{BP}. In line 31 of Search protocol in Π_{BP}, S along with sending TS to C, stores TS at $\mathbf{D}[\mathsf{label}_w]$. From TS, C retrieves the search results in similar fashion as in Π_{BP}. Recomputation of tags in Search protocol in Π_{BP} is needed because ver_w gets updated. While in Π_{WBP}, since the tags are independent of ver_w, recomputation of tags is not required. Hence, the second round of communication is not needed.

The changes we make in Π_{BP} in constructing Π_{WBP} induces additional leakage which can be shown to be bounded by the leakage profile of WBP. The proof of Theorem 4.1 can be easily adapted to prove security of Π_{WBP}.

The crucial observation from our constructions is that efficient backward private schemes can be realized without involving complex cryptographic primitives. The simplicity of design in our backward private constructions is an appealing feature, particularly from the implementation perspective.

4.5 Comparative Analysis

In Table 1, we provided a comparison of our schemes with some prior and concurrent works [7, 20, 40]. On that line, we conclude this section with a comparative study of the currently available BP-secure DSSE schemes. The goal is to figure out the scenarios in which each of these constructions would fit best. Let us first consider the scenario where the requirement is to achieve minimal information leakage. The candidate constructions are Orion [20] and Moneta [7] as they satisfy strong notion of backward privacy (BPIP). As the search time of Orion is quasi-optimal in n_w (linear in n_w upto a logarithmic factor), it may appear to be more suitable than Moneta in such scenarios. However, Orion may not be practical for very large databases as Path-ORAM [39] is used as a building block in its construction, which limits the applicability of Orion in such scenarios [32]. Constructions Mitra [20], Fides [7] and Π_{BP}-prime (Section 4.2) satisfy the next level of backward privacy. Mitra and Π_{BP}-prime are essentially the same, make use of symmetric primitives only and as a result are very efficient in practice. But, these constructions still suffer from significant communication overhead.

In order to overcome the above limitations, one can look to trade security a bit for performance, while at the same time ensure that the notion of forward and backward privacy is preserved. Constructions $\mathsf{Diana}_{\mathsf{del}}$ [7], Janus [7], Janus++ [40], Horus [20] and Π_{WBP} (Section 4.2) satisfy the notion of weak backward privacy (WBP). The communication and computation complexity of search and update protocols of $\mathsf{Diana}_{\mathsf{del}}$ isn't optimal (See Table 1). To improve upon the communication overhead of Search protocol, a single round-trip Janus framework [7] was proposed. It was instantiated using asymmetric and symmetric puncturable encryption scheme in Janus [7] and Janus++ [40] respectively. However, the computational complexity of search protocol in Janus framework is $O(n_w \cdot d_w)$, which is unreasonably high ($n_w \cdot d_w >> o'_w$). Horus, a modified version of Orion, was proposed in order to improve the number of round trips in the Search protocol ($O(log(N))$ to $O(log(d_w))$). But Horus suffers from the same scalability issue as Orion. Π_{WBP} is a single-round trip DSSE scheme that achieves optimal communication complexity, makes use of symmetric primitives only and is very efficient in practice. However, the computation complexity of the Search protocol is not quasi-optimal in n_w. Moreover, the notion of weak backward privacy can only be used in scenarios where reinsertion of keyword-document is not allowed.

BP-II (Definition 3.6) along with satisfying the intuitive notion of backward privacy, allows the reinsertion of keyword-document pairs. The corresponding construction, Π_{BP} (Section 4.3) is the first DSSE scheme that satisfies the notion of backward privacy in the general setting that achieves optimal communication complexity, makes use of symmetric primitives only and is very efficient in practice. The only limitation is that the search time in Π_{BP} is not quasi-optimal in n_w which seems to be the cost that one has to pay to achieve optimal communication complexity in Update and Search protocol.

5 Implementation Results

In this section, we discuss the implementation results of schemes Π_{FP} and Π_{BP}. Π_{FP} being currently the most efficient forward private scheme in literature, serves as a benchmark to evaluate the performance of other DSSE schemes. The implementation results for Π_{BP} gives a fair indication about the performance of Π_{BP}-prime and Π_{WBP} as their asymptotic complexity are same and they also make use of light-weight symmetric primitives. Further, the concrete computation cost of Π_{WBP} reduces further as the tags need not be recomputed. We have implemented the schemes in C++. For pseudo random functions F_d, F_t and pseudo random permutation $\mathsf{G}_{\mathsf{tag}}$, we use AES, and for hash functions H_1 and H_2, we use SHA-256. We use the AES and SHA-256 function available in OpenSSL library [41] in our code. Maps $\mathbf{T}$ and $\mathbf{W}$ are stored using RocksDB [17].

All our experiments were performed on a desktop computer with an Intel Core i5 4460 3.20GHz CPU and

Implementation	Time (s)	Time per pair * (μs)	Strg. at S (GB)	Strg. at C (MB)
Π_{FP} (in main memory)	164.66	4.49	2.8	2.4
Π_{FP} (in ext. disk)	172.66	4.71	2.8	2.4
Π_{BP} (in main memory)	182.90	4.98	2.8	2.4
Π_{BP} (in ext. disk)	193.74	5.28	2.8	2.4

$*$ - document-keyword pair, Strg. = Storage

Table 2: EDB Creation

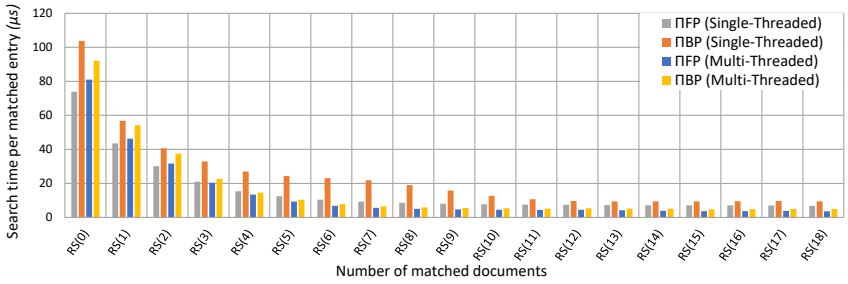


Figure 13: Average per entry search time (Main memory)

8GB RAM running Ubuntu 16.04 LTS. Our code is designed to run as a single program as we are interested in determining the performance of Search and Update protocols of our constructions.

We used Enron email dataset [1] to create EDB on which we perform our search and update operations. We wrote a python code to extract keywords from the mails in Enron email dataset using NLTK library [34]. The number of documents, number of keywords and number of document-keyword pairs in our dataset are 517,401, 212,020 and 36,688,028 respectively.

EDB Creation EDB was created to store all the document-keyword pairs extracted from the Enron email dataset. The computational works and I/O latency required for EDB creation are parallelized using thread pool. Table 2 shows the time taken to create an EDB, the time taken to process each document-keyword pair and the size of EDB and **W** just after EDB creation. For in memory results, only map **T** was mounted on to main memory.

For each entry in Table 2, we ran our experiment 10 times and computed the average. The time taken to create EDB in main memory and external disk is similar for both the schemes. The time taken to create EDB for Π_{BP} is reasonably close to that of Π_{FP}. The per document-keyword processing time is very less for both the schemes as only symmetric primitives are used in our constructions.

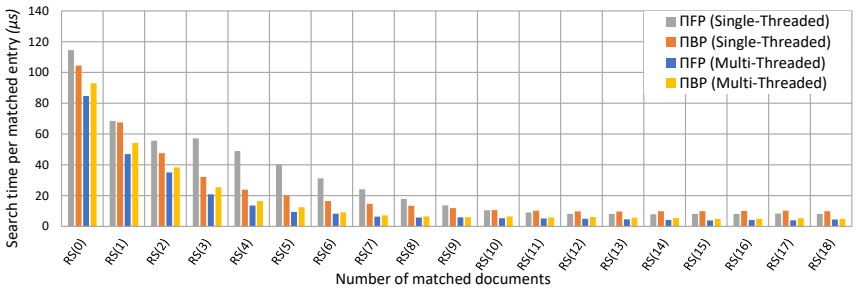


Figure 14: Average per entry search time (External disk)

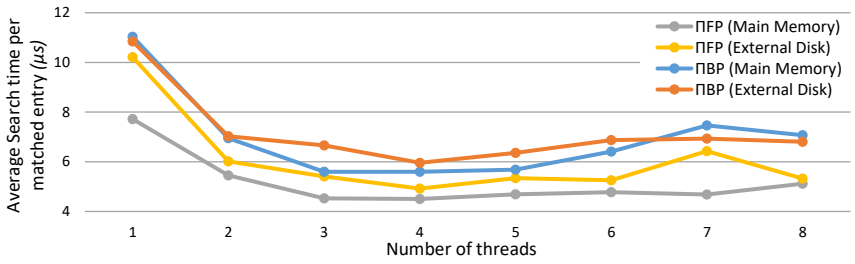


Figure 15: Average per entry search time (# Threads)

EDB Search To evaluate the search performance, we searched all the keywords extracted from the Enron email dataset just after EDB creation and measured the overall time taken for the Search protocol. The performance of both the schemes when $\mathbf{T}$ is mounted on to main memory and when $\mathbf{T}$ is stored on external disk was evaluated. Figure 13 describes the search time per matched entry (stpme) based on the number of documents returned in the search results for both the schemes when $\mathbf{T}$ is mounted on to main memory. It illustrates the performance in single threaded and multi-threaded environment. Figure 14 describes the same when $\mathbf{T}$ is stored on the external hard disk. In Figures 13 and 14,

$$\mathrm{RS}(i) = \begin{cases} \mathsf{DB}(w) = 1 & \text{if } i = 0 \\ 2^{i-1} < \mathsf{DB}(w) \leq 2^i & \text{if } 1 \leq i \leq 18. \end{cases}$$

Figure 15 illustrates that the stpme for both the schemes is affected by the number of threads used to perform the search operation. For searches matching less number of documents, the cost is high because of one time computations such as storage access in computation of token at C's end, creation of threads, access to map $\mathbf{D}$, etc., which is amortized for searches matching large number of documents. The performance of both the schemes improve significantly in multi-threaded environment, thus, highlighting the potential for parallelism in practice. The schemes performed the best when the number of threads were around the number of cores in the processor, i.e., 4. The performance of Π_{BP} is comparable to the performance of Π_{FP}. The stpme in Π_{BP} was 4.50 (resp. 4.89) μs for queries whose result set size was in the interval $(2^{17}, 2^{18}]$ and the average stpme was 5.59 (resp. 5.95) μs for all queries when $\mathbf{T}$ was mounted on to main memory (resp. stored on external disk).

EDB Dynamic Environment To show the performance of search queries in dynamic environment, i.e., where search queries are interspersed with update queries, we identified the set of keywords matching more than 80k documents from the extracted document-keyword pairs, say $\mathsf{S}_{80\mathrm{k}}$. An initial EDB was constructed using extracted document-keyword pairs apart from those corresponding to keywords in $\mathsf{S}_{80\mathrm{k}}$. The document-keyword pairs corresponding to keywords in $\mathsf{S}_{80\mathrm{k}}$ are then utilized to perform update queries dynamically. Figure 16 describes the stpme with regard to the probability (p) of search queries on keywords in $\mathsf{S}_{80\mathrm{k}}$. This implies that the update queries occur with probability 1-p, of which, with probability 0.1 it is a del query. The performance evaluation includes the time required to process del operation in search queries. For both the schemes, Figure 16 illustrates that the average stpme decreases in scenarios where search queries are frequent, as both the schemes exploit the locality introduced by $\mathbf{D}$. The average stpme in Π_{BP} turns out to be 1.53 μs, when the probability of search query is 0.0005.

The prototype implementations of Π_{FP} and Π_{BP} indicate that the cost of achieving backward privacy over and above forward privacy is substantially small. This makes Π_{BP} a suitable candidate for practical deployment.

Communication Cost As all our constructions make use of symmetric primitives only, the communication cost becomes the main performance bottleneck. We now compare the communication cost of Π_{BP}-prime, Π_{BP} and Π_{WBP} as a function of the nature of update queries. Figure 17 depicts the communication cost of the trio based on the number of documents returned in the search results and the probability with which an

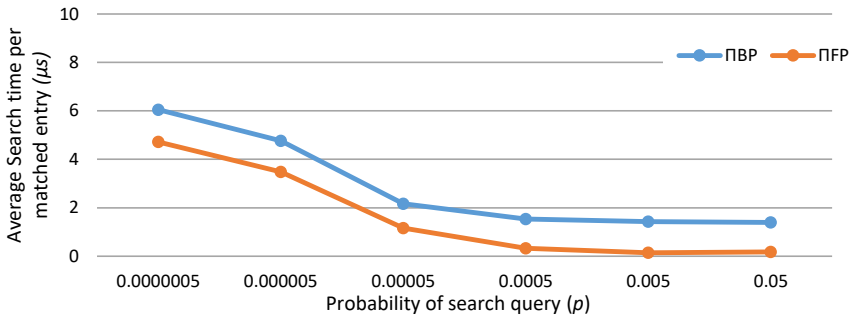


Figure 16: Average per entry search time (Dynamic Environment)

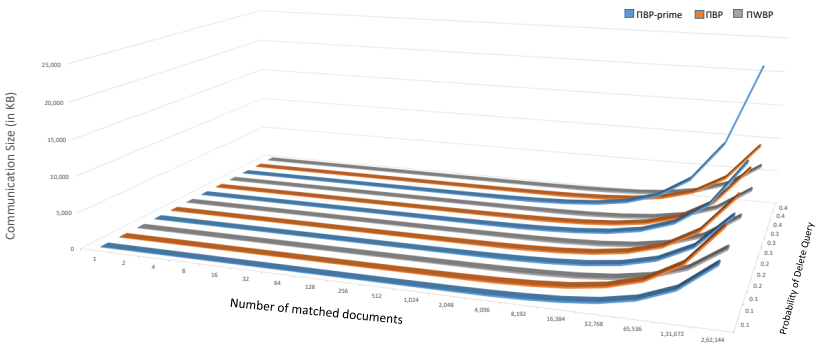


Figure 17: Communication Cost: Number of matching documents vs. Probability of delete queries

update query is a del query. As can be observed, for Π_{BP} and Π_{WBP} the communication cost depends only upon the result set size and is independent of the nature of update queries. Therefore, the communication complexity is the same for all types of update query distribution for Π_{BP} and Π_{WBP}. However, that is not the case for Π_{BP}-prime, as can be observed from Figure 17, the communication complexity increases with increase in the percentage of delete queries. But note that if the percentage of delete query is less, then the communication cost of Π_{BP}-prime is less than Π_{BP} in practice. Concretely, Π_{WBP} has the least communication overhead among all our backward private constructions in practice. As already mentioned in Section 4.2, Π_{BP}-prime avoids re-encryption of entries and thus improves upon the communication cost of Mitra*. Hence, the communication cost of our constructions fair reasonably well against the most efficient construction until this work.

6 Conclusion

The main contribution of this paper is to propose three efficient backward private DSSE schemes, viz., Π_{BP}-prime, Π_{BP} and Π_{WBP}. We start with revisiting the existing definitions of backward privacy and propose alternative formulations of leakage for backward privacy, viz., BP-I and BP-II. The proposed constructions achieve practical efficiency by using light weight symmetric cryptographic components only. In particular, our construction Π_{BP} is the first backward private scheme in the general setting that achieves optimal communication complexity using symmetric cryptographic primitives only. The main takeaway from this work is that efficient backward private schemes can be realized without involving complex cryptographic primitives. The simplicity of their design make our backward private constructions even more appealing, particularly from the implementation perspective. On the definition front, an interesting question arising out of this study is to analyze the real-world consequences of the leakages incurred by constructions satisfying the definitions of backward privacy, viz., BPIP, BPUP, BP-I, BP-II or even WBP (in restricted setting). On

the construction front, an interesting problem to pursue is to design an efficient, single roundtrip, response revealing backward private scheme in the general setting ideally with optimal communication complexity.

References

[1] "Enron Email Dataset," https://www.cs.cmu.edu/ enron/, Accessed: 2018-05-14.

[2] M. A. Abdelraheem, T. Andersson, and C. Gehrmann, "Inference and Record-Injection Attacks on Searchable Encrypted Relational Databases," *IACR Cryptology ePrint Archive*, vol. 2017, p. 24, 2017.

[3] M. Bellare and P. Rogaway, "The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs," in *EUROCRYPT*, ser. LNCS, vol. 4004. Springer, 2006, pp. 409–426.

[4] D. Boneh and B. Waters, "Constrained Pseudorandom Functions and Their Applications," in *ASIACRYPT*, ser. LNCS, vol. 8270. Springer, 2013, pp. 280–300.

[5] R. Bost, "$\Sigma o \varphi o \varsigma$: Forward Secure Searchable Encryption," in *ACM CCS*. ACM Press, 2016, pp. 1143–1154.

[6] ——, "Searchable Encryption: New Constructions of Encrypted Databases," Ph.D. dissertation, 2018.

[7] R. Bost, B. Minaud, and O. Ohrimenko, "Forward and Backward Private Searchable Encryption from Constrained Cryptographic Primitives," in *ACM CCS*. ACM Press, 2017, pp. 1465–1482.

[8] E. Boyle, S. Goldwasser, and I. Ivan, "Functional Signatures and Pseudorandom Functions," in *PKC*, ser. LNCS, vol. 8383. Springer, 2014, pp. 501–519.

[9] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-Abuse Attacks Against Searchable Encryption," in *ACM CCS*. ACM Press, 2015, pp. 668–679.

[10] D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, "Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation," in *NDSS*. The Internet Society, 2014.

[11] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner, "Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries," in *CRYPTO*, ser. LNCS. Springer, 2013, vol. 8042, pp. 353–373.

[12] Y. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *ACNS*, ser. Lecture Notes in Computer Science, vol. 3531, 2005, pp. 442–455.

[13] M. Chase and S. Kamara, "Structured Encryption and Controlled Disclosure," in *ASIACRYPT*, ser. LNCS, vol. 6477. Springer, 2010, pp. 577–594.

[14] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, "Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions," in *ACM CCS*. ACM Press, 2006, pp. 79–88.

[15] M. Etemad, A. Küpçü, C. Papamanthou, and D. Evans, "Efficient Dynamic Searchable Encryption with Forward Privacy," *PoPETs*, vol. 2018, no. 1, pp. 5–20, 2018.

[16] S. Faber, S. Jarecki, H. Krawczyk, Q. Nguyen, M. Rosu, and M. Steiner, "Rich Queries on Encrypted Data: Beyond Exact Matches," in *ESORICS*, ser. LNCS, vol. 9327. Springer, 2015, pp. 123–145.

[17] Facebook, "RocksDB: A persistent key-value store for fast storage environment," https://rocksdb.org/, Accessed: 2018-05-14.

[18] S. Garg, P. Mohassel, and C. Papamanthou, "TWORAM: Efficient Oblivious RAM in Two Rounds with Applications to Searchable Encryption," in *CRYPTO*, ser. LNCS, vol. 9816. Springer, 2016, pp. 563–592.

[19] C. Gentry, "A Fully Homomorphic Encryption Scheme," Ph.D. dissertation, Stanford, CA, USA, 2009, aAI3382729.

[20] J. Ghareh Chamani, D. Papadopoulos, C. Papamanthou, and R. Jalili, "New constructions for forward and backward private symmetric searchable encryption," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. ACM, pp. 1038–1055.

[21] O. Goldreich, S. Goldwasser, and S. Micali, "How to Construct Random Functions (Extended Abstract)," in *FOCS*. IEEE Computer Society Press, 1984, pp. 464–479.

[22] O. Goldreich and R. Ostrovsky, "Software Protection and Simulation on Oblivious RAMs," *J. ACM*, no. 3, pp. 431–473, 1996.

[23] S. Goldwasser, S. Micali, and R. L. Rivest, "A "paradoxical" solution to the signature problem (extended abstract)," in *FOCS*. IEEE Computer Society, 1984, pp. 441–448.

[24] ——, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM J. Comput.*, vol. 17, no. 2, pp. 281–308, 1988.

[25] M. D. Green and I. Miers, "Forward Secure Asynchronous Messaging from Puncturable Encryption," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 2015, pp. 305–320.

[26] S. Kamara and T. Moataz, "Boolean Searchable Symmetric Encryption with Worst-Case Sub-linear Complexity," in *EUROCRYPT*, ser. LNCS, vol. 10212. Springer, 2017, pp. 94–124.

[27] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic Searchable Symmetric Encryption," in *ACM CCS*. ACM Press, 2012, pp. 965–976.

[28] A. Kiayias, S. Papadopoulos, N. Triandopoulos, and T. Zacharias, "Delegatable Pseudorandom Functions and Applications," in *ACM CCS*. ACM Press, 2013, pp. 669–684.

[29] K. S. Kim, M. Kim, D. Lee, J. H. Park, and W. Kim, "Forward Secure Dynamic Searchable Symmetric Encryption with Efficient Updates," in *ACM CCS*. ACM Press, 2017, pp. 1449–1463.

[30] N. Koblitz and A. Menezes, "Another look at security definitions," *Adv. in Math. of Comm.*, vol. 7, no. 1, pp. 1–38, 2013.

[31] A. Menezes and N. P. Smart, "Security of signature schemes in a multi-user setting," *Des. Codes Cryptography*, vol. 33, no. 3, pp. 261–274, 2004.

[32] M. Naveed, "The fallacy of composition of oblivious RAM and searchable encryption," *IACR Cryptology ePrint Archive*, vol. 2015, p. 668, 2015.

[33] M. Naveed, M. Prabhakaran, and C. A. Gunter, "Dynamic Searchable Encryption via Blind Storage," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 2014, pp. 639–654.

[34] NLTK Project, "Natural Language Toolkit," https://www.nltk.org/, Accessed: 2018-05-14.

[35] T. Pornin and J. P. Stern, "Digital signatures do not guarantee exclusive ownership," in *ACNS*, ser. LNCS, vol. 3531, 2005, pp. 138–150.

[36] D. X. Song, D. Wagner, and A. Perrig, "Practical Techniques for Searches on Encrypted Data," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 2000, pp. 44–55.

[37] X. Song, C. Dong, D. Yuan, Q. Xu, and M. Zhao, "Forward Private Searchable Symmetric Encryption with Optimized I/O Efficiency," *IEEE Transactions on Dependable and Secure Computing*, 2018.

[38] E. Stefanov, C. Papamanthou, and E. Shi, "Practical Dynamic Searchable Encryption with Small Leakage," in *NDSS*. The Internet Society, 2014.

[39] E. Stefanov, M. van Dijk, E. Shi, C. W. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path ORAM: An Extremely Simple Oblivious RAM Protocol," in *ACM CCS*. ACM Press, 2013, pp. 299–310.

[40] S.-F. Sun, X. Yuan, J. K. Liu, R. Steinfeld, A. Sakzad, V. Vo, and S. Nepal, "Practical backward-secure searchable encryption from symmetric puncturable encryption," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. ACM, 2018, pp. 763–780.

[41] The OpenSSL Project, "OpenSSL Cryptography and SSL/TLS Toolkit," https://www.openssl.org/, Accessed: 2018-05-14.

[42] S. Vaudenay, "Digital signature schemes with domain parameters: Yet another parameter issue in ECDSA," in *ACISP*, ser. LNCS, vol. 3108. Springer, 2004, pp. 188–199.

[43] Y. Zhang, J. Katz, and C. Papamanthou, "All Your Queries Are Belong to Us: The Power of File-Injection Attacks on Searchable Encryption," in *USENIX Security Symposium*. USENIX Association, 2016, pp. 707–720.

A Correctness and Security of Π_{FP}

A.1 Proof of Correctness

Theorem A.1. *If* F_d, F_t *are secure* PRF*s and* H_1 *is a collision-resistant hash function then* Π_{FP} *is correct.*

Proof. We use the games G_0 and G_1. The games are identical to Π_{FP} (see Figure 6) except for the following changes:

1. In the Setup algorithm, we initialize the sets LabSet_1, LabSet_2 to null set and set boolean variable bad to false.
2. We remove line 5 and 6 of Update protocol and after line 4 in the Update protocol, we add the following code:

   G_0, $\boxed{\mathsf{G}_1}$

   5. $\boxed{\mathsf{k}_w \leftarrow \mathsf{Key}_t[w, \mathsf{ver}_w]}$
   6. $\mathsf{label} \leftarrow \mathsf{H}_1(\mathsf{k}_w \| \mathsf{c}_w)$
   7. **if** $\mathsf{label} \in \mathsf{LabSet}_1$ **then**
   8. $\quad \mathsf{bad} \leftarrow \mathsf{true}, \mathsf{label} \xleftarrow{\$} \{0,1\}^\lambda \setminus \mathsf{LabSet}_1$
   9. $\mathsf{LabSet}_1 \leftarrow \mathsf{LabSet}_1 \cup \{\mathsf{label}\}$
3. We replace line 5 in the Search protocol with the following code:

   G_0, $\boxed{\mathsf{G}_1}$

   5. $\boxed{\mathsf{label}_w \leftarrow \mathsf{Key}_d[w]}$
   6. **if** $\exists\, w'$ s.t $(w', \mathsf{label}_w) \in \mathsf{LabSet}_2 \wedge w' \neq w$ **then**
   7. $\quad \mathsf{bad} \leftarrow \mathsf{true}, \mathsf{label}_w \xleftarrow{\$} \{0,1\}^\lambda \setminus \mathsf{LabSet}_2$
   8. $\mathsf{LabSet}_2 \leftarrow \mathsf{LabSet}_2 \cup \{(w, \mathsf{label}_w)\}$
4. We replace line 7 in the Search protocol with the following code:

   G_0, $\boxed{\mathsf{G}_1}$

   7. $\boxed{\mathsf{k}_w \leftarrow \mathsf{Key}_t[w, \mathsf{ver}_w]}$

In the modification snippets, G_1 includes the box code and G_0 does not. The first game G_0 will output 1, only if bad is set, as repeated labels in maps $\mathbf{T}$ and $\mathbf{D}$ are the only source of incorrectness. G_0 produces an

identical distribution to real game when bad is not set. If the value assigned to label is repeated, G_0 replaces it with new value which hasn't been assigned to any label up till now.

Let us denote the event 'the flag bad is set to true' in G_0 by $\mathbb{E}_0$. This gives,

$$\Pr[\mathbf{DSSECor}^{\Pi_{\mathsf{FP}}}_{\mathcal{A}}(\lambda) = 1] \leq \Pr[\mathbb{E}_0]. \tag{18}$$

In G_1, every call to PRFs F_t and F_d are answered using tables Key_t and Key_d respectively. The entries in table Key_t are referred by (w, ver) and entries in table Key_d are referred by w.

Let us denote the event 'the flag bad is set to true' in G_1 by $\mathbb{E}_1$.

There exists an adversary $\mathcal{B}_1$ that can distinguish F_t from a truly random function and/or an adversary $\mathcal{B}_2$ that can distinguish F_d from a truly random function such that:

$$\begin{aligned} |\Pr[\mathbb{E}_1] - \Pr[\mathbb{E}_0]| &\leq \mathbf{Adv}^{\mathsf{prf}}_{\mathsf{F}_t,\mathcal{B}_1}(\lambda) + \mathbf{Adv}^{\mathsf{prf}}_{\mathsf{F}_d,\mathcal{B}_2}(\lambda) \\ &\leq \mathsf{neg}(\lambda). \end{aligned} \tag{19}$$

The event $\mathbb{E}_1$ occurs only when the newly picked label value was already picked earlier in G_1. $\mathbb{E}_1$ occurs in line 7 of modification 3, if the same label is generated for more than one keyword. Since, the labels are picked uniformly at random in line 5 and the number of keywords, m, is a polynomial in security parameter, by the birthday bound we can conclude that the probability that the two labels for map $\mathbf{D}$ are equal is at most $\frac{m^2}{2^\lambda}$, i.e., $\mathsf{neg}(\lambda)$.

Further, the key k_w is picked uniformly at random (see line 5 of modification 2) and the number of updates on $\mathbf{T}$, say q, is a polynomial in security parameter. By the birthday bound we can conclude that the probability that the two keys are equal is at most $\frac{q^2}{2^\lambda}$, i.e., $\mathsf{neg}(\lambda)$. Therefore, only with negligible probability the input to H_1 is repeated. So, if $\mathbb{E}_1$ occurs in line 8 of modification 2, one can find collision in the hash function H_1. Since, H_1 is collision resistant this happens with negligible probability. Therefore,

$$\Pr[\mathbb{E}_1] \leq \mathsf{neg}(\lambda). \tag{20}$$

From (18), (19) and (20), we get

$$\Pr[\mathbf{DSSECor}^{\Pi_{\mathsf{FP}}}_{\mathcal{A}}(\lambda) = 1] \leq \mathsf{neg}(\lambda). \tag{21}$$

□

A.2 Security Proof

Theorem A.2. *If $\mathsf{F}_d, \mathsf{F}_t$ are secure* PRF*s and $\mathsf{H}_1, \mathsf{H}_2$ are hash functions modeled as random oracles outputting 2λ and μ bits respectively then Π_{FP} is* FP-I *secure (Definition 3.1).*

Proof. We structure our proof using a sequence of games G_0 to G_4. G_0 will compute a distribution identical to $\mathbf{Real}^{\Pi_{\mathsf{FP}}}_{\mathcal{A}}(\lambda)$ and G_4 will compute a distribution that can be simulated perfectly given $\mathcal{L}$, i.e., its distribution is identical to $\mathbf{Ideal}^{\Pi_{\mathsf{FP}}}_{\mathcal{A},\mathsf{Sim}}(\lambda)$ and the intermediate games are hybrids.

Game G_0: G_0 is exactly identical to $\mathbf{Real}^{\Pi_{\mathsf{FP}}}_{\mathcal{A}}(\lambda)$.

$$\Pr[\mathbf{Real}^{\Pi_{\mathsf{FP}}}_{\mathcal{A}}(\lambda) = 1] = \Pr[\mathsf{G}_0 = 1]. \tag{22}$$

Game G_1: In G_1, every call to PRFs F_t and F_d are answered using tables Key_t and Key_d respectively. The entries in table Key_t are referred by (w, ver) and entries in table Key_d are referred by w. If there exists an adversary $\mathcal{A}$ that is able to distinguish between games G_0 and G_1, we can construct an adversary $\mathcal{B}_1$ that

Setup()

1. $\mathbf{W}, \mathbf{T}, \mathbf{D} \leftarrow$ empty map
2. $\mathsf{bad} \leftarrow \mathsf{false}$
3. return ($\mathsf{EDB} = (\mathbf{D}, \mathbf{T})$, $\mathsf{st_C} = (\mathsf{k}_t, \mathsf{k}_d, \mathbf{W})$) to (S, C)

Update$(\mathsf{op}, w, \mathsf{ind}, \mathsf{st_C}; \mathsf{EDB})$

Update$_\mathsf{C}(\mathsf{op}, w, \mathsf{ind}, \mathsf{st_C})$

1. $(\mathsf{ver}_w, \mathsf{c}_w) \leftarrow \mathbf{W}[w]$
2. **if** $(\mathsf{ver}_w, \mathsf{c}_w) = \perp$ **then**
3. $\quad \mathsf{ver}_w \leftarrow 0,\ \mathsf{c}_w \leftarrow -1$
4. $\mathsf{c}_w \leftarrow \mathsf{c}_w + 1,\ \mathsf{k}_w \leftarrow \mathsf{Key}_t[w, \mathsf{ver}_w]$
5. $\mathsf{val}_r \xleftarrow{\$} \{0,1\}^\lambda$
6. **if** $\mathbf{H}_1[\mathsf{k}_w, \mathsf{c}_w] \neq \perp$ **then**
7. $\quad \mathsf{bad} \leftarrow \mathsf{true}, \boxed{\mathsf{val}_r \leftarrow \mathsf{H}_1(\mathsf{k}_w \| \mathsf{c}_w)}$
8. $\mathsf{Hash}_1[w, \mathsf{ver}_w, \mathsf{c}_w] \leftarrow \mathsf{val}_r$
9. $\mathsf{b} \leftarrow 0(\mathsf{op} = \mathsf{add})/1(\mathsf{op} = \mathsf{del})$
10. $\mathsf{pad} \leftarrow \mathsf{H}_2(\mathsf{k}_w \| \mathsf{c}_w)$
11. $\mathsf{e} \leftarrow \mathsf{b} \| \mathsf{ind} \oplus \mathsf{pad}$
12. $\mathbf{W}[w] \leftarrow (\mathsf{ver}_w,\ \mathsf{c}_w)$
13. Send $(\mathsf{val}_r, \mathsf{e})$ to S

Search$(w, \mathsf{st_C}; \mathsf{EDB})$

Search$_\mathsf{C}(w, \mathsf{st_C})$:

1. $(\mathsf{ver}_w, \mathsf{c}_w) \leftarrow \mathbf{W}[w]$
2. **if** $(\mathsf{ver}_w, \mathsf{c}_w) = \perp$ **then**
3. $\quad$ return $\emptyset$
4. **else**
5. $\quad \mathsf{label}_w \leftarrow \mathsf{Key}_d[w]$
6. $\quad$ **if** $\mathsf{c}_w \neq -1$ **then**
7. $\quad\quad \mathsf{k}_w \leftarrow \mathsf{Key}_t[w, \mathsf{ver}_w]$
8. $\quad\quad$ **for** $i = 0$ to c_w **do**
9. $\quad\quad\quad \mathbf{H}_1[\mathsf{k}_w, i] \leftarrow \mathsf{Hash}_1[w, \mathsf{ver}_w, i]$
10. $\quad\quad \mathsf{ver}_w \leftarrow \mathsf{ver}_w + 1, \mathsf{c}_w \leftarrow -1$
11. $\quad\quad \mathbf{W}[w] \leftarrow (\mathsf{ver}_w,\ \mathsf{c}_w)$
12. $\quad$ **else**
13. $\quad\quad \mathsf{k}_w \leftarrow \perp$
14. $\quad$ Send $(\mathsf{label}_w, \mathsf{k}_w, \mathsf{c}_w)$ to S

$\mathsf{H}_1(\mathsf{k} \| \mathsf{c})$

1. $\mathsf{val} \leftarrow \mathbf{H}_1[\mathsf{k}, \mathsf{c}]$
2. **if** $\mathsf{val} = \perp$ **then**
3. $\quad \mathsf{val} \xleftarrow{\$} \{0,1\}^\lambda$
4. $\quad$ **if** $\exists w, \mathsf{ver}$ s.t. $\mathsf{ver} = \mathbf{W}[w].\mathsf{ver}_w \wedge \mathsf{k} = \mathsf{Key}_t[w, \mathsf{ver}] \wedge \mathsf{c} \leq \mathbf{W}[w].\mathsf{c}_w$ **then**
5. $\quad\quad \mathsf{bad} \leftarrow \mathsf{true}, \boxed{\mathsf{val} \leftarrow \mathsf{Hash}_1[w, \mathsf{ver}, \mathsf{c}]}$
6. $\quad \mathbf{H}_1[\mathsf{k}, \mathsf{c}] \leftarrow \mathsf{val}$
7. return val

Figure 18: Games G_2 and G_2' (Theorem A.2). G_2' includes the box code and G_2 does not.

can distinguish F_t from a truly random function and/or an adversary $\mathcal{B}_2$ that can distinguish F_d from a truly random function. Formally, there exist adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$, such that

$$\begin{aligned} |\Pr[\mathsf{G}_0 = 1] - \Pr[\mathsf{G}_1 = 1]| &\leq \mathbf{Adv}^{\mathsf{prf}}_{\mathsf{F}_t, \mathcal{B}_1}(\lambda) + \mathbf{Adv}^{\mathsf{prf}}_{\mathsf{F}_d, \mathcal{B}_2}(\lambda) \\ &\leq \mathsf{neg}(\lambda). \end{aligned} \tag{23}$$

Game G_2: In G_2, instead of calling H_1 to generate label in the Update protocol, we pick random strings. Then, during the Search protocol, the random oracle H_1 is programmed accordingly to ensure consistency.

Table Hash_1 and $\mathbf{H}_1$ are used to simulate the random oracle H_1, the entries in the table Hash_1 are referred by $(w, \mathsf{ver}, \mathsf{c})$ and in the table $\mathbf{H}_1$ by (k, c).

Figure 18 formally describes G_2, and an intermediate game G_2'. In G_2', H_1 is never programmed to two different values for the same input, thus, ensuring consistency. Instead of storing the randomly picked value in table Hash_1 at position $(w, \mathsf{ver}_w, \mathsf{c}_w)$, it first checks whether H_1 is already programmed at value (k_w, c_w) which can happen if there was a query to the random oracle H_1 with input $(k_w \| \mathsf{c}_w)$. If the check is true, the value $\mathsf{H}_1(\mathsf{k}_w \| \mathsf{c}_w)$ is stored in $\mathsf{Hash}_1[w, \mathsf{ver}_w, \mathsf{c}_w]$ else the randomly picked value is stored in $\mathsf{Hash}_1[w, \mathsf{ver}_w, \mathsf{c}_w]$. The random oracle when needed in the Search protocol in line 9 or by an adversary's query to the random oracle H_1 in line 5 is lazily programmed in G_2, so that the outputs are consistent throughout.

The only difference between game G_1 and G_2' is how we model the random oracle H_1. The outputs of H_1 is perfectly indistinguishable in both these games, therefore,

$$\Pr[\mathsf{G}_2' = 1] = \Pr[\mathsf{G}_1 = 1]. \tag{24}$$

Let us denote the event 'the flag bad is set to true' in G_2' by $\mathbb{E}_1$. The games G_2' and G_2 are also perfectly identical unless the event $\mathbb{E}_1$ occurs, and we can apply *identical-until-bad* technique to bound the distinguishing advantage between G_2' and G_2.

$$|\Pr[\mathsf{G}_2' = 1] - \Pr[\mathsf{G}_2 = 1]| \leq \Pr[\mathbb{E}_1]. \tag{25}$$

Setup()

1. $\mathbf{W}, \mathbf{T}, \mathbf{D} \leftarrow$ empty map
2. $\mathsf{u} \leftarrow 0$
3. $\mathbf{Update}, \mathsf{STs} \leftarrow$ empty table
4. $\mathsf{STs}[w][0] \leftarrow -1, \quad \forall w \in \mathsf{W}$
5. return ($\mathsf{EDB} = (\mathbf{D}, \mathbf{T})$, $\mathsf{st_c} = (\mathbf{W})$) to (S, C)

Update$(\mathsf{op}, w, \mathsf{ind}, \mathsf{st_C}; \mathsf{EDB})$

Update$_\mathsf{C}(\mathsf{op}, w, \mathsf{ind}, \mathsf{st_C})$

1. Append $(\mathsf{u}, \mathsf{op}, \mathsf{ind})$ to $\mathbf{Update}[w]$
2. $\mathsf{Hash}_1[\mathsf{u}] \xleftarrow{\$} \{0,1\}^{2\lambda}$
3. $\mathsf{Hash}_2[\mathsf{u}] \xleftarrow{\$} \{0,1\}^{\mu}$
4. Send $(\mathsf{Hash}_1[\mathsf{u}], \mathsf{Hash}_2[\mathsf{u}])$ to S
5. $\mathsf{u} \leftarrow \mathsf{u} + 1$

Search$(w, \mathsf{st_C}; \mathsf{EDB})$

Search$_\mathsf{C}(w, \mathsf{st_C})$:

1. Append u to $\mathsf{STs}[w]$
2. $(\mathsf{empty}, \mathsf{ver}_w, \mathsf{c}_w$ $\{\mathsf{ind}_{w,\mathsf{c}}, \mathbf{H}_{1,w,\mathsf{c}}, \mathbf{H}_{2,w,\mathsf{c}}\}_{0\le \mathsf{c} \le \mathsf{c}_w}) \leftarrow$ $\mathsf{GetData}(\mathbf{Update}, \mathsf{STs}, \mathsf{Hash}_1, \mathsf{Hash}_2)$
3. **if** $\mathsf{empty} = \mathsf{true}$ **then**
4. return $\emptyset$
5. **else**
6. $\mathsf{label}_w \leftarrow \mathsf{Key}_d[w]$
7. **if** $\mathsf{c}_w \neq -1$ **then**
8. $\mathsf{k}_w \leftarrow \mathsf{Key}_t[w, ver_w]$
9. **for** $i = 0$ to c_w **do**
10. $\mathsf{H}_1(\mathsf{k}_w, i) \leftarrow \mathbf{H}_{1,w,i}$
11. $\mathsf{H}_2(\mathsf{k}_w, i) \leftarrow \mathbf{H}_{2,w,i} \oplus \mathsf{ind}_{w,i}$
12. **else**
13. $\mathsf{k}_w \leftarrow \perp$
14. Send $(\mathsf{label}_w, \mathsf{k}_w, \mathsf{c}_w)$ to S
15. $\mathsf{u} \leftarrow \mathsf{u} + 1$

GetData$(\mathbf{Update}, \mathsf{STs}, \mathsf{Hash}_1, \mathsf{Hash}_2)$

1. **if** $\mathbf{Update}[w] = \perp$
2. $\mathsf{empty} \leftarrow \mathsf{true}$
3. **else**
4. $\mathsf{empty} \leftarrow \mathsf{false}$
5. $\mathsf{ver}_w \leftarrow -1$
6. **for** $i = 1$ to $|\mathsf{STs}[w]| - 1$ **do**
7. **if** $\exists\, \mathsf{u}, \mathsf{op}, \mathsf{ind}$ s.t. $(\mathsf{u}, \mathsf{op}, \mathsf{ind}) \in$ $\mathbf{Update}[w] \wedge \mathsf{u} > \mathsf{STs}[w][i-1] \wedge$ $\mathsf{u} < \mathsf{STs}[w][i]$ **then**
8. $\mathsf{ver}_w \leftarrow \mathsf{ver}_w + 1$
9. $\mathsf{c}_w \leftarrow -1$
10. $ul \leftarrow |\mathsf{STs}[w]| - 1$
11. $\mathsf{Buf}_w \leftarrow \emptyset$
12. **for** $\mathsf{u} = \mathsf{STs}[w][ul-1]$ to $\mathsf{STs}[w][ul]$ **do**
13. **if** $\exists\, \mathsf{op}, \mathsf{ind}$ s.t. $(\mathsf{u}, \mathsf{op}, \mathsf{ind})$ $\in \mathbf{Update}[w]$ **then**
14. $\mathsf{c}_w \leftarrow \mathsf{c}_w + 1$
15. $\mathsf{b} \leftarrow 0(\mathsf{op} = \mathsf{add})/1(\mathsf{op} = \mathsf{del})$
16. $\mathsf{ind}_{w,\mathsf{c}_w} \leftarrow \mathsf{b} \| \mathsf{ind}$
17. $\mathbf{H}_{1,w,\mathsf{c}_w} \leftarrow \mathsf{Hash}_1[\mathsf{u}]$
18. $\mathbf{H}_{2,w,\mathsf{c}_w} \leftarrow \mathsf{Hash}_2[\mathsf{u}]$
19. return $(\mathsf{empty}, \mathsf{ver}_w, \mathsf{c}_w,$ $\{\mathsf{ind}_{w,\mathsf{c}}, \mathbf{H}_{1,w,\mathsf{c}}, \mathbf{H}_{2,w,\mathsf{c}}\}_{0\le \mathsf{c} \le \mathsf{c}_w})$

Figure 19: Game G_4 (Theorem A.2)

The event $\mathbb{E}_1$ occurs in line 7 of Update protocol and in line 5 of H_1 algorithm. The former captures the fact that adversary has already queried random oracle H_1 at input $(\mathsf{k}_w \| \mathsf{c}_w)$ and the latter captures the fact that the adversary queries random oracle H_1 on a valid input $(\mathsf{k} \| \mathsf{c})$ and the value k is not currently revealed to adversary. Since, the value k_w is picked uniformly at random, the probability with which event $\mathbb{E}_1$ occurs is negligible. Using (24) and (25), we can conclude:

$$|\Pr[\mathsf{G}_1 = 1] - \Pr[\mathsf{G}_2 = 1]| \le \mathsf{neg}(\lambda). \tag{26}$$

Game G_3: In G_3, what we did for H_1 in game G_2, we do for H_2. Using the same arguments, we can conclude:

$$|\Pr[\mathsf{G}_2 = 1] - \Pr[\mathsf{G}_3 = 1]| \le \mathsf{neg}(\lambda). \tag{27}$$

Game G_4: In G_4 (see Figure 19), we abstract out the information that needs to be simulated by the simulator in order to output transcripts identical to G_3. Using GetData algorithm in G_4, one keeps track of the randomly generated label and pad differently than in G_3. In Search protocol, the random oracles are programmed identically to that in G_3. Queries to random oracles H_1 and H_2 can be simulated by outputting random values.

The transcripts outputted by Update protocol is identical to that of Update protocol in G_3, as we output fresh random strings in the Update protocol.

Next, we describe the Search protocol in G_4. Based on tables **Update** and STs, the value of the following components: empty, ver_w, $\mathsf{c}_w$and $\{\mathsf{ind}_{w,\mathsf{c}}, \mathbf{H}_{1,w,\mathsf{c}}, \mathbf{H}_{2,w,\mathsf{c}}\}_{0\le \mathsf{c} \le \mathsf{c}_w}$ is determined using GetData algorithm. Flag empty is set to 1 if **Update** is empty. ver_w is the count of searches for which there was an update on map $\mathbf{T}$ corresponding to keyword w after the previous search. The loop in line 6 of GetData determines the number of times version number is updated, i.e., value of ver_w. Here, c_w denotes the count of

updates on map $\mathbf{T}$ corresponding to keyword w after the previous search. $\{\mathsf{ind}_{w,\mathsf{c}}\}_{0\le \mathsf{c}\le \mathsf{c}_w}$ are the document identifier values along with operation bit that have been added to $\mathbf{T}$ after the previous search operation and the values $\{\mathbf{H}_{1,w,\mathsf{c}}, \mathbf{H}_{2,w,\mathsf{c}}\}_{0\le \mathsf{c}\le \mathsf{c}_w}$ are used to simulate the random oracles consistently with the response given at the time of update queries. The loop in line 12 of GetData computes the values of c_w and $\{\mathsf{ind}_{w,\mathsf{c}}, \mathbf{H}_{1,w,\mathsf{c}}, \mathbf{H}_{2,w,\mathsf{c}}\}_{0\le \mathsf{c}\le \mathsf{c}_w}$. As the value of components are computed correctly and consistently w.r.t. previous queries, the transcripts of Search protocol is identical to that of Search protocol in G_3. Therefore, we conclude that:

$$\Pr[\mathsf{G}_4 = 1] = \Pr[\mathsf{G}_3 = 1]. \tag{28}$$

Simulator Sim: Finally, we construct a simulator that given the leakage profile $\mathcal{L}$ simulates game G_4 correctly. It is easy to see that, Sim can simulate Update protocol correctly. Instead of using keyword w, Sim uses the counter $\overline{w}$=min $\mathsf{sp}(w)$ uniquely mapped from w using $\mathcal{L}_{\mathsf{Search}}$ in simulating the Search protocol (line 6 and line 8). In line 2 of Search protocol, Sim uses $\mathsf{sp}(w)$ and $\mathsf{Hist}(w)$ instead of STs and **Update** as input to the GetData algorithm. Thus, Sim is able to produce transcripts of output of Search and Update protocols identical to G_4. Hence, we conclude that:

$$\Pr[\mathbf{Ideal}^{\Pi_{\mathsf{FP}}}_{\mathcal{A},\mathsf{Sim}}(\lambda) = 1] = \Pr[\mathsf{G}_4]. \tag{29}$$

By connecting all the games, we conclude

$$\left|\Pr[\mathbf{Real}^{\Pi_{\mathsf{FP}}}_{\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{Ideal}^{\Pi_{\mathsf{FP}}}_{\mathcal{A},\mathsf{Sim}}(\lambda) = 1]\right| \le \mathsf{neg}(\lambda).$$

□

B Security Proof of Π_{BP}-prime

Theorem B.1. *If* F_d, F_t *are secure* PRF*s,* $\mathcal{E}$ *is* RCPA *secure and* H_1 *is a hash function modeled as random oracle outputting* 2λ *bits then* Π_{BP}-prime *is* BP-I *secure (Definition 3.5).*

Proof. We structure our proof using a sequence of games G_0 to G_4. G_0 will compute a distribution identical to $\mathbf{Real}^{\Pi_{\mathsf{BP}}\text{-}\mathsf{prime}}_{\mathcal{A}}(\lambda)$ and G_4 will compute a distribution that can be simulated perfectly given $\mathcal{L}$, i.e., its distribution is identical to $\mathbf{Ideal}^{\Pi_{\mathsf{BP}}\text{-}\mathsf{prime}}_{\mathcal{A},\mathsf{Sim}}(\lambda)$ and the intermediate games are hybrids.

Game G_0: G_0 is exactly identical to $\mathbf{Real}^{\Pi_{\mathsf{BP}}\text{-}\mathsf{prime}}_{\mathcal{A}}(\lambda)$.

$$\Pr[\mathbf{Real}^{\Pi_{\mathsf{BP}}\text{-}\mathsf{prime}}_{\mathcal{A}}(\lambda) = 1] = \Pr[\mathsf{G}_0 = 1]. \tag{30}$$

Game G_1: In G_1, every call to PRF F_t and F_d are answered using tables Key_t and Key_d respectively. The entries in table Key_t are referred by (w, ver) and entries in table Key_d are referred by w. If there exists an adversary $\mathcal{A}$ that is able to distinguish between games G_0 and G_1, we can construct an adversary $\mathcal{B}_1$ that can distinguish F_t from a truly random function and/or an adversary $\mathcal{B}_2$ that can distinguish F_d from a truly random function. Formally, there exists adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$, such that

$$\begin{aligned} |\Pr[\mathsf{G}_0 = 1] - \Pr[\mathsf{G}_1 = 1]| &\le \mathbf{Adv}^{prf}_{\mathsf{F}_t,\mathcal{B}_1}(\lambda) + \mathbf{Adv}^{prf}_{\mathsf{F}_d,\mathcal{B}_2}(\lambda) \\ &\le \mathsf{neg}(\lambda). \end{aligned} \tag{31}$$

Game G_2: In G_2, instead of calling H_1 to generate label in the Update protocol, we pick random strings. Then, during the Search protocol, the random oracle H_1 is programmed accordingly, to ensure consistency.

Table Hash_1 and $\mathbf{H}_1$ are used to simulate the random oracle H_1, the entries in the table Hash_1 are referred by $(w, \mathsf{ver}, \mathsf{c})$ and in the table $\mathbf{H}_1$ by (k, c).

Figure 20 formally describes G_2, and an intermediate game G'_2. In G'_2, H_1 is never programmed to two different values for the same input, thus, ensuring consistency. Instead of storing the randomly picked value in table Hash_1 at position $(w, \mathsf{ver}_w, \mathsf{c}_w)$, it first checks whether H_1 is already programmed at value (k_w, c_w)

Setup()

1. $\mathsf{k}_e \leftarrow \mathcal{E}.\mathsf{Gen}()$
2. $\mathbf{W}, \mathbf{T}, \mathbf{D} \leftarrow$ empty map
3. $\mathsf{bad} \leftarrow \mathsf{false}$
4. return ($\mathsf{EDB} = (\mathbf{D}, \mathbf{T})$, $\mathsf{st_C} = (\mathsf{k}_e, \mathsf{k}_t, \mathsf{k}_d, \mathbf{W})$) to (S, C)

Update$(\mathsf{op}, w, \mathsf{ind}, \mathsf{st_C}; \mathsf{EDB})$

Update$_\mathsf{C}(\mathsf{op}, w, \mathsf{ind}, \mathsf{st_C})$

1. $(\mathsf{ver}_w, \mathsf{c}_w) \leftarrow \mathbf{W}[w]$
2. **if** $(\mathsf{ver}_w, \mathsf{c}_w) = \perp$ **then**
3. $\quad \mathsf{ver}_w \leftarrow 0,\ \mathsf{c}_w \leftarrow -1$
4. $\mathsf{c}_w \leftarrow \mathsf{c}_w + 1$
5. $\mathsf{k}_w \leftarrow \mathsf{Key}_t[w, \mathsf{ver}_w]$
6. $\mathsf{val}_r \xleftarrow{\$} \{0,1\}^\lambda$
7. **if** $\mathbf{H}_1[\mathsf{k}_w, \mathsf{c}_w] \neq \perp$ **then**
8. $\quad \mathsf{bad} \leftarrow \mathsf{true}, \boxed{\mathsf{val}_r \leftarrow \mathsf{H}_1(\mathsf{k}_w \| \mathsf{c}_w)}$
9. $\mathsf{Hash}_1[w, \mathsf{ver}_w, \mathsf{c}_w] \leftarrow \mathsf{val}_r$
10. $\mathsf{b} \leftarrow 0$ (op=add)/ 1 (op=del)
11. $\mathsf{e} \leftarrow \mathcal{E}.\mathsf{Enc}(\mathsf{k}_e, \mathsf{b} \| \mathsf{ind})$
12. $\mathbf{W}[w] \leftarrow (\mathsf{ver}_w,\ \mathsf{c}_w)$
13. Send $(\mathsf{val}_r, \mathsf{e})$ to S

Search$(w, \mathsf{st_C}; \mathsf{EDB})$

Round 1

Search$_\mathsf{C}(w, \mathsf{st_C})$:

1. $(\mathsf{ver}_w, \mathsf{c}_w) \leftarrow \mathbf{W}[w]$
2. **if** $(\mathsf{ver}_w, \mathsf{c}_w) = \perp$ **then**
3. $\quad$ return $\emptyset$
4. **else**
5. $\quad \mathsf{label}_w \leftarrow \mathsf{Key}_d[w]$
6. $\quad$ **if** $\mathsf{c}_w \neq -1$ **then**
7. $\quad\quad \mathsf{k}_w \leftarrow \mathsf{Key}_t[w, \mathsf{ver}_w]$
8. $\quad\quad$ **for** $i = 0$ to c_w **do**
9. $\quad\quad\quad \mathbf{H}_1[\mathsf{k}_w, i] \leftarrow \mathsf{Hash}_1[w, \mathsf{ver}_w, i]$
10. $\quad\quad \mathsf{ver}_w \leftarrow \mathsf{ver}_w + 1$
11. $\quad\quad \mathsf{c}_w \leftarrow -1$
12. $\quad\quad \mathbf{W}[w] \leftarrow (\mathsf{ver}_w,\ \mathsf{c}_w)$
13. $\quad$ **else**
14. $\quad\quad \mathsf{k}_w \leftarrow \perp$ //No need of round 2
15. $\quad$ Send $(\mathsf{label}_w, \mathsf{k}_w, \mathsf{c}_w)$ to S

Round 2

Search$_\mathsf{C}(w, \mathsf{st_C})$:

16. Receive $(\mathsf{PSet}_w, \mathsf{AList})$ from S
17. $(\mathsf{ver}_w, \mathsf{c}_w) \leftarrow \mathbf{W}[w]$
18. $\mathsf{AuxSet} \leftarrow \mathsf{PSet}_w$
19. **for** $\mathsf{c} \leftarrow 0$ to $|\mathsf{AList}|$ **do**
20. $\quad (\mathsf{b} \| \mathsf{ind}) \leftarrow \mathcal{E}.\mathsf{Dec}(\mathsf{k}_e, \mathsf{AList}[\mathsf{c}])$
21. $\quad\quad$ **if** $\mathsf{b} = 0$
22. $\quad\quad \mathsf{AuxSet} \leftarrow \mathsf{AuxSet} \cup \{\mathsf{ind}\}$
23. $\quad\quad$ **else**
24. $\quad\quad \mathsf{AuxSet} \leftarrow \mathsf{AuxSet} \setminus \{\mathsf{ind}\}$
25. return AuxSet

$\mathsf{H}_1(\mathsf{k} \| \mathsf{c})$

1. $\mathsf{val} \leftarrow \mathbf{H}_1[\mathsf{k}, \mathsf{c}]$
2. **if** $\mathsf{val} = \perp$ **then**
3. $\quad \mathsf{val} \xleftarrow{\$} \{0,1\}^\lambda$
4. $\quad$ **if** $\exists w, \mathsf{ver}$ s.t. $\mathsf{ver} = \mathbf{W}[w].\mathsf{ver}_w \wedge \mathsf{k} = \mathsf{Key}_t[w, \mathsf{ver}] \wedge \mathsf{c} \leq \mathbf{W}[w].\mathsf{c}_w$ **then**
5. $\quad\quad \mathsf{bad} \leftarrow \mathsf{true}, \boxed{\mathsf{val} \leftarrow \mathsf{Hash}_1[w, \mathsf{ver}, \mathsf{c}]}$
6. $\quad \mathbf{H}_1[\mathsf{k}, \mathsf{c}] \leftarrow \mathsf{val}$
7. return val

Figure 20: Games G_2 and G_2' (Theorem B.1). G_2' includes the box code and G_2 does not.

which can happen if there was a query to the random oracle H_1 with input $(k_w \| \mathsf{c}_w)$. If the check is true, the value $\mathsf{H}_1(\mathsf{k}_w \| \mathsf{c}_w)$ is stored in $\mathsf{Hash}_1[w, \mathsf{ver}_w, \mathsf{c}_w]$ else the randomly picked value is stored in $\mathsf{Hash}_1[w, \mathsf{ver}_w, \mathsf{c}_w]$. The random oracle when needed in the Search protocol in line 9 or by an adversary's query to random oracle H_1 in line 5 is lazily programmed in G_2, so that the outputs are consistent throughout.

The only difference between game G_1 and G_2' is how we model the random oracle H_1. The outputs of H_1 is perfectly indistinguishable in both these games, therefore,

$$\Pr[\mathsf{G}_2' = 1] = \Pr[\mathsf{G}_1 = 1]. \tag{32}$$

Let us denote the event 'the flag bad is set to true' in G_2' by $\mathbb{E}_1$. The games G_2' and G_2 are also perfectly identical unless the event $\mathbb{E}_1$ occurs, and we can apply *identical-until-bad* technique to bound the distinguishing advantage between G_2' and G_2.

$$|\Pr[\mathsf{G}_2' = 1] - \Pr[\mathsf{G}_2 = 1]| \leq \Pr[\mathbb{E}_1]. \tag{33}$$

The event $\mathbb{E}_1$ occurs in line 8 of Update protocol and in line 5 of H_1 algorithm. The former captures the fact that the adversary has already queried random oracle H_1 at input $(\mathsf{k}_w \| \mathsf{c}_w)$ and the latter captures the fact that the adversary queries random oracle H_1 on a valid input $(\mathsf{k} \| \mathsf{c})$ and the value k is not currently revealed to adversary. Since, the value k_w is picked uniformly at random, the probability with which event $\mathbb{E}_1$ occurs is negligible. Using (32) and (33), we can conclude:

$$|\Pr[\mathsf{G}_1 = 1] - \Pr[\mathsf{G}_2 = 1]| \leq \mathsf{neg}(\lambda). \tag{34}$$

Game G_3: In G_3, instead of computing ciphertexts corresponding to actual op bit 'b' and document identifier ind, we replace them with randomly chosen ciphertexts from the ciphertext space. We also store the plaintext-ciphertext pairs in order to simulate the view of round 2 of the Search protocol correctly. If there exists an

Setup()

1. $\mathbf{W}, \mathbf{T}, \mathbf{D} \leftarrow$ empty map
2. $\mathsf{u} \leftarrow 0$
3. $\mathbf{Update}, \mathsf{STs} \leftarrow$ empty table
4. $\mathsf{STs}[w][0] \leftarrow -1, \quad \forall w \in \mathsf{W}$
5. return ($\mathsf{EDB} = (\mathbf{D}, \mathbf{T})$, $\mathsf{st_c} = (\mathsf{k}_e, \mathbf{W})$) to (S, C)

Update$(\mathsf{op}, w, \mathsf{ind}, \mathsf{st_C}; \mathsf{EDB})$

Update$_\mathsf{C}(\mathsf{op}, w, \mathsf{ind}, \mathsf{st_C})$

1. Append $(\mathsf{u}, \mathsf{op}, \mathsf{ind})$ to $\mathbf{Update}[w]$
2. $\mathsf{Hash}_1[\mathsf{u}] \xleftarrow{\$} \{0,1\}^{2\lambda}$
3. $\mathsf{e} \xleftarrow{\$} \mathcal{C}$
4. Send $(\mathsf{Hash}_1[\mathsf{u}], \mathsf{e})$ to S
5. $\mathsf{u} \leftarrow \mathsf{u} + 1$

GetData$_{r1}(\mathbf{Update}, \mathsf{STs}, \mathsf{Hash}_1)$

1. **if** $\mathbf{Update}[w] = \perp$
2. $\quad \mathsf{empty} \leftarrow \mathsf{true}$
3. **else**
4. $\quad \mathsf{empty} \leftarrow \mathsf{false}$
5. $\quad \mathsf{ver}_w \leftarrow -1$
6. $\quad$ **for** $i = 1$ to $|\mathsf{STs}[w]| - 1$ **do**
7. $\quad\quad$ **if** $\exists\, \mathsf{u}, \mathsf{op}, \mathsf{ind}$ s.t. $(\mathsf{u}, \mathsf{op}, \mathsf{ind}) \in \mathbf{Update}[w] \wedge \mathsf{u} > \mathsf{STs}[w][i-1] \wedge \mathsf{u} < \mathsf{STs}[w][i]$ **then**
8. $\quad\quad\quad \mathsf{ver}_w \leftarrow \mathsf{ver}_w + 1$
9. $\quad \mathsf{c}_w \leftarrow -1$
10. $\quad ul \leftarrow |\mathsf{STs}[w]| - 1$
11. $\quad$ **for** $\mathsf{u} = \mathsf{STs}[w][ul-1]$ to $\mathsf{STs}[w][ul]$ **do**
12. $\quad\quad$ **if** $\exists\, \mathsf{op}, \mathsf{ind}$ s.t. $(\mathsf{u}, \mathsf{op}, \mathsf{ind}) \in \mathbf{Update}[w]$ **then**
13. $\quad\quad\quad \mathsf{c}_w \leftarrow \mathsf{c}_w + 1$
14. $\quad\quad\quad \mathbf{H}_{1,w,\mathsf{c}_w} \leftarrow \mathsf{Hash}_1[\mathsf{u}]$
15. return $(\mathsf{empty}, \mathsf{ver}_w, \mathsf{c}_w, \{\mathbf{H}_{1,w,\mathsf{c}}\}_{0 \le \mathsf{c} \le \mathsf{c}_w})$

Search$(w, \mathsf{st_C}; \mathsf{EDB})$

Round 1

Search$_\mathsf{C}(w, \mathsf{st_C})$:

1. Append u to $\mathsf{STs}[w]$
2. $(\mathsf{empty}, \mathsf{ver}_w, \mathsf{c}_w, \{\mathbf{H}_{1,w,\mathsf{c}}\}_{0 \le \mathsf{c} \le \mathsf{c}_w}) \leftarrow$ GetData$_{r1}(\mathbf{Update}, \mathsf{STs}, \mathsf{Hash}_1)$
3. **if** $\mathsf{empty} = \mathsf{true}$ **then**
4. $\quad$ return $\emptyset$
5. **else**
6. $\quad \mathsf{label}_w \leftarrow \mathsf{Key}_d[w]$
7. $\quad$ **if** $\mathsf{c}_w \neq -1$ **then**
8. $\quad\quad \mathsf{k}_w \leftarrow \mathsf{Key}_t[w, \mathsf{ver}_w]$
9. $\quad\quad$ **for** $i = 0$ to c_w **do**
10. $\quad\quad\quad \mathsf{H}_1[\mathsf{k}_w, i] \leftarrow \mathbf{H}_{1,w,i}$
11. $\quad$ **else**
12. $\quad\quad \mathsf{k}_w \leftarrow \perp$
13. $\quad$ Send $(\mathsf{label}_w, \mathsf{k}_w, \mathsf{c}_w)$ to S

Round 2

Search$_\mathsf{C}(w, \mathsf{st_C})$:

14. Receive $(\mathsf{PSet}_w, \mathsf{AList})$ from S
15. $\mathsf{AuxSet} \leftarrow$ GetData$_{r2}(\mathsf{PSet}_w, \mathbf{Update}, \mathsf{STs})$
16. return AuxSet
17. $\mathsf{u} \leftarrow \mathsf{u} + 1$

GetData$_{r2}(\mathsf{PSet}_w, \mathbf{Update}, \mathsf{STs})$

1. $\mathsf{AuxSet} \leftarrow \mathsf{PSet}_w$
2. $ul \leftarrow |\mathsf{STs}[w]| - 1$
3. $\quad$ **for** $\mathsf{u} = \mathsf{STs}[w][ul-1]$ to $\mathsf{STs}[w][ul]$ **do**
4. $\quad\quad$ **if** $\exists\, \mathsf{op}, \mathsf{ind}$ s.t. $(\mathsf{u}, \mathsf{op}, \mathsf{ind}) \in \mathbf{Update}[w]$ **then**
5. $\quad\quad\quad$ **if** $\mathsf{op} = \mathsf{add}$ **then**
6. $\quad\quad\quad\quad \mathsf{AuxSet} \leftarrow \mathsf{AuxSet} \cup \{\mathsf{ind}\}$
7. $\quad\quad\quad$ **else**
8. $\quad\quad\quad\quad \mathsf{AuxSet} \leftarrow \mathsf{AuxSet} \setminus \{\mathsf{ind}\}$
9. return AuxSet

Figure 21: Game G_4 (Theorem B.1)

adversary $\mathcal{A}$ that is able to distinguish between games G_2 and G_3, we can construct an adversary $\mathcal{B}$ that breaks the RCPA security of $\mathcal{E}$.

Game G_4: In G_4 (see Figure 21), we abstract out the information that needs to be simulated by the simulator in order to output transcripts identical to G_3. Using GetData$_{r1}$ and GetData$_{r2}$ algorithms in G_4, one keeps track of label differently than in G_4. In Search protocol, the random oracle is programmed identically to that in G_4. Queries to random oracle H_1 can be simulated by outputting random values.

As we output fresh random strings in Update protocol, the transcripts of Update protocol is identical to that of Update protocol in G_3.

Next, let us describe the Search protocol in G_4. Based on tables **Update** and STs, the value of following components: empty, ver_w, c_w and $\{\mathbf{H}_{1,w,\mathsf{c}}\}_{0 \le \mathsf{c} \le \mathsf{c}_w}$ are determined using GetData$_{r1}$ algorithm. Flag empty is set to 1 if **Update** is empty. ver_w is the count of searches for which there was an update on map $\mathbf{T}$ corresponding to keyword w after the previous search. The loop in line 6 of GetData$_{r1}$ determines the number of times version number is updated, i.e., value of ver_w. Here, c_w denotes the count of updates on map $\mathbf{T}$ corresponding to keyword w after the previous search and the values $\{\mathbf{H}_{1,w,\mathsf{c}}\}_{0 \le \mathsf{c} \le \mathsf{c}_w}$ are used to simulate the random oracle consistently with the response given at the time of update queries. The loop in line 11 of GetData$_{r1}$ computes the values of c_w, and $\{\mathbf{H}_{1,w,\mathsf{c}}\}_{0 \le \mathsf{c} \le \mathsf{c}_w}$. Given PSet_w, STs and **Update**, we determine the current set of documents matching the keyword w, i.e., AuxSet=DB(w) in GetData$_{r2}$ which is then used to simulate the output of round 2 of Search protocol. As the value of components are determined

correctly and consistently, the transcripts of Search protocol is identical to that of Search protocol in G_3. Therefore, we conclude that:

$$\Pr[\mathsf{G}_4 = 1] = \Pr[\mathsf{G}_3 = 1]. \tag{35}$$

Simulator Sim: Finally, we construct a simulator that given the leakage profile $\mathcal{L}$ simulates game G_4 correctly. Sim can simulate Update protocol correctly as in G_4. Instead of using keyword w, Sim uses the counter $\overline{w}$=min $\mathsf{sp}(w)$ uniquely mapped from w using $\mathcal{L}_{\mathsf{Search}}$ in simulating the Search protocol (line 6 and line 8). In line 2 of Search protocol, Sim uses $\mathsf{sp}(w)$ and $\mathsf{Updates}(w)$ instead of STs and **Update** as input to the $\mathsf{GetData}_{r1}$ algorithm. Note that in $\mathsf{GetData}_{r1}$, we only use the timestamps of the search and update queries in order to perform the simulation. Also, the output of $\mathsf{GetData}_{r2}$ (line 15) is the current set of documents matching the keyword w, i.e., $\mathsf{DB}(w)$ which is a part of leakage profile, $\mathcal{L}_{\mathsf{Search}}$. Thus, Sim is able to produce transcripts of output of Search and Update protocols identical to G_4. Hence, we conclude that:

$$\Pr[\mathbf{Ideal}^{\Pi_{\mathsf{BP}\text{-}\mathsf{prime}}}_{\mathcal{A},\mathsf{Sim}}(\lambda) = 1] = \Pr[\mathsf{G}_4]. \tag{36}$$

By connecting all the games, we conclude

$$\left|\Pr[\mathbf{Real}^{\Pi_{\mathsf{BP}\text{-}\mathsf{prime}}}_{\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{Ideal}^{\Pi_{\mathsf{BP}\text{-}\mathsf{prime}}}_{\mathcal{A},\mathsf{Sim}}(\lambda) = 1]\right| \leq \mathsf{neg}(\lambda).$$

□