Exploiting Update Leakage in Searchable Symmetric Encryption

Jacob Haltiwanger
Virginia Tech
Department of Computer Science
Blacksburg, VA, USA
jacobshalt@vt.edu

Thang Hoang
Virginia Tech
Department of Computer Science
Blacksburg, VA, USA
thanghoang@vt.edu

ABSTRACT

Dynamic Searchable Symmetric Encryption (DSSE) provides efficient techniques for securely searching and updating an encrypted database. However, efficient DSSE schemes leak some sensitive information to the server. Recent works have implemented *forward and backward privacy* as security properties to reduce the amount of information leaked during update operations. Many attacks have shown that leakage from search operations can be abused to compromise the privacy of client queries. However, the attack literature has not rigorously investigated techniques to abuse *update leakage*.

In this work, we investigate update leakage under DSSE schemes with forward and backward privacy from the perspective of a passive adversary. We propose two attacks based on a maximum likelihood estimation approach, the UFID Attack and the UF Attack, which target forward-private DSSE schemes with no backward privacy and Level II backward privacy, respectively. These are the first attacks to show that it is possible to leverage the frequency and contents of updates to recover client queries. We propose a variant of each attack which allows the update leakage to be combined with search pattern leakage to achieve higher accuracy. We evaluate our attacks against a real-world dataset and show that using update leakage can improve the accuracy of attacks against DSSE schemes, especially those without backward privacy.

CCS CONCEPTS

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

KEYWORDS

Searchable Encryption; Inference Analysis; Database Privacy

ACM Reference Format:
Jacob Haltiwanger and Thang Hoang. 2024. Exploiting Update Leakage in Searchable Symmetric Encryption . In *Proceedings of the Fourteenth ACM Conference on Data and Application Security and Privacy (CODASPY '24), June 19–21, 2024, Porto, Portugal.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3626232.3653260

1 INTRODUCTION

Cloud computing has experienced substantial growth in the past decade. Improvements in networking infrastructure and lower costs of storage hardware have given way to a number of remote storage solutions. These options promise high availability and enable convenient access to files across multiple devices. Cheap remote storage options have been rapidly increasing in popularity as a way for users to offload files and free up storage space on their local devices. However, storing personal files on an untrusted server presents security concerns. The server has access to the client's entire file set and queries in plaintext. One way to mitigate these concerns is to encrypt each file before uploading to the server. However, this approach breaks user-friendly search functionality because keyword matching becomes impossible under traditional encryption.

Searchable Symmetric Encryption (SSE) is one method to address the privacy concerns of remote data outsourcing while retaining search functionality in optimal time. In a basic SSE scheme, the client first *tokenizes* the keywords and uses the tokens to create a search index. Then, the client protects the database with symmetric encryption and sends the encrypted database and search index to the server. To perform a query, the client sends a search token to the server, who checks the index and returns the corresponding files. The client can then decrypt and view the files locally. *Dynamic* Searchable Symmetric Encryption (DSSE) [7, 15, 21, 24] is a convenient generalization of SSE that allows the client to privately update the database. All forms of encrypted search are designed to provide two kinds of privacy to the users: data privacy and query privacy. *Data privacy* implies that the server is unable to learn the plaintext of the encrypted files that it is storing on behalf of the client. *Query privacy* implies that the server is unable to learn which keywords a client is searching or updating. Over time, SSE scheme development research has focused on more efficient constructions [9, 15, 21], more expressive queries [6, 14, 26, 37, 38], multi-user settings [25, 32, 36], and stronger privacy properties [3, 4, 7, 12, 17].

Although SSE/DSSE significantly improves the privacy of remote storage, all efficient schemes leak some sensitive information to the server. Curtmola et al. [9] introduced the notion of *leakage functions*, which formally characterize the information that the server can learn from SSE operations. An SSE scheme must only leak information that is captured in one of the scheme's leakage functions. Most efficient SSE schemes leak the *access pattern*, or the file identifiers returned in response to a search, and almost all SSE schemes leak the *search pattern*, or whether two search operations target the same keyword. Update operations can also leak sensitive information in DSSE schemes. *Forward privacy* and *backward privacy* are important security properties that were introduced to make dynamic schemes more secure. They imply that the server cannot learn whether newly inserted files match past queries or whether deleted files match future queries, respectively.

Many *leakage-abuse attacks* have shown that access and search pattern leakage can be abused by an adversarial server to violate the privacy of SSE schemes with certain background information

CODASPY '24, June 19–21, 2024, Porto, Portugal

ACM ISBN 979-8-4007-0421-5/24/06
https://doi.org/10.1145/3626232.3653260

Table 1: Comparison of passive attacks against SSE/DSSE

Attack Name	Attack Type*	Targets DSSE?**	Target Leakage†	Requirements		
				Known queries	Known files	Aux
IKK [22]	Known-Data	✗	Co-occurrence Pattern	✓	✓	✗
Count [5]	Known-Data	✗	Co-occurrence, Volume Patterns	✓	✓	✗
SelVolAn [2]	Known-Data	✗	Volume Pattern	✗	✓	✗
Refined Score [10]	Hybrid	✗	Co-occurrence Pattern	✓	✗	✓
Graph Match [31]	Inference	✗	Co-occurrence Pattern	✗	✗	✓
Frequency [27]	Inference	✗	Search Pattern	✗	✗	✓
SAP [28]	Inference	✗	Search, Volume Patterns	✗	✗	✓
IHOP [29]	Inference	✗	Co-occurrence, Search, Volume Patterns	✗	✗	✓
FMA [39]	Inference	✓	Search, Volume Patterns	✗	✗	✓
VIA, PVIA [39]	Known-Data	✓	Volume Pattern	✗	✓	✗
LVIA [39]	Inference	✓	Volume Pattern	✗	✗	✓
Our UF	Inference	✓	Search Pattern, Update Freq.	✗	✗	✓
Our UFID	Inference	✓	Search Pattern, Update Freq. & Contents	✗	✗	✓

*Known-data attacks require the attacker to know some portion of the documents, queries, or both, in plaintext. Inference attacks use statistically-similar auxiliary information about the target leakage to assist in the attack.
**Attacks which do not target DSSE can be modified to work against DSSE, but the necessary changes are non-trivial, especially for stronger schemes.
†Co-occurrence pattern leaks the number of files which both contain a keyword. Search pattern leaks whether two queries target the same keyword. Volume pattern typically means the number of files returned in response to a query (SelVolAn, FMA, and VIA use a higher-leakage notion called file volume, which leaks the number of keywords in each file containing the queried keyword).
Both of our attacks use search pattern leakage to infer update frequency. The UFID Attack additionally uses the file identifiers associated with each update. In Section 4, we present a variant of each attack, SP+UF and SP+UFID respectively, which can exploit both the search pattern and update leakage directly.

and assumptions about client behavior. Most attacks in the field are aimed at compromising query privacy [2, 5, 10, 22, 28, 29, 33, 39, 40]. Most attacks are *passive* (e.g., [5, 10, 22]), meaning the adversary only needs to observe leakage and possess some background information. Some attacks are *active* (e.g., [2, 33, 40]), meaning the adversary can send chosen queries, add files to the database, or perform other SSE operations. A summary of leakage profiles that are commonly leaked (and commonly targeted by attacks) is below:

- **Access Pattern**: After a query for keyword w, reveals the identifiers of the files that contain w (This is commonly called Access Pattern in the SSE literature, but called Result Pattern in adjacent fields).
- **Co-occurrence Pattern**: After queries for keywords w_i and w_j, reveals which file identifiers contain both w_i and w_j. Can be inferred from access patterns.
- **Volume Pattern** After a query for keyword w, reveals how many documents were returned, i.e., size of the access pattern for w. Some works include a stronger notion of *file volume*, or the number of keywords per file containing w.
- **Search Pattern** Given two queries τ_i and τ_j, reveals whether both queries target the same keyword.

See Table 1 for a summary of passive attacks which exploit these different kinds of leakage. These works focus heavily on leakage during the search operation, because it is the primary source of leakage in SSE schemes. In fact, almost all passive attacks only target static SSE [5, 10, 22, 28, 29, 31], and do not consider how to attack DSSE schemes where the client performs both search and update operations. While a handful of active attacks [2, 33, 40] are powerful against DSSE schemes without forward privacy, all such attacks only exploit leakage from the search operation. Leakage from client updates has generally been overlooked in the attack literature. Thus, the defense literature has been working to improve the efficiency of backward privacy [4, 7, 17, 42] not in response to any attacks, but instead under the collective intuition that less leakage must result in better security. Without any attacks targeting DSSE to evaluate against, defense researchers are unsure whether backward privacy successfully defends against any attacks [42]. This has left the field with the following open question:

Can leakage related to DSSE update operations be directly exploited to improve the accuracy of query-recovery attacks, even in the presence of forward and backward privacy?

1.1 Our Contributions

We answer this question affirmatively with two query-recovery attacks: the UFID Attack and the UF Attack. Our attacks aim to directly compromise the update leakage exclusive to dynamic SSE schemes. We also show how to improve accuracy by combining search pattern and update leakage, resulting in a variant of each attack, the SP+UFID Attack and SP+UF Attack, respectively. The UFID and SP+UFID Attacks achieve high query recovery rates against forward-private DSSE with no backward privacy, while the UF and SP+UF Attacks can recover queries from schemes with forward privacy and Level II backward privacy. These attacks use the Maximum Likelihood Estimation technique explored by Oya and Kerschbaum [28] as a building block because it is an efficient and elegant tool to exploit frequency information.

Previous passive attacks have typically focused on targeting static SSE schemes [2, 5, 22, 28, 29, 31]. Instead, our work investigates how a passive adversary can use the frequency of updates and their associated file identifiers to compromise the security of dynamic SSE schemes. Our work is also the first to exploit update frequency against backward-private DSSE schemes at Level II and Level III.

The very recent work by Xu et al. [39] proposes the first passive attacks to target DSSE, and their attacks work against forward- and backward-private schemes. However, their attacks only use update leakage to infer information about traditionally-exploited leakage from searches. Our work is different because we focus on exploiting this update leakage directly via frequency analysis.

While there are a handful of Level I backward-private SSE schemes [1, 16, 17, 42, 43], they are not strong candidates for widespread adoption because the oblivious data structures necessary to build them incur heavy performance costs. As such, our attacks can compromise query privacy of nearly all efficient DSSE schemes.

We evaluate our attacks under realistic settings with a real-world dataset and show that incorporating update leakage improves Oya and Kerschbaum's recent frequency attack, SAP, [28] by 5% in schemes with forward privacy and Level II or Level III backward-privacy, and up to 25% in schemes with only forward privacy. Therefore, our work provides significant theoretical and empirical evidence that a DSSE scheme which leaks information about the frequency of searches and updates enables a passive adversarial server to compromise client privacy *more* effectively than the static search-only setting traditionally considered by the attack literature.

2 RELATED WORK

SSE and Defenses. Song et al. [34] introduced the first practical techniques for searching over encrypted data. Curtmola et al. [9] formalized definitions related to the security of SSE schemes.

Forward and *backward privacy* are important security properties for dynamic SSE schemes introduced by Stefanov et al. [35]. Forward privacy reduces the leakage after update operations, but requires some efficiency tradeoffs. Stefanov et al. gave the first forward-private scheme, and forward privacy was later formally defined by Bost [3]. Works such as those by Zhang et al. [41] focus on creating efficient forward-private schemes.

Later, Bost et al. [4] formalized backward privacy, an even stronger security notion organized into three levels, I through III, with Level I leaking the least information. Backward privacy limits the amount of information the server can learn about deleted documents, but again comes at an efficiency cost. Despite several works [7, 12, 17] focusing on implementation of efficient forward- and backward-private schemes, designing a practically-efficient Level I backward-private scheme remains an open challenge.

A different line of work investigates strategies to defend against the most common types of attacks against SSE schemes. These works typically involve probabilistically hiding one of the leakage profiles introduced above. Demertzis et al. [13] developed a scheme which can reduce volume and access pattern leakage by padding response volume based on adjustable security parameters. Amjad et al. [1] craft two forward- and backward-private volume-hiding DSSE schemes from dynamic volume-hiding encrypted multi-maps. The work of Zhao et al. [42] has a similar focus with stronger backward privacy and better asymptotic efficiency at the cost of imperfect correctness. Access-pattern-hiding countermeasures such as the differentially-private approach presented by Chen et al. [8] typically provide security through the inclusion of false positives and false negatives, which can increase security but harm the efficiency and correctness of the scheme. These types of probabilistic countermeasures are often harder to make provable security guarantees about, but provide practical security against attacks that require the leakage they attempt to hide.

Leakage-Abuse Attacks. Islam, Kuzu, and Kantarcioglu [22] were the first to demonstrate that it is possible to violate query privacy in searchable encryption. By combining ground truth data with query result co-occurrence, they could frame query recovery as a quadratic optimization problem that can be approximated via simulated annealing. The field collectively refers to their work as the IKK attack. Cash et al. [5] proposed a similar but more efficient approach that required less ground truth information.

Pouliot and Wright [31] frame the query recovery problem as a graph matching problem. Their attack can work with ground truth data or similar data, but it may be difficult to obtain similar co-occurrence data in practice. Damie et al. [10] developed an attack that performs well under scenarios in which the attacker only has some known queries and a distributionally similar auxiliary dataset.

Liu et al. [27] proposed an attack which uses query frequency information combined with open-source data and background knowledge about the dataset to recover client queries. Oya and Kerschbaum [28] combine frequency information with response volume leakage to construct the SAP Attack, which solves a linear assignment problem via a maximum likelihood estimation approach. Later, Oya and Kerschbaum [29] introduced the IHOP Attack, which makes use of frequency and query dependency information to iteratively solve a quadratic optimization problem, improving upon the result from the SAP Attack.

Attacks Against DSSE. Zhang et al. [40] were the first to consider an *active* adversary in detail. They proposed a file-injection attack, where the adversary adds carefully crafted files to the database and observes the access patterns to recover client queries with high accuracy. They concluded that forward privacy is a vital security property for defeating file-injection attacks, although Salmani and Barker [33] later showed that file-injection attacks can be modified to compromise the security of forward-private schemes. Although this class of attacks specifically targets dynamic SSE schemes, our work differs in two ways. First, file-injection attacks require an active adversary, while ours uses a weaker, passive adversary. Second, file-injection attacks target access pattern leakage, while our work targets search pattern and update leakage.

Concurrent to our work, Xu et al. [39] presented techniques for an adversary to track leakage exploited by many attacks against static SSE to the dynamic setting while the client issues updates. While this process is simple for defenseless DSSE schemes, they moved the field forward by presenting techniques that work even in the presence of forward- and Level I backward-private schemes. They propose three attacks which use information about the total number of updates per keyword to recover search and volume patterns, which are exploited by many of the existing static SSE attacks. Our work differs because we focus on exploiting information about the frequency and contents of the updates directly to compromise forward-private schemes with backward privacy equal to or worse than Level II. In addition, we show that our approach can make attacks *more* powerful against DSSE schemes, while their work only showed that DSSE is similarly vulnerable to static SSE.

Table 2: Summary of notation

General Parameters	
Δ	Keyword universe $\Delta \doteq [w_1, w_2, \ldots, w_n]$.
n	Total number of keywords, $n \doteq \lvert\Delta\rvert$.
w_i	The ith keyword in Δ, with $i \in [n]$.
N_D	Number of documents in the encrypted dataset.
id_i	ith file identifier, with $1 < i \leq N_D$
ρ	Number of observation time intervals.
Adversary Observations	
m	Number of search tags (derived from search pattern leakage).
γ_j	jth tag, with $j \in [m]$.
η_k	Number of queries sent in the kth time interval, with $k \in \rho$.
$\boldsymbol{\eta}$	Vector $\boldsymbol{\eta} \doteq [\eta_1, \eta_2, \ldots, \eta_\rho]$.
$f_{j,k}$	Search frequency of γ_j in the kth time interval, with $k \in \rho$.
$\mathbf{f}_j$	Search frequency vector of γ_j, $\mathbf{f}_j \doteq [f_{j,1}, \ldots, f_{j,\rho}]$.
$\mathbf{F}$	Search frequency matrix of all tags (size $m \times \rho$).
μ_k	Number of update requests in the kth time interval.
$\boldsymbol{\mu}$	Vector $\boldsymbol{\mu} \doteq [\mu_1, \mu_2, \ldots, \mu_\rho]$.
$\omega_{j,k}$	Number of update requests for γ_j in the kth time interval.
$\boldsymbol{\omega}_j$	Vector $\boldsymbol{\omega} \doteq [\omega_{j,1}, \omega_{j,2}, \ldots, \omega_{j,\rho}]$.
$\boldsymbol{\Omega}$	Update frequency matrix of all tags (size $m \times \rho$).
$U_{j,y,k}$	Update frequency for jth tag in the yth file in in time interval k.
$\mathbf{U}_{y,k}$	Update frequency for all tags in the yth file in time interval k (size m).
$\mathbf{U}_k$	Update frequency for all tags in all files in time interval k (size $m \times N_D$).
$\mathbf{U}$	(Update, file ID) pair frequency matrix of all tags (size $m \times N_D \times \rho$).
Auxiliary Information	
$\tilde{f}_{i,k}$	Search frequency of w_i in the kth time interval, with $k \in \rho$.
$\tilde{\mathbf{f}}_i$	Search frequency vector of w_i, $\mathbf{f}_i \doteq [f_{i,1}, \ldots, f_{i,\rho}]$.
$\tilde{\mathbf{F}}$	Search frequency matrix of all keywords (size $n \times \rho$).
$\tilde{\omega}_{i,k}$	Update frequency of w_i in the kth time interval.
$\tilde{\boldsymbol{\omega}}_i$	Update frequency vector of w_i, $\tilde{\boldsymbol{\omega}}_i \doteq [\omega_{i,1}, \ldots, \omega_{i,\rho}]$.
$\tilde{\boldsymbol{\Omega}}$	Update frequency matrix of all keywords (size $n \times \rho$).
$\tilde{U}_{i,y,k}$	Update freq. for ith keyword in the yth file in time interval k.
$\tilde{\mathbf{U}}_{y,k}$	Update freq. for all keywords in the yth file in time interval k (size m).
$\tilde{\mathbf{U}}_k$	Update freq. for all keywords in all files in time interval k (size $m \times N_D$).
$\tilde{\mathbf{U}}$	(Update, file ID) pair frequency matrix of all keywords (size $n \times N_D \times \rho$).
Attack Goal	
$p(j)$	Index of the keyword that an attack assigns to γ_j.
$\mathbf{P}$	Permutation matrix, $\mathbf{P}_{p(j),j} = 1$, else 0 (size $n \times m$).

3 PRELIMINARIES

Notation. Given an SSE database with N_D documents, Δ is the keyword universe of size n for which the client issues search or update queries. If the adversary believes two queries correspond to the same keyword, she assigns them the same *tag*. Let m be the number of search tags. Search pattern leakage is a necessary assumption for tag assignment. The adversary will observe ρ time intervals of query and response traces between the client and server. $\mathbf{f}_j$ and $\boldsymbol{\omega}_j$ hold all information from all ρ time intervals about the frequency of the searches and updates, respectively, on tag j. Matrices $\mathbf{F}$ and $\boldsymbol{\Omega}$ of size $(m \times \rho)$ store search/update frequency information for all tags for all weeks. $\mathbf{U}$ is size $(m \times N_D \times \rho)$, where U_j, y, k gives the frequency of updating the jth tag into the yth file in the kth time interval. $\tilde{\mathbf{F}}, \tilde{\boldsymbol{\Omega}}$, and $\tilde{\mathbf{U}}$ are the auxiliary counterparts to these matrices. $p(j)$ is the keyword that an attack assigns to a tag. Matrix $\mathbf{P}$ is the full injective mapping of keywords from Δ to tags. In other words, it contains $p(j)\ \forall\ j \in \{1, m\}$. Let $\mathcal{P}$ be the set of all valid permutation matrices $\mathbf{P}$. See Table 2 for notation used by the attacks.

3.1 System Model

We consider a client-server file storage service where a client wishes to offload a private database to save local storage space. The client wishes to perform two types of operations over the database. The first is *single-keyword queries*, where the server returns all files which contain a particular keyword. The second is *updates*, where the client adds or deletes a file identifier from the entry of a keyword in the search index. Below we briefly explain the operation of a DSSE scheme, then present formal definitions and notation that we will use to define our threat and attack models.

Dynamic Searchable Symmetric Encryption. First, the client sets up the scheme by using the files to generate an inverted search index. The index maps each keyword to all of the identifiers of all the documents that contain it. Then, the client *tokenizes* each keyword w in the inverted index by computing its one-way trapdoor τ_w. After tokenization, the inverted index is known as the encrypted search index. Next, the client uses symmetric encryption to encrypt the entire database. The client concludes the setup phase by sending the encrypted search index and the encrypted database to the server.

When the client wishes to perform a search query for keyword w, she sends its search token τ_w to the server. The server checks the index for the row containing τ_w and sends the corresponding files back to the client for decryption.

To perform an update on keyword w, the client sends a tuple $(\mathsf{op}, \tau_w, \mathsf{id})$ to the server. The server finds the row in the index containing τ_w or creates a new row if one doesn't exist. The server looks at op to determine whether the operation is an addition or deletion. The server then performs the specified operation on file id in the position for τ_w in the search index. These operations are formally defined below:

Definition 3.1. (Dynamic Searchable Symmetric Encryption) A *dynamic searchable symmetric encryption scheme* Σ consists of an algorithm Setup and two protocols, Search and Update, between a client and server.

- $\mathsf{Setup}(\mathsf{DB})$ is an algorithm run by the client to initialize the DSSE scheme. It takes as input the initial database DB and outputs $(\mathsf{EDB}, K, \sigma)$, where EDB is an encrypted form of DB, K is the master secret key, and σ is the client's internal state.
- $\mathsf{Search}(K, \sigma, w; \mathsf{EDB})$ is a protocol consisting of a client algorithm $\mathsf{Search}_C(K, \sigma, w)$ which takes as input the master key, internal state, and keyword w, and a server algorithm $\mathsf{Search}_S(\mathsf{EDB})$, which takes the encrypted database as input and outputs $\mathsf{DB}(w)$.
- $\mathsf{Update}(K, \sigma, \mathsf{op}, \mathsf{in}; \mathsf{EDB})$ is a protocol consisting of two algorithms. $\mathsf{Update}_C(K, \sigma, \mathsf{op}, \mathsf{in})$ is a client algorithm which takes as input the master key, internal state, operation $\mathsf{op} \in \{\mathsf{add}, \mathsf{del}\}$, and input in, which consists of a file identifier id and a keyword w. $\mathsf{Update}_S(\mathsf{EDB})$ is a server algorithm which takes the encrypted database as input and adds or deletes one or more keyword-document pairs from the encrypted search index.

A DSSE scheme is considered correct iff the search protocol returns the correct result for all queries except with negligible probability.

This definition of updates extends to the case where the client wants to add or remove a file from the database; this operation would be decomposed into an individual keyword/document update for each keyword in the file.

3.2 Threat and Attack Models

In our paper, the adversary is a server that is *honest-but-curious*; it will follow the DSSE protocol exactly but attempts to learn the underlying keywords of all searches and updates sent by the client.

Search Patterns. We consider SSE schemes which leak the *search pattern*. Informally, search pattern leakage allows the adversary to learn query equality, or whether two search tokens were generated by the same keyword. This could be in the form of *explicit* search patterns, where keyword w is converted to the same search token regardless of the client's internal state σ, or *implicit* search patterns, where the adversary can reliably tell whether two tokens match a keyword by looking for unique patterns in other types of leakage. We formally define search pattern leakage as follows [9]:

Definition 3.2. (Search Pattern) Given a keyword universe Δ, database EDB, and query history $H = (\mathsf{EDB}, \Delta)$ of length n, the search pattern is a symmetric binary matrix $S(H)$ such that for $1 \leq i, j \leq n$, $S(H)_{i,j} = 1$ if $w_i = w_j$, and 0 otherwise.

Search patterns are hard to hide because they can be inferred from other types of leakage. By observing client queries and their responses, the adversary can determine whether two different queries were for the same keyword with high probability. For example, if the adversary observes that two queries, q_i and q_j, have the same access pattern, she can assume that q_i and q_j were generated by the same keyword.

Tracking Search Patterns in DSSE. In DSSE schemes, search patterns are harder to track because file additions and deletions can change a token's access patterns or response volumes over time. Prior works have presented techniques to overcome this obstacle. Salmani and Barker [33] create n bit-strings of length N_D, where the jth bit is 1 if w_i is contained in file j. The server saves these bit-strings. If a file is added to the database, copies of the bit-strings are made and a bit for the new file is prepended to the string. The server also maintains a deletion bit-string, where the jth bit is 1 if file j has been deleted by the client. When the server receives a search token, it takes its bit-string and compares it bit-wise with the deletion bit-string and all of the original bit-strings. If the comparison returns zero, then the tokens correspond to the same keyword, thus search patterns are recovered. Xu et al. [39] estimate query equality by computing differences in response similarity. We refer to the papers by both authors for a more thorough treatment of this topic. For the purposes of our work, we assume the adversary is capable of tracking query equality in the DSSE environment.

Update History. When a client issues an update for keyword w in a DSSE scheme, the server can learn which keyword is being updated, the timestamp of the update, whether the operation is an addition or deletion, and which file is being updated. The adversary can keep track of this data continuously for all keywords in a leakage profile called UpHist described by Bost [3]. More generally, $\mathsf{UpHist}(w) = (u, \mathsf{op}, \mathsf{id})$, where u is a timestamp starting at 0 and incremented with each operation, op is the operation type, and id is the identifier of the updated file. Below we discuss security properties that can reduce the leakage of update operations to less than $\mathsf{UpHist}(\mathsf{w})$.

Forward Privacy. It is desirable to prevent the server from building UpHist after every update. Forward privacy is a security property that helps mitigate leakage during update operations. Informally, forward privacy guarantees that the server cannot learn whether the keywords of a newly added document match any previous queries. In a forward-private scheme, updates on keyword w are only revealed to the server after the client has searched for w. Formally, forward privacy is defined as follows [3]:

Definition 3.3. (Forward Privacy) An $\mathcal{L}$-adaptively secure DSSE is forward-private iff the leakage function $\mathcal{L}^{\mathsf{Updt}}$ can be written as:

$$\mathcal{L}^{\mathsf{Updt}}(\mathsf{op}, \mathsf{in}) = \mathcal{L}'(\mathsf{op}, \{(\mathsf{id_i}, \mathsf{w_i})\}$$

where $\mathcal{L}'$ is stateless and the set $\{(\mathsf{id_i}, \mathsf{w_i})\}$ contains all updated documents as the number of modified keywords $\mathsf{w_i}$ in updated document $\mathsf{id_i}$.

Under this definition, the most information that an adversary can learn immediately after a file is added to the database in a forward-private scheme is the identifier id of the file, the number of keywords in the file, and whether the update was an addition or deletion. In this work, we only consider SSE schemes that are at least forward-private.

Backward Privacy. Backward privacy is a strong security property for DSSE schemes. Informally, backward privacy ensures that the server does not learn whether deleted files match a search query. Bost et al. [4] were the first to rigorously characterize backward privacy, which they organize into three levels. We recap them in order of decreasing strength, noting that each level is strictly stronger than the next. At each level, a search for keyword w reveals:

Level I. Backward Privacy with Insertion Pattern: All the documents currently containing w ($\mathsf{TimeDB}(w)$), when they were inserted, and the total number of updates on w (a_w).

Level II. Backward Privacy with Update Pattern: All the documents currently containing w ($\mathsf{TimeDB}(w)$), when they were inserted, and when all the updates on w happened ($\mathsf{Updates}(w)$).

Level III. Weak Backward Privacy: All the documents currently containing w ($\mathsf{TimeDB}(w)$), when they were inserted, and when all the updates on w happened with information about which deletion updates cancelled which addition updates ($\mathsf{DelHist}(w)$).

From a query list Q in the form (u, w), where u is a timestamp starting at 0 and incremented with every query, we can construct the necessary leakage profiles, following Bost et al [4]. $\mathsf{TimeDB}(w)$ is the list of all documents matching w with a timestamp of when they were inserted in the database and it excludes any documents which have been deleted. Formally,

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

$\mathsf{Updates}(w)$ is the collection of update timestamps on w. Formally,

$$\mathsf{Updates}(w) = \{u \mid (u, \mathsf{add}, (w, \mathsf{id})) \text{ or } (u, \mathsf{del}, (w, \mathsf{id})) \in Q\}.$$

$\mathsf{DelHist}(w)$ is the list of timestamps for all deletion operations with the timestamp of the inserted entries they remove. Formally,

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

Definition 3.4. (Backward Privacy) An $\mathcal{L}$-adaptively secure DSSE achieves Level I, II, or III backward privacy iff the search and update leakage functions $\mathcal{L}^{\mathsf{Search}}$, $\mathcal{L}^{\mathsf{Updt}}$ can be written as:

Level I: $\mathcal{L}^{\mathsf{Updt}}(\mathsf{op}, w, \mathsf{id}) = \mathcal{L}'(\mathsf{op})$
$\mathcal{L}^{\mathsf{Search}}(w) = \mathcal{L}''(\mathsf{TimeDB}(w), a_w)$,
Level II: $\mathcal{L}^{\mathsf{Updt}}(\mathsf{op}, w, \mathsf{id}) = \mathcal{L}'(\mathsf{op}, w)$
$\mathcal{L}^{\mathsf{Search}}(w) = \mathcal{L}''(\mathsf{TimeDB}(w), \mathsf{Updates}(w))$,
Level III: $\mathcal{L}^{\mathsf{Updt}}(\mathsf{op}, w, \mathsf{id}) = \mathcal{L}'(\mathsf{op}, w)$
$\mathcal{L}^{\mathsf{Search}}(w) = \mathcal{L}''(\mathsf{TimeDB}(w), \mathsf{DelHist}(w))$,
where $\mathcal{L}'$ and $\mathcal{L}''$ are stateless.

Clarifying Update Leakage. The *update operation leakage* of a DSSE scheme is the information leaked during an update operation. This information is formally defined by a scheme's leakage function $\mathcal{L}^{\mathsf{Updt}}$. However, under a typical DSSE use-case, the client will perform both searches and updates. Although state of the art schemes [7, 17, 42] leak little to no information while performing an update for keyword w, the next search for w must leak additional information about the previous updates, otherwise the server could not return the correct results to the client [4]. To avoid confusion when discussing update-related leakage in DSSE schemes, it is useful to distinguish between leakage incurred at update time and the additional leakage after searches. For the rest of this paper, we refer to the extra leakage about previous updates that occurs after searches as *post-search update leakage*.

The post-search update leakage of a keyword w is all leakage about previous update operations on w that exists in $\mathcal{L}^{\mathsf{Search}}(w)$. For schemes that are forward- or backward-private, post-search update leakage is always strictly greater than update operation leakage. For schemes that are not backward-private, the post-search update leakage for w is $\mathsf{UpHist}(w)$.

3.3 Maximum Likelihood Estimation

Oya and Kerschbaum developed the SAP Attack [28], which can be viewed as a combination of a search frequency attack and response volume attack. Although their work targeted the static setting, they use an elegant mathematical approach that is applicable to the dynamic setting. Of particular relevance is the Maximum Likelihood Estimation (MLE) approach that they use to analyze search frequency, which we will refer to as the Search Frequency Attack. We recap this technique below.

The adversary uses *tags* to keep track of the implicit search pattern leakage. She assigns two keywords the same tag if she believes they correspond to the same keyword, using techniques like access pattern comparison described in Section 3.2.

The Search Frequency Attack aims to find the keyword-tag mapping $\mathbf{P}$ that maximizes the probability of observing $\mathbf{F}$, η, and N_D given the auxiliary frequency information $\tilde{\mathbf{F}}$. Formally, the attack solves the optimization problem

$$\mathbf{P} = \underset{\mathbf{P} \in \mathcal{P}}{\operatorname{argmax}} \Pr(\mathbf{F}, \boldsymbol{\eta}, N_D | \tilde{\mathbf{F}}, \mathbf{P})$$

Search Frequency Analysis. The SAP attack makes two assumptions in order to use search frequency as input to the optimization function. The first assumption is that the number of queries $\boldsymbol{\eta}$ for each time interval follows an arbitrary distribution $\Pr(\boldsymbol{\eta})$. The second assumption is that the client chooses keywords to query independently from all other queries, with probabilities given by the auxiliary frequency information $\tilde{\mathbf{F}}$. Under these conditions, the number of queries for each time interval $k \in \rho$ follows a multinomial distribution with η_k trials and probabilities given by the frequency vector $\tilde{f_k}$. The formal relationship is expressed below.

$$\begin{aligned}\Pr(\mathbf{F}, \boldsymbol{\eta} | \tilde{\mathbf{F}}, \mathbf{P}) &= \Pr(\boldsymbol{\eta}) \cdot \Pr(\mathbf{F} | \tilde{\mathbf{F}}, \boldsymbol{\eta}, \mathbf{P}) \\ &= \Pr(\boldsymbol{\eta}) \cdot \prod_{k=1}^{\rho} \Pr(\mathbf{f}_k | \tilde{\mathbf{f}}_k, \eta_k, \mathbf{P}) \\ &= \Pr(\boldsymbol{\eta}) \cdot \prod_{k=1}^{\rho} \eta_k! \prod_{j=1}^{m} \frac{(\tilde{f}_{p(j),k})^{\eta_k f_{j,k}}}{(\eta_k f_{j,k})!}\end{aligned} \tag{1}$$

Oya et al. choose to maximize the logarithm of Equation 1 because it is more precise than maximizing Equation 1 directly. Taking the log, we have:

$$\begin{aligned}&\sum_{k=1}^{\rho} \sum_{j=1}^{m} \log\left((\tilde{f}_{p(j),k})^{\eta_k f_{j,k}}\right) \\ &= \sum_{k=1}^{\rho} \sum_{j=1}^{m} \eta_k f_{j,k} \log\left(\tilde{f}_{p(j),k}\right)\end{aligned} \tag{2}$$

Equation 2 allows for the construction of a *cost matrix* that can be used to numerically express the optimization problem. The (i, j)th entry in the cost matrix is as follows:

$$(\mathbf{C}_f)_{i,j} = -\sum_{k=1}^{\rho} \eta_k f_{j,k} \cdot \log(\tilde{f}_{i,k})$$

4 ATTACK DETAILS

Our goal is to show that post-search update leakage can be exploited to improve the accuracy of attacks on DSSE schemes. To accomplish this goal, we use the Maximum Likelihood Estimation building block described in Section 3.3.

4.1 No Backward Privacy

The post-search update leakage of a scheme with forward privacy and no backward privacy is $\mathsf{UpHist}(w) = (u, \mathsf{op}, \mathsf{id})$. Note that for DSSE schemes with neither forward nor backward privacy, the attack methods described in this section can be used even if the client never searches and only issues updates. We refer to the attack presented in this section as the UFID Attack.

When choosing the update input, we assume the client chooses the keyword w and file identifier id independently from any other updates with probabilities given by $\tilde{\mathbf{U}}$. We finally assume that the number of updates for each tag in file y in a time interval $k \in \rho$ follows a multinomial distribution with μ_k trials and probabilities given by frequency vector $\tilde{\mathbf{U}}_{y,k}$. These assumptions allow us to formally model the probability of observing $\mathbf{U}$ and $\boldsymbol{\mu}$ given $\tilde{\mathbf{U}}$:

$$\begin{aligned}\Pr(\mathbf{U}, \boldsymbol{\mu} | \tilde{\mathbf{U}}, \mathbf{P}) &= \Pr(\boldsymbol{\mu}) \cdot \Pr(\mathbf{U} | \tilde{\mathbf{U}}, \boldsymbol{\mu}, \mathbf{P}) \\ &= \Pr(\boldsymbol{\mu}) \cdot \prod_{k=1}^{\rho} \Pr(\mathbf{U}_k | \tilde{\mathbf{U}}_k, \mu_k, \boldsymbol{P}) \\ &= \Pr(\boldsymbol{\mu}) \cdot \prod_{k=1}^{\rho} \prod_{y=1}^{N_D} \Pr(\mathbf{U}_{y,k} | \tilde{\mathbf{U}}_{y,k}, \mu_{y,k}, \boldsymbol{P})\end{aligned}$$

Because of our assumption mentioned above, we can apply the probability mass function (PMF) of the multinomial distribution to express the probability as an explicit formula:

$$\prod_{k=1}^{\rho}\prod_{y=1}^{N_D}\mu_{y,k}!\prod_{j=1}^{m}\frac{(\tilde{U}_{p(j),y,k})^{\mu_{y,k}U_{j,y,k}}}{(\mu_{y,k}U_{j,y,k})!} \tag{3}$$

We now take the logarithm of Equation 3 to obtain our update frequency cost matrix. We can ignore any terms which are not dependent on $\mathbf{P}$ because they do not affect the optimization problem.

$$\sum_{k=1}^{\rho}\sum_{y=1}^{N_D}\sum_{j=1}^{m}\mu_{y,k}U_{j,y,k}\cdot\log(\tilde{U}_{p(j),y,k})$$

We are now ready to construct the cost matrix. The entry in the (i,j)th position is:

$$(\mathbf{C}_{\mathsf{UFID}})_{i,j} = -\sum_{k=1}^{\rho}\sum_{y=1}^{N_D}\mu_{y,k}U_{j,y,k}\cdot\log(\tilde{U}_{i,y,k})$$

4.2 Level II Backward Privacy

The UFID Attack becomes unusable in the presence of backward privacy, because the file identifier information is no longer leaked. However, Level II and III backward-private schemes still leak the frequency of updates on a keyword w from the post-search update leakage Updates(w). We refer to the attack constructed from the methodology described below as the UF Attack.

The assumptions are analogous to the UFID Attack: assume the client chooses the keyword w independently from any other updates with probabilities given by $\tilde{\Omega}$. The number of updates for each tag in time interval $k \in \rho$ follows a multinomial distribution with μ_k trials and probabilities given by frequency vector $\tilde{\boldsymbol{\omega}}_k$. Therefore, the probability of observing Ω and $\boldsymbol{\mu}$ given $\tilde{\Omega}$:

$$\begin{aligned}\Pr(\Omega,\boldsymbol{\mu}|\tilde{\Omega},\mathbf{P}) &= \Pr(\boldsymbol{\mu})\cdot\Pr(\Omega|\tilde{\Omega},\boldsymbol{\mu},\mathbf{P})\\ &= \Pr(\boldsymbol{\mu})\cdot\prod_{k=1}^{\rho}\Pr(\boldsymbol{\omega}_k|\tilde{\boldsymbol{\omega}}_k,\mu_k,\mathbf{P})\end{aligned}$$

Again, we apply the PMF of the multinomial distribution to achieve an explicit equation:

$$= \Pr(\boldsymbol{\mu})\cdot\prod_{k=1}^{\rho}\mu_k!\prod_{j=1}^{m}\frac{(\tilde{\omega}_{p(j),k})^{\mu_k\omega_{j,k}}}{(\mu_k\omega_{j,k})!}$$

Now we are ready to take the logarithm, ignoring terms independent of $\mathbf{P}$:

$$\sum_{k=1}^{\rho}\sum_{j=1}^{m}\mu_k\omega_{j,k}\cdot\log(\tilde{\omega}_{p(j),k})$$

We can now create the cost matrix for update frequency information:

$$(\mathbf{C}_{\mathsf{UF}})_{i,j} = -\sum_{k=1}^{\rho}\mu_k\omega_{j,k}\cdot\log(\tilde{\omega}_{i,k})$$

As described, the UF Attack will work on schemes with Level II backward privacy or worse. Because this attack uses time interval information, it will fail on Level I backward-private schemes because they do not leak timestamps.

4.3 Exploiting Both Search and Update Leakage

While the UFID and UF Attacks against schemes with at least forward privacy require the client to issue searches in order to obtain the necessary post-search update leakage, they only incorporate the update leakage into their cost matrices. From the adversary's perspective, it is desirable to combine the search frequency information with the update frequency information to increase the potential accuracy of query recovery attacks. From the searches that the adversary observes, she can construct $\mathbf{C}_f$. From the post-search update leakage, she can construct $\mathbf{C}_{\mathsf{UF}}$ in the presence of Level II backward privacy or $\mathbf{C}_{\mathsf{UFID}}$ if there is no backward privacy. These matrices can be combined with element-by-element multiplication, also known as the Hadamard product. For two independent events A and B with probabilities given by matrices $P(A)$ and $P(B)$, the Hadamard product matrix given by $(P(A)\odot P(B))_{ij} = (P(A))_{ij}(P(B))_{ij}$ can be statistically interpreted as the joint probability $P(A\cap B)$ of both events occurring. If we assume the occurrence of a search does not affect the probability of an update occurring, we can treat the cost matrices as independent and apply the Hadamard product. For schemes with no backward privacy:

$$\mathbf{C}_{\mathsf{SP+UFID}} = \mathbf{C}_f \odot \mathbf{C}_{\mathsf{UFID}}$$

For schemes with Level II backward privacy:

$$\mathbf{C}_{\mathsf{SP+UF}} = \mathbf{C}_f \odot \mathbf{C}_{\mathsf{UF}}$$

This yields a new version of each attack, referred to as the SP+UFID Attack and SP+UF Attack, respectively. See Section 5.3 for our empirical evaluation of the effectiveness of these versions of the attacks compared to their baseline versions.

Following Oya and Kerschbaum [28], we express the maximization as an unbalanced assignment problem, which is efficiently solved by the Hungarian Algorithm. If $\mathrm{tr}(\mathbf{A})$ is the trace of matrix $\mathbf{A}$, the assignment problem for all attacks described in this section is as follows:

$$\mathbf{P} = \underset{\mathbf{P}\in\mathcal{P}}{\operatorname{argmin}}\ \mathrm{tr}(\mathbf{P}^T(\mathbf{C})),$$

where $\mathbf{C}$ is the cost matrix of the corresponding attack.

5 EVALUATION

We evaluate our proposed attack methods against a range of experiments and compare the results with the search frequency portion of the SAP attack by Oya and Kerschbaum [28], described in Section 3.3 and hereafter referred to as the Search Frequency Attack.

Implementation. We implement our experiments using roughly 1,000 lines of single-threaded Python 3.6.8 code on a server running CentOS Stream 8 with 125 GB of RAM and an Intel(R) Xeon(R) Platinum 8630Y CPU with 2.4 GHz clock speed. We use NumPy's implementation of probabilistic functions and SciPy's implementation of the Hungarian algorithm. Our code is available at https://github.com/vt-asaplab/DSSE-Attacks.

5.1 Methodology

Experiment Setting. We re-implement the Search Frequency Attack and implement our proposed attacks under new assumptions. Oya et al. demonstrated their SAP attack under the assumption that the adversary has fine-grained probabilities about the client's

query frequencies for all of the keywords that the adversary is interested in. We feel this assumption is unlikely to arise in a real-world scenario for two reasons.

First, the adversary must explicitly develop a *predicted keyword universe* for all of the keywords that she believes the client will query. Then the adversary must gather auxiliary information about each keyword in Δ_{pred}. As Damie et al. [10] point out, any attack operating under this model will be unable to recover any keywords which are contained in the client's *true keyword universe* Δ_{real} but absent from Δ_{pred}. We claim that constructing a good Δ_{pred} is difficult in practice except in the case of a security breach (i.e., a scheme that was set up incorrectly but later patched, leaking some query traces) or the adversary already has significant information about the client's database (perhaps through a side-channel attack or surveillance). The latter case is outside the scope of this work.

Second, if the auxiliary probabilities $\tilde{\mathbf{F}}$ selected by the adversary do not align closely with the underlying query behavior $\mathbf{F}$ of the client, the attack will be less likely to recover keywords. We feel that taking auxiliary data from Google Trends is unlikely to accurately represent the query behavior of clients of SSE services, especially if the client is querying over a specialized database.

For these reasons, we implement all experiments under the assumption that the adversary has access to some real query and update traces from the client (i.e., the security breach scenario). Although access to real query and update traces is a strong assumption, we feel that it is more likely to occur in practice than Google Trends data closely predicting the behavior of real SSE clients. We note that this is simply the strategy we use to create auxiliary data to simulate a realistic setting; our attacks still function as similar-data inference attacks and do not require known data.

Generating the Auxiliary Data. Because there are no publicly available datasets including real timestamped client query or update information, we generate auxiliary client query and update traces using data from Google Trends. We use the same Google Trends probabilities as the Oya et al. evaluation [28] to keep our evaluation consistent with theirs[1]. For each of the 3000 most common keywords in each dataset, the frequency is obtained from Google Trends for each week in a 260-week span. These relative frequencies are then normalized and used for auxiliary data generation.

Because we only consider forward-private schemes, we remove all updates from the auxiliary and observed traces if their corresponding keyword or tag is not searched in a later week. We model the client's choice of which file identifier id to associate an update with by sampling from a normal distribution with center c and standard deviation d. We assume each keyword has a unique center c. d is a parameter of the experiment. $d = 0$ implies that a keyword is always updated into the same file. The choice of which file to update becomes more random as d increases. There are many other valid ways to simulate client updates, but we are unable to use real-world data for this choice due to the lack of publicly available database-update datasets including timestamps and file identifiers.

Processing the Auxiliary Data. All experiments are conducted using 100 time intervals of search and update traces. For each keyword w_i in Δ, we repeat the following procedure: in the first time interval, we take its normalized frequency value from the the first week in the Google Trends dataset and use it as the chance that w_i will appear in the first week of query or update traces. This process is repeated for the remaining 99 weeks.

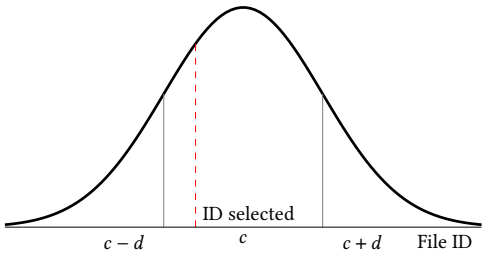


Figure 1: Diagram of how the client chooses which file to update. Each keyword has a different center c. Standard deviation d is a parameter of the experiment. The dashed line represents a random sample from the distribution. The client will choose the file ID id **at this location to include in the update tuple** $(\text{op}, w, \text{id})$**. c and d are not known to the adversary.**

At this point, 100 weeks of query and update traces have been generated. We refer to the first 50 weeks as the auxiliary period because the traces from these weeks will be left in plaintext, as in the security breach scenario described previously. The adversary will use them to populate the auxiliary frequency matrices $\tilde{\mathbf{F}}$, $\tilde{\Omega}$, and $\tilde{\mathbf{U}}$.

For the next 50 weeks, the server may only access traces which have been processed to reveal no more information than a correct SSE scheme, perhaps due to a security patch. This interval is known as the observation period. The adversary never has access to any keyword-tag pairs; she only sees keyword traces during the auxiliary period, and only sees tokenized traces during the observation period. Our approach differs from Oya et al. because we do not allow the adversary to access any of the Google Trends data, only the auxiliary traces.

The adversary effectively has access to outdated frequency information because the attack goal is to use the data from the auxiliary period to recover queries in the observation period. We note that the adversary only has access to keywords in the auxiliary period and only has access to tags in the observation period, and therefore never has access to ground truth information.

Attack Accuracy. For each experiment, accuracy of all attacks is measured by the total number of searches recovered divided by the total number of searches, including duplicates, in the observation period. This is known as weighted query recovery, and is the standard accuracy metric used by attack papers [2, 5, 10, 28, 29].

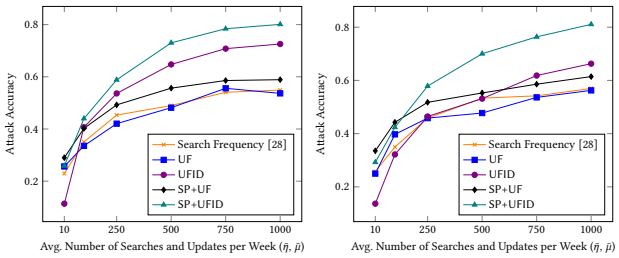
5.2 Experimental Setup

Dataset. To evaluate our attack, we use the 30,109 files in the 'sent-mail' directory of the Enron dataset[2], as is standard in the field [10, 28, 29, 40]. We also use the first 66,491 files after September 2001 of the Lucene dataset[3] for the first experiment. After removal of English stopwords, we pick a random subset of size n of the 3,000

[1] From https://github.com/simon-oya/USENIX21-sap-code/tree/master/datasets_pro

[2] https://www.cs.cmu.edu/~enron/

[3] https://mail-archives.apache.org/mod_mbox/lucene-java-user/



(a) Enron dataset (b) Lucene dataset

Figure 2: Effect of increasing number of searches and updates per week in both the auxiliary and observation periods.

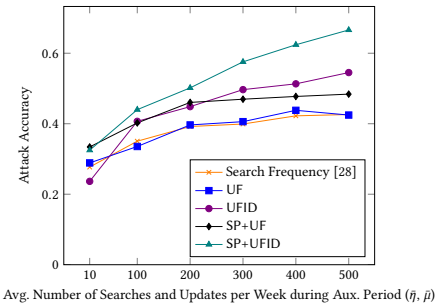
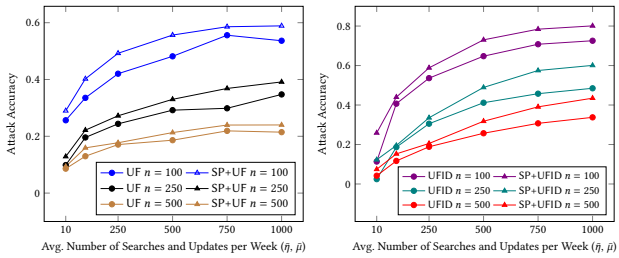


Figure 3: Effect of increasing auxiliary data on attack accuracy. During the auxiliary period, $\bar{\eta}$ and $\bar{\mu}$ are set to the values shown on the x-axis. During the observation period, $\bar{\eta}$, $\bar{\mu} = 100$ for all experiments.



(a) UF Attacks (b) UFID Attacks

Figure 4: Effect of keyword universe size on attack accuracy.

most commonly appearing keywords in the dataset to build our keyword universe Δ.

Experiment Parameters. We run each experiment with settings of keyword universe size $n = 100$, average searches and updates per week of $\bar{\eta}, \bar{\mu} = 100$, number of weeks for the auxiliary and observation periods each $\rho = 50$, number of files in the dataset $N_D =$ 30,109, and standard deviation of the distribution that selects files $d = 0.0033N_D$ ($\approx$100 for $N_D =$ 30,109). Except for the independent variable, these values are used for all experiments.

Following Oya et al., we generate the actual number of searches and updates per week by sampling from a Poisson distribution centered at $\bar{\eta}$ and $\bar{\mu}$, respectively.

To increase the confidence in our results, we run each experiment 30 times, using a different random seed for each. The seed affects which keywords are selected to be in Δ, which file identifiers are selected by the process simulating client updates (Figure 1), and the actual values for η and μ for each week. After the inputs are selected, the attacks are deterministic. For each data point in the following figures, we report the average result of the 30 trials.

5.3 Experimental Results

Amount of Auxiliary Information. Figure 2 shows that the accuracy of all attacks increases as the average number of searches and updates per week increases. We note that our implementations of the attacks require more searches per week to obtain the same accuracy reported by Oya and Kerschbaum. This is due to the change in attack setting; because our adversary only has access to auxiliary traces, the client must issue more searches before the adversary can get an accurate prediction of the true probability that a client will query each keyword.

The UF Attack achieves similar accuracy to the Search Frequency Attack. This is because update frequency alone is no more powerful than search frequency leakage. The SP+UF Attack achieves roughly five percent higher accuracy at all intervals. Both versions of the UFID Attack perform better on average than the other attacks, although the magnitude of this improvement increases as the amount of information increases. We conjecture that this is because adding more information is more likely to fill empty spaces in $\mathbf{U}$ and $\tilde{\mathbf{U}}$ when compared to Ω and $\tilde{\Omega}$, because the UFID Attacks track frequencies at a finer granularity.

In Figure 2(b), we run our attacks against the Lucene dataset to verify the effectiveness of our attacks does not arise from some unique quality of the Enron dataset. From Enron to Lucene, the accuracy of the Search Frequency Attack is stable, whereas our attacks lose a small but noticeable amount of accuracy. This is likely because the Lucene dataset has many more files than Enron, which makes our matrices Ω and $\mathbf{U}$ more sparse. However, it is clear that all attacks shown are effective against both datasets.

Figure 3 shows that increasing the amount of auxiliary information can significantly improve the accuracy of attacks, even if the amount of searches and updates observed by the adversary does not increase.

In Figure 4, we compare our attacks across different keyword universe sizes. As with all attacks against SSE, we notice a decrease in accuracy as more keywords are included. We note that the keyword universe does not have to be as large as the entire set of keywords in the database, but instead can be hand-selected by the adversary. While selecting keywords that the target client is likely to query is a hard problem in practice, it could reduce the size of the keyword universe to dozens or hundreds.

Number of Files in Database. Figure 5 shows the runtime scaling of the UFID Attacks with respect to N_D when all other variables are held constant. We define runtime as the amount of time it takes to build the cost matrix and optimize it with the Hungarian algorithm[4]. The size of the UFID Attacks' data structures is larger than their UF Attack counterparts by a factor of N_D. This figure provides

[4]UFID Attack is not shown in Figure 5 because the difference in runtime between versions is negligible.

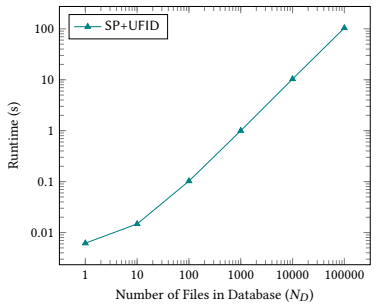
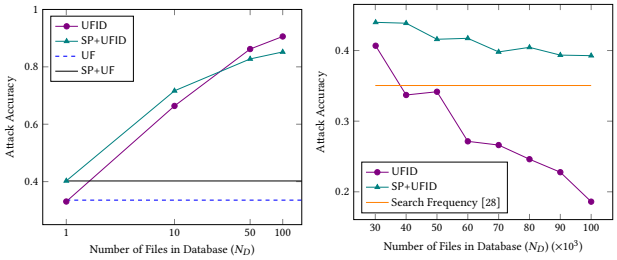


Figure 5: Effect of database size on runtime of UFID Attacks.



(a) Small database (b) Large database

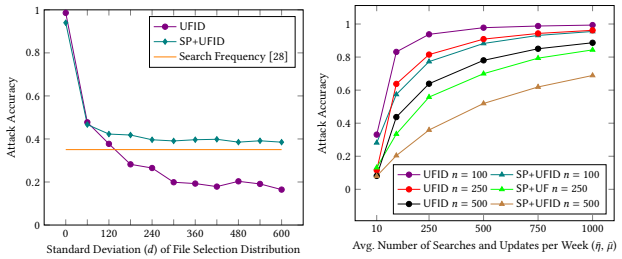
Figure 6: Effect of adding files to a database on accuracy of UFID Attacks.

experimental verification that the UFID Attacks run in reasonable time for realistic experiment parameters.

Figure 6(a) experimentally verifies that the UF Attack is a special case of the UFID Attack where there is only one file in the database. This is expected because the adversary does not have file identifier information in Level II backward-private schemes. Starting from a database with only one file, the adversary begins to learn more useful information from the file identifier leakage as the number of files increases.

Figure 6(b) shows that adding more files to the database will eventually degrade the accuracy of the UFID Attacks. We argue that the value of the file identifier information decreases if there are too few or too many files in the database because the relationship between keywords and file identifiers in updates becomes less unique. The SP+UFID Attack maintains an improvement over search pattern leakage alone, even for databases with 100,000 files.

File ID Distribution. Figure 7 investigates the effect that the choice of which file to update has on the accuracy of the UFID Attacks. Recall that we simulate this choice by sampling from a normal distribution as described in Figure 1. We do not expect a normal distribution to accurately model real-world client updates; indeed, one can imagine real-world scenarios where client behavior would be better modeled by other distributions. Instead, we use the normal distribution to provide experimental insight about the effect of the *uniqueness* of the update distribution. For both versions of the attack, we find that very low values of d result in very high recovery rates. On the other hand, a standard deviation of 600 implies that 95% of the updates occur over a range of 2400 different file identifiers, or 8% of the database size N_D. Even under these conditions, the SP+UFID



(a) Normal distribution (b) Zipfian distribution

Figure 7: Effect of file selection distribution on accuracy of UFID attacks.

Attack still slightly outperforms the Search Frequency Attack. These results suggest that the relationship between keywords and the files that are updated is an important factor in these attacks.

In Figure 7(b), unlike all other experiments, the client chooses the file identifier to associate with a keyword by sampling from a Zipfian distribution with a parameter of $a = 1.1$. We assume the most frequently updated file is different for each keyword. The Zipfian distribution leads to much more pronounced couplings of keywords and file identifiers than the normal distribution, causing the attacks to be remarkably effective under these conditions. We point out that the SP+UFID Attack is less effective than the UFID Attack, presumably because adversary's information about the client's updates is of such high quality with this distribution that including the search frequencies is no longer advantageous.

5.4 Discussion

Practicality of Attacks. Our results suggest that DSSE use cases with frequent updates and at least semi-frequent searches, such as encrypted email, could be very vulnerable to our attacks. Other use cases that are infrequently accessed by the client, such as backup file storage, are less likely to be compromised by our attacks.

Any DSSE scheme is likely to receive some amount of updates from the client, otherwise the user would simply opt for a static SSE scheme instead. As such, we encourage future works to consider post-search update leakage matching the security properties of the target scheme to be part of the standard leakage in their analysis. We encourage deeper investigation into techniques to compromise DSSE schemes so defense researchers are more capable of empirically evaluating the strength of their defenses against attacks that explicitly target DSSE.

In practice, the major obstacle for an attacker is obtaining auxiliary data. Most attack papers only offer a short explanation about how an attacker might obtain known documents, queries, or auxiliary data. We agree with the sentiment of Damie et al. [11] that high-quality ground truth data, or even statistically-similar auxiliary data, is difficult to obtain in practice, even in the case of a security breach. However, as we argued earlier, a security breach is one of the only practical ways for an attacker to form a comprehensive keyword universe Δ.

In this work, we have made our best attempt to evaluate our frequency-based attacks under realistic conditions, following strategies from related works [27–29]. However, as Damie et al. [10] point

out, it is difficult to evaluate frequency-based attacks without real timestamped search and update datasets[5].

Nevertheless, our attacks expose weaknesses in DSSE schemes which had not previously been exploited. It would be unwise to dismiss leakage-abuse attacks as impractical for three reasons. First, even attacks which are only theoretical motivate stronger designs [13, 17, 19, 42]. Second, the security versus performance balance of real commercial-scale systems are unlikely to lean in the direction of the security properties that provably defeat ours and related attacks. Third, the complexities of implementing large systems, such as those that led to the recent devastating attacks by Gui et al. [20] targeting the logging system of MongoDB's Queryable Encryption, can greatly assist an adversary in gaining auxiliary data to compromise the privacy of SSE protocols.

Countermeasures. A growing line of research on SSE defenses have focused on obfuscating access patterns and search patterns [8, 13, 30]. While effective, very few of these works extend their proposed methods to construct dynamic SSE schemes, generally because it is challenging to do so under real-world efficiency constraints [13]. Zhao et al. [42] provide a Level I backward-private scheme that obfuscates search patterns, however, its correctness is inversely proportional to the strength of the security parameter and the authors do not provide an empirical evaluation of its performance. The SWiSSSE scheme designed by Gui et al. [19] appears to be the most efficient (and only end-to-end) defense to our attacks. A DSSE scheme instantiated with frequency-smoothing techniques like PANCAKE by Grubbs et al. [18] would defeat the attacks proposed in this paper, but potentially not all attacks that could be designed with similar methods (e.g., a dynamic extension of IHOP [29] that includes update leakage). As such, attackers can use update leakage to compromise all efficient DSSE schemes more effectively than attacks that ignore this leakage. For this reason, we encourage researchers to investigate techniques to improve the practical efficiency of Level I backward privacy for DSSE.

Directions for Future Work. It is possible to incorporate the post-search update leakage from Level I backward-private schemes into an attack. Recall that the post-search update leakage of Level I schemes is the time that each document matching w was inserted into the database and the total number of updates on w. It would be challenging to exploit this information effectively without a stronger attack model. We leave this as an open research question.

Our attacks do not exploit all possible update leakage. For Level III schemes, one could utilize the deletion information from DelHist. For weaker schemes, it is possible to exploit the type of operation op. Motivated by the results of our work and others [28, 29], we are of the opinion that including more information to attacks will generally improve their accuracy. We encourage attack researchers to pursue this line of research further to incorporate update leakage from DSSE schemes into existing attacks and new attacks alike.

While leakage-obfuscation defenses have proven effective against a variety of attacks, there appears to be a heavier focus on volume-hiding [1, 19, 23, 30, 42] than search pattern obfuscation [19, 42]. While effectively obfuscating the search pattern is an open challenge [4, 28], it would defeat our attacks and several others [27–29, 39], so we believe it is a worthwhile direction to pursue further.

[5]In fact, almost all SSE attack evaluations, even those for non-frequency-based attacks, choose to model client query behavior due to the lack of real-world data.

6 CONCLUSION

In this work, we propose two novel query-recovery attacks and demonstrate that including traditionally-overlooked leakage from updates in Dynamic Searchable Symmetric Encryption (DSSE) can compromise query privacy more effectively than a strategy that only uses leakage from searches. We show that the backward privacy security property prevents the strongest versions of these attacks. We hope our work encourages researchers to improve the efficiency of backward privacy so clients of future real-world encrypted search systems can benefit from enhanced privacy.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their feedback which improved the quality of this work. This material is based upon work supported by the National Science Foundation under Grants Number 194649, an unrestricted gift from Robert Bosch, 4-VA, and the Commonwealth Cyber Initiative (CCI) — an investment in the advancement of cyber R&D, innovation, and workforce development. For more information about CCI, visit www.cyberinitiative.org.

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. Privacy Enhancing Technologies Symposium, Vol. 2023. 417–436. https://doi.org/10.56553/popets-2023-0025

[2] Laura Blackstone, Seny Kamara, and Tarik Moataz. 2020. Revisiting Leakage Abuse Attacks. ISOC Network and Distributed System Security Symposium. In *NDSS 2020*. https://dx.doi.org/10.14722/ndss.2020.23103

[3] Raphael Bost. 2016. Sophos: Forward Secure Searchable Encryption. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 1143–1154. https://doi.org/10.1145/2976749.2978303

[4] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. 2017. Forward and Backward Private Searchable Encryption from Constrained Cryptographic Primitives. 1465–1482. https://doi.org/10.1145/3133956.3133980

[5] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. 2016. Leakage-Abuse Attacks Against Searchable Encryption. In *15th ACM Conference on Computer and Communications Security (CCS)*. https://eprint.iacr.org/2016/718.pdf

[6] David Cash, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Roşu, and Michael Steiner. 2013. Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries. In *Advances in Cryptology – CRYPTO 2013*, Ran Canetti and Juan A. Garay (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 353–373.

[7] Javad Ghareh Chamani, Dimitrios Papadopoulos, Mohammadamin Karbasforushan, and Ioannis Demertzis. 2022. Dynamic Searchable Encryption with Optimal Search in the Presence of Deletions. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 2425–2442. https://www.usenix.org/conference/usenixsecurity22/presentation/chamani

[8] Guoxing Chen, Ten-Hwang Lai, Michael K. Reiter, and Yinqian Zhang. 2018. Differentially Private Access Patterns for Searchable Symmetric Encryption. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications* (Honolulu, HI, USA). IEEE Press, 810–818. https://doi.org/10.1109/INFOCOM.2018.8486381

[9] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. 2006. Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions. In *13th ACM Conference on Computer and Communications Security (CCS)*. https://eprint.iacr.org/2006/210.pdf

[10] Marc Damie, Florian Hahn, and Andreas Peter. 2021. A Highly Accurate Query-Recovery Attack against Searchable Encryption using Non-Indexed Documents. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 143–160. https://www.usenix.org/conference/usenixsecurity21/presentation/damie

[11] Marc Damie, Jean-Benoist Leger, Florian Hahn, and Andreas Peter. 2023. The statistical nature of leakage in SSE schemes and its role in passive attacks.

Cryptology ePrint Archive, Paper 2023/1883. https://eprint.iacr.org/2023/1883 https://eprint.iacr.org/2023/1883.

[12] Ioannis Demertzis, Javad Ghareh Chamani, Dimitrios Papadopoulos, and Charalampos Papamanthou. 2020. Dynamic Searchable Encryption with Small Client Storage. In *NDSS 2020*. https://doi.org/10.14722/ndss.2020.24423

[13] Ioannis Demertzis, Dimitrios Papadopoulos, Charalampos Papamanthou, and Saurabh Shintre. 2020. SEAL: Attack Mitigation for Encrypted Databases via Adjustable Leakage. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 2433–2450. https://www.usenix.org/conference/usenixsecurity20/presentation/demertzis

[14] Ioannis Demertzis, Stavros Papadopoulos, Odysseas Papapetrou, Antonios Deligiannakis, Minos Garofalakis, and Charalampos Papamanthou. 2018. Practical Private Range Search in Depth. *ACM Trans. Database Syst.* 43, 1, Article 2 (mar 2018), 52 pages. https://doi.org/10.1145/3167971

[15] Mohammad Etemad, Alptekin Küpçü, Charalampos Papamanthou, and David Evans. 2017. Efficient Dynamic Searchable Encryption with Forward Privacy. arXiv:1710.00208 [cs.CR]

[16] Sanjam Garg, Payman Mohassel, and Charalampos Papamanthou. 2016. TWORAM: Efficient Oblivious RAM in Two Rounds with Applications to Searchable Encryption. In *Proceedings, Part III, of the 36th Annual International Cryptology Conference on Advances in Cryptology — CRYPTO 2016 - Volume 9816*. Springer-Verlag, Berlin, Heidelberg, 563–592. https://doi.org/10.1007/978-3-662-53015-3_20

[17] Javad Ghareh Chamani, Dimitrios Papadopoulos, Charalampos Papamanthou, and Rasool Jalili. 2018. New Constructions for Forward and Backward Private Symmetric Searchable Encryption. 1038–1055. https://doi.org/10.1145/3243734.3243833

[18] 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 *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 2451–2468. https://www.usenix.org/conference/usenixsecurity20/presentation/grubbs

[19] Zichen Gui, Kenneth G. Paterson, Sikhar Patranabis, and Bogdan Warinschi. 2020. SWiSSSE: System-Wide Security for Searchable Symmetric Encryption. Cryptology ePrint Archive, Paper 2020/1328. https://eprint.iacr.org/2020/1328 https://eprint.iacr.org/2020/1328.

[20] Zichen Gui, Kenneth G. Paterson, and Tianxin Tang. 2023. Security Analysis of MongoDB Queryable Encryption. In *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, Anaheim, CA, 7445–7462. https://www.usenix.org/conference/usenixsecurity23/presentation/gui

[21] Kun He, Jing Chen, Qinxi Zhou, Ruiying Du, and Yang Xiang. 2021. Secure Dynamic Searchable Symmetric Encryption With Constant Client Storage Cost. *IEEE Transactions on Information Forensics and Security* 16 (2021), 1538–1549. https://doi.org/10.1109/TIFS.2020.3033412

[22] Mohammad Islam, Mehmet Kuzu, and Murat Kantarcioglu. 2012. Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation. In *NDSS 2012*. https://www.ndss-symposium.org/wp-content/uploads/2017/09/06_1.pdf

[23] Seny Kamara and Tarik Moataz. 2019. Computationally Volume-Hiding Structured Encryption. 11477 (2019), 183–213. https://doi.org/10.1007/978-3-030-17656-3_7

[24] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. 2012. Dynamic Searchable Symmetric Encryption. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (Raleigh, North Carolina, USA) *(CCS '12)*. Association for Computing Machinery, New York, NY, USA, 965–976. https://doi.org/10.1145/2382196.2382298

[25] Aggelos Kiayias, Ozgur Oksuz, Alexander Russell, Qiang Tang, and Bing Wang. 2016. Efficient Encrypted Keyword Search for Multi-user Data Sharing. In *Computer Security – ESORICS 2016*, Ioannis Askoxylakis, Sotiris Ioannidis, Sokratis Katsikas, and Catherine Meadows (Eds.). Springer International Publishing, Cham, 173–195.

[26] Mehmet Kuzu, Mohammad Saiful Islam, and Murat Kantarcioglu. 2012. Efficient Similarity Search over Encrypted Data. In *2012 IEEE 28th International Conference on Data Engineering*. 1156–1167. https://doi.org/10.1109/ICDE.2012.23

[27] Chang Liu, Liehuang Zhu, Mingzhong Wang, and Yu an Tan. 2014. Search pattern leakage in searchable encryption: Attacks and new construction. *Information Sciences* 265 (2014), 176–188. https://doi.org/10.1016/j.ins.2013.11.021

[28] Simon Oya and Florian Kerschbaum. 2021. Hiding the Access Pattern is Not Enough: Exploiting Search Pattern Leakage in Searchable Encryption. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 127–142. https://www.usenix.org/conference/usenixsecurity21/presentation/oya

[29] Simon Oya and Florian Kerschbaum. 2022. IHOP: Improved Statistical Query Recovery against Searchable Symmetric Encryption through Quadratic Optimization. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 2407–2424. https://www.usenix.org/conference/usenixsecurity22/presentation/oya

[30] 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 *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (London, United Kingdom) *(CCS '19)*. Association for Computing Machinery, New York, NY, USA, 79–93. https://doi.org/10.1145/3319535.3354213

[31] David Pouliot and Charles V. Wright. 2016. The Shadow Nemesis: Inference Attacks on Efficiently Deployable, Efficiently Searchable Encryption. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 1341–1352. https://doi.org/10.1145/2976749.2978401

[32] Cédric Van Rompay, Refik Molva, and Melek Önen. 2018. Secure and Scalable Multi-User Searchable Encryption. Cryptology ePrint Archive, Paper 2018/090. https://eprint.iacr.org/2018/090 https://eprint.iacr.org/2018/090.

[33] Khosro Salmani and Ken Barker. 2021. Don't Fool Yourself with Forward Privacy, Your Queries STILL Belong to Us!. In *Proceedings of the Eleventh ACM Conference on Data and Application Security and Privacy* (Virtual Event, USA) *(CODASPY '21)*. Association for Computing Machinery, New York, NY, USA, 131–142. https://doi.org/10.1145/3422337.3447838

[34] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. 2000. Practical Techniques for Searches on Encrypted Data. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy (SP '00)*. IEEE Computer Society, USA, 44.

[35] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. 2014. Practical Dynamic Searchable Encryption with Small Leakage. In *NDSS 2014* (San Diego, CA, USA). http://dx.doi.org/10.14722/ndss.2014.23298

[36] Qiang Tang. 2014. Nothing is for Free: Security in Searching Shared & Encrypted Data. Cryptology ePrint Archive, Paper 2014/362. https://eprint.iacr.org/2014/362 https://eprint.iacr.org/2014/362.

[37] Jianfeng Wang, Hua Ma, Qiang Tang, Jin Li, Hui Zhu, Siqi Ma, and Xiaofeng Chen. 2013. Efficient Verifiable Fuzzy Keyword Search over Encrypted Data in Cloud Computing. *Computer Science and Information Systems* 10 (04 2013), 667–684. https://doi.org/10.2298/CSIS121104028W

[38] Qian Wang, Meiqi He, Minxin Du, Sherman S. M. Chow, Russell W. F. Lai, and Qin Zou. 2018. Searchable Encryption over Feature-Rich Data. *IEEE Transactions on Dependable and Secure Computing* 15, 3 (2018), 496–510. https://doi.org/10.1109/TDSC.2016.2593444

[39] Lei Xu, Leqian Zheng, Chengzhi Xu, Xingliang Yuan, and Cong Wang. 2023. Leakage-Abuse Attacks Against Forward and Backward Private Searchable Symmetric Encryption. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security* (Copenhagen, Denmark) *(CCS '23)*. Association for Computing Machinery, New York, NY, USA, 3003–3017. https://doi.org/10.1145/3576915.3623085

[40] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. 2016. All Your Queries Are Belong to Us: The Power of File-Injection Attacks on Searchable Encryption. In *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, Austin, TX, 707–720. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/zhang

[41] Zhongjun Zhang, Jianfeng Wang, Yunling Wang, Yaping Su, and Xiaofeng Chen. 2019. Towards Efficient Verifiable Forward Secure Searchable Symmetric Encryption. In *Computer Security – ESORICS 2019: 24th European Symposium on Research in Computer Security, Luxembourg, September 23–27, 2019, Proceedings, Part II* (Luxembourg, Luxembourg). Springer-Verlag, Berlin, Heidelberg, 304–321. https://doi.org/10.1007/978-3-030-29962-0_15

[42] Yongjun Zhao, Huaxiong Wang, and Kwok-Yan Lam. 2021. Volume-Hiding Dynamic Searchable Symmetric Encryption with Forward and Backward Privacy. Cryptology ePrint Archive, Paper 2021/786. https://eprint.iacr.org/2021/786 https://eprint.iacr.org/2021/786.

[43] Cong Zuo, Shi-Feng Sun, Joseph K. Liu, Jun Shao, and Josef Pieprzyk. 2019. Dynamic Searchable Symmetric Encryption with Forward and Stronger Backward Privacy. Cryptology ePrint Archive, Paper 2019/1055. https://eprint.iacr.org/2019/1055 https://eprint.iacr.org/2019/1055.