Leakage-Abuse Attacks Against Searchable Encryption

David Cash
Rutgers University
110 Frelinghuysen Road
Piscataway, NJ 08854
david.cash@cs.rutgers.edu

Paul Grubbs
Skyhigh Networks Inc. & Cornell University
Ithaca, NY 14853
pag225@cornell.edu

Jason Perry*
Lewis University
One University Parkway
Romeoville, IL 60446
perryjn@lewisu.edu

Thomas Ristenpart
Cornell Tech
111 8th Avenue #302
New York, NY 10011
ristenpart@cornell.edu

ABSTRACT

Schemes for secure outsourcing of client data with search capability are being increasingly marketed and deployed. In the literature, schemes for accomplishing this efficiently are called Searchable Encryption (SE). They achieve high efficiency with provable security by means of a quantifiable leakage profile. However, the degree to which SE leakage can be exploited by an adversary is not well understood.

To address this, we present a characterization of the leakage profiles of in-the-wild searchable encryption products and SE schemes in the literature, and present attack models based on an adversarial server's prior knowledge. Then we empirically investigate the security of searchable encryption by providing query recovery and plaintext recovery attacks that exploit these leakage profiles. We term these *leakage-abuse attacks* and demonstrate their effectiveness for varying leakage profiles and levels of server knowledge, for realistic scenarios. Amongst our contributions are realistic *active* attacks which have not been previously explored.

Keywords

Searchable encryption; leakage

1. INTRODUCTION

Encryption protects data stored at an untrusted service provider, but introduces complications. Amongst these, the service provider is unable to process the encrypted data as freely and efficiently as it can plaintext data, making access cumbersome for the data owner. To address the difficulty of retrieving encrypted text efficiently, increasingly practitioners turn to *searchable encryption (SE)* schemes, first introduced by Song, Wagner, and Perrig [19]. An SE scheme encrypts a set of documents in a way that allows the data owner to delegate search capabilities to the provider without decrypting the documents. For example, using SE, a user may encrypt her email, store it at the remote provider, and later have the provider fulfill keyword search queries. If the SE scheme is *dynamic*, then she will also be able to add encrypted documents efficiently [12].

Many SE constructions can be implemented using only symmetric cryptography, and some are even legacy-compatible, requiring no modification at the provider. A commonly deployed example of the latter, and one suggested in recent research papers [9, 15], is to append to a conventional encryption of the document a sequence of outputs of a secret-keyed pseudorandom function (PRF) on individual keywords. Search is performed by submitting the PRF output of the desired search term, and documents can easily be added or removed.

All efficient SE constructions expose some information, called *leakage*, about the plaintext to the service provider. Typically, SE schemes allow the provider to learn about the underlying data by observing statistics like the number and frequency of encrypted documents accessed during searches. Song et al. recommended periodic re-encryption to address what they call *statistical attacks*, but did not investigate this further. Islam, Kuzu, and Kantarcioglu [10] (IKK) initiated the study of the empirical security of SE by showing that a user's keyword search queries can be guessed with high accuracy when used on a dataset known to an honest-but-curious service provider. Their attacks, and the more general statistical attacks alluded to by Song et al., are possible even when a scheme has been proven secure under a standard assumption. In other words, the attacks are permitted by the security definition.

The risk posed by leakage has not been closely scrutinized beyond the work of IKK. The current literature leaves a practitioner with few concrete recommendations for configuring and deploying SE. The risk is not merely abstract; several deployments of SE may be easily broken depending on how they are used.

Our contributions. The present paper studies the leakage of SE in order to understand its practical security. We consider a range of threats including IKK's query-identification setting, and against various approaches to building SE. We enumerate new threat models that describe attacks against suggested SE use-cases, such as encrypted email, and explore the efficacy of several attacks against different constructions of SE. In each case, we design attacks that exploit the leakage rather than any particular construction. We term these *leakage abuse attacks* (LAAs), and explore several LAAs

*Work completed while at Rutgers University.

CCS'15, October 12–16, 2015, Denver, Colorado, USA.

DOI: http://dx.doi.org/10.1145/2810103.2813700.

Objective	Prior Knowledge	Pas/Act	Min Leakage	Known Constructions	Where
Query Recovery	Fully Known Docs	Passive	L1	All	[10], § 4.2
Query Recovery	Partially Known Docs	Passive	L1	All	§ 4.4
Plaintext Recovery	Known Doc Subset	Passive	L3	–	§ 5.1
Plaintext Recovery	Distributional	Chosen Doc	L3	–	§ 5.2
Plaintext Recovery	Distributional	Chosen Doc	L2	Shadowcrypt, Mimesis	§ 5.2

Figure 1: Summary of successful leakage-abuse attacks on searchable encryption schemes. In all cases the adversary sees the transcript of communications to the server, in addition having the indicated knowledge and, in the last two cases, the ability to mount active chosen document attacks. Leakage levels L1, L2, and L3 are ordered in increasing amount of leakage; see § 2.

against leakage profiles for constructions that are found in the literature and deployed in practice.

New threat models. We investigate a range of threat models for SE. In terms of adversarial goals, we focus on *search query recovery*, following IKK, as well as attacks that may perform *(partial) plaintext recovery*. The latter has not before been looked at despite an intuitive relationship with query recovery (as queries are by definition part of the plaintext) and the fact that such attacks are likely to be more damaging in practice. For each of these adversarial goals, we also explore a number of adversarial capability models, for example the kind of partial information about the document set known to attackers and whether they can insert chosen documents.

Broader treatment of SE. This work considers a range of SE approaches, including schemes that leak more information to servers than the academic schemes that followed from the early SE works [6, 20]. Weaker schemes are used in practice due to their simplicity and compatibility with legacy servers, and have also been suggested for use in a number of recent research systems as well [9, 15]. The essential intuitive difference in leakage is that they leak information to the server even before a search is issued, while the more advanced schemes hide almost all information about plaintexts until keyword searches begin. In the body we make this precise by defining a series of *leakage levels* L1–L4, where L1 is the least amount of leakage corresponding to the Curtmola et al. security notion, L2 leaks all the information of L1 and more, and so on with L4 the most leaky.

Query recovery attacks. We build leakage-abuse attacks against a variety of leakage models. As mentioned, LAA attacks work against any scheme admitting the targeted amount of leakage (or more). We perform experiments using public email data sets as stand-ins for confidential emails. For a summary of the attack results in this paper, see the table in Figure 1.

We first revisit the query recovery attack setting of IKK in which the adversary knows the entire plaintext document set and we target L1 leakage, the hardest setting. We emphasize that while knowing all the plaintexts exactly does not seem very realistic, such knowledge does not make query recovery trivial and IKK uses a relatively complex attack that uses simulated annealing to attempt to match queries with keywords based on the pattern of which documents are returned. We give a significantly simpler, faster, and more accurate attack that we call the count attack. The observation is that a large fraction of keywords will match against a unique number of documents, and so an adversary in the IKK setting can simply count the number of documents returned by each keyword and compare to the number of documents matched by a query. When multiple keywords return the same number of documents, one can disambiguate using the pattern of keywords across returned documents.

It seems unlikely that adversaries in practice will know all documents. On the other hand, assuming knowledge of no documents is a step too far: the adversary may incidentally know that a user has one or more widely-circulated emails in her repository, for example. We explore relaxing the assumption of full document set knowledge to partial knowledge, and show that the IKK attack does not perform well even when just a small fraction of documents are unknown to the attacker; our count attack's efficacy degrades as well but not so severely. We also explore whether an obvious potential countermeasure to the count attack, padding out the number of documents returned for queries, bears fruit: unfortunately our experiments indicate that a slightly modified variant of the count attack recovers query information even if the amount of padding increases the storage overhead by 8x.

Partial plaintext recovery attacks. We also explore, for the first time, *active attacks* in which an adversary can induce a user to insert chosen documents. We consider these attacks to be a realistic threat against deployments such as automatically-updated email repositories. In this setting, an adversary may simply send the user an email and learn from the subsequent leakage. We stress that the email is, in every technical sense, a valid email from a sender unconnected to the adversary, so preventing such an attack would require semantic analysis or some other major change to the deployment. In the body we discuss how active attacks might be mounted against other systems such as those appearing in [9, 15].

We show that either knowing a few documents or being able to insert documents enable easy attacks that extract significant information about plaintexts in the L2 and L3 leakage settings. As one example using the Enron email dataset for simulations, we show that a server that happens to know even a single email sent to 500 Enron employees learns on average 35% of the keywords of all other encrypted emails.

Summary. We are the first to investigate the implications of common SE leakage models related to schemes used in industry and in academic works [9, 15]. Our experimental analyses show that efficient and easy-to-mount attacks can reveal a significant amount of information about email plaintexts. We expect the attacks will work against other document types that are similar to email (e.g., tweets, free-text web site form data, etc.), though the exact efficacy will vary with context.

The consequences for the security of deployed systems is less clear. In contexts where searchability is requisite, SE schemes provably revealing only L1 or, at least, L2 leakage is clearly better than not having any encryption at all. Future work should explore the incorporation of countermeasures: periodically re-encrypting content, adding noise at the expense of query performance, or attempting to make practical mechanisms from the oblivious RAM literature [8]. At the end of this paper we provide some additional reflection on the implications of these attacks.

Versions of this paper. In April 2018, Seny Kamara and Tarik Moataz pointed out an inconsistency between our prose description of the count attack with partial knowledge and the attack implementation: our implementation assumed a few queries were known a priori, but this assumption was not clearly explained in the paper. In the course of correcting these results we discovered some

methodological errors, due to software bugs, in the experiments. The Apache email corpus was processed incorrectly and experiments in Section 4 which should have been randomized were run with a fixed random seed. This version (hereafter, v2) corrects these problems with the previous version of the paper (hereafter, v1) and introduces changes to the attacks with padding and partial knowledge in Sections 4.3 and 4.4 respectively.

The change to the count attack with padding is in the preprocessing step. In v1 of this paper, the preprocessing step which prunes query-to-keyword mappings was run only on a small fraction of queries. In v2, the pruning is run on every query. See Section 4.3 for more details. This change substantially increases the accuracy of the attack. For example, in v1 the accuracy of the attack on the Enron dataset was decreased with 15% padding, but in v2 the attack performs perfectly until around 200% padding.

The change to the count attack with partial knowledge is twofold: (1) adding a preprocessing step to prune incorrect mappings and (2) using a limited brute-force search of query-to-keyword mappings. In v1, the only preprocessing step was guessing queries which can be uniquely identified from their counts. In v2 we use the adversary's partial knowledge to estimate confidence intervals for the count of each keyword in the full corpus. Then, preprocessing eliminates query-to-keyword mappings where the query's count falls outside the confidence interval for the keyword.

In v1 the count attack was run only a single time. In v2 we brute-force every mapping for a small number of keywords, run the count attack for each possible mapping, and return the best consistent solution. See Section 4.4 for more details. These two changes increase the average reconstruction rate of the partial knowledge attack in v2 by 3-5% compared to v1. The results in v2 are without known queries.

All count attack results in Section 3 are as good or better in v2 than in v1. Some of the attacks on the correctly-processed Apache corpus in Section 5 of v2 are slightly worse than in v1. Overall, the qualitative conclusion of this work is unchanged: the inherent leakage of searchable encryption leads to simple and efficient attacks that can be very damaging in practical settings.

2. SE SCHEMES AND THEIR LEAKAGE

In this section we recall what are SE schemes, give an overview of common schemes, and explain their leakage profiles. We will incorporate as well some discussion of schemes used to build systems in the research literature and in commercial products, as well as how threat models map onto real systems. First we fix some notation from the SE literature.

SE basics. We let $\mathbf{D} = (D_1, ..., D_n)$ denote a collection of $n > 0$ plaintext documents, where each document is a variable-length string from a known character set. We denote the length of a document in characters by $|D_i|$. A keyword extraction procedure takes as input a character string D_i and outputs a vector W_i, the keywords, where each component is a character string. A typical keyword extraction will first parse D_i into words, drop common words such as articles and propositions, stem the remaining individual words, and remove duplicate keywords. We assume keyword extraction is deterministic and, looking ahead, known to the adversary. We denote by c_i the count of unique keywords in, or equivalently the dimension of, W_i. Let $\mathbf{W} = (W_1, \ldots, W_n)$ be the ordered list of all the documents' keyword vectors.

An SE scheme consists of encryption, search, and (possibly) update algorithms. The encryption algorithm takes as input a secret key K, documents $\mathbf{D}$ and emits a ciphertext. Search takes as input a secret key K, a keyword w, and outputs a query message. If a scheme includes the update algorithm, it is known as a *dynamic* SE scheme. The update algorithm takes as input K and a document D, and outputs an update message. We assume non-interactive settings in which query and update messages are sent to the server, the latter executing some algorithm and returning a result. The client can then process the result, e.g., decrypting recovered documents due to a search query. Several examples of schemes are given below.

2.1 A Leakage Hierarchy for SE

Starting with the work of Curtmola et al. [6], research on SE formalize security by defining a *leakage profile* that characterizes the information an adversary may learn. One proves that an adversary's view during an attack can be simulated given the leakage profile. A long line of work follows in this vein [3, 4, 6, 11–14, 16, 17, 21]. Such analyses rule out adversarial attacks that obtain information not captured by the leakage model. However, this does not clarify what an attacker can learn from the information that is leaked. The goal of our work is to do exactly that.

We developed a scale for characterizing leakage, defining a set of four leakage profiles. The attacks we will present in later sections target any scheme with a given leakage profile or greater. The leakage of SE schemes often includes additional items of information (c.f., Figure 1 of [4]), such as the number of documents, number of unique keywords, and when the same query is repeated. When such items are not implied by a given leakage profile, we ignore them in our attacks. We note, however, that certain statistics like the plaintext lengths, when queries repeat, and even a query's time of day or client IP address may lead to improved attacks beyond the ones we explore.

The definitions of leakage profiles follow along with the description of schemes that instantiate them. The descriptions progress in order from greatest leakage (L4) to least (L1). We also stress that the same leakage profile may arise for two very different schemes (we note an example below for L2). Below we explain leakages simultaneously with example classes of schemes that achieve them. For the sake of brevity we will typically only informally describe the portions of SE schemes that are not relevant to our results.

2.1.1 In-place SE schemes

The first two schemes we describe are called *in-place* because they involve direct uploading of encrypted document data, and the server searches by iterating over keywords on a per-document basis. These are the simplest to implement, and have the distinct advantage of being compatible with many existing storage and search APIs. This eases the route to deployment, at the cost of more leakage.

Full-text substitution cipher. This is the simplest type of searchable encryption. The client parses each input document and performs keyword extraction to produce keywords W_i that are in order with repeats. Then, it applies a deterministic cipher E to each word. The resulting collection of ciphertexts is uploaded. The client searches by sending the enciphered version of a keyword to the server. The server scans the encrypted documents to match the term ciphertext or, alternatively, it may use an inverted index precomputed over the individual encrypted keywords in the ciphertext.

This scheme supports not only keyword search queries, but also boolean and phrase searches. However, stemming, wildcard, and approximate-match searching are not supported. This is because the value indexed by the server is a pseudorandom token which has no relation to the plaintext. So, for example, a wildcard search like "s*" would fail because the tokens for keywords starting with "s" do not themselves start with "s".

This simple in-place scheme has the largest leakage of any that we discuss. We define its leakage profile as:

L4: *Full plaintext under deterministic word-substitution cipher.* The server learns the pattern of locations in the text where each word occurs and its total number of occurrences in the document collection.

Schemes with L4 leakage support searching for specific phrases, among other kinds of queries beyond exact keyword match. Companies including Skyhigh Networks [18] and CipherCloud [5], whose products are discussed further below, make use of schemes with L4 leakage. That said, our attacks focus on the more restricted leakage models below; they apply against schemes with L4 leakage as well.

We note that the original searchable encryption scheme of Song et al. [19] is a deterministic encryption scheme like that described above, but with the additional use of a separate stream cipher for each keyword. The cipher iterates sequentially over each occurrence of the keyword in the document set. Thus, in the uploaded data each occurrence of a keyword has a different ciphertext, and the server initially cannot observe the pattern of repeated keywords. To search, the client sends a ciphertext corresponding to the first location of the search term, and the server iterates the cipher to find all word matches in the document set. The word occurrence pattern is revealed progressively as queries are issued. Thus if all keywords are queried this becomes equivalent to L4 leakage. We omit for brevity defining a leakage model reflective of this scheme, and only comment that it would be less leakage than L4, an orthogonal amount of leakage to L2 and L2 defined below, and strictly more leakage than L1 (also defined below).

Appended-keywords SE. Another class of SE schemes encrypts each document D_i using conventional randomized symmetric encryption, and appends to the resulting ciphertext as well an encoding of the values

$$F_K(W_i[1]) \ , \ \ldots \ , \ F_K(W_i[c_i])$$

where K is a secret key and F_K is a pseudorandom function (PRF) such as HMAC. The document ciphertext and its keyword hashes are uploaded as-is to the server. Search on a keyword w is easy: compute $F_K(w)$ and request the server to perform a search on it.

As discussed at length in [9,15], the benefit of appended-keyword SE schemes is that they are legacy-compatible: the server can perform indexing on the uploaded keyword hashes, addition and removal of keywords is straightforward, etc. Also, and unlike the simple deterministic encryption scheme above, here one can perform stemming during keyword extraction.

Such a scheme provides no additional hiding of occurrence patterns prior to search. Thus co-occurrence relationships (that a keyword appears in a particular subset of documents), counts of the number of unique indexed keywords, ciphertext lengths, and the order of keyword appearance in the can be learned by the server immediately upon upload. More formally, we have the following leakage profile:

L3: *Fully-revealed occurrence pattern with keyword order.* Intuitively, this leakage profile fully reveals the pattern of keyword occurrences in documents, in the order of their first appearance, but not the occurrence counts within a document. Formally, using the notation from above and fixing some order over all possible keywords $w_1, \ldots, w_N$, the profile outputs the sequence of sets

$$\{(i,j) : W_i[j] = w_1\}, \ldots, \{(i,j) : W_i[j] = w_N\} \ .$$

So the first set includes all pairs (i, j) where the first keyword is the j-th term in the processed version of document i.

We note that if one were to change the scheme above to additionally include repeats (say, to allow frequency-informed search), then the scheme would have L4 leakage (assuming the same keyword extraction algorithm).

In some implementations of the appended PRF scheme the client will sort the PRF values before uploading. This is actually better for security, as information about the order of first occurrences of keywords in each document is not leaked. The leakage is captured by the following model.

L2: *Fully-revealed occurrence pattern.* This profile is similar to leakage profile L3 in that the occurrence patterns of keywords are revealed for every term, yet not in document order. Formally, if the documents collectively contain terms $\bigcup_i W_i = \{w_1, \ldots, w_N\}$, then the profile leaks the full collection of sets

$$\{i : w_1 \in W_i\}, \ldots, \{i : w_N \in W_i\}.$$

We consider both ordered and unordered appended PRF schemes because we believe the difference in security is not widely appreciated. See also the discussion below on research systems use of appended-PRF schemes.

Use of appended-PRF SE. Due to their simplicity, these schemes have seen a variety of use. Commercial encryption products from Skyhigh Networks [18], CipherCloud [5], bitglass [2], and others use these schemes (or variants of them) to support keyword search on encrypted data uploaded to cloud services. An enterprise customer works with one of these encryption providers to arrange for their employees' connections to cloud services like SalesForce, Box, Dropbox, and others to be intercepted by their managed encryption proxy service. The proxy performs encryption on the clients' behalf using a key originally owned by the enterprise customer. The appended-PRF SE, being legacy compatible, works well within the existing APIs of the cloud services to support both insertion and search over ciphertexts. The security goal is to ensure that the cloud service, should it be compromised, cannot learn information about the customer's data.

Appended-PRF schemes have also been suggested for use in a number of recent academic research prototypes. The goal of ShadowCrypt [9] is to secure on the client side a user's inputs to a malicious web page. It does this using ShadowDOM, a new standard for building DOM trees (or sub-trees) on top of the existing webpage. The browser should enforce that the ShadowDOM cannot be scripted by the (malicious) webpage. To use ShadowCrypt, a user simply clicks a lock next to the field and enters his or her input as usual. Then, the ShadowCrypt plugin encrypts the input before the actual webpage has access to it. ShadowCrypt supports searching on the encrypted input by applying a keyed pseudorandom function (PRF) to each of the keywords and appending the resulting values to the end of the ciphertext. The paper leaves ambiguous whether PRF values should be sorted; the open-source prototype does indeed sort the values. A diagram of highlighting ShadowCrypt's architecture when it comes to search is shown in Figure 3.

Mimesis Aegis [15] aims to allow inputing confidential data into untrusted mobile applications. It does so by arranging for the operating system to interpose a transparent encryption layer on top of the GUI of mobile applications. As with ShadowCrypt where the web page is untrusted, here the application may be malicious, and still the encryption should prevent the application from learning

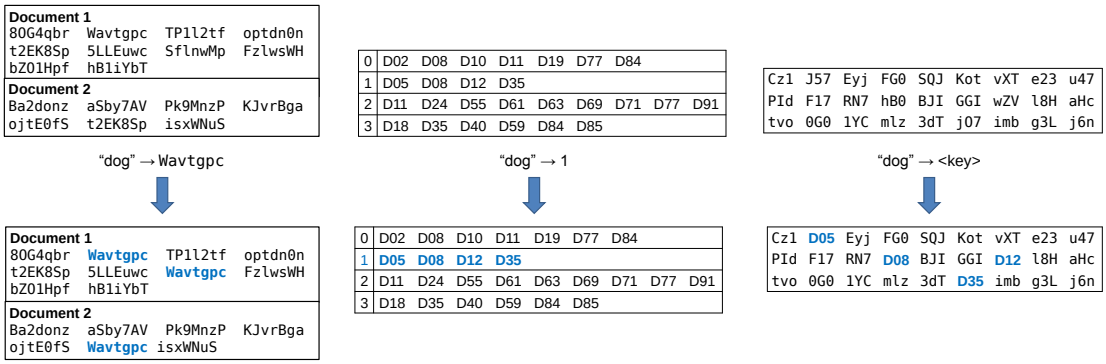

Figure 2: Server's view of SE data structures for the same plaintexts before and after searching for the keyword 'dog'. From left to right, the schemes are examples of leakage profiles L4, L2, L1.

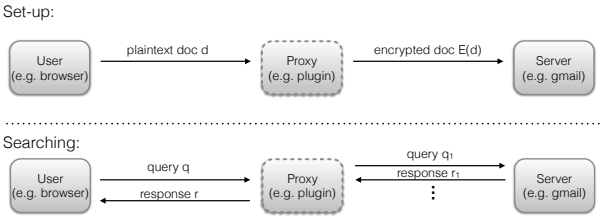

Figure 3: Architecture for a legacy-compliant SE scheme.

information about the plaintext data. To enable searching, Mimesis Aegis employs a technique called easily-deployable efficiently-searchable symmetric encryption, or EDESE. The work discusses different ways of accomplishing this functionality, but for the purposes of our security analysis they are all equivalent to the technique used by ShadowCrypt. They also leave implicit whether ordering of hash values is critical to security, though we their constructions do sort the PRF values[1] and meeting their definition of security for EDESE would seem to rule out schemes with L3 leakage.

2.1.2 *Encrypted-index SE*

Most academic work has been on (what we call) encrypted-index SE [3,4,6,11–14,16,17,20,21]. In these schemes, clients construct an encrypted index before upload.

To describe these schemes, we need to introduce additional notation. For these schemes, the client generates an *inverted index* $\mathbf{I}$. This is (abstractly) an $m \times n$ matrix where entry $\mathbf{I}_{i,j} = 1$ iff document D_j contains word w_i. All other entries are zero. The $m \times m$ *co-occurrence count matrix* $\mathbf{C}$ contains in location $\mathbf{C}_{i,j}$ the number of documents in which w_i and w_j both occur. This can be normalized to produce an empirical *co-occurrence probability matrix* $\hat{\mathbf{C}}$.

Searching the index $\mathbf{I}$ requires the client to generate a per-query *trapdoor*, which is sent to the server. The trapdoor can be thought of as allowing the server to decrypt only those document identifiers corresponding to the search term. In some cases, a special search protocol is performed with the trapdoor by the server on the encrypted index. For instance, the protocol of [4] employed an interactive protocol to carry out Boolean searching. Unlike in-place schemes, index-based schemes inherently hide the document word order.

Unencrypted inverted index. In this scheme, the inverted index $\mathbf{I}$ (table of document IDs) is sent unencrypted to the server. We assume that the document ID's are meaningless to the server (e.g. randomly assigned serial numbers). The client randomly permutes the rows of the index before uploading, perhaps according to a PRP, so the server does not learn the correspondence between rows and keywords. The trapdoor for a query is simply an index to a row of $\mathbf{I}$, and the server fulfills the query by returning the selected row to the client.

Since this scheme is essentially giving to the server the exact document-term matrix, the server can, prior to any queries being issued, directly observe the length of all result sets and also construct the co-occurrence matrix $\mathbf{C}$ for all keywords. In fact, this scheme has leakage profile L2, equivalent to the appended-hash scheme described above. Note that in this scheme, Boolean queries can be fulfilled at no additional security loss, since the server sees the entire index at upload time.

Encrypted inverted index, no result length hiding. This scheme differs from the above in that each row of the index is encrypted as a whole (say with a block cipher in some randomized chaining mode), so that no repeated document IDs can be read from the index before queries are issued. However, the length of each row is not hidden; before any queries are issued, the server can observe the result *count* for every row.

The client searches by sending a trapdoor allowing the server to decrypt the index row corresponding to the keyword. As queries are fulfilled, the server gains information about the overlap of documents in result sets.

In this scheme the server can fulfill Boolean queries by obtaining the decryption key for every keyword in the query. However, this results in the server learning additional document ID's beyond those satisfying the query.

We do not define a separate leakage profile for this scheme, but consider it under profile L1, described below.

Fully length-hiding SE. In this strongest class of searchable encryption schemes that we consider, the length of individual result sets is hidden in the initially uploaded index, so that before queries are issued the server learns nothing except the total size of the index. It was suggested by Song et al. [19] that the length of result lists could be partially hidden by padding the shorter lists to a fixed size. A solution to completely hide result lengths without using

[1] In fact they insert the PRF values into a bloom filter, to lessen the amount of storage required. This has the same effect as sorting them.

padding was first presented in Curtmola et al. [6], by the use of an interwoven linked list data structure. Recent work has shown that it is possible to fulfill Boolean queries in this setting while revealing less information than the entire row for all words in the query [4].

We define the leakage of fully-length-hiding SE, the smallest leakage profile we consider, as L1:

L1: *Query-revealed occurrence pattern.* Intuitively, this profile reveals the same information as L2, but only for terms that have been queried. This is the class of leakage achieved by the schemes of [6] and its derivatives. In this profile, initially only basic size information (e.g. the total length of the documents) is leaked. When a trapdoor is sent, the server learns the *access pattern* of the query, which in the case of keyword search means the identifiers of documents containing w.

Formally, when $\mathbf{D}$ is processed into $\mathbf{W}$ and encrypted, and a sequence of queries $q_1, q_2, \cdots, q_Q$ is issued, then the leakage profile includes the sequence of sets

$$\{i : q_1 \in W_i\}, \{i : q_2 \in W_i\}, \ldots \{i : q_Q \in W_i\}.$$

Depending on the setting, the leakage function may permute the indices i randomly to hide the initial correspondence between the plaintexts and ciphertexts. Thus, for each queried term, the profile leaks the number of documents containing the term, and for multiple queries, the profile reveals which documents they have (or don't have) in common.

See Figure 2 for an intuitive picture of what the server "sees" before and after a query is issued, for schemes with leakage profiles L4, L2 and L1,

Use of encrypted-index SE. While at the time of writing are unaware of any deployment of these schemes in practice, recent work has shown that they scale to large datasets [4]. Given this and the signficiantly improved leakage profile over the in-use appended-PRF schemes, we expect that practitioners will increasingly transition to using some form of encrypted-index SE.

2.2 Attack Models

We classify attack models along two axes. First, we consider the *mode* of the attack, meaning whether the server is passively or actively mounting attacks, and second, we specify the prior *knowledge* of the server regarding the documents and queries.

Attack modes. In SE the server receives the encrypted dataset and query requests from a client, and an adversarial server may use its position to extract private information. We classify the following three attack modes (where the last two may be utilized simultaneously in one attack). In each case, the adversary is a server following the SE protocols, but it may take actions to induce the client to run with certain inputs.

- An *honest-but-curious* server follows the protocol and takes no actions beyond those of an honest server, and attempts to learn about the plaintext of documents or terms that were queried.
- An active adversary can carry out a *chosen-document attack* in which it tricks the client into including a chosen document in the document set.
- An adversarial server may mount a *chosen-query attack* by inducing the client into issuing certain queries, thereby revealing whatever is leaked by that query. Our attacks will not use this attack mode, as it seems less realistic, but it conceivable that a deployment scenario would enable this ability in some form.

We note that the danger of active attacks have not, to the best of our knowledge, been given a treatment in academic literature on SE. Unfortunately active attacks seem to be relatively straightforward to mount in most SE deployment settings, particularly so for chosen-document attacks. (Note that for the appended-PRF schemes a chosen-query attack can be mounted using a chosen-document attack with a single-word document.) Consider using SE to automatically encrypt and back-up one's email inbox. The malicious server can then arrange for an email message of their choosing to be sent to the victim. As long as it is not deleted by a spam filter before going to the victim's inbox, this will succeed in forcing SE to be applied to the message with resulting ciphertext subsequently observed by the server.

As another example consider ShadowCrypt. Recall that here users click on a button within a webpage to indicate to the browser that it should encrypt the data in some form field. The web page is considered malicious. Here click-jacking-type techniques can be used to mount chosen-document attacks. The web page has an "encrypt field" button hidden behind an innocuous-looking frame that the user must click in normal interaction with the site. The web page can use such a technique to have the user unknowingly instruct ShadowCrypt to encrypt data of the web page's choosing. The authors acknowledge that click-jacking attacks are possible against their system, and indeed they might give rise to more direct attack vectors than we consider here. Nevertheless this serves to show that active attacks are often going to be possible for determined adversaries.

Adversary knowledge. One of our theses is that an SE server's prior knowledge of the documents and queries may enable the extraction of more information. We specify the following possible types of prior knowledge. Depending on the setting, multiple types of knowledge may be available to the adversary.

- *Distributional query knowledge* models a typical case where an adversary has some idea about the queries being issued. For instance, if encrypted chat logs are being searched, an adversary can refer to typical user behavior regarding such searches to inform its attack.
- *Known queries* occur when the server knows some of the terms input by the client for a search. In practice an adversarial server may use contextual information about the client's behavior to accurately guess some queries. Formally, in this setting we will draw some queries $q'_1, \ldots, q'_k$ from a distribution that all become known to the adversary, and then other queries $q_1, \ldots, q_Q$ from a distribution that the adversary will not know.
- *Distributional document knowledge* models the contextual information an adversary will have about the documents (e.g. whether they are emails, or corporate sales documents, or something else). Formally we will model this by considering a distribution on documents. In our experiments, we model this by dividing a large data set into training and test sets, and give the training set to the attacker.
- *Known documents* models scenarios where an attacker will know certain plaintext documents or perhaps have significant information about them. For example, an attacker might know that a widely-distributed email exists in a user's repository. In our attack experiments we model this situation by drawing documents as above, and then either hand-selecting likely known documents or choosing some at random.

- *Fully-known document set* is the setting explored by IKK [10], where all of the documents are known to the adversary, and only some or all of the queries are unknown.

Attack objectives. We next identify possible objectives for an adversary controlling the server in an SE scheme.

- *Query recovery (QR)* is the goal of determining the plaintext of queries that have been issued by the client. This was the objective of the first known attack on SE by Islam et al. [10]. QR can be considered in any setting where some of the queries are unknown, including cases where the documents are fully or partially known.

- *Partial plaintext recovery (PR).* In PR attacks, the adversary's goal is to reconstruct as much as possible of the client's indexed documents, primarily by learning a mapping of keywords to their encrypted versions. A PR attack may reveal plaintexts as a "bag-of-words" or in document order; for the attacks we present, this depends on the scheme and is independent of the attack method itself.

We point out that there is an intimate relationship between query recovery and partial plaintext recovery, as any query that is revealed immediately indicates that a document contains that particular keyword. If a full document is revealed, then queries against it are narrowed down to a set of keywords.

Of course, query and plaintext recovery are not the only attack objectives: for completeness we briefly point out two others, *document presence* and *document identification*. In a document presence attack, the adversary simply wishes to determine whether a known plaintext document is present in the client's index. In a document identification attack, the server seeks to find the correspondence of known documents to the document IDs revealed by the SE scheme. Also note that attack objectives are interrelated. In particular, a successful plaintext recovery attack will make document identification trivial.

In the subsequent sections we describe our attack results for meaningful combinations of attack objective, leakage profile, and server knowledge. See the table in Figure 1 for a summary.

3. EXPERIMENTAL METHODOLOGY

We investigated the vulnerability of the described leakage profiles by means of simulated query recovery and plaintext recovery attacks, using two separate publicly available email datasets. The first is emails from 150 employees of the Enron corporation from 2000-2002, available online [1]. In order to focus on intra-company email, following the approach of [10], we took emails from each employee's sent mail folder, resulting in 30,109 total documents. The on-disk size of the data set is 50 megabytes. The second dataset we used is a subset of the Apache mailing list archives. Specifically, we used the "java-user" mailing list from the lucene project for the years 2001-2011. This consists of around 38,000 emails, with an on-disk size of 244 megabytes.

For each dataset, one email message is considered as one document. Fixed-size vocabularies were established by taking the most frequently occurring keywords from each dataset, after removal of 200 stopwords. The typical vocabulary size used in our experiments was 5000, which represents the upper limit at which the attack of the prior work succeeds, as seen in Figure 4. This gives an average of 93 words per document in the Enron corpus, and 291 in the Apache dataset. We will further discuss the impact of vocabulary size on our results as we present them.

Keywords are stemmed using the standard Porter stemming algorithm. Stemming is a crucial feature of usable search functionality, as it provides more flexible matching, e.g., a search for "cat" will also match the word "cats". Stemming has a two-sided effect on the attacks that we evaluated. On one hand, it limits an attack to reconstructing only the stems of plaintext words, causing loss of information relative to the original plaintext. On the other hand, stemming reduces the total vocabulary size and increases repetitions of terms, making our attacks easier.

4. QUERY RECOVERY ATTACKS

As stated above, the adversary's goal in query recovery attacks is to recover the correct plaintext keywords corresponding to queries that the client has issued to the SE server. These attacks apply to any of the leakage profiles defined above (L1 or greater), being originally designed for the fully-hiding SE schemes described in Section 2.1.2. Though the attacks work under the lowest leakage profile, they require the server to have more extensive knowledge of the document set that is indexed. These attacks also depend on multiple queries being issued, so that term access patterns can be correlated.

4.1 Prior Work: The IKK Attack

Islam et al. [10] give the first successful experimental attack on searchable encryption that we are aware of. Theirs is a query recovery attack on SE (leakage profile L1), using full document knowledge and partial query knowledge. They give experimental results using the Enron email dataset, achieving recovery rates of over 80% of issued queries for some vocabulary sizes.

The attack assumes that the server knows a fixed set of m potential search terms, and the server's knowledge of $\mathbf{D}$ is distributional, in the form of the $m \times m$ matrix $\hat{\mathbf{C}}$ of word co-occurrence probabilities. Note that this can be computed exactly if the server knows the true document-term matrix for the indexed documents.

The server mounts the attack by observing the document access pattern revealed by the client's queries. Let q be the number of unique query tokens observed. These are used to construct a $q \times q$ term co-occurrence matrix $\mathbf{C}_t$ and its normalized version $\hat{\mathbf{C}}_t$. This is a permutation of an unknown submatrix of $\hat{\mathbf{C}}$ (approximate if the server's knowledge of $\mathbf{D}$ is not exact.) Then, simulated annealing is used to find the best match of $\hat{\mathbf{C}}_t$ to $\hat{\mathbf{C}}$. The output is a mapping of rows of $\mathbf{C}_t$ to rows of $\hat{\mathbf{C}}$; this is the set of guesses for the query terms. We note that in leakage profile L2 or greater, the attack can be carried out using the entire keyword set prior to observing any queries.

The authors give experimental results for the Enron email dataset. The keywords are taken to be the m most common stemmed words in the document set after stopword removal, and the queries are chosen uniformly at random from among these. They measure recovery rate as the percent of unique queries correctly guessed. In all but one of their experiments, they additionally assume 15% of the queries are known by the attacker. With 20 known queries, the recovery rate reported varies from near perfect with 500 keywords to approximately 0.65 with 2500 keywords, for a constant 150 unique queries. We re-implemented the attack to confirm these results and make further comparisons.

One strength of the attack, henceforth called the IKK attack, is that the success rate is largely independent of the number of queries issued. The median number of queries in the results of [10] is 150, which we also use as a default in our experiments. We consider that this many queries could be observed by an adversarial in a reasonable amount of time, and it is a sufficient number for the attacks to be effective.

A significant weakness of the attack is that the adversary's advantage does depend strongly on the number m of keywords under consideration. The authors of [10] do not report on experiments with keyword sets larger than 2500. In our experiments, the annealing attack performs poorly on queries for vocabulary sizes over 5000. This is shown in Figure 4. Thus the attack of [10], while valuable for illustrating the vulnerabilities of SE, is fairly non-scalable.

Another weakness is the reliance on known queries. They perform one experiment with varying numbers of known queries for $m = 1500$ and 150 unique queries. In that parameter regime, they show the number of known queries does not affect recovery rate. We replicated this experiment with higher values of m and observed that their attack is very sensitive to the number of known queries—for example, with $m = 4500$ and 15% of queries known they recover about 25% of 150 unique queries (see Figure 4). With only 5% of queries known, however, their recovery rate drops to zero.

As will be shown in Section 4.4, while the knowledge about the documents required by the IKK attack is only "distributional", in the form of keyword co-occurrence probabilities, in practice the accuracy of this knowledge required for the attack to succeed is so high that we believe it can only be obtained by a server that has explicit knowledge of the true document set. Thus, in the next section we present a simpler and more effective attack that can be carried out when the server does have more explicit knowledge of the indexed documents.

4.2 Query Recovery with a Counting Attack

When the adversarial server, in addition to knowing the co-occurrence pattern of keywords, also knows the number of documents in the indexed set that match each keyword—the *result lengths*—a much more efficient and accurate query recovery attack is applicable. This length-based query recovery attack is applicable to all schemes with leakage profile L1, and requires no numerical optimization techniques.

Attack Description. As in the IKK attack scenario, the adversarial server knows a keyword co-occurrence matrix $\mathbf{C}$ of size $m \times m$. In addition, we assume that the server knows, for each keyword w, the *number* of matching documents $\mathbf{count}(w)$ in the true document set. Note that all this information can be easily computed if the server has access to the explicit document set.

The first observation is that if any keyword with a unique result count is queried using trapdoor q, then a server with knowledge of the true document set can immediately recover the query, by finding the word w such that $\mathbf{count}(w) = \mathbf{count}(q)$.

In the Enron email dataset with the same setup as [10], if we consider the 500 most common non-stopwords as keywords, 63% of them have a unique result count. If 2000 keywords are considered, then 24% are unique. Note that due to the Zipfian character of word distribution in natural-language text, the unique result counts will come from the more frequently occurring keywords. If queries are assumed to be drawn uniformly from this set of keywords, then these percentages can be considered a baseline rate for query recovery when the server knows result counts. Also note that the larger the document collection, the larger the number of keywords that have a unique result count.

Of course, many key content words are singletons and can never be recovered using unique counts alone. Beginning with this baseline knowledge, our attack algorithm proceeds to recover queries that do not have a unique result count by comparing term co-occurrence counts. See Algorithm 1.

Input: Unencrypted keyword index Index, observed query tokens $\mathbf{t}$ and results
Initialize known query map K with queries (q,k) having unique result lengths;
Compute co-occurrence counts C_q for observed queries and C_I for Index;
while *size of K is increasing* **do**
  for *each unknown query* $q \in \mathbf{t} - K$ **do**
    Set candidate keywords $S \subseteq K = \{s : \mathbf{count}(s) = \mathbf{count}(q)\}$;
    for $s \in S$ **do**
      for *known queries* $(q',k) \in K$ **do**
        if $C_q[q,q'] \neq C_I[s,k]$ **then**
          remove s from S;
        end
      end
    end
    if *one word s remains in S* **then**
      add (q,s) to K;
    end
  end
end

Algorithm 1: The count attack algorithm.

In contrast to the IKK attack, in the count attack the adversary knows which queries have been correctly reconstructed with very high probability. This is because the queries with non-unique result lengths are matched using co-occurrences with other queries, and co-occurrence information is a vector with dimension equal to the number of observed queries. Concretely, the co-occurrence vector is hundreds or thousands of dimensions, so the probability of a false positive (i.e., an incorrectly-reconstructed query whose co-occurrence vector matches in every dimension with the correct reconstruction) is very low for queries appearing in sufficiently many documents. However, an important caveat is that this informal argument does *not* hold for very rare keywords: for example, take two rare keywords which both only occur together in a single document. Neither result length nor co-occurrence information can distinguish queries for such keywords.

The count attack also has another advantageous property not shared by IKK: even for queries that are not uniquely reconstructed, the adversary has partial information about the true keyword in the form of the candidate set. In the example given above the two rare keywords cannot be uniquely reconstructed, but the count attack's candidate set for a query for one of them would only contain two keywords.

Count Attack Analysis. Figure 4 shows the query reconstruction results from our implementation of the count attack in the same setup as [10]. Crucially, the count attack is not given any known queries. IKK knows 14% of the queries initially. The graph shows results for varying number of keywords from 500 to 6500, in each case assuming that 10% of the keywords under consideration have been queried. The queries in each trial were chosen uniformly at random from the set of possible keywords. We performed 25 trials for the count attack for each keyword set size. We also performed 25 trials for the IKK attack when the keyword set size was less than 3500, but above 3500 we only performed 15 trials for IKK. This is because the attack scales very poorly for large keyword set sizes. Concretely, for 6500 keywords a single trial of IKK takes over ten hours. In most of their experiments IKK assume 15% of queries are known initially instead of 14%. We re-ran IKK for keyword

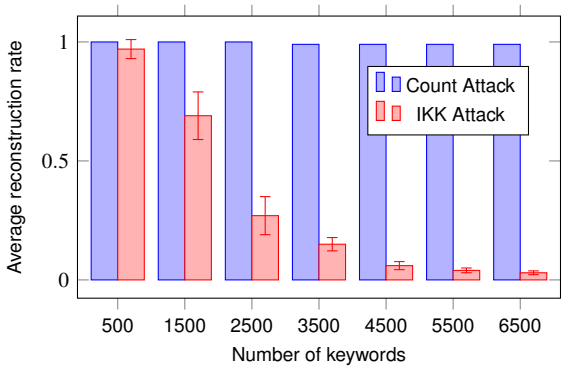

Figure 4: Comparison of IKK and Count attacks for query reconstruction by vocabulary size using the Enron dataset with 10% of keywords queried. IKK attack knows 14% of queries initially. Count attack has no known queries. Error bars show one standard deviation.

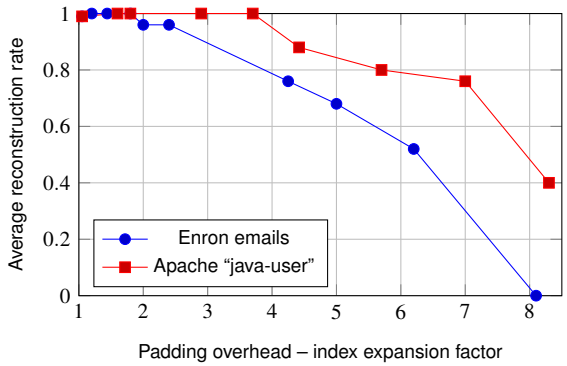

Figure 5: Reconstruction rate for the count attack, modified to handle padding as described in Section 4.3. 5000 keywords considered, 10% of keywords queried.

set sizes up to 3500 with 15% of queries known; IKK's average reconstruction rate increased slightly but the overall trend did not change.

Our experiments indicate that once even a small number of queries is initially disambiguated, the co-occurrence counting phase can successfully recover nearly all the queries. The worst performance was for 6500 keywords, where in the worst trial the count attack failed to reconstruct only 5 of 650 keywords.

4.3 Padding Countermeasures

We then considered whether the effectiveness of the count attack could be decreased by a client who pads the index with additional entries for bogus document ID's, to disguise the true counts. These bogus ID's can be filtered from search results by the client. We consider a scheme in which the number of entries in each index row is padded up to the nearest multiple of an integer n. This can be thought of as constructing "buckets" within which keywords will have the same result length. Changing the padding size n allows us to adjust the space-security tradeoff.

Qualitatively, this affects the count attack in two ways: First, it reduces the number of unique result lengths, increasing the number of candidate matches for a given query. Second, it prevents the co-occurrence counting stage of the count attack from being carried out exactly: The number of co-occurrences of two keywords in the padded index may exceed the co-occurrence count of the corresponding keywords computed from the true document set. The attacker must allow for this and cannot eliminate as many candidates.

The count attack as described above completely fails once the padding size n is increased to the point that no keyword has a unique result count; in this case there are no initial known queries with which to bootstrap the co-occurrence counting. However, we can modify the algorithm to allow the attack to be carried out with reduced information. This *generalized count attack* has two modifications: Firstly, knowing that the padding can cause additional "false co-occurrences", the co-occurrence count matches within a window as large as the maximum number of false co-occurrences, rather than requiring an exact match. Secondly, we remove the algorithm's dependence on initially finding a query with a unique result count. This is done by, for every query, making an initial guess mapping the query to one of the candidate keywords of matching result length, and then running the remainder of the algorithm. If the guess is wrong, the co-occurrence counting phase detects an inconsistency, and the next candidate will be tried. If any guess lets the algorithm recover nearly every query, the algorithm terminates and returns that query mapping. Though this approach can cause the count attack to be run tens of thousands of times, pre-computing relevant data structures like the co-occurrence matrix lets the attack run in a fraction of a second. The guessing phase is simple to parallelize as well.

Figure 5 shows the effect of padding on the reconstruction rate generalized count attack over 25 trials. The 25 trials were composed of five different random paddings of the index, with five random query transcripts each. We do not show error bars. The variance is high in all trials over 2.5x overhead, but this is because the attack either recovers nothing or everything.

Up to a padding level that increases the index size by about 140% for the Enron data, and 330% for the Apache data, the attack's success rate is unaffected. Even after this, the attack remains fairly effective—for the Enron dataset, increasing the size of the index by over 500% only prevents learning all keywords in half the trials. We conjecture that this is because keyword co-occurrences are very high-dimensional (a single keyword's co-occurrence vector is the size of the keyword alphabet), so even with substantial padding, the co-occurrence information can uniquely identify a keyword.

Our implementation of the attack fails completely with more than a 700% overhead, but we believe this is not inherent—a more advanced attack should be able to recover queries accurately even above 700% by, for example, brute-forcing combinations of two or three queries instead of just one. Though this would be computationally expensive, it is easy to parallelize.

4.4 Query Recovery from Partially Known Documents

In this section we analyze the above attacks on L1 leakage (with slight modifications) in the case when the server has only partial knowledge of the document set. This will help us to better understand the server knowledge level required to carry out these attacks.

Analysis of the IKK attack with partial knowledge. Because the IKK attack only directly makes use of word co-occurrence probabilities, technically it does not require the server to have explicit knowledge of the documents themselves. However, the effectiveness of the attack depends strongly on the accuracy of the co-occurrence matrix. The authors discuss this and give results of an experiment in which random Gaussian noise is added to the co-occurrence

matrix. In their experiment, the recovery rate drops from 85% to 65% at the first increment of added noise.

To understand better how real-world limitations on a server's knowledge affect the accuracy of the attack, we devised a new experiment. Instead of adding noise to the co-occurrence matrix, we assume the server knows only a certain *fraction* of the true documents, and computes the co-occurrence probabilities from the documents it knows. We duplicated the experiments of [10] with the server's co-occurrence matrix constructed from a randomly chosen subset of the true documents. See the results in Figure 6, which relates the recovery rate to the percentage of the dataset known to the server. In brief, these results indicate that unless the server has access to 99% of the true document data, the IKK attack performs quite poorly.

Generalized count attack for partial knowledge. We also modified the generalized count attack described in the previous section to allow query reconstruction when the server does not know the full document set. As with the case of a padded index, this requires the algorithm to allow keyword candidates within a window of co-occurrence counts, rather than requiring exact equality. The size of this window is proportional to the number of unknown documents, so for example if the server knows only 50% of the documents in a large corpus the window will be very large. Because a large window makes it difficult to eliminate candidate mappings, we use a greedy heuristic which brute-forces all possible mappings for a small number of queries and returns the mapping which maximizes the number of disambiguated queries.

To prune the set of possible mappings for each query, we use two tools. The first is estimating a confidence interval for the count of each keyword, using the adversary's partial knowledge of the documents. The second is a pruning phase similar to the one used in the padded index attack above. We describe each in turn.

Confidence intervals and Hoeffding's inequality. The adversary's partial knowledge of the documents can be used to estimate, for each keyword, the probability that it occurs in a document. Once the adversary estimates the probability, it can use a tail bound to construct a confidence interval for the count of that keyword in the full corpus (i.e., the number of documents returned by a query on the full corpus for that keyword). One such tail bound is Hoeffding's inequality.

THEOREM 4.1 (HOEFFDING'S INEQUALITY). *Let X_i, be i.i.d. Bernoulli RVs with* $\Pr[X_i = 1] = p$ *for all* $i \in [1,\ldots,n]$. *Let* $X = \frac{1}{n}\sum_{i=1}^{n} X_i$. *Then* $\Pr[\,|X - p| \geq \varepsilon\,] \leq 2e^{-2n\varepsilon^2}$.

The adversary can use Hoeffding's inequality to eliminate candidate keywords for a query q as follows. First, the adversary performs a preprocessing step: for each keyword k in the subset of the corpus known to the adversary, it computes the count of k (call it c_k^s) in that subset, then computes the estimate $\hat{p}_k = c_k^s/np_{pk}$ of its probability of occurring in a document.

From this, the adversary can estimate the *expected* count of k (call this random variable C_k) in the entire corpus as $\mathrm{E}[C_k] = n\hat{p}_k = c_k^s/p_{pk}$. Then the adversary applies Hoeffding's inequality to compute a lower and upper bound on C_k which should hold with high probability, assuming the number of occurrences of a keyword in a set of documents is well-modeled by a binomial distribution. (In our experiment we chose $\varepsilon = \sqrt{0.5n\log 40}$, which when plugged into the Hoeffding bound gives $\Pr\left[\,|C_k - c/p_{pk}| \leq \varepsilon\right] \geq 0.95$.)

To complete the preprocessing, for each keyword k the adversary computes

$$LB_k = \frac{c_k^s}{p_{pk}} - \sqrt{0.5n\log 40}$$

and

$$UB_k = \frac{c_k^s}{p_{pk}} + \sqrt{0.5n\log 40}$$

as the confidence interval on the true count of keyword k in the full corpus. Note that while here we're considering the probability for a single keyword, the Dvoretzky-Kiefer-Wolfowitz inequality implies an identical (up to constants) bound holds for the counts of all keywords simultaneously.

Once the adversary has computed a confidence interval for each keyword, computing the candidate set for a query q which returns r documents is straightforward: the adversary takes it to be

$$S_q = \left\{k' \in K \,|\, LB_{k'} \leq r \leq UB_{k'}\right\}$$

Note that nothing has required the adversary see any co-occurrence information—in fact this attack works unmodified if only the *size* of the results list is known. Thus, even putting the search index in ORAM or FHE (both of which leak results list size) will not prevent the adversary from learning some information about queries using this approach.

Pruning the candidate set. Once the adversary computes the set S_q in the previous step, it can remove some of the elements from consideration by running the count attack once for each possible keyword for q in the candidate set. For those mappings which result in an inconsistency, the adversary removes the keyword from S_q. As discussed above, if the server knows a relatively small fraction of the documents the window outside which a co-occurrence must fall to trigger an inconsistency is large; this means very few candidates can be eliminated in this step.

Results. The results of this attack are found in the second plot of Figure 6. Each point in that figure is the mean of the recovery rate over at least 25 random trials, and the error bars show one standard deviation. The 25 trials were composed of five random subsets of documents, which act as the server's partial knowledge, with five query transcripts per subset. The count attack performs better than IKK and requires no known queries. Our adaptation of the count attack to the partial knowledge setting does result in false positives—concretely, for 50% knowledge the false positive rate is higher than the recovery rate. For all other trials the false positive rate is lower than the recovery rate. Since IKK tries to recover every query simultaneously, their false positive rate is high as well.

We emphasize that these experiments are for the case where the server knows a subset of the true documents. We also conducted experiments in which the server is trained on a random subset of documents, while the client queries a disjoint subset from the same corpus—in other words, the *unknown documents* case. Here both our attack and the IKK attack fail completely. We believe this is not inherent, and a more advanced statistical attack could recover queries in this setting as well.

Our results support the conclusion that these attacks on L1 leakage require a significant amount of server knowledge of the document set, but are nonetheless possible with less-than-perfect knowledge.

Future directions. All the count attack variants presented in this section are very simple and can be improved substantially. One direction for future work on query recovery attacks is using more advanced techniques from information retrieval and machine learning. One such technique is latent semantic indexing [7], which uses dimensionality reduction on inverted indexes (e.g., with SVD) to embed both keywords and documents into a low-dimensional "semantic" space. One way this could be applied is to project the ad-

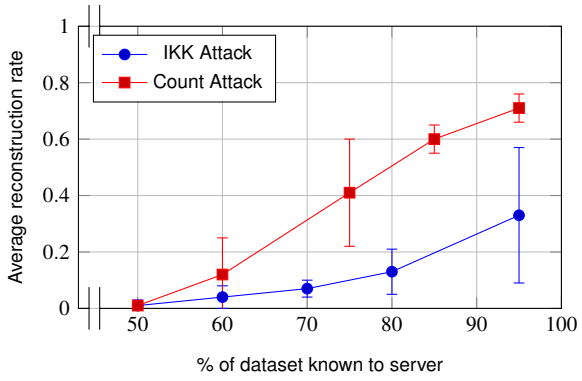

Figure 6: Average reconstruction rate when server has partial knowledge of true document set for the Enron dataset with 500 keywords and 150 queried uniformly. 5% of queries are known by IKK. The count attack has no known queries. Error bars show one standard deviation.

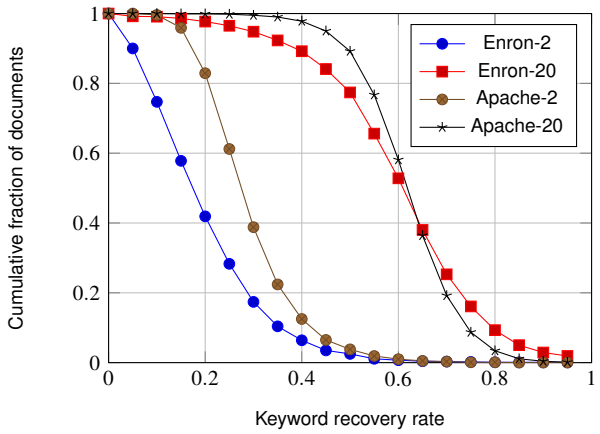

Figure 7: Keyword recovery rates for Enron and Java-user datasets from known documents for an in-place SE scheme with ordered keyword hashes.

versary's observed query-document occurrence matrix into a low-dimensional semantic space, then match each (projected) query to a (projected) keyword from the adversary's prior knowledge of the documents.

We conducted some experiments which used confidence intervals (computed using Hoeffding's inequality as described above) on co-occurrence counts to prune candidates more aggressively. This aggressive pruning performs well sometimes: when the server knows 60% or more of the documents the attack is both faster and recovers more queries than the simpler pruning which takes the co-occurrence window to be its worst-case maximum size. Below 60%, however, this approach fails because the adversary's estimate of the co-occurrence probabilities from its known documents is too poor: the algorithm "prunes" correct guesses often.

In light of the good performance of the confidence interval approach for estimating *query* probabilities (as opposed to co-occurrence probabilities), it may seem curious that using confidence intervals for co-occurrence counts does not perform well. However, this phenomenon is easily understood by observing that co-occurrences are a higher-dimensional estimation problem than keyword counts, so obtaining an estimate of co-occurrences with the same level of accuracy as keyword counts requires more samples. We conjecture that pruning candidate mappings using confidence intervals can work with a more careful approach, but we leave the details to future work.

5. PARTIAL PLAINTEXT RECONSTRUCTION ATTACKS

In this section we present attacks that enable the adversary to learn the mappings of keywords of the client's documents to the ciphertexts stored by the SE scheme. This keyword recovery can then be used to partically reconstruct the plaintexts of stored documents, either as a "bag of words" in appended-keywords schemes, or ordered plaintext if document text is encrypted using in-order word hashes.

These attacks exploit leakage profiles L2 and L3 or higher. For instance, they apply to searchable encryption schemes that store encrypted words on a per-document basis using a PRF or hash function, as in the "in-place" schemes described in Section 2.1.1. We show realistic known-document (passive) and chosen-document (active) attacks in this scenario. We present the success of the attacks quantitatively in the form of percentage of total non-stopwords recovered, and then demonstrate how this allows a highly revealing partial plaintext reconstruction.

5.1 Known-Document Attacks

This subsection presents and analyzes two keyword-based plaintext recovery attacks by a passively adversarial server that correctly executes the SE scheme algorithms and that knows the plaintext of a small number of the stored documents. The first attack is against L3 leakage and the second against L2.

5.1.1 Order of Hashes Known (L3)

To start with the simpler case, we consider a scheme in which the order of appended hashed keywords is not changed from the order in which the keywords appear in the document—that is, leakage profile L3. In this case, the attack is almost trivial: the server can immediately observe the hash values of all the indexed keywords in the known documents. Unlike the experiments of Section 4, there is no need to assume a fixed vocabulary beforehand; the vocabulary is derived from the known documents. We present the results of statistical experiments quantifying the advantage gained by an attacking server in this scenario.

Random Documents. To quantify the fraction of plaintext keywords learned by an adversarial server that knows a small number of the stored documents, we plotted the fraction of documents in our datasets for which a given percentage of keywords is recovered, at 5% intervals, when the adversary is given either 2 or 20 random known documents. Each rate is averaged over 10 random selections of known documents. Results for the two datasets are summarized in cumulative style in Figure 7. The curves that fall further to the right are indicative of a larger percentage of documents having high keyword recovery rates. For example, in the Apache "java-user" dataset, if the server knows only two email encryptions, 80% of the documents will have 20% of their keywords exposed.

As mentioned, the Enron and Apache datasets have over 30,000 and 38,000 documents, respectively, so the known documents are a very small fraction of the amount of data revealed. We conjecture that the attack is more effective for the Apache dataset because the documents have a common topic; in contrast, the Enron dataset

```
 The attached contract is ready for
signature.  Please print 2 documents and
have Atmos execute both and return same to
my attention.  I will return an original
for their records after ENA has signed.  Or
if you prefer, please provide me with the
name / phone # / address of your customer
and I will Fed X the Agreement.
```

```
attach contract signatur pleas print 2
document have execut both same will origin
ena sign prefer provid name agreement
```

Figure 8: (Top) An example plaintext email from the Enron corpus. (Bottom) The stemmed words recovered by our attack when given 20 randomly selected known emails.

consists of emails on different topics from many different employees.

To gauge the ability of a human attacker to gain information from this reconstruction attack, we visually inspected the output of the attack on a random selection of Enron emails, printing out in document order the keyword stems learned from a 20 known document trial, omitting stopwords. The result for a sample email is shown in Figure 8, together with the original plaintext. Note that potentially sensitive information has been revealed, including the name of a company (ENA) involved in a contract. We conclude that this attack reveals sufficient information for a human inspecting the output to gain a strong sense of document content.

Known Public Documents. Though choosing documents at random is important for statistically understanding the power of attacks, the source of a real-world known-document attack would likely not be emails chosen uniformly at random. A more probable source might be a message that has a wide distribution, such as a company-wide announcement. The more recipients an email has, the more likely it is that its plaintext will become known to an attacker.

To test this, we ran the same experiment with a single email from the Enron dataset that was sent to 500 recipients. It was an announcement sent to an entire division, four paragraphs long, with 832 unique keywords, containing an announcement of an upcoming survey of the organization by an outside consulting group. From this single document, an average of 35% of the indexed keywords in every document could be recovered. If it is the case that publicly distributed emails are longer than average, with large vocabularies relative to brief, person-to-person emails, then this attack is potentially even more damaging than the statistics using random documents indicate.

5.1.2 Order of Hashes Unknown (L2)

In leakage profile L2, for example, an in-place scheme when the keyword hashes are stored in randomized order, a server that knows a number of document plaintexts cannot immediately determine the mapping of every keyword hash in the document, though it knows the *set* of hashes that correspond to the keywords in the document.

We can quantify the ambiguity of words in the known documents. A keyword which the server knows the hash of has ambiguity 1 (no ambiguity). If the number of indexed keywords in a document known to the server is c_1, then for each word there are c_1 different possibilities for its hash. If this is the only document known to the server, we say the average ambiguity of all the keywords in the known document set is c_1.

When more than one document is known by the server, the ambiguity can be reduced, again by use of the co-occurrence pattern of keywords in the known documents. We omit a full analysis of this approach. A more powerful attack for unknown order of hashes is possible when the server is able to plant documents, as shown in section 5.2.2.

5.2 Active Attacks

We now consider the power of an attacking server who, in addition to observing the client's uploaded data, can "plant" documents that will be processed by the client and added to the uploaded dataset. Note that the protocol itself is not attacked; the server still fulfills requests following the rules of the scheme. This attack model can be seen as the analogue of the chosen-plaintext attack model (CPA) for encryption.

We can easily imagine real-world scenarios where attacks of this nature can be carried out—it can be as simple as a malicious server sending a client an email, which is then indexed by the client and uploaded to the server. As another example, a malicious server might make an invisible-to-the-user attack on ShadowCrypt using UI redressing.

The attacks described below apply to in-place schemes, as described in Section 2.1.1. We assume that by observing the payload length and/or the time at which documents are uploaded, the adversary can determine which index or database entry corresponds to the document he planted. For each attack we measure the level of plaintext recovery as percentage of keywords reconstructed.

5.2.1 Hash order known (L3), chosen document

Recall that searchable encryption schemes with leakage profile L3 preserve the document order of encrypted keywords stored at the server, either by deterministic word-based encryption of the document itself or by storing keyword hashes in their order of appearance. In this case then an adversarial server can carry out a simple but devastating attack. He can plant a single document in the database with any desired set of keywords, and then from its encryption can learn all of the hashes of those keywords. We do not explore this simplistic, though obviously very damaging attack, further.

5.2.2 Hash order unknown (L2), chosen documents

Lastly, we consider schemes in which the hashes of keywords are stored at the server in random order. As in the known-document case, an adversary who plants a chosen document will only learn the *set* of keyword hashes for the document, not the one-to-one mapping of keywords to hashes. Clearly, the server can learn a single word unambiguously by planting a one-word document. Adopting a more sophisticated strategy, an adversary may seek to insert documents so as to maximize the "yield" of keywords learned per inserted document, and minimize the number of potential errors. We present and analyze an attack based on frequency analysis of a related corpus, which allows the server to trade off error probability against the size and number of inserted documents, giving an effective and flexible attack strategy.

Attack description. From a known related corpus, the adversary generates a list of (possibly stemmed) keywords ranked by frequency. Fixing a document size k, the adversary divides the ranked keyword list into k equal-sized slices. He then generates a k-word document by choosing the top word from each slice. The goal is to maximize the *frequency distance* between keywords in a given document.

The adversary also computes the frequency distribution of keyword hashes in the data uploaded by the client. After observing

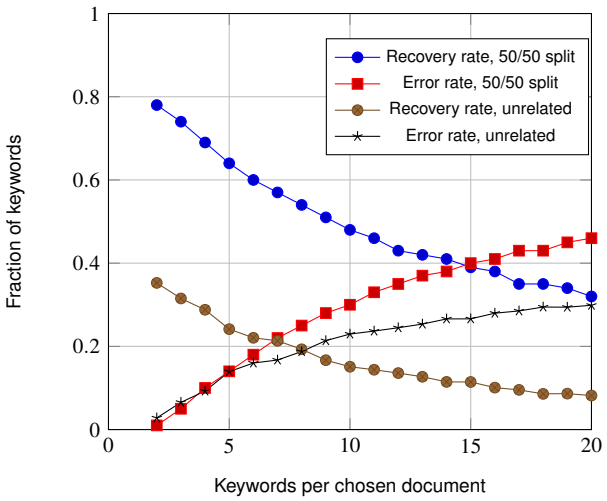

Figure 9: Apache keyword recovery rate and error rate for planted documents, unknown order of keyword hashes, 5000 keywords considered.

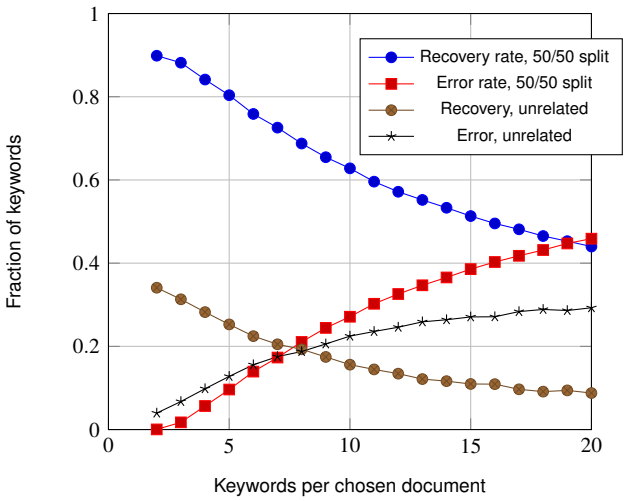

Figure 10: Enron keyword recovery rate and error rate for planted documents, unknown order of keyword hashes, 5000 keywords considered.

the hashes of the words in the planted document, the adversary ranks them by their frequency in the uploaded dataset, and guesses that the hashes correspond to the keywords of the same rank based on his own corpus. All k guesses are correct as long as there are no *rank reversals* in their keyword frequencies between the two datasets. The adversary can repeat this process for all known keywords or for as many documents as he is able to insert.

Experimental setup. We used two setups, the first one to model an adversarial server that has access to a closely related corpus, the second to model a server having access only to unrelated corpus in the same language. For the first, we divided a single data set in half, with the server "training" on 50% of the documents to learn keyword frequencies, while the client processes the documents in the other half and uploads encrypted keywords to the server. For the second, we used keyword sets and frequencies from the Enron dataset to attack the Apache dataset, and vice versa.

In each experiment, we generated chosen documents to cover all the keywords known to the server and measured the average recovery rate as well as error rate of client keywords. The plots in Figure 10 and Figure 9 show the tradeoff between number of words per chosen document and error rate for the Enron and Apache datasets, respectively. The data are averaged over 10 runs per dataset for the 50/50 split experiment. The variance was low in all experiments. As expected, the larger the number of slices/keywords per document k, the greater the probability of error, as the words in a document will be closer in frequency, increasing the likelihood of a rank reversal between the client and server datasets.

The recovery rate is lower than 1 minus the error rate, because not all of the client-indexed keyword hashes will occur in the server's dataset; these are not a source of errors, as the adversary knows he has no information on them.

For the single-dataset 50% split scenario, at around $k = 19$ keywords per document, the error rate becomes higher than the recovery rate, and thus the attacker will gain very little information. Also as expected, for the unrelated-corpus scenario, the recovery rate starts much lower, and a smaller slice size must be used to avoid too many errors. However, the error rate is not worse in the cross-dataset experiment, showing that word frequency rank is fairly consistent across these two email domains.

To recover w keywords, the adversary must plant at least w/k documents, more to allow for the error rate. The choice of document size must be informed by the attacker's desired error rate and the probability of detection. At any rate, the attack allows even an adversary who can plant a small number of documents to learn a sensitive selection of keywords.

6. CONCLUSION

We close with a summary of the lessons learned from this work. Our attacks begin to map the risks in using SE at various leakage levels.

First, we conclude that it is dangerous to attempt to protect queries on known document sets using SE schemes, even those proven to only admit L1 leakage. This reinforces the work of IKK, our attacks improve on theirs to show that L1 leakage can be used to learn query values even when the search space becomes very large. In this case one might consider more secure but unfortunately significantly slower primitive like oblivious ORAM [8].

Second, we conclude that leakage levels L2 and L3 are risky for deployments that enable active insertion of documents or insert a few publicly-known documents. Both are a threat against outsourced email storage, and the latter may be applicable in other scenarios (e.g., a user downloads and stores public documents). We note that future work is likely to show damaging L2 and L3 leakage attacks that are passive, relying only on more nuanced statistical analysis of an index's revealed co-occurrence relationships.

Our second conclusion motivates the usage of SE with L1 level leakage over weaker variants. But we cannot conclude that L1 is necessarily safe. In settings where a large fraction of the possible keywords are queried, L1 leakage devolves to that of L2 leakage. Future work may seek leakage levels even lower than L1, for example by accepting some small amount of false positives and/or false negatives in search.

Acknowledgements

The authors would like to thank Seny Kamara and Tarik Moataz for bringing an inconsistency in a prior draft to the authors' attention. Their help is greatly appreciated.

Cash was supported in part by NSF grant CNS-1453132. This work was done in part while Cash was visiting the Simons Institute for the Theory of Computing, supported by the Simons Foundation and by the DIMACS/Simons Collaboration in Cryptography through NSF grant CNS-1523467. Ristenpart was supported in part by NSF grants CNS-1546033, CNS-1330308, and CNS-1065134 and a generous gift from Microsoft. Perry was supported in part by DARPA under agreement number FA8750-13-2-005.

The opinions of this paper do not necessarily reflect those of the authors' employers or the funding agencies that supported this work.

7. REFERENCES

[1] Enron email dataset. `https://www.cs.cmu.edu/~./enron/`. Accessed: 2015-05-13.

[2] Bitglass. Security, Compliance, and Encryption. `http://www.bitglass.com/solutions/salesforce-encryption`.

[3] D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *NDSS 2014*, San Diego, California, USA, Feb. 23–26, 2014. The Internet Society.

[4] D. Cash, S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 353–373, Santa Barbara, CA, USA, Aug. 18–22, 2013. Springer, Berlin, Germany.

[5] CipherCloud. Cloud Data Encryption. `http://www.ciphercloud.com/technologies/encryption/`.

[6] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In A. Juels, R. N. Wright, and S. Vimercati, editors, *ACM CCS 06*, pages 79–88, Alexandria, Virginia, USA, Oct. 30 – Nov. 3, 2006. ACM Press.

[7] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[8] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM*, 43(3):431–473, 1996.

[9] W. He, D. Akhawe, S. Jain, E. Shi, and D. Song. Shadowcrypt: Encrypted web applications for everyone. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 1028–1039. ACM, 2014.

[10] M. S. Islam, M. Kuzu, and M. Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *19th Annual Network and Distributed System Security Symposium, NDSS 2012*. The Internet Society, 2012.

[11] S. Kamara and C. Papamanthou. Parallel and dynamic searchable symmetric encryption. In A.-R. Sadeghi, editor, *FC 2013*, volume 7859 of *LNCS*, pages 258–274, Okinawa, Japan, Apr. 1–5, 2013. Springer, Berlin, Germany.

[12] S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In T. Yu, G. Danezis, and V. D. Gligor, editors, *ACM CCS 12*, pages 965–976, Raleigh, NC, USA, Oct. 16–18, 2012. ACM Press.

[13] K. Kurosawa. Garbled searchable symmetric encryption. In N. Christin and R. Safavi-Naini, editors, *FC 2014*, volume 8437 of *LNCS*, pages 234–251, Christ Church, Barbados, Mar. 3–7, 2014. Springer, Berlin, Germany.

[14] K. Kurosawa and Y. Ohtaki. How to update documents verifiably in searchable symmetric encryption. In M. Abdalla, C. Nita-Rotaru, and R. Dahab, editors, *CANS 13*, volume 8257 of *LNCS*, pages 309–328, Paraty, Brazil, Nov. 20–22, 2013. Springer, Berlin, Germany.

[15] B. Lau, S. Chung, C. Song, Y. Jang, W. Lee, and A. Boldyreva. Mimesis aegis: A mimicry privacy shield–a systems approach to data privacy on public cloud. In *Proceedings of the 23rd USENIX conference on Security Symposium*, pages 33–48. USENIX Association, 2014.

[16] M. Naveed, M. Prabhakaran, and C. A. Gunter. Dynamic searchable encryption via blind storage. In *2014 IEEE Symposium on Security and Privacy*, pages 639–654, Berkeley, California, USA, May 18–21, 2014. IEEE Computer Society Press.

[17] W. Ogata, K. Koiwa, A. Kanaoka, and S. Matsuo. Toward practical searchable symmetric encryption. In K. Sakiyama and M. Terada, editors, *IWSEC 13*, volume 8231 of *LNCS*, pages 151–167, Okinawa, Japan, 2013. Springer, Berlin, Germany.

[18] I. Skyhigh Networks. Skyhigh for Salesforce. `https://www.skyhighnetworks.com/product/salesforce-encryption/`.

[19] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 14-17, 2000*, pages 44–55. IEEE Computer Society, 2000.

[20] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy*, pages 44–55, Oakland, California, USA, May 2000. IEEE Computer Society Press.

[21] E. Stefanov, C. Papamanthou, and E. Shi. Practical dynamic searchable encryption with small leakage. In *NDSS 2014*, San Diego, California, USA, Feb. 23–26, 2014. The Internet Society.