Leakage-Abuse Attacks Against Forward and Backward Private Searchable Symmetric Encryption

Lei Xu
Nanjing University of Science and Technology
Nanjing, China
City University of Hong Kong
Hong Kong, China
xuleicrypto@gmail.com

Leqian Zheng
City University of Hong Kong
Hong Kong, China
leqizheng2-c@my.cityu.edu.hk

Chengzhi Xu
Nanjing University of Science and Technology
Nanjing, China
xcz@njust.edu.cn

Xingliang Yuan
Monash University
Melbourne, Australia
xingliang.yuan@monash.edu

Cong Wang
City University of Hong Kong
Hong Kong, China
congwang@cityu.edu.hk

ABSTRACT

Dynamic searchable symmetric encryption (DSSE) enables a server to efficiently search and update over encrypted files. To minimize the leakage during updates, a security notion named forward and backward privacy is expected for newly proposed DSSE schemes. Those schemes are generally constructed in a way to break the linkability across search and update queries to a given keyword. However, it remains underexplored whether forward and backward private DSSE is resilient against practical leakage-abuse attacks (LAAs), where an attacker attempts to recover query keywords from the leakage passively collected during queries.

In this paper, we aim to be the first to answer this question firmly through two non-trivial efforts. First, we revisit the spectrum of forward and backward private DSSE schemes over the past few years, and unveil some inherent constructional limitations in most schemes. Those limitations allow attackers to exploit query equality and establish a guaranteed linkage among different (refreshed) query tokens surjective to a candidate keyword. Second, we refine volumetric leakage profiles of updates and queries by associating each with a specific operation. By further exploiting update volume and query response volume, we demonstrate that all forward and backward private DSSE schemes can leak the same volumetric information (e.g., insertion volume, deletion volume) as those without such security guarantees. To testify our findings, we realize two generic LAAs, i.e., frequency matching attack and volumetric inference attack, and we evaluate them over various experimental settings in the dynamic context. Finally, we call for new efficient schemes to protect query equality and volumetric information across search and update queries.

CCS CONCEPTS

• **Security and privacy → Management and querying of encrypted data**; **Cryptanalysis and other attacks**.

KEYWORDS

searchable encryption, forward and backward privacy, vulnerabilities, leakage attacks

1 INTRODUCTION

Dynamic searchable symmetric encryption (DSSE) [9, 37, 38, 47] enables a server to efficiently search and update over encrypted files. During queries and updates (file addition and deletion), the server only knows precisely defined leakage, but not the contents of queries and files. In the past decade, various DSSE schemes have actively been proposed with regard to improved functionality [16, 33, 44, 54], security [49, 60, 66], and efficiency [4, 9, 37, 47]. Among others, minimizing the leakage for DSSE is the most emerging objective in the literature. A security notion of DSSE named forward and backward privacy (FP/BP) [7, 60] has become essential and standard for newly designed DSSE schemes [1, 7, 10, 11, 61, 62]. It echos the intrinsic security goals of SSE, i.e., no information about data should be learned if it has not been searched.

At a high level, forward privacy implies that newly added files cannot be associated with queries in the past, and the goal is to protect previous queries and files during updates to them; while backward privacy implies that subsequent queries cannot be executed over already deleted files, and the goal is to protect files and updates to them during queries. To achieve those security notions, all existing forward and backward private DSSE schemes are constructed in a way to break the linkability across queries and updates to a given keyword. As a result, leakage in those schemes

is expected to only reveal minimally necessary information in updates and queries, such as the number of updates and the number of matched results for a query.

Because of the above security features, a common understanding today is that FP/BP-DSSE may possibly mitigate file injection attacks [3, 57, 71, 72]. In these attacks (aka chosen-data attacks), an active attacker can inject carefully chosen files into the encrypted dataset, causing each keyword to manifest unique patterns that could potentially lead to the recovery of encrypted query keywords. However, beyond file injection attacks - a specific type of leakage-abuse attacks (LAAs) [8, 32], the study on those modern DSSE schemes and their resilience against more generic passive LAAs remains largely underexplored.

Leakage-abuse attacks (LAAs) against searchable symmetric encryption (SSE) schemes [3, 8, 30–32, 50, 67] show the implication and impact of leakage in real-world adversarial settings, where an attacker could have access to the background information of the underlying dataset. By formalizing the relationship between such background knowledge and the adversarial view using matrices or bi-partite graphs, it is demonstrated that the fundamental reason why LAAs can succeed is: the underlying characteristics of a dataset remain intact after the encryption of SSE [3]. Throughout queries, the attacker can identify these unique and structurally-invariant characteristics (e.g., query response lengths, query co-occurrence counts) that exist both in unencrypted and encrypted datasets, so as to facilitate query recovery.

Implicitly, for any LAA to be effective, the attackers have to determine which collected leakage patterns (from query tokens and query responses) are mapped to the underlying keywords. Here, the query token refers to the encrypted representation of the keyword to be queried, acting as the credential for the search operation. The primary purpose of using tokens in SSE is to enable the server to conduct keyword searches on encrypted data stored without disclosing the underlying plaintexts. We note that all passive LAAs investigate static SSE, where updates are excluded. In the static setting, the mapping between the set of query tokens and their respective responses, and the set of candidate keywords in the plaintext database are bijective. Namely, each query token and its response are exactly associated with a keyword. This makes the query recovery equivalent to finding the best bijective mapping between those two sets, using the aforementioned structural-equivalent leakage and the attacker's background knowledge.

In dynamic settings, this is no longer the case due to refreshed query tokens and responses, which are demanded by forward and backward security notions. As seen in Fig. 1, there exist multiple individual query tokens and their respective responses related to the same candidate keyword. That is, the mapping from the set of observed query tokens and their responses to the set of candidate keywords becomes surjective but no longer bijective. Moreover, the client may add and/or remove files without the control of an attacker. That naturally obfuscates the exposed leakage from queries and updates and further increases the difficulty of passive LAAs.

1.1 Our Contributions

Given the above observations and challenges, in this paper, we aim to first answer the following open question: Is forward and
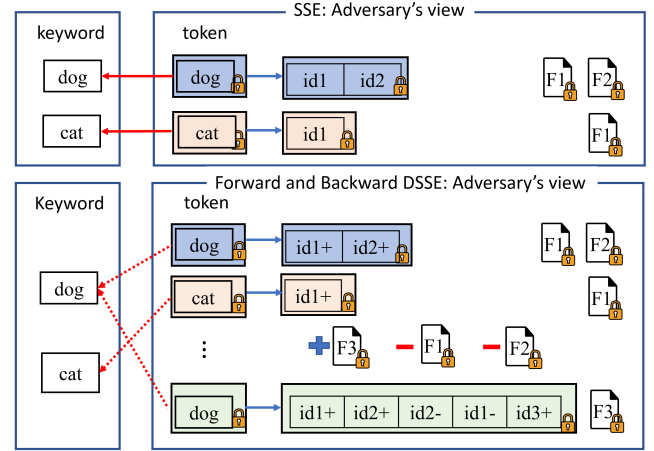

Figure 1: Difference between static SSE and forward and backward private DSSE

backward dynamic SSE more resilient against LAAs, compared to static SSE? To answer this question, we thoroughly revisit the spectrum of existing forward and backward private DSSE schemes over the past few years [1, 7, 10, 11, 61, 62]. Our conclusion presents a seemingly counterintuitive perspective to the above question: *those DSSE schemes with advanced security properties are also vulnerable to LAAs to a similar extent, compared to the traditional static schemes.*

Our key observations lay from two aspects. First, there are inherent constructional limitations in specific schemes that allow attackers to establish guaranteed linkage between refreshed query tokens after encrypted updates and their historical counterparts. Second, more generally, the leakage during updates, especially the volume of updates, if not protected well, will allow attackers to combine it with other leakage patterns (some are refined and formalized in this work) to group those refreshed query tokens and their responses that are more likely to be associated with a candidate keyword. In other words, both observations enable the attacker to effectively group seemingly unrelated query tokens (and their respective query responses) together. The grouping of query tokens transforms the aforementioned surjective mapping into a bijective one between the set of query tokens (and responses) and the set of candidate keywords. Thus, the attacker can readily launch known LAAs using the structurally-invariant leakage against forward and backward private DSSE, just like static SSE.

Below, we elaborate on our technical contributions.
Refining Leakage Profiles for DSSE. The first is how to identify and exploit the structurally-invariant leakage in the dynamic setting. Learning from prior LAAs, we reckon that the core information for exploitation expects to be the volumetric information of queries and updates. We note that existing LAAs targeting static SSE exploit singular leakage profiles, which are insufficient for DSSE. As opposed to merely using a response volume pattern, we refine and formalize the volumetric information further by associating it with specific operations. In particular, volumetric leakage patterns are defined as *insert volume pattern, delete volume pattern, response volume pattern, and update volume pattern.* The update volume pattern refers to that of both addition and deletion operations.

Table 1: A comparison of existing attacks and our attacks against (D)SSE schemes

Attack Setting[1]		Attack	Leakage Exploited[2]	Attack Target[3]
Passive against SSE	Known-data	Count attack [8]	co, rlen	Naïve
		Subgraph^{ID} [3]	rid	Naïve
		Subgraph^{VL} [3]	fvol	Naïve
		VolAn [3]	tvol	Naïve
		SelVolAn [3]	fvol, rlen	Naïve
		LEAP [50]	co	Naïve
	Sampled-data	IKK [31]	co	Naïve
		SAP [51]	rlen, qeq	CLRZ [13], SEAL [15], PPYY [56]
		IHOP [52]	qcor, co, qeq	CLRZ [13], Pancake [24], OSSE [59]
		Gui et al. [30]	co	CLRZ [13], PPYY [56], PRT-EMM[34]
Active against DSSE	Chosen-data	Decoding&Binary [3]	tvol	Naïve
		Zhang et al. [72]	rid	Naïve
		BVA [71]	fvol	SEAL [15], ShieldDB [64]
		BVMA [71]	rlen, qeq	SEAL[15], ShieldDB [64]
Passive against DSSE	Sampled-data	FMA	qeq, fvol	FP/BP-DSSE [7, 10, 11, 61, 62], VH/FP/BP-EMM [1]
	Known-data	VIA and PVIA	fvol, rlen	FP/BP-DSSE [7, 10, 11, 61, 62], VH/FP/BP-EMM [1]
	Sampled-data	LVIA	rlen	FP/BP-DSSE [7, 10, 11, 61, 62]

[1] Chosen-data attacks require the adversary to choose some of the data in the data collection. Known-data attacks require knowledge of a subset of the data collection. Sampled-data attacks require a statistical distribution of the data collection, such as keyword distribution of the data collection or the frequency of the data collection to be accessed;

[2] co, rid, fvol, tvol, rlen, qcor, and qeq denote the co-occurrence pattern, response identifier pattern, file volume pattern, total volume pattern, response length pattern, query correlation pattern, and query equality pattern, respectively;

[3] It lists the constructions that the attack targets. Naïve, FB/BP and VH refer to SSE constructions that leak full access pattern, achieve forward privacy/backward privacy, and response length hiding, respectively. Here, VH denotes the response length (the number of files matching the query) hiding.

It is worth noting that forward and backward private DSSE schemes do not separately leak insert volume and delete volume in updates, as operations are hidden during updates. But by exploiting query equality, update volume, and response volume, we find that those two leakage profiles can trivially be derived, as demonstrated in Section 4.2. It turns out that all forward and backward private DSSE schemes leak the same volumetric information as those without such security guarantees. We also show that those refined leakage profiles can help identify clients' updates, differentiating whether the client executes deletion operations or not. Inspired by very recent work [30], we further consider a system-wide leakage, i.e., the file volume pattern, which is inherent in any SSE schemes during file retrieval. We find that one can exploit this leakage to predict whether or not two different query tokens are for the same keyword, even if they originally cannot be linked from a specific scheme, e.g., ORAM-like constructions.

Revisiting Forward and Backward Private DSSE Schemes. With the above build leakage framework, we thoroughly revisit forward and backward private DSSE schemes [1, 7, 10, 11, 61, 62] in the literature. As mentioned, we discover that several schemes allow one to link refreshed query tokens of the same keyword from the specific construction level. In some forward private constructions, it is known that a query token can normally be computed by that of the latter one. In backward privacy, schemes can be classified into three levels, from the strongest to the weakest, according to the

security they can provide. In particular, except for schemes with the strongest security level (mostly ORAM-like constructions), the equality of queries can directly or indirectly be leaked, e.g., query tokens with respect to the same keyword share some common components. More details can be found later in Section 4.3. Once query equality can be determined, connections between queries and updates to a given keyword can also be derived.

Realizing LAAs against Forward and Backward Private DSSE. Based on the above stepping stones, we realize two generic passive query recovery attacks, frequency matching attack (FMA) and volumetric inference attack (VIA), under different settings and assumptions. In particular, FMA assumes that only the query distribution information is known [14, 51]. Unlike prior work, FMA first builds query equality information from observed queries and then determines the frequencies of linked queries that have the same underlying keyword. After that, the attacker can match the collected query frequencies with the known query distribution for query recovery. It is noteworthy that the attack performance of FMA is intrinsically augmented by the heterogeneous query distributions across distinct temporal intervals in dynamic settings. To be specific, when the query distribution pertains solely to a singular time interval, the corresponding recovery rate is a mere 1.8%. In contrast, when the query distributions are provided across 24 temporal intervals, the enhanced recovery rate spans a range of 63.5% to 93.4% across diverse settings.

Lei Xu, Leqian Zheng, Chengzhi Xu, Xingliang Yuan, & Cong Wang

VIA adapts to common assumptions of prior LAAs [3, 8, 50], and considers attackers with different levels of background knowledge, i.e., a fraction of files in updates between two linked queries, or the distribution of the dataset. After deriving query equality, VIA exploits our refined leakage profiles to recover the observed queries. Experimental results indicate that the recovery rate can reach up to 36.8%, even when only 50% of the inserted files are accessible to the attacker and up to 30% of the files are removed by the client. Furthermore, the recovery rate can increase up to 96% when more background knowledge is provided to the attacker. Table 1 overviews the recent attacks against SSE schemes. As summarized, our work is the first to focus on passive LAAs against DSSE schemes with forward and backward privacy.

Takeaway Messages. This work uncovers often-overlooked leakage patterns in existing forward and backward private dynamic SSE schemes. By incorporating dynamic operations into the traditional volumetric leakage and query equality leakage patterns, we refine the previous leakage patterns and bridge the gap between static SSE and dynamic SSE schemes. With this new leakage framework, we demonstrate how to recover queries in passive settings, illustrating that forward and backward privacy are insufficient and vulnerable to leakage-abuse attacks. Our research highlights the need for improved security measures against the proposed LAAs, urging the development of efficient schemes to safeguard query equality and volumetric information during search and update queries.

2 RELATED WORK

Dynamic Searchable Symmetric Encryption. As mentioned above, DSSE is motivated to support fundamental database operations like addition and deletion. Like static SSE, since the first DSSE constructions given by Kamara et al. [38], research interests on DSSE still focuses on asking for better trade-offs between the efficiency with lower search complexity and client storage overhead [4, 9, 36, 37, 47], stronger security property with less leakage [10, 49, 60, 63, 66] and expressiveness of supported queries [16, 33, 44, 54]. Despite the numerous constructions being proposed, including those that support forward and backward privacy, it has been pointed out by Patel et al. [55] that they either rely on the use of oblivious RAM with at least logarithmic cost [20] or tolerate specific leakages to obtain efficiency improvement [21, 35].

Leakage Abuse Attack and Suppressions. Leakage abuse attack (LAA) is a widely concerned threat that explores the real-world security problem of SSE in deployment. The main idea of LAA is to leverage the structurally-invariant characteristics that can be observed by the adversary to match their background knowledge to recover the query or reconstruct the clear dataset. The first seminar work on LAA was proposed by Islam et al. [31] and later improved by Cash et al. [8]. After that, a line of works under different adversarial models are proposed to attack SSE and other property-preserving secure search schemes [2, 17, 19, 25–28, 39–43, 46, 48, 50–53, 58, 68]. To mitigate LAA, the primary task is to suppress the above leakages [15, 24, 56, 65, 70], and the notion of forward and backward privacy was born in this context [7, 10, 11, 61, 62, 67]. Recent years have witnessed increasing efforts on forward and backward private DSSE with the hope of being efficient [11, 61]. And most of these follow-up works initiate the studies with the same security

notion like [7]. In addition to these works, there are also other works attempting to find more secure BP-DSSE schemes or define stronger backward security notions. For example, Chamani et al. [11]'s construction Orion does not reveal the total number of updates associated with the given keyword. The state-of-the-art, volume hiding, forward and backward private DSSE schemes were introduced by Amjad et al. [1]. They tolerate query equality patterns to obtain efficiency improvements.

3 PRELIMINARIES

We present a background of dynamic searchable encryption and its corresponding security notions.

Basic Notations. Let $\mathcal{W}$ be a keyword space and $\mathbf{D}$ be a file collection. Let $DB = \{(id_i, W_i)\}$ be a set of records that describe the correlations between keywords and files, where W_i denotes the set of all keywords appearing in $D_i \in \mathbf{D}$ and id_i represents the identifier of D_i. We define $DB(w)$ as the set of files containing w, i.e., $DB(w) = \{id_i : w \in W_i\}$. Lastly, we introduce the intersection of two collections. Given two collections $X = \{x_1, \ldots, x_m\}$ and $Y = \{y_1, \ldots, y_n\}$, we define their intersection Z as the collection of elements, including duplicities, appearing in both collections. For example, given $X = \{1, 1, 2, 2\}$ and $Y = \{1, 2, 2\}$, then their intersection is $Z = X \cap Y = \{1, 2, 2\}$. Likewise, their union is defined as $W = X \cup Y = \{1, 1, 2, 2\}$. For ease of presentation, we refer to the number of items in a set as $|\cdot|$ and the number of items in a collection as $\#(\cdot)$. Clearly, we have $\#(D_i) \geq |W_i|$.

Dynamic Searchable Symmetric Encryption. A dynamic searchable symmetric encryption scheme $DSSE = (Setup, Search, Update)$ is a tuple of one algorithm and two protocols between the client and the server proceeding as follows:

- $(K, \sigma, EDB) \leftarrow Setup(DB)$ is a probabilistic polynomial-time algorithm that takes DB as inputs and outputs a secret key K and a secret state σ for the client and an encrypted database EDB for the server.
- $(R, \perp) \leftarrow Search(K, \sigma, q, EDB)$ is a protocol run between the client with the key K, its state σ, and a query q as inputs, and the server with EDB as inputs. At the end of the protocol, the client receives a set of files R and the server receives nothing. In this paper, we only consider single-keyword search schemes where q refers to a keyword w.
- $(\sigma', EDB') \leftarrow Update(K, \sigma, op, in, EDB)$ is protocol run between the client with the key K, its state σ, an update operation op, and an update data in as inputs, and the server with the encrypted database EDB as input. At the end of the protocol, the client will update its state as σ', and the server will update its encrypted database as EDB'. The update operation op is taken from $\{add, del\}$ denoting the addition and the deletion of a keyword-identifier pair, respectively.

A DSSE scheme is *perfectly correct* if the search protocol returns all files matching the query. Notice that we focus on the DSSE schemes with perfect correctness in this work.

Security. The security of DSSE defines what an adversary learns in a DSSE scheme. It can be formalized as the indistinguishable model between a real word game SSE_{Real} and an ideal world game SSE_{Ideal} with predefined leakage profiles. These leakage profiles are usually modeled as triple $\mathcal{L} = (\mathcal{L}_{Setp}, \mathcal{L}_{Srch}, \mathcal{L}_{Updt})$, defining what

information from Setup, Search, Update leaks to the adversary. A DSSE scheme requires that the adversary learns nothing about the issued query and the content of the encrypted database except the above-admitted leakage profiles.

Forward and Backward Private DSSE. Despite the above security notion, DSSE also requires forward and backward privacy due to the disclosed update pattern and access pattern.

For forward privacy, it requires that the newly added file/keyword pairs cannot be linked by previous query tokens. This implies that previous query tokens cannot be used to search the above newly added entries. In existing constructions, such a notion is usually achieved by refreshing the query token. While for backward privacy, it focuses on the privacy of entries which are added and deleted later. In consideration of the balance between security and efficiency, Bost et al. [6] classify backward privacy into three levels, ranked from the strongest to the weakest. The details of these levels are as follows:

- Type-I. Backward privacy with insertion pattern: reveals the files that currently match keyword w, along with the time of their insertion and the total number of updates.
- Type-II. Backward privacy with update pattern: leaks the files currently matching keyword w, the time of their insertion, and the time of all updates made on w (excluding their content) are revealed.
- Type-III. Weak backward privacy: discloses the files that currently match keyword w, along with the time of their insertion, the time of all updates made on w, and which deletion update corresponds to which insertion update.

The formal security notion of the DSSE scheme, forward privacy, and three levels of backward privacy are given in our full version [69].

4 REFINING LEAKAGE PROFILES

In this section, we will refine the volumetric leakage profiles further to accommodate both insert and delete operations in dynamic settings. The primary purpose of making this refinement is to analyze what forward and backward private DSSE leaks from a volumetric perspective, which has not drawn sufficient attention.

4.1 Modeling Leakage

Given a DB and a query list Q in the form of (μ, w), we first present leakage patterns that have been defined in [6], where μ is the timestamp and w denotes the search keyword. Specifically,

- TimeDB(w) denotes the list of all files that match the query q, excluding deleted ones, together with the timestamp of when each file was inserted. Formally,

$$\text{TimeDB}(w) = \{(\mu, \text{id}) | (\mu, \text{add}, (w, \text{id})) \in Q$$
$$\text{and } \forall \mu', (\mu', \text{del}, (w, \text{id})) \notin Q\}.$$

- Updates(w) denotes the list of the timestamps of updates on w. Formally,

$$\text{Updates}(w) = \{\mu | (\mu, \text{add}, (w, \text{id}))$$
$$\text{or } (\mu, \text{del}, (w, \text{id})) \in Q\}.$$

- DelHist denotes the list of timestamps for all deletion operations and the timestamp of the inserted entry it removes.

Formally,

$$\text{DelHist}(w) = \{(\mu^{\text{add}}, \mu^{\text{del}}) | \exists \text{ id s.t. } (\mu^{\text{del}}, \text{del}, (w, \text{id}))$$
$$\in Q \text{ and } (\mu^{\text{add}}, \text{add}, (w, \text{id})) \in Q\}.$$

With the update operations and their corresponding timestamp in mind, next, we put forward some new types of leakage that provide a more detailed description of volumetric information in dynamic settings. The following are the details:

- **Update length pattern** denoted as ulen(w) = |Updates(w)| outputs the number of updates (i.e., add and del) made on keyword w.
- **Insert length pattern** denoted as ilen(w) = |Updates(w)| − |DelHist(w)| outputs the number of insertions (i.e., op = add) made on w.
- **Delete length pattern** denoted as dlen(w) = |DelHist(w)| outputs the number of deletions (i.e., op = del) made on w.
- **Response length pattern** (aka result length pattern) denoted as rlen(w) = |TimeDB(w)| outputs the number of files matching keyword w.
- **File volume pattern** denoted as fvol(w) = $\{\#D_{\text{id}} | \text{id} \in$ TimeDB(w)\} outputs the volume of each file containing w. Usually, we measure the volume by the number of keywords each file contains.
- **Response similarity pattern** denoted as rsp(w_i, w_j) outputs the similarity between two query responses. Here, we measure the similarity by the extended Jaccard similarity coefficient for collections, i.e., rsp(w_i, w_j) = #(fvol(w_i) ⋒ fvol(w_j))/#(fvol(w_i) ⋓ fvol(w_j)).
- **Query equality pattern** (aka search pattern) denoted as qeq(w_i, w_j) = $\mathbf{1}(w_i = w_j)$ indicates whether two queries are targeting the same keyword. $\mathbf{1}(\cdot)$ is the indicator function outputting 1 if the input evaluates to true and 0 otherwise.

As seen above, we first link the volumetric information with specific update operations and incurring timestamps and define the insertion length patterns, update length patterns and delete length patterns. Besides, we also introduce a new notion termed response similarity pattern, which is used to quantify the similarity of the query responses. Later, we will show that these refined leakage patterns will play a critical role in evaluating the security of the forward and backward DSSE schemes.

4.2 Volumetric Leakages in BP-DSSE

The above section recalls the definition and the difference between the three-level of backward security in terms of access patterns and timestamps. In this section, we will revisit the above three tuples from the volumetric aspect and interpret them from a new perspective. The reason why we are doing this is that a line of volume attacks have been proposed in the last few years and they are shown to cause devastating damage to existing searchable encryption schemes. And we find that these damages are getting worse in dynamic scenarios, because the volume information changes over time (as seen in Figure 4), and the adversary can use therein difference to help recover the query further.
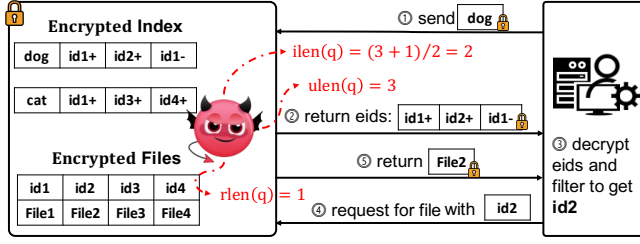
Figure 2: Example of extracting volumetric leakages from $B(\Sigma)$ **or** $B(O)$ **[7]**

Combined with the definition listed in Section 3, we now show the volumetric leakage in different levels of backward private notions. Here we pay our attention to four types of volumetric information, which are defined as follows:

- n_w - the number of files matching w
- u_w - the total number of updates on w
- a_w - the number of insertions on w
- d_w - the number of deletions on w.

Observe that since the total number of updates is the sum of insertions and deletions, and the response length is the difference between insertions and deletions, thus we have the following equations, i.e.,

$$\begin{cases} n_w = a_w - d_w \\ u_w = a_w + d_w \end{cases} \tag{1}$$

Then if one has any two of these four values, the equation can be transformed into a system of two binary linear equations. Solving this equation system can obtain the other two values. In all three levels of backward privacy notions, we have observed that each of them leaks two of the aforementioned values in their search procedures. Specifically,

- Type-I backward private constructions disclose volumetric information u_w and $n_w = |\text{TimeDB}(w)|$,
- Type-II backward private constructions reveal volumetric information $n_w = |\text{TimeDB}(w)|$ and $u_w = |\text{Updates}(w)|$,
- Type-III backward private constructions leak volumetric information $n_w = |\text{TimeDB}(w)|$ and $d_w = |\text{DelHist}(w)|$.

This implies that all three-level backward privacy notions share the same volumetric leakage. For example, for the Type-I backward private constructions revealing n_w and u_w, we can obtain

$$\begin{cases} n_w = a_w - d_w \\ u_w = a_w + d_w \end{cases} \implies \begin{cases} a_w = (u_w + n_w)/2 \\ d_w = (u_w - n_w)/2 \end{cases}. \tag{2}$$

Similarly, we can get all four types of volumetric information for the other two levels of backward private construction. To this end, once an attacker possesses one of the above four types of background knowledge, it can use the corresponding leakage to infer the information of the query. It could be noted that the ways to derive the above leakages differ in different constructions. Following is an example to show how to derive the above volumetric information in a classical backward private DSSE construction proposed by Bost and Fouque [6].

Example: To clearly present these leakages, we take Bost et al.'s generic two-roundtrip solution [7] as an example to show how

to extract volumetric leakage from the constructions. It should be noted that their solutions can provide at least Type II BP and will achieve Type-I backward privacy if ORAM or TWORAM techniques are adopted. For the purposes of presentation, we refer to the generic solution as $B(\Sigma)$ and the ORAM-based solution as $B(O)$. Since the ORAM also does not protect volumetric information, we choose the former as an example. We first review the main idea of it. The diagram in Figure 2 shows the index structure and query operation in $B(\Sigma)$. The client builds the encrypted index for the tuple $(w, \text{id}\|\text{op})$ from an arbitrary SSE scheme. Note that, here a label op (aka "+", "−" in the figure) is attached to the file identifier to remark the operation to the tuple is addition or deletion.

Combined with Figure 2, we now show how to obtain the above leakages. In the query stage, one can retrieve the encrypted files containing the given keyword by a two-roundtrip interaction. Firstly, the client uses the generated token to fetch the encrypted indices that match the query with the help of the server. In this round, the total number of updates on the keyword ulen = 3 can be observed. Once these indices are obtained, the client executes the decrypt operation locally and filters out those which have not been deleted before. Then it sends these undeleted indices to the server, and the server returns the corresponding encrypted files to the client. This procedure discloses the actual number of files in the results, i.e., rlen = 1. According to Equation (2), it is easy to know that the real number of files containing the given keyword injected before is 2.

4.3 Query Equality Pattern

Another leakage pattern we want to revisit is the query equality pattern, which is the key to computing the query frequency pattern. In previous works, one can always obtain the query equality pattern by checking whether they keep the same search token. While in the context of forward and backward private DSSE, it does not work because the search token in most schemes will be refreshed upon the update operations. Nevertheless, this does not mean that forward and backward private DSSE will not leak query equality patterns. As pointed out by Kammar [21], if an adversary observes that some query q_1 has the largest volume v, then observes an edit operation, and observes that some query q_2 now has volume $v + 1$, it can deduce that $\text{qeq}(q_1, q_2) = 1$ and the query equality is leaked. The following parts provide more systematic thoughts on rebuilding the above query equality pattern in forward and backward private DSSE schemes.

Generally, there is no "one-size-fits-all" approach to determine the qeq due to the different instantiation principles of FB-DSSE schemes and therein query status. By surveying existing constructions, we find that one can rebuild the query equality pattern through these two aspects: 1) *the relations of tokens*; 2) *the similarity of responses*.

For the former one, throughout recent forward and backward construction, we find that the relations of tokens with respect to the same keyword can be classified into the following four categories:

- *exactly same* means that the tokens with respect to the same queried keyword are the same.
- *partially same* means that tokens with respect to the same keyword share some same components.

Table 2: Leakage of representative FP/BP-DSSE schemes

Scheme	qeq[2]	rlen	ulen	ilen	dlen	fvol	Security
B(O) [7][3]	$\overset{?}{\approx}$	●	●	●	●	●	
Moneta [7]	$\overset{?}{\approx}$	●	●	●	●	●	Type I
Orion [11]	$\overset{?}{\approx}$	●	○	○	○	●	
B(Σ) [7][4]	$\equiv$	●	●	●	●	●	
Fides [7]	$\overset{C}{=}$	●	●	●	●	●	
Aura [61]	$\overset{\mathcal{P}}{=}$	●	●	●	●	●	Type II
Mitra [11]	$\overset{\mathcal{P}}{=}$	●	●	●	●	●	
2ch$_{FB}$ [1]	$\overset{\mathcal{P}}{=}$	○	○	○	○	●	
Janus [7]	$\overset{\mathcal{P}}{=}$	●	●	●	●	●	
Janus++ [62]	$\overset{\mathcal{P}}{=}$	●	●	●	●	●	Type III
OSSE [10]	$\overset{\mathcal{P}}{=}$	●	●		●	●	
LLSE [10]	$\overset{\mathcal{P}}{=}$	●	●	●	●	●	

Column headers: Leakage Pattern[1] (qeq[2], rlen, ulen, ilen, dlen, fvol) and Security.

[1] The listed leakage patterns are defined when these constructions are used in file-retrieval applications, where ● denotes that the listed scheme exposes the corresponding leakage pattern, and ○ signifies that it does not;

[2] $\overset{?}{\approx}$ denotes that tokens are irrelevant but we can infer qeq with their response similarity. $\equiv, \overset{C}{=}, \overset{\mathcal{P}}{=}$ refer to exactly same, computationally same, and partially same, respectively, which describes the relations of tokens for the same keyword in listed constructions;

[3] The generic backward-private scheme initiated with ORAM;

[4] The generic two-roundtrip backward-private scheme initiated with an arbitrary SSE scheme.

- *computationally same* means that one token can be computed by that of the later one.
- *irrelevant* indicates that no correlation can be found between the two queries.

With these relations, one can determine qeq of the queries by checking whether their tokens satisfy one of the above first three conditions. If yes, we have qeq = 1, and 0 otherwise.

Table 2 summarizes the leakage patterns revealed by selected FP/BP-DSSE constructions and how they expose the query equality pattern. As seen, B(Σ) [5] uses the same token for the same keyword and encrypts them to prevent the server from learning indices. The plaintext indices can only be obtained and processed by the client. Thus, the server cannot get any sensitive information about updates without the client. A typical example of partially same tokens is Sun et al. [62]'s scheme, whose tokens with respect to the same keywords are partially same. Tokens in this scheme are in the form of $T_w = (F(K, w), msk)$, where $F(K, w)$ is the same with respect to the same keyword. They keep the freshness of tokens by updating msk after each query to guarantee forward privacy.

Another way we give through query response is proposed for schemes (like Moneta [7]) that do not leak the relations of the tokens. In this case, the client always deletes the indices searched before and rebuilds a new one. To deal with this, we usually leverage the similarity between the responses to build the equality pattern. For robustness, here we introduce another way of using file volume patterns to compute the response similarity. Specifically, for any

two queries q and q' with file volume leakage fvol_q and $\mathrm{fvol}_{q'}$, we denote their response similarity as

$$\mathrm{rsp}(q, q') = \#(\mathrm{fvol}_q \cap \mathrm{fvol}_{q'})/\#(\mathrm{fvol}_q \cup \mathrm{fvol}_{q'}). \qquad (3)$$

Based on the fact that queries for the same information usually have similar responses, here we suggest estimating the query equality pattern by using a threshold method. Specifically, we define

$$\mathrm{qeq}(q, q') = \begin{cases} 1, & \mathrm{rsp}(q, q') \geq \delta \\ 0, & \mathrm{rsp}(q, q') < \delta \end{cases} \qquad (4)$$

where the threshold is characterized by δ, which can be inferred from the growth rate of data volume. Specifically, we set $\delta = \min\{|DB_{\mu_i}|/|DB_{\mu_j}|, |DB_{\mu_j}|/|DB_{\mu_i}|\}$ in this work, where $|DB_{\mu_i}|$ denotes the size of dataset at time μ_i. In reality, the volume of updated data is relatively small compared to the entire database, thus we can set $\delta > 0.5$ in this work. The same setting will be adopted in our experiment.

4.4 Attack Model and Overview

In Section 2, we have shown that the potential risk brought by leakage to static SSE has been confirmed by many works. Does this still hold for dynamic SSE (with forward and backward security)? Before giving a formal answer, we first overview the attack model. **Targets and Attack Settings.** Differing from the previous attacks on DSSE schemes, we focus on **passive attacks**. The attacker captures certain metadata (also known as auxiliary information) of encrypted datasets, such as data distribution, query frequency, etc. He is allowed to monitor the transcripts between the client and the server and exploits his background knowledge (e.g., distribution, statistics) about the updates. Like most of the existing LAAs, our goal is to recover the observed queries by the disclosed leakage.

In practice, it is rational to consider the attacker can access the background knowledge of updates. For instance, in the context of the COVID-19 pandemic, statistics like daily increases in the number of new cases can be obtained from social media. This allows attackers to gain volumetric information related to updates, even if the underlying database of medical records is encrypted. Geospatial datasets could also serve as an example, as the location information is often publicly available. For instance, the US Places of Worship dataset records all places of worship across states. The data classifies meters by the restriction type and includes location descriptions. Additionally, if one adopts SSE techniques to build encrypted databases, then the transaction log of the underlying database engine would reveal information such as when and how encrypted files are updated w.r.t. which (deterministically) encrypted keywords. There is indeed work [29] about how to exploit transaction logs against SSEs, while the attacking target is not forward/backward private.

Attack Models. Based on different attack settings, auxiliary data needed, and leakage profile exploited, LAAs can be classified into different categories. In this paper, we focus on the following two common types of attacks:

- **Frequency Matching Attack (FMA).** Given a series of queries and query distribution in the period that encompasses all target queries, FMA utilizes the equality pattern to recover the query information. The key to this attack is
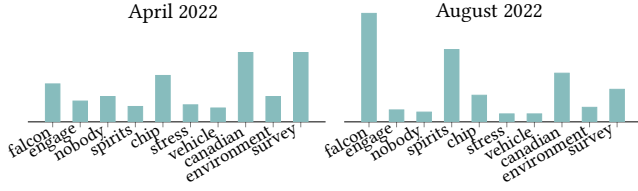
Figure 3: Query distribution of 10 randomly selected keywords in April and August 2022

to utilize exposed leakages to calculate the query equality pattern as well as the query frequency.
- **Volumetric Inference Attack (VIA).** Given a series of queries, VIA exploits volumetric leakage to recover the query. To perform the attack, the attacker must possess some prior knowledge of the encrypted dataset, such as a fraction of plaintext data collection or its distribution. This prior knowledge can be the dataset before the time μ_0 or the updated information between two linked queries that have the same underlying keyword.

From the target of the attack, both two attacks discussed are commonly referred to as query recovery attacks. In the subsequent sections, we will comprehensively validate our observations of forward and backward DSSEs by presenting the above-mentioned two types of attacks. We would like to clarify that these attacks work with leakages derived from FP/BP notions rather than specific constructions. We acknowledge that some schemes provide stronger security guarantees than FP/BP. We will thoroughly present the scope and details of each attack in the following sections.

5 FREQUENCY MATCHING ATTACK

We start our work with the FMA attack, which matches the collected query frequencies with the known query distribution for query recovery. The key to this attack is to build the query equality for observed queries, and then the frequency of each query performed can be calculated. Intuitively, the success of FMA significantly hinges on the diversity of the query frequency. Affected by factors such as seasonality, trending topics, news events, day of the week, and month of the year [23], the query distribution usually changes over time, providing necessary diversity (as seen in Figure 3). This diversity can be leveraged to identify the query further by comparing the reconstruction spaces and computing their intersection in multiple rounds. More details are offered below.

Attack Assumption. Let $q = \{q_1, \ldots, q_t\}$ be observed queries and $\mathcal{D} = \{d_{[t_i, t_{i+1}]} \sim (p_{w_1}^{(i)}, \cdots, p_{w_{|W|}}^{(i)})\}_{i=1}^{\tau}$ be distribution of queries over $\mathcal{W}$ in different time intervals. In FMA, we consider a passive and persistent attacker who is additionally aware of the auxiliary information of the keyword frequency requested. The goal of FMA is to recover the true keywords that generated the query sequence $q = \{q_1, \ldots, q_t\}$.

Attack Description. As mentioned, the fundamental principle of FMA includes identifying the query equality of these queries, calculating query frequency, and comparing them with the provided distribution to recover queries.

In accordance with this principle, our first step is to leverage the approach provided in Section 4.3 to build the query equality for

queries within the same time interval (lines 19-25). Once the query equality pattern is built, we put queries with the query equality pattern of 1 in the same group. For example, given query sequences $\tilde{q}$ with query distribution $\tilde{d} \sim (\tilde{p}_{w_1}^{(i)}, \cdots, \tilde{p}_{w_{|W|}}^{(i)})$ in a certain time interval, if $\mathrm{qeq}(\tilde{q}_i, \tilde{q}_j) = 1$, $\tilde{q}_i$ and $\tilde{q}_j$ will be grouped together. Once the groups, denoted by $\mathcal{G} = \{G\}$, are generated, our second step is to build the candidate set for each group (lines 26-28). In fact, the query frequency of the keyword issuing in a group is the size of that group. According to this, keywords whose requested frequencies are closest to the size of G will be assigned as the candidates for queries in G. Namely, for $\tilde{q} \in G$, we have

$$\mathrm{MM}[q] = \arg \min_{w \in \mathcal{W}} |\#(G) - \#(\tilde{q}) \cdot \tilde{p}_w|$$

where MM is the multi-map from the query to its candidates.

By applying the above procedure to query sequences at different time intervals, a candidate set can be generated for each query (lines 1-9). In this process, the query keyword can be determined if only one element remains in the set (lines 5-8). For queries that do not meet this condition, it continues to narrow down candidates across different time intervals. Specifically, we select queries related to the same keyword in different time intervals based on their query equality and compute the intersection of their candidate set (line 11). If the candidate set remains only one element, it is the output as the information for the query. The pseudo-code of FMA is detailed in the Algorithm 1.

Note that, FMA is effective to forward and backward DSSE schemes that hide responses or re-encrypt indices after the query. This is because refreshing the query response does not alter file volume information, thus preserving the query equality information.

6 VOLUMETRIC INFERENCE ATTACK

The second attack is the volumetric inference attack (VIA) which exploits volumetric information to recover the query. For attack practicality, we will consider attackers with different levels of knowledge.

6.1 VIA: Volumetric Inference Attack

We initiate the attack from a baseline assumption analog to that of existing LAAs where the attacker has knowledge of the added files between two linked queries that have the same underlying keyword [3, 8]. Studying the vulnerability of VIA under this assumption is of theoretical significance in evaluating the strength of the proposed scheme against a powerful adversary. We then gradually constrain this assumption to design more practical attacks.

Attack Assumption. Let q denote a query performed on an encrypted database EDB that the attacker is targeting. Let $\tilde{q}$ be a query requesting the same keyword after q. According to the analysis in Section 4.3, such a $\tilde{q}$ can always be identified if it exists. Let $\overline{D}$ be the file collections inserted into EDB between q and $\tilde{q}$, which is known to the attacker. Suppose a sequence of operations is performed on $\overline{D}$ by the client, including additions that add all files from $\overline{D}$ into EDB and deletions that remove some of the existing entries from EDB. Here, we follow a slightly different assumption compared to existing works [3, 30]: the attacker knows $\overline{D}$ are injected files, but not whether they are deleted in later operations. As mentioned, in VIA, an attacker is assumed to be able to monitor the transcripts between the client and the server. Here we take the constructions

Algorithm 1: Frequency Matching Attack

Input: Encrypted database EDB, keyword space $\mathcal{W}$, a query list $\boldsymbol{q} = \{q_1, \ldots, q_t\}$, and query distribution in different time intervals $\mathcal{D} = \{\boldsymbol{d}_{[t_i, t_{i+1}]} \sim (p_{w_1}^{(i)}, \cdots, p_{w_{|\mathcal{W}|}}^{(i)})\}_{i=1}^{\tau}$.

Output: A map M from $\boldsymbol{q}$ to $w \in \mathcal{W}$.

1 Initialize an empty map M and an empty multi-map MM*
2 Partition $\boldsymbol{q}$ to $\boldsymbol{q}^{(1)}, \cdots, \boldsymbol{q}^{(\tau)}$ according to $\{t_i, t_{i+1}\}_{i \in [\tau]}$
3 **for** query sequences $\boldsymbol{q}^{(i)}$ incur in $[t_i, t_{i+1}]$ **do**
4 $\quad$ Run MM$^{(i)} \leftarrow$ Candidate_Gen$(f, \boldsymbol{q}^{(i)})$
5 $\quad$ **if** MM$^{(i)}[q] = 1$ **then**
6 $\quad\quad$ Set M$[q] \leftarrow$ MM$^{(i)}[q]$ and remove q from $\boldsymbol{q}$
7 $\quad\quad$ Remove MM$^{(i)}[q]$ from MM$^{(i)}$
8 $\quad$ **end**
9 **end**
10 **for** remained $q \in \boldsymbol{q}$ **do**
11 $\quad$ Set MM$^*[q] = \bigcap_{q' \in \boldsymbol{q}}$ MM$^{(i)}[q']$ where qeq$(q, q') = 1$
12 $\quad$ **if** $|$MM$^*[q]| = 1$ **then**
13 $\quad\quad$ Add all $(q',$ MM$^*[q])$ to M* where qeq$(q, q') = 1$
14 $\quad$ **end**
15 **end**
16 **return** query-keyword maps M
17 **Function** Candidate_Gen$(\tilde{\boldsymbol{d}}, \tilde{\boldsymbol{q}})$:
18 $\quad$ Initialize an empty multi-map MM and an initial cluster $\mathcal{G} = \{G_1 = \{\tilde{q}_1\}\}$
19 $\quad$ **for** each $\tilde{q}_i \in \tilde{\boldsymbol{q}}$ **do**
20 $\quad\quad$ **if** $\exists \tilde{q}_j \in G_k, G_k \in \mathcal{G}$ s.t. qeq$($EDB$, \tilde{q}_i, \tilde{q}_j) = 1$ **then**
21 $\quad\quad\quad$ Add $\tilde{q}_i$ into G_k **else**
22 $\quad\quad\quad\quad$ Create a new group $G_{|\mathcal{G}+1|} = \{\tilde{q}_j\}$
23 $\quad\quad\quad$ **end**
24 $\quad\quad$ **end**
25 $\quad$ **end**
26 $\quad$ **for** each $\tilde{q} \in G \in \mathcal{G}$ **do**
27 $\quad\quad$ Set MM$[\tilde{q}] \leftarrow \arg\min_{w \in \mathcal{W}} |\#(G) - \#(\tilde{\boldsymbol{q}})\tilde{p}_w|$
28 $\quad$ **end**
29 $\quad$ **return** query-keyword multi-maps MM

Algorithm 2: Toy Volumetric Inference Attack

Input: Encrypted database EDB, two linked queries $(q, \tilde{q})$, a multi-map $V = \{(w, \{\#f\}_{f \in \overline{D}[w]})\}$, where $\overline{D}$ is the file collection inserted between q and $\tilde{q}$, a map $L = \{(w, |\overline{D}[w]|)\}$

Output: The goal is to recover $\boldsymbol{q}$.

1 Count $u_q \leftarrow$ ulen$($EDB$, q)$ and $n_q \leftarrow$ rlen$($EDB$, q)$
2 Count $u_{\tilde{q}} \leftarrow$ ulen$($EDB$, \tilde{q})$ and $n_{\tilde{q}} \leftarrow$ rlen$($EDB$, \tilde{q})$
3 Compute $a_q = (u_q + n_q)/2$ and $a_{\tilde{q}} = (u_{\tilde{q}} + n_{\tilde{q}})/2$
4 Set $C_q \leftarrow \{w : L[w] = a_{\tilde{q}} - a_q\}$ as the candidates
5 **if** $\#(C_q) = 1$ **then**
6 $\quad$ Map q to $w \in C_q$
7 **else**
8 $\quad$ Initialize a collection $\boldsymbol{v} \leftarrow$ fvol$[$EDB$, \tilde{q}]/$fvol$[$EDB$, q]$
9 $\quad$ **for** each $w \in C_q$ **do**
10 $\quad\quad$ remove w from C_q if $\exists v \in \boldsymbol{v}$ s.t. $v \notin V[w]$
11 $\quad$ **end**
12 **end**
13 Map q to $w \in C_q$ if $\#C_q = 1$. Otherwise, return $\bot$

Now, if there is only one keyword in the candidate set, the attacker directly returns it as the output of the query (lines 5-6). Otherwise, the attacker further proceeds with the candidates by comparing the file volumes, removing the keyword whose corresponding file volumes do not include the response file volumes from the candidate set (lines 7-11). More precisely, it narrows down the candidates to

$$C_q = C_q / \{w : \exists v \in \text{fvol}[\text{EDB}, \tilde{q}]/\text{fvol}[\text{EDB}, q], \text{ s.t. } v \notin V[w]\}$$

where fvol$[$EDB$, q]$ and fvol$[$EDB$, \tilde{q}]$ are collections of file volumes returned by q and $\tilde{q}$. If only one candidate remains, the attacker maps it to q. Otherwise, return $\bot$ (line 13).

6.2 Attacks with Limited Prior Knowledge

VIA describes a baseline attack to recover the query from insert volume patterns and file volume patterns under the standard forward and backward notions. Generally, the assumption that all added files are known to the adversary is strong. Besides, if the cleartext is not a public dataset, an attacker might not be able to get the file volumes. For these constraints, following we limit the basic assumption step by step, making our attack more practical.

PVIA: Attacking with Partial Inserted Files. Firstly, we consider the *partially file known case* that only a fraction of files $\overline{D}^{\dagger} \subset \overline{D}$ are known to the adversary as most of existing works [3, 50]. Then the attack inputs become to $V[w] = \{\#f\}_{f \in \overline{D}^{\dagger}[w]}$ and $L[w] = |\overline{D}^{\dagger}[w]|$. With reference to recommendations in [3], we assume that known files are sampled uniformly from the entire file collection. The known file rate is parameterized by a value $\alpha \in \{0, 1\}$. Under this setting, attackers with a fraction of files cannot perform equality matching with revealed leakage because the information derived from the known files will be inconsistent with that of the query response. To solve this problem, we modify the strategy (line 4) to build the initial candidate set for query q as

$$C_q := \{w : \alpha a_q \leq L[w] \leq a_q\} \tag{5}$$

having Type-I backward privacy as an example to illustrate our attacks.

Attack Description. Let $(q, \tilde{q})$ be a pair of linked queries mentioned above, let $L[w] = |\overline{D}(w)|$ be a map that records the number of files containing w and $V[w] = \{\#f\}_{f \in \overline{D}[w]}$ be multi-maps storing the size of the file containing w. Here both L and V are known to the attacker. Then the attacker can run Algorithm 2 to recover q.

The first step is to extract the insert volume of the above two queries (lines 1-3). As all BP-DSSE constructions leak the update length pattern and response length pattern, for each pair of $q_i, \tilde{q}_i$, an attacker can learn their insert lengths

$$a_q = (u_q + n_q)/2 \text{ and } a_{\tilde{q}} = (u_{\tilde{q}} + n_{\tilde{q}})/2$$

where $u_q = $ ulen$($EDB$, q)$ and $u_{\tilde{q}} = $ ulen$($EDB$, \tilde{q})$ are update lengths on q and $\tilde{q}$ and $n_q = $ rlen$($EDB$, q)$ and $n_{\tilde{q}_i} = $ rlen$($EDB$, \tilde{q})$ are the response lengths. With the insert length, the attacker can calculate the insert length $a_{\tilde{q}} - a_q$ of the keyword between q and $\tilde{q}$ and use it to initialize the set of candidates C_q for q (line 4). Specifically, it defines
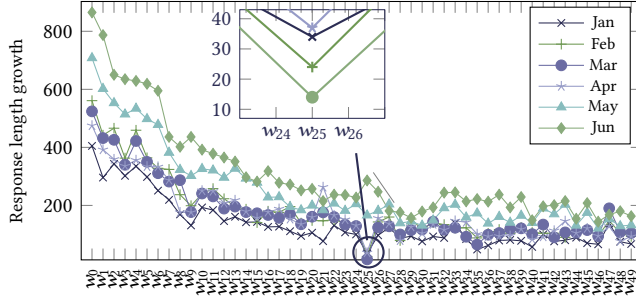
$$C_q = \{w : a_{\tilde{q}} - a_q = L[w]\}.$$

Figure 4: The growth of the number of files with respect to 50 randomly selected keywords in half a year

and choose the keyword

$$\arg \max_{w \in C_q} |(\text{fvol}[\text{EDB}, \tilde{q}]/\text{fvol}[\text{EDB}, q]) \cap V[w]|$$

as the attack result (line 13).

LVIA: Attacking with Data Distribution Only. To follow up on the above principle, we constrain the assumption further to the scenario where the attacker only captures the knowledge of data distribution which records the probability of a keyword appearing in a file. For ease of presentation, we directly parameterized this knowledge as the response length of each keyword in the dataset. Previous work [8] has demonstrated that the harm of response length to query privacy is mild, owing to the fact that only a small proportion of queries hold a unique response length. However, the above result was obtained under static conditions. As shown in Figure 4, the update frequency of different keywords varies significantly across different time intervals in dynamic settings.

Based on the above insight, we further optimize our attack by proceeding in rounds to reduce the number of candidates. Specifically, we can first obtain the candidate sets of the target $C_q^{(1)}, C_q^{(2)}, \cdots, C_q^{(n)}$ in different time intervals. Here, we choose the keyword candidates within a window of response length. The attack maps q to the keyword that appears most frequently in the above candidates, i.e.,

$$w = \arg \max_{w \in W} \left\{ |C_w| : C_w = \left\{ C_q^{(i)} \right\}_{w \in C_q^{(i)}} \right\},$$

where $\#C_w$ is the number of candidate sets containing w. Intuitively, large θ implies more elements in the candidate set and may require more rounds to select the correct candidate.

6.3 Attacks without Prior Knowledge of Updates

Previously, we have considered scenarios in which attackers possess prior knowledge of updates. Here we consider another scenario where they cannot touch such information but have access to some historical knowledge of the encrypted database. Formally, we assume that the attacker possesses prior knowledge of the encrypted dataset before time μ_0. This is a similar assumption as the one in LAAs against static SSE, where data breaches could happen previously or databases were not encrypted before. Following, we show that IVA can be adapted to this scenario after minor modification.

Similarly, the first step is to find a linked query $\tilde{q}$ to the target query q, which occurs closest and after time μ_0. In this case, the disclosed leakage from the query response of $\tilde{q}$ is closest to that

derived from the attacker's prior knowledge, and the goal becomes recovering $\tilde{q}$. Following this principle, we use the approaches mentioned before to check the query equality between q and previous queries. And if no such a $\tilde{q}$ can be found, we set $\tilde{q} = q$. Once the $\tilde{q}$ is determined, we initialize the candidates as

$$C_q = C_{\tilde{q}} = \{w : a_{\tilde{q}} \geq L[w]\},$$

where $a_{\tilde{q}}$ denotes the number of updates related to $\tilde{q}$ before $\tilde{q}$ happens. After that, we further narrow down the candidate set by leveraging the volume leakage and execute

$$C_q = C_q/\{w : \exists v \in \text{fvol}[\text{EDB}, \tilde{q}] \text{s.t. } v \notin V[w]\}$$

Likewise, if only one candidate remains, the attacker outputs it as the attack result. The abovementioned approach can also be leveraged to tailor the PVIA and LVIA to suit the setting in this section. However, the attack performance of LVIA may be significantly weakened because the diversity of data distribution in updates cannot be used.

7 EXPERIMENT EVALUATIONS

To validate our theoretical findings, we implement and evaluate the effectiveness of the proposed attacks under different attack settings.

7.1 Experiment Setup

As with prior work, we focus on measuring attack accuracy by the proportion of recovered queries. All experiments are conducted on a server running Ubuntu using Intel Xeon 5218R CPU@2.10GHZ with 188G RAM[1].

Experiment datasets. The test datasets are extracted from two commonly-used email systems, Enron [18] and Lucene [45]. We select 500, 1000, 2000, and 3000 most frequent words outside of the stemming as the keyword space and use these keywords and associated identifiers to build the encrypted database. For ease of presentation, we denote the encrypted database as En_i and Lu_i for Enron and Lucene datasets, respectively, where i refers to the number of selected keywords.

Update operation. Our experiments focus on two fundamental update operations: file addition and deletion. For additions, we simulate the dynamicity of the encrypted datasets by using emails from different time intervals. Specifically, if a query is generated in July 2012, the query response only includes the items that are inserted before that time. For deletions, we randomly delete existing emails with a predefined deletion ratio, where each file has the same probability of being deleted.

Auxiliary data. Like most existing works, we assume that the passive attacker learns some auxiliary data from the dataset. Here we consider the following three types of auxiliary data:

- **Query distribution.** It records how frequently a keyword is queried in a period of time. We derive the query distribution from Google Trend [22]. For dynamicity concerns, query distribution here is defined monthly.
- **Partial dataset.** A subset of files in the clear is assumed to be known by the attacker. This is the most commonly assumed information an attacker possesses in the literature on attacking an encrypted dataset.

---

[1]The code is publicly available at https://github.com/FB-Attack/FB-Attack

(a) T1 and T2 with different τ ($\lambda = 120K$)

(b) T1 and T2 with different datasets

(c) T1 and T2 with different λ ($\tau = 24$)

(d) T1 and T2 with different γ ($\tau = 24$)

(e) T2 with different τ and γ ($\gamma = 12K$)
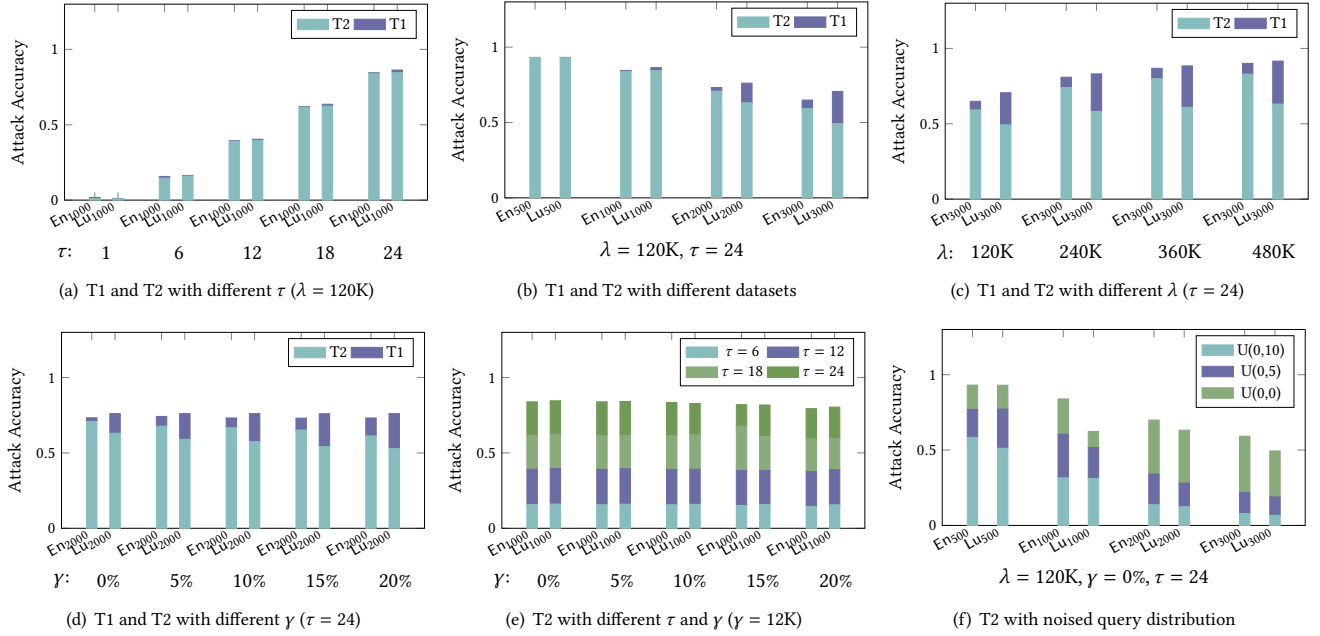
(f) T2 with noised query distribution

Figure 5: Attack results on query frequency matching attack. The backward private schemes that leak the query equality pattern are denoted as T1, while those that do not are denoted as T2. Let $\gamma \in \{0\%, 5\%, 10\%, 15\%, 20\%\}$ be the deletion rates and $\lambda \in \{120K, 240K, 360K, 480K\}$ be the number of queries issued in the total of query time intervals. Let $\tau \in \{1, 6, 12, 18, 24\}$ be the number of time intervals of query distributions given to the attacker. $U[0, n]$ implies the query distribution the attacker learned is perturbed by noise uniformly sampled from $[0, n]$. We use $U[0, 0]$ to denote the case where no noise is introduced.

- **Data distribution.** This statistical information characterizes the distribution of keywords across the entire dataset. Equivalently, it can be represented by the number of files related to each keyword.

7.2 Results on Frequency Matching Attacks

We now present our evaluations on query frequency matching attacks introduced in Section 5, which exploits the query frequency information to recover the query. In this experiment, the query distributions on keywords are known to the attacker. The detailed settings are given below.

Here, we consider two kinds of forward and backward private constructions, the schemes that leak the query equality pattern by relations of tokens are denoted as T1, while those that do not are denoted as T2. For the latter, the attacker must build the query equality by their response similarity. Without loss of generality, we set $\delta = 0.6$ here. To effectively model the dynamic nature of query distributions, we divide the entire query time into $\tau \in \{1, 6, 12, 18, \ldots, 24\}$ intervals, wherein queries within each interval adhere to a distinct distribution. Similar to Oya et al.'s work [51], we sample the number of queries in each interval from a Poisson distribution with parameter λ/τ, where $\lambda \in \{120K, 240K, 360K, 480K\}$ is the expected total number of queries; we also assume that the number of queries for each keyword in each time interval follows a multinomial distribution, with the probabilities derived from the query frequency of the corresponding time interval of Google Trends [22]. The deletion rate in the experiment is parameterized by $\gamma \in \{0\%, 5\%, 10\%, 15\%, 20\%\}$.

We start with a comparative experiment to show the advantage of the frequency matching attack, which exploits the diversity of query distribution in dynamic scenarios. By setting τ from 1 to 24, FMA gains a significant improvement in the recovery rate, as illustrated in Figure 5(a). For schemes of T1 type, the recovery rate increases from 1.8% to 84.5%, showing an increase of 82.7%. And for schemes of T2 type, we obtained a comparable outcome, thus highlighting the benefit of utilizing response similarity to link queries executed on identical keywords. Moreover, it is observed that the recovery rate for schemes of T2 type diminishes marginally as compared to that of T1. This is because rebuilding the query equality in T2 has a certain probability of being wrong.

Further, we repeat this experiment on different datasets to verify the above results. The results in Figure 5(b) show that the recovery rate drops to 70.6% for T1 and 49.3% for T2 when attacking Lu_{3000}. The main reason for this gap is that as the keyword space increases, the disparity in query frequency for each keyword diminishes, and in some cases, there is no difference at all, while the total number of queries remains unchanged. In addition, the average number of queries for each keyword is also reduced. As the number of queries is sampled, a small number of queries is insufficient to reflect the real-world query distribution. And the reason why the drop is more significant in T2 is that even for the identical keyword, their query responses can be significantly different in different stages due to frequent updates, which leads to incorrect query equality.

The above analysis shows that the number of observed queries may also influence attack performance. We continue examining the
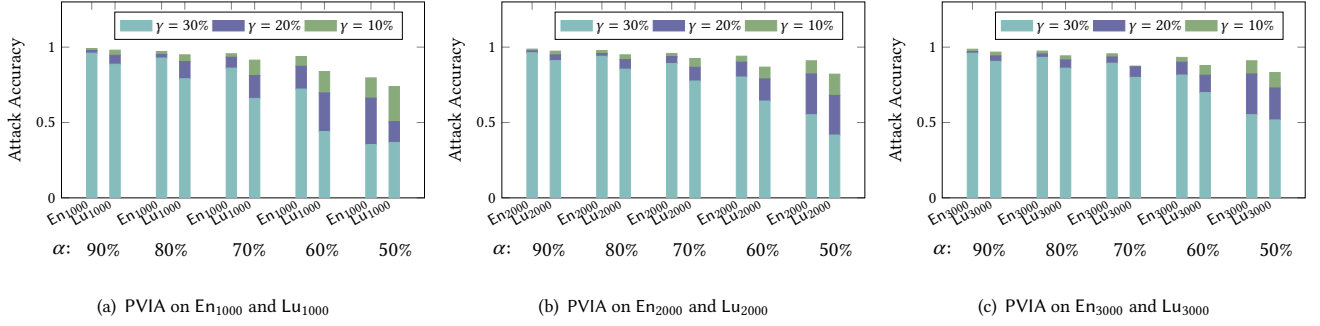
(a) PVIA on En_{1000} and Lu_{1000}

(b) PVIA on En_{2000} and Lu_{2000}

(c) PVIA on En_{3000} and Lu_{3000}

Figure 6: Attack results of PVIA **with different known file rates (α) and deletion rates (γ), where $\alpha = 90\%$ indicates that 10% files are removed from the selected dataset and set as the attack's input and $\gamma = 30\%$ means the client deletes 30% files from the encrypted dataset.**

recovery rate under varying numbers of queries. Figure 5(c) lists the recovery rates of attacks given different numbers of queries. The recovery rate of T1 goes from 64.9% to 90.1% when the query volume grows from 120K to 480K. This result tells us that an attacker needs to collect as many queries as possible to launch the frequency matching attack, because only a large number of queries can get close to the true distribution due to the central limit theorem (CLT).

As data deletion is allowed in DSSE constructions, we also evaluate the ability of our attack to cope with this situation. We randomly remove a fraction of files from the dataset based on a specified rate, denoted as γ. The rest parameters are $\lambda = 120K$ and $\tau = 24$. As depicted in Figure 5(d), the recovery rate of attacks on T1 schemes and En_{2000} remains stable at around 73.3%, with little variation. This is because the T1 schemes can establish query equality through the relations of query tokens, which will not affect the group results. While for T2 schemes, the recovery rate decreased from 70.9% to 61.3% with the delete rate increases. The main reason is that the delete operations further enlarge the difference of query response associated with the same keyword, leading to incorrect query equality. We conducted more experiments with various parameter settings to investigate the impact of data deletion. Figure 5(e) shows that quite a few queries can be recovered under these settings, suggesting that FMA remains highly effective in compromising query privacy even in the presence of data deletions.

In reality, the query distribution possessed by attackers may not be entirely accurate, which can also affect the effectiveness of the proposed attack. To evaluate the performance of our attack against this case, we perturb the query distribution by adding a noise that follows the uniform distribution $U(0, n)$ and take it as the attacker's input. Likewise, we set $\lambda = 120K$, $\tau = 24$ and $n \in \{5, 10\}$. In this setting, the average number of queries per keyword in a month will be 10 for En_{500} and less than 2 for En_{3000}. It is comparable to or even less than the magnitude of the noise. Unlike the above experiments, we cannot perform frequency matching to identify the potential candidates. Keywords whose query frequency is around (may not equal) that of their observations (group size) should be put into the candidate set. Figure 5(f) illustrates the attack results on datasets with various sizes. The results show that a certain number of queries can still be recovered under this weaker assumption, reflecting the vulnerability of DSSE in dynamic settings. For example, for

query distribution perturbed by noise from distribution $U[0, 10]$, the recovery rate of queried on dataset En_{500} is 58.4%.

7.3 Results on Volumetric Inference Attacks

In this section, we detail the valuation for several volumetric inference attacks elaborated in Section 6 which are all based on volumetric leakages. We run VIAs under different settings and collect the corresponding recovery rates.

PVIA **attack.** Recall that a PVIA attacker is assumed to have a subset of files in the clear and exploits response length, update length, and file size patterns. Response length and file size patterns could be obtained analogously as described in [30]. As for the update length pattern, we assume that each keyword will be queried after insertion in each time interval. This assumption allows for the convenient computation of the update length pattern in each time interval during our evaluations. Otherwise, an attacker could also leverage timestamp information to obtain the update length pattern alternatively. Namely, our assumption is only for simplicity in our evaluations and could be omitted in real-world scenarios. Suppose α and γ be the known file rate and the delete rate, respectively, then the accuracy of attacking different datasets with different file known rates and deletion rates are presented in Figure 6.

As seen in Figure 6(a), when fixing $\gamma = 30\%$, with α decreasing from 90% to 30%, the recovery rate on En_{1000} drops from 96% to 35.4%. On the other hand, when fixing $\alpha = 60\%$, as γ increases from 10% to 30%, the recovery rate reduces from 83.8% to 44%. The results show that both the known file rate and deletion rate significantly impact accuracy. Notably, this impact is achieved by affecting the volumetric information, particularly the length type information, including insert length and response length. While for the leakage of file volume, the unique combinations of file volumes can still be used to connect the query and keyword related to the same information. For example, if a file with a volume of 5 is returned for a query but the keyword "urgent" never appears in files with such a volume, we can remove "urgent" from the candidate set of the query. It remains worthwhile for us to validate the argument further. More attacks on other datasets can be seen in Figure 6(b) and Figure 6(c).

LVIA **attack.** The attack of LVIA is similar to VIA except that the auxiliary information captured by the adversary pertains to the
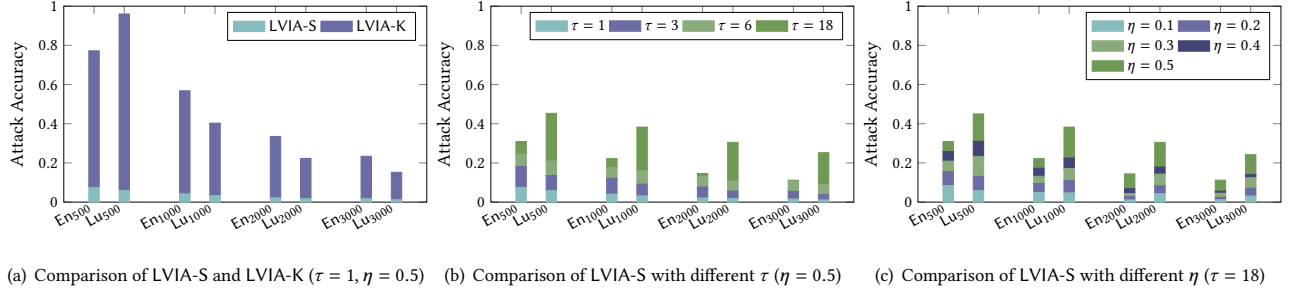
(a) Comparison of LVIA-S and LVIA-K ($\tau = 1, \eta = 0.5$)   (b) Comparison of LVIA-S with different τ ($\eta = 0.5$)   (c) Comparison of LVIA-S with different η ($\tau = 18$)

Figure 7: Attack results of LVIA with different attack settings. LVIA-S **and** LVIA-K **denote running** LVIA **is run in a sampled-data setting and known-data setting, respectively.** τ **denotes the number of groups (aka time intervals) that the dataset is partitioned, where** $\tau = 1$ **denotes the static case.** η **is the fraction of the dataset sampled to generate the data distribution.**

data distribution rather than partial files in the targeted dataset. Therefore, the only available information that can be utilized is the length of the query response. Regarding this, we use a similar dataset to generate auxiliary information in the following experiments. Specifically, we partition the Enron/Lucene dataset into τ groups based on the time intervals each document belongs to. Within each interval, we further divide the documents into two halves: half of the documents serve as the attack target (i.e., data to be updated by the client in each interval), while (a fraction of) the other half is used to generate auxiliary distributions for the adversary. Here, we also parameterize the number of intervals as τ and set $\tau \in \{1, 2, 3, 6, 18\}$, where $\tau = 1$ denotes the static case for prior passive LAAs. It is clear that increasing τ implies more refined distribution changes available to the adversary[2]. Based on this setting, we run a series of comparative experiments to investigate the influence of different attack settings on attack performance.

The first experiment studies performance differences of LVIA between the known-data and sampled-data settings, where the sampled-data setting (LVIA-S) follows the description above and the known-data setting (LVIA-K) assumes the adversary possesses the response length information of each keyword in the database. The results shown in Figure 7(a) indicate a notable degradation in the performance of the attack when using noised auxiliary distributions (i.e., sampled-data) compared to the exact auxiliary distributions. For instance, the recovery rate on En_{500} reduces from 77.2% to 7.5%, which is consistent with our insight given in Section 6.2 and related works [3]. Namely, the impact of response length on query privacy is relatively mild in real-world applications, especially when the background knowledge is not consistent with the derived leakage from the encrypted database.

The experimental findings above show that the length-type leakage is somewhat inadequate regarding attack effectiveness. Fortunately, this shortcoming can be effectively alleviated in dynamic scenarios. As mentioned, the data distributions in different time intervals are normally different, providing adversaries with more information to refine leakage patterns and enhance attack accuracy. Briefly, we can identify candidate sets for an encrypted query in different time intervals and then select the most probable one (i.e., the one with the highest frequency of occurrence) as the recovered plaintext keyword. In light of this observation, we repeat LVIA-S

---

[2]However, having an excessive number τ of groups implies fewer documents in the auxiliary dataset of each group, which results in more noise in the auxiliary dataset.

attack with $\tau \in \{1, 2, 3, 6, 18\}$ for Enron dataset from December 1st, 1999 to February 27th, 2001, and Lucene dataset from June 1st, 2006 to November 30th. As depicted in Figure 7(b), the recovery rate exhibits a notable increase from 5.9% (corresponding to the static case, $\tau = 1$) to 44.9% (corresponding to the dynamic case, $\tau = 18$) when conducting the attack on Lu_{500} as expected.

Finally, to examine the impact of data distribution quality on the attack accuracy, we utilize a fraction of the dataset (whose size is parameterized by the proportion η) as the auxiliary dataset to generate the data distribution. As illustrated in Figure 7(c), while maintaining a constant number of time intervals $\tau = 18$, the recovery rate of LVIA-S diminishes from 44.9% to 5.9% as η decreases from 0.5 to 0.1. This decline in accuracy occurs because the discrepancy between the data distributions obtained from the targeted dataset and the auxiliary dataset widens with the reduction of η.

7.4 Effectiveness on Prior Countermeasures

The above evaluations offer a comprehensive analysis of existing FP/BP-DSSEs to ascertain how to combine leakages across updates. This allows us to adapt existing LAAs targeting static settings and make them effective against FP/BP-DSSEs. At this point, one may wonder if the above attacks can withstand any countermeasures similar to other LAAs. Indeed, the attacks presented in our work serve as an illustrative example of how to conduct passive attacks in FP/BP settings. Consequently, existing attacks (e.g., SAP [51], IHOP [52], etc.) could be adapted by leveraging our proposed leakage framework to implement LAAs in dynamic settings. Therefore, the ability to withstand countermeasures in dynamic settings relies on whether the static LAAs, to be adapted to dynamic settings, can also withstand the same countermeasures. If the countermeasure is exceedingly strong, such as the almost zero-leakage dynamic construction proposed by George et al. [21], capable of thwarting all existing attacks, our attack will also fail. If the design protects only volume patterns [1], then frequency-based attacks would definitely work. Generally, one can transform strong attacks in static settings (e.g., SAP and IHOP) via our leakage framework for dynamic settings, and those transformed attacks (targeting specific dynamic constructions) will still be effective if no countermeasures are adopted correspondingly. In our experiments, we have evaluated our attacks w.r.t various leakage patterns, which implies the (in)effectiveness of different countermeasures, such as volume-hiding dynamic constructions [1] and others.

8 DISCUSSION ON COUNTERMEASURES

Most recently, researchers have been aware of the importance of suppressing leakages in both update and search processes and made efforts to investigate effective techniques to suppress aforementioned leakages [1, 21, 70] in dynamic scenarios. Intuitively, one can use both Oblivious RAM (ORAM)-like techniques and padding approaches to solve the above problem. ORAM-like techniques can be employed to conceal the search pattern and access pattern, whereas padding approaches can be used to eliminate the volume pattern. Unfortunately, the above techniques incur heavy communication, computation, or storage cost. Specifically, ORAM introduces a polylogarithmic search overhead, while worst-case padding can result in worst-case linear search time or quadratic index size [15].

The above challenge motivates researchers to seek new efficient solutions. In a recent study, George et al. [21] extend their concept of (almost) zero leakage security from static to dynamic scenarios and proposed three new dynamic structure encryption schemes that could provide asymptotically better performance than using black-box ORAM simulation. The first two proposed solutions prioritize efficiency over achieving zero-leakage or perfect correctness, whereas the last solution addresses all the aforementioned issues at the cost of increased complexity and computational overhead. Aside from the above work, Amjad et al. [1] focus on hiding volume pattern for forward and backward private constructions. Their work tolerates the query equality pattern to obtain further efficiency improvement, implying that their work is only suitable for applications that access each datum uniformly.

For security concerns, a simple and straightforward approach to address the problem stated in our work is to apply George et al. [21]'s third construction here. However, all the above schemes are based on encrypted multi-maps (EMM). In prior works for file retrieving, EMMs typically refer to the inverted index, which aids in storing identifiers (value) and the corresponding keyword (label) [12]. Therefore, the above solution only hides the leakages from one component of the overall DSSE scheme, which is the encrypted search index, and similar leakage patterns that occurred in the file retrieval phase can still be exploited to recover encrypted queries. The same insight has also been shown by Gui et al. [30] in the context of static settings.

To solve this problem, one possible solution is to substitute the stored values with the encrypted files instead of the identifiers in construction [21]. Note that in this approach, one also needs to do padding for files such that all share the same volume to hide file volume information. It is not hard to see that such an approach incurs at least $O(|\mathcal{W}| \max_{w \in \mathcal{W}} |\mathrm{DB}[w]| \cdot \max_{f \in \mathbf{D}} \#(f))$ storage cost and $O(\max_{w \in \mathcal{W}} |\mathrm{DB}[w]| \cdot \max_{f \in \mathbf{D}} \#(f))$ communication cost per query. For instance, let's consider using the Enron dataset as an example to create a multi-map for a dataset consisting of 30,204 emails and 73,991 keywords. In this case, the total storage overhead would increase by about $10^7 \times$. Regarding this matter, finding a feasible approach that deserves to be deployed may still be a long-term endeavor.

9 CONCLUSION

In this paper, we revisit existing forward and backward private DSSE schemes and systematically analyze their limitations with regard to leakage-abuse attacks. By refining leakages, specifically for volumetric information of updates and queries and query equality, it is demonstrated that DSSE schemes supporting forward and backward privacy, are still vulnerable to LAAs. The considerable recovery rates underscore the need for solutions that completely thwart query equality leakage and protect volumetric leakage during updates, queries, and file retrieval. By connecting to known results of prior arts, we observe that the solutions that can fully resolve the privacy problem in DSSE encounter severe performance issues, while the efficient solutions are still susceptible to real-world attacks. We expect that this work will deepen the understanding of the real-world security strength of forward and backward private DSSE schemes and shed light on the future design space for safeguarding DSSE schemes.

REFERENCES

[1] Ghous Amjad, Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. 2023. Dynamic Volume-Hiding Encrypted Multi-Maps with Applications to Searchable Encryption. In *Proc. of PETS*.
[2] Vincent Bindschaedler, Paul Grubbs, David Cash, Thomas Ristenpart, and Vitaly Shmatikov. 2018. The Tao of Inference in Privacy-Protected Databases. *PVLDB* 11, 11 (2018), 1715–1728.
[3] Laura Blackstone, Seny Kamara, and Tarik Moataz. 2020. Revisiting Leakage Abuse Attacks. In *Proc. of NDSS*.
[4] Angèle Bossuat, Raphael Bost, Pierre-Alain Fouque, Brice Minaud, and Michael Reichle. 2021. SSE and SSD: page-efficient searchable symmetric encryption. In *Proc. of CRYPTO*.
[5] Raphael Bost. 2016. ∑οφος: Forward Secure Searchable Encryption. In *Proc. of the ACM CCS*.
[6] Raphael Bost and Pierre-Alain Fouque. 2017. Thwarting Leakage Abuse Attacks against Searchable Encryption – A Formal Approach and Applications to Database Padding. Cryptology ePrint Archive.
[7] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. 2017. Forward and Backward Private Searchable Encryption from Constrained Cryptographic Primitives. In *Proc. of ACM CCS*.
[8] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. 2015. Leakage-Abuse Attacks Against Searchable Encryption. In *Proc. of ACM CCS*.
[9] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. 2014. Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation. In *Proc. of NDSS*.
[10] Javad Ghareh Chamani, Dimitrios Papadopoulos, Mohammadamin Karbasforushan, and Ioannis Demertzis. 2022. Dynamic Searchable Encryption with Optimal Search in the Presence of Deletions. In *Proc. of USENIX Security*.
[11] Javad Ghareh Chamani, Dimitrios Papadopoulos, Charalampos Papamanthou, and Rasool Jalili. 2018. New Constructions for Forward and Backward Private Symmetric Searchable Encryption. In *Proc. of ACM CCS*.
[12] Melissa Chase and Seny Kamara. 2010. Structured Encryption and Controlled Disclosure. In *Proc. of ASIACRYPT*.

[13] Guoxing Chen, Ten-Hwang Lai, Michael K. Reiter, and Yinqian Zhang. 2018. Differentially Private Access Patterns for Searchable Symmetric Encryption. In *Proc. of IEEE INFOCOM*.

[14] Marc Damie, Florian Hahn, and Andreas Peter. 2021. A Highly Accurate Query-Recovery Attack against Searchable Encryption using Non-Indexed Documents. In *Proc. of USENIX Security*.

[15] Ioannis Demertzis, Dimitrios Papadopoulos, Charalampos Papamanthou, and Saurabh Shintre. 2020. SEAL: Attack Mitigation for Encrypted Databases via Adjustable Leakage. In *Proc. of Usenix Security*.

[16] Ioannis Demertzis, Stavros Papadopoulos, Odysseas Papapetrou, Antonios Deligiannakis, and Minos N. Garofalakis. 2016. Practical Private Range Search Revisited. In *Proc. of ACM SIGMOD*.

[17] F. Betül Durak, Thomas M. DuBuisson, and David Cash. 2016. What Else is Revealed by Order-Revealing Encryption?. In *Pro. of ACM CCS*.

[18] Enron Email Dataset. 2015. Online at: https://www.cs.cmu.edu/~./enron.

[19] Francesca Falzon, Evangelia Anna Markatou, Akshima, David Cash, Adam Rivkin, Jesse Stern, and Roberto Tassia. 2020. Full Database Reconstruction in Two Dimensions. In *Proc. of ACM CCS*.

[20] Sanjam Garg, Payman Mohassel, and Charalampos Papamanthou. 2016. TWORAM: Efficient Oblivious RAM in Two Rounds with Applications to Searchable Encryption. In *Proc. of CRYPTO*.

[21] Marilyn George, Seny Kamara, and Tarik Moataz. 2021. Structured Encryption and Dynamic Leakage Suppression. In *Proc. of EUROCRYPT*.

[22] Google. 2022. Google Trends. https://trends.google.com/trends/?geo=HK.

[23] Google. 2023. FAQ about Google Trends data. https://support.google.com/trends/answer/4365533?hl=en.

[24] Paul Grubbs, Anurag Khandelwal, Marie-Sarah Lacharité, Lloyd Brown, Lucy Li, Rachit Agarwal, and Thomas Ristenpart. 2020. Pancake: Frequency Smoothing for Encrypted Data Stores. In *Proc. of USENIX Security*.

[25] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. 2018. Pump up the Volume: Practical Database Reconstruction from Volume Leakage on Range Queries. In *Proc. of ACM CCS*.

[26] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. 2019. Learning to Reconstruct: Statistical Learning Theory and Encrypted Database Attacks. In *Proc. of IEEE S&P*.

[27] Paul Grubbs, Kevin Sekniqi, Vincent Bindschaedler, Muhammad Naveed, and Thomas Ristenpart. 2017. Leakage-Abuse Attacks against Order-Revealing Encryption. In *Proc. of IEEE S&P*.

[28] Zichen Gui, Oliver Johnson, and Bogdan Warinschi. 2019. Encrypted Databases: New Volume Attacks against Range Queries. In *Proc. of ACM CCS*.

[29] Zichen Gui, Kenny Paterson, and Tianxin Tang. 2023. Security Analysis of MongoDB Queryable Encryption. In *Proc. of USENIX Security*.

[30] Zicheng Gui, Kenneth G. Paterson, and Sikhar Patranabis. 2023. Rethinking Searchable Symmetric Encryption. In *Proc. of IEEE S&P*.

[31] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. 2012. Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation. In *Proc. of NDSS*.

[32] Seny Kamara, Abdelkarim Kati, Tarik Moataz, Thomas Schneider, Amos Treiber, and Michael Yonli. 2022. SoK: Cryptanalysis of Encrypted Search with LEAKER - A framework for LEakage AttacK Evaluation on Real-world data. In *Proc. of IEEE EuroS&P* .

[33] Seny Kamara and Tarik Moataz. 2017. Boolean Searchable Symmetric Encryption with Worst-Case Sub-linear Complexity. In *Proc. of EUROCRYPT*.

[34] Seny Kamara and Tarik Moataz. 2019. Computationally Volume-Hiding Structured Encryption. In *Proc. of EUROCRYPT*.

[35] Seny Kamara, Tarik Moataz, and Olga Ohrimenko. 2018. Structured Encryption and Leakage Suppression. In *Proc. of CRYPTO*.

[36] Seny Kamara, Tarik Moataz, Andrew Park, and Lucy Qin. 2021. A Decentralized and Encrypted National Gun Registry. In *Proc. of IEEE S&P*.

[37] Seny Kamara and Charalampos Papamanthou. 2013. Parallel and Dynamic Searchable Symmetric Encryption. In *Proc. of FC*.

[38] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. 2012. Dynamic searchable symmetric encryption. In *Proc. of ACM CCS*.

[39] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. 2016. Generic Attacks on Secure Outsourced Databases. In *Proc. of ACM CCS*.

[40] Evgenios M. Kornaropoulos, Charalampos Papamanthou, and Roberto Tassia. 2019. Data Recovery on Encrypted Databases With k-Nearest Neighbor Query Leakage. In *Proc. of IEEE S&P*.

[41] Evgenios M. Kornaropoulos, Charalampos Papamanthou, and Roberto Tassia. 2020. The State of the Uniform: Attacks on Encrypted Databases Beyond the Uniform Query Distribution. In *Proc. of IEEE S&P*.

[42] Evgenios M. Kornaropoulos, Charalampos Papamanthou, and Roberto Tassia. 2021. Response-Hiding Encrypted Ranges: Revisiting Security via Parametrized Leakage-Abuse Attacks. In *Proc. of IEEE S&P*.

[43] Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. 2018. Improved Reconstruction Attacks on Encrypted Data Using Range Query Leakage. In *Proc. IEEE S&P*.

[44] Shangqi Lai, Sikhar Patranabis, Amin Sakzad, Joseph K. Liu, Debdeep Mukhopadhyay, Ron Steinfeld, Shifeng Sun, Dongxi Liu, and Cong Zuo. 2018. Result Pattern

[45] Hiding Searchable Encryption for Conjunctive Queries. In *Proc. of ACM CCS*.

[45] Lucene Email Dataset. 1999. Online at: https://lucene.apache.org/.

[46] Evangelia Anna Markatou, Francesca Falzon, Roberto Tassia, and William Schor. 2021. Reconstructing with Less: Leakage Abuse Attacks in Two Dimensions. In *Proc. of ACM CCS*.

[47] Brice Minaud and Michael Reichle. 2022. Dynamic Local Searchable Symmetric Encryption. In *Proc. of CRYPTO*.

[48] Muhammad Naveed, Seny Kamara, and Charles V. Wright. 2015. Inference Attacks on Property-Preserving Encrypted Databases. In *Proc. of ACM CCS*.

[49] Muhammad Naveed, Manoj Prabhakaran, and Carl A. Gunter. 2014. Dynamic Searchable Encryption via Blind Storage. In *Proc. of IEEE S&P*.

[50] Jianting Ning, Xinyi Huang, Geong Sen Poh, Jiaming Yuan, Yingjiu Li, Jian Weng, and Robert H. Deng. 2021. LEAP: Leakage-Abuse Attack on Efficiently Deployable, Efficiently Searchable Encryption with Partially Known Dataset. In *Proc. of ACM CCS*.

[51] Simon Oya and Florian Kerschbaum. 2021. Hiding the Access Pattern is Not Enough: Exploiting Search Pattern Leakage in Searchable Encryption. In *Proc. of USENIX Security*.

[52] Simon Oya and Florian Kerschbaum. 2022. IHOP: Improved Statistical Query Recovery against Searchable Symmetric Encryption through Quadratic Optimization. In *Proc. of USENIX Security*.

[53] Yanjun Pan, Alon Efrat, Ming Li, Boyang Wang, Hanyu Quan, Joseph S. B. Mitchell, Jie Gao, and Esther M. Arkin. 2020. Data inference from encrypted databases: a multi-dimensional order-preserving matching approach. In *Proc. of ACM Mobihoc*.

[54] Vasilis Pappas, Fernando Krell, Binh Vo, Vladimir Kolesnikov, Tal Malkin, Seung Geol Choi, Wesley George, Angelos D. Keromytis, and Steven M. Bellovin. 2014. Blind Seer: A Scalable Private DBMS. In *Proc. of IEEE S&P*.

[55] Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. 2020. Lower Bounds for Encrypted Multi-Maps and Searchable Encryption in the Leakage Cell Probe Model. In *Proc. of CRYPTO*.

[56] Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. 2019. Mitigating Leakage in Secure Cloud-Hosted Data Structures: Volume-Hiding for Multi-Maps via Hashing. In *Proc. of ACM CCS*.

[57] Rishabh Poddar, Stephanie Wang, Jianan Lu, and Raluca Ada Popa. 2020. Practical Volume-Based Attacks on Encrypted Databases. In *Proc. of IEEE EuroS&P*.

[58] David Pouliot and Charles V. Wright. 2016. The Shadow Nemesis: Inference Attacks on Efficiently Deployable, Efficiently Searchable Encryption. In *Proc. of ACM CCS*.

[59] Zhiwei Shang, Simon Oya, Andreas Peter, and Florian Kerschbaum. 2021. Obfuscated Access and Search Patterns in Searchable Encryption. In *Proc. of NDSS*.

[60] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. 2014. Practical Dynamic Searchable Encryption with Small Leakage. In *Proc. of NDSS*.

[61] Shifeng Sun, Ron Steinfeld, Shangqi Lai, Xingliang Yuan, Amin Sakzad, Joseph K. Liu, Surya Nepal, and Dawu Gu. 2021. Practical Non-Interactive Searchable Encryption with Forward and Backward Privacy. In *Proc. of NDSS*.

[62] Shifeng Sun, Xingliang Yuan, Joseph K. Liu, Ron Steinfeld, Amin Sakzad, Viet Vo, and Surya Nepal. 2018. Practical Backward-Secure Searchable Encryption from Symmetric Puncturable Encryption. In *Proc. of ACM CCS*.

[63] Viet Vo, Shangqi Lai, Xingliang Yuan, Shifeng Sun, Surya Nepal, and Joseph K. Liu. 2020. Accelerating Forward and Backward Private Searchable Encryption Using Trusted Execution. In *Proc. of ACNS*.

[64] Viet Vo, Xingliang Yuan, Shi-Feng Sun, Joseph K. Liu, Surya Nepal, and Cong Wang. 2023. ShieldDB: An Encrypted Document Database With Padding Countermeasures. *IEEE Trans. Knowl. Data Eng.* 35, 4 (2023), 4236–4252.

[65] Jianfeng Wang, Shifeng Sun, Tianci Li, Saiyu Qi, and Xiaofeng Chen. 2022. Practical Volume-Hiding Encrypted Multi-Maps with Optimal Overhead and Beyond. In *Proc. of ACM CCS*.

[66] Tianhao Wang and Yunlei Zhao. 2016. Secure Dynamic SSE via Access Indistinguishable Storage. In *Proc. of ACM AsiaCCS*.

[67] Lei Xu, Huayi Duan, Anxin Zhou, Xingliang Yuan, and Cong Wang. 2021. Interpreting and Mitigating Leakage-Abuse Attacks in Searchable Symmetric Encryption. *IEEE Trans. Inf. Forensics Secur.* 16 (2021), 5310–5325.

[68] Lei Xu, Xingliang Yuan, Cong Wang, Qian Wang, and Chungen Xu. 2019. Hardening Database Padding for Searchable Encryption. In *Proc.of IEEE INFOCOM*.

[69] Lei Xu, Leqian Zheng, Chengzhi Xu, and Xingliang Yuan Cong Wang and. 2023. Leakage-Abuse Attacks Against Forward and Backward Private Searchable Symmetric Encryption. *CoRR* (2023).

[70] Lei Xu, Anxin Zhou, Huayi Duan, Cong Wang, Qian Wang, and Xiaohua Jia. 2022. Toward Full Accounting for Leakage Exploitation and Mitigation in Dynamic Encrypted Databases. *IEEE Trans. Dependable Secur. Comput.* (2022).

[71] Xianglong Zhang, Wei Wang, Peng Xu, Laurence T. Yang, and Kaitai Liang. 2023. High Recovery with Fewer Injections: Practical Binary Volumetric Injection Attacks against Dynamic Searchable Encryption. In *Proc. of USENIX Security*.

[72] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. 2015. All Your Queries Are Belong to Us: The Power of File-Injection Attacks on Searchable Encryption. In *Proc. of USENIX Security*.