The Shadow Nemesis: Inference Attacks on Efficiently Deployable, Efficiently Searchable Encryption

David Pouliot
Portland State University
Portland, OR 97207
dpouliot@cs.pdx.edu

Charles V. Wright
Portland State University
Portland, OR 97207
cvwright@cs.pdx.edu

ABSTRACT

Encrypting Internet communications has been the subject of renewed focus in recent years. In order to add end-to-end encryption to legacy applications without losing the convenience of full-text search, ShadowCrypt and Mimesis Aegis use a new cryptographic technique called "efficiently deployable efficiently searchable encryption" (EDESE) that allows a standard full-text search system to perform searches on encrypted data. Compared to other recent techniques for searching on encrypted data, EDESE schemes leak a great deal of statistical information about the encrypted messages and the keywords they contain. Until now, the practical impact of this leakage has been difficult to quantify.

In this paper, we show that the adversary's task of matching plaintext keywords to the opaque cryptographic identifiers used in EDESE can be reduced to the well-known combinatorial optimization problem of weighted graph matching (WGM). Using real email and chat data, we show how off-the-shelf WGM solvers can be used to accurately and efficiently recover hundreds of the most common plaintext keywords from a set of EDESE-encrypted messages. We show how to recover the tags from Bloom filters so that the WGM solver can be used with the set of encrypted messages that utilizes a Bloom filter to encode its search tags. We also show that the attack can be mitigated by carefully configuring Bloom filter parameters.

Keywords

Security; Efficiently Deployable Efficiently Searchable Encryption; Encrypted Email

1. INTRODUCTION

Encrypting Internet communications has been the subject of renewed focus in recent years. Encrypting only the links (e.g. with TLS) is no longer considered sufficient; instead, recent work has focused on encrypting communications from end-to-end between client devices. Whenever a message is transmitted or stored on a remote service, it is first encrypted with a key that is kept secret from the service provider. This approach protects against malicious or compromised providers.

ShadowCrypt [17] and Mimesis Aegis [22] propose a novel approach for adding end-to-end encryption to cloud based communication services that do not support it natively, including Gmail, Facebook, and Twitter. To do this, both ShadowCrypt and Mimesis interpose themselves between the user interface of the legacy application and the human user. ShadowCrypt works with web applications and builds on the Shadow DOM, an emerging standard for isolating components on a web page from each other in the browser. Mimesis Aegis works with Android applications by injecting a transparent window over the top of the legacy app. In both cases, the new security layer captures the user's input and encrypts it before handing it off to the legacy app. Similarly, when the app retrieves encrypted data from the cloud, the security layer transparently decrypts it and displays the plaintext to the user.

At the same time, the large number of messages that users send and receive each day can lead to a form of "information overload" [31] where it becomes very difficult for the user to manage their messages, e.g. to find a message that they received in the past. The most effective tool for combating information overload is full-text search. Highly effective search is an important feature of many successful apps such as Gmail. In order to achieve widespread deployment, any new security layer must not remove the user's ability to efficiently search their stored messages. There are many cryptographic techniques for searching on encrypted data. Many of these have formal proofs that they conform to rigorous definitions of security [12] based on semantic security [15]. Unfortunately these constructions cannot be deployed on top of the legacy services that ShadowCrypt and Mimesis aim to support. Instead, ShadowCrypt and Mimesis propose the use of simplified constructions that the Mimesis authors call *efficiently deployable, efficiently searchable encryption* (EDESE).

EDESE.

In order to achieve reasonable performance, all practical techniques for searching on encrypted data leak some amount of information about the encrypted database and the search queries. This leakage profile is typically small,

CCS'16, October 24 - 28, 2016, Vienna, Austria

ISBN 978-1-4503-4139-4/16/10...$15.00

DOI: http://dx.doi.org/10.1145/2976749.2978401

precisely defined, and rigorously verified with formal proofs of security. However, the *implications* of the leakage—that is, what other sensitive information can be inferred from it—are not yet well understood.

In order to achieve backwards compatibility with legacy systems, EDESE schemes like those used in Mimesis and ShadowCrypt have a particularly aggressive leakage profile. To enable a legacy service or application to search on encrypted messages, EDESE attaches to each message a list of opaque identifiers—here we call them "tags"—that correspond to the set of keywords in the message. Each tag t is computed as a pseudorandom function of a keyword w, using a secret key k. When the user initiates a search for a keyword w, Mimesis and ShadowCrypt intercept the user's input and replace the plaintext query with one or more tags $t = F_k(w)$ before returning control back to the legacy app. On the back end, the legacy application can use any standard full-text indexing technique to keep track of the list of messages that contain each tag. Given a search request for tag t, the legacy app's back end consults its index and returns the encrypted documents to the front end. When the front end goes to display the search results, Mimesis and ShadowCrypt again intercept the user interface to decrypt the messages and display the plaintext to the user.

The practical benefits of EDESE are clearly compelling. It allows users to immediately begin encrypting their communications, without changing providers or losing familiar application user interfaces. The EDESE approach also has much lower startup costs compared to other techniques for searching encrypted data. According to a case study by Grubbs [16], implementation and deployment of a conventional symmetric searchable encryption system requires around one person-year of effort.

On the other hand, EDESE reveals a great deal of information that could be useful to an adversary, namely: (*i*) the relative frequency of each tag in the corpus and (*ii*) the frequency with which each pair of tags occurs together in the same message. In contrast, with a conventional SSE scheme, the adversary is only allowed to learn the relationship between a tag and a document when the user performs a search for corresponding keyword. The additional leakage from EDESE provides the opportunity for statistical inference attacks whereby the adversary uses known word frequencies to match the observed tags back to the keywords that they represent.

Recent works make conflicting claims about the security of SSE and EDESE schemes against such inference attacks. As a result, it has been difficult so far to determine whether the gains in efficiency and convenience that EDESE offers are worth the increased risk of inference attacks. Up until now, there have been no known attacks against the version of EDESE used in Mimesis, and the one known attack against the EDESE used in ShadowCrypt only works against a weakened version of the scheme. We will refer to the efficiently searchable encryption scheme used in Shadowcrypt as SC-ESE and the scheme proposed in Mimesis as MA-ESE.

Our Contributions.

In this paper, we show that the risks of using EDESE schemes are much greater than previously known. We describe new inference attacks that enable an honest but curious adversary to recover the plaintext keywords for hundreds of the most common tags. Our attacks are the first inference attacks that are effective against the full-strength version of SC-ESE, and the first ever inference attack against the more sophisticated MA-ESE construction used in Mimesis Aegis.

Our threat model is conservative, in that we give the adversary access to only the ciphertext messages and the search tags. Unlike the standard adversary model for searchable encryption, our adversary has no access to encryption or decryption oracles. Notably, unlike previous inference attacks, our techniques are effective even when the adversary has no *a priori* knowledge of the search queries, the plaintext messages, or the plaintext keywords for any of the tags.

A key insight underlying our approach is that the SC-ESE adversary's task can be reduced to well-known combinatorial optimization problems based on graph matching: *weighted graph matching* and *labeled graph matching*. Although these graph matching problems are in NP, there exist several efficient solvers that can find good approximate solutions in polynomial time. Another key observation of our work is that constructions like MA-ESE that use Bloom filters for encrypted search must be careful in how they configure the Bloom filter's parameters. If the filter parameters are chosen carelessly, or with only efficiency in mind, then the adversary can use an additional pre-processing step to apply the graph matching attacks against the bits in the Bloom filter.

Using real email and chat data, we show how these solvers can be used to efficiently and accurately recover the list of keywords for messages encrypted with MA-ESE and SC-ESE. For example, for several users in the Enron email corpus, the attack can recover more than 900 of the top 1000 most common keywords. In a corpus of chat messages from the Ubuntu Linux project, it recovers more than half of the top 500 keywords. Recovering so many of the top keywords would enable the adversary to perform a variety of interesting analyses on the encrypted documents, such as grouping similar documents together in clusters or identifying the *sentiment* (positive/negative, happy/sad/angry) expressed in each message.

To mitigate against our attacks, we propose and evaluate a new strategy based on careful tuning of the Bloom filter parameters to reduce the information leaked by the tags. Experimental results show that an efficient choice of parameters is sufficient to break our current attack, while better protection is possible at the cost of increased space overhead. However, we caution that our proposed defense does not eliminate the information leakage entirely. Given a sufficient amount of data, it is still possible that a clever adversary might be able to reverse-engineer the Bloom filter.

Beyond the immediate impact to efficiently searchable encryption, our results here may have implications for the security of other encrypted search systems that use Bloom filters, such as SADS [26, 25] and BlindSeer [24, 13], and for systems that perform symmetric searchable encryption over natural language documents [14, 12, 10].

2. BACKGROUND AND RELATED WORK

Full-Text Indexing.

To enable efficient full-text search on collections of documents, one standard approach is to construct an *inverted index*. For each keyword, the index contains a list of all the documents that contain that word [32]. The most common words like *the, a, and, or, of*, etc. are called "stop

words" and are typically excluded from the index. Excluding these extremely common words significantly reduces the space requirements for the index without sacrificing much expressive power. Since stop words appear in almost every document, they are not usually useful as search terms. Recent searchable encryption schemes like [10] work by essentially encrypting, and then selectively revealing, parts of an inverted index. In contrast, EDESE schemes work by letting an existing full-text search system include cryptographic tags in its inverted index.

Bloom Filters.

Bloom filters [6] are probabilistic data structures that represent sets and support membership queries. For applications that can tolerate a small false positive rate, Bloom filters offer a space-efficient alternative to the full inverted index. Conceptually, the Bloom filter is an array or bit vector of m bits, all initially set to zero. To insert an element x into the set, we hash x with each of k hash functions and set each of the k bits $b_i = h_i(x), 1 \leq i \leq k$ to one in the Bloom filter. To check if an item z is in the set, we check if $h_i(z) = 1$ for all $1 \leq i \leq k$.

The standard BF construction described by Bloom allows anyone to check for the presence of an item in the filter. This is not desirable for indexing encrypted data; it could be leveraged by an attacker to perform a dictionary attack. Therefore EDESE schemes use a pseudorandom function (PRF) to set the bits in the Bloom filter. More formally, let $f_1, f_2, ..., f_k$ be a family of k pseudorandom functions. Let $F(w)$ be the set of bits in the Bloom filter that correspond to keyword w, ie $F(w) = \{f_i(w) : i \in [1, k]\}$. Only someone who has the secret symmetric key can compute the PRF. This prevents simple brute-force dictionary attacks on the hash function. In practice, the PRF can be instantiated as a truncated message authentication code (MAC), sometimes also called a "keyed hash." For example, Mimesis Aegis uses HMAC-SHA256 to set bits in a Bloom filter of size 2^{24}.

Searching on Encrypted Data.

Song, Wagner, and Perrig proposed the first scheme for searching on encrypted data [28]. Goh [14] and Curtmola, Garay, Kamara, and Ostrovsky [12] formalized definitions of security and described new schemes that met those definitions. Goh was also the first to propose use of Bloom filters for encrypted search. The EDESE construction used in Mimesis is very similar to a weaker version of Goh's scheme. Early SSE schemes assumed a relatively static document corpus and offered somewhat limited performance. Current SSE schemes can handle dynamic data [19] and offer good performance even on very large data sets [10]. However, the security definitions for SSE require that the server must engage in a cryptographic protocol with the client to execute searches on their behalf. Conventional SSE schemes are therefore not compatible with existing "legacy" applications and services.

Efficiently Searchable Encryption.

Bellare, Boldyreva, and O'Neill [4] introduced the notion of efficiently searchable encryption (ESE) and presented a new definition of security, called PRIV, for such schemes. They showed that a simple hash-and-encrypt construction is secure in this model when the data has high min-entropy. Amanatidis, Boldyreva, and O'Neill [3] introduced the related notion of efficiently searchable authenticated encryption (ESAE) which also guarantees the authenticity of ciphertexts. They prove that a simple MAC-and-encrypt scheme satisfies this definition. Unlike conventional searchable encryption, ESE and ESAE schemes can be efficiently indexed and searched by an unsecured full-text search system. The tagging scheme used in ShadowCrypt's SC-ESE is conceptually similar to the construction in [3].

Inference Attacks on Encrypted Data.

In an inference attack, the adversary uses some outside "auxiliary" information to exploit leakage from a cryptographic construction in order to infer the value of some hidden data. The original statistical inference attack was al-Kindi's frequency analysis [1], first proposed in the 9^{th} century AD. It was developed to break classical crypto schemes such as substitution ciphers, and it is still widely used to illustrate the weakness of these schemes in introductory cryptography courses. More recent inference attacks have also targeted efficient schemes for storing and searching records in relational databases [27] [23] and anonymized packet traces [7]. Interestingly, a new analysis by Lacharité and Paterson [21] proves that frequency analysis is the maximum likelihood estimator for deterministic encryption.

Islam, Kuzu, and Kantarcioglu [18] presented the first inference attack against symmetric searchable encryption. They noted that over time, as more searches are performed in an SSE, the server can see which tags tend to occur together in the same documents. In the IKK attack, the adversary observes the frequency of co-occurrence for each pair of words in some corpus of training data, then uses this information to map tags in the encrypted corpus back to plaintext words. IKK prove that the adversary's task is an NP Complete combinatorial optimization problem, but they also demonstrate that simulated annealing can be used to recover most of the top few hundred keywords in under 14 hours. We apply a similar approach for our attack against EDESE, but our attack is even stronger. We provide a reduction to a well-known problem that has been studied for 30 years and has good off-the-shelf solvers.

Unfortunately the IKK experiments use the same data for the target and for the adversary's auxiliary information—this implicitly assumes that the adversary has perfect knowledge of the word co-occurrence frequencies. Cash, Grubbs, Perry, and Ristenpart (CGPR) attempted to reproduce the IKK experiments [9], and their results show that the accuracy of the IKK attack degrades quickly as the adversary's knowledge of the encrypted corpus decreases. CGPR present new, simpler attacks that outperform the IKK attack but also require knowledge of a large fraction of the target data. CGPR also show an inference attack against a weakened version of SC-ESE that reveals the order in which the tags appear in the plaintext document. Our attacks work against the full-strength scheme that hides the original ordering.

3. GRAPH MATCHING

Weighted Graph Matching.

The weighted graph matching (WGM) problem is a well known combinatorial optimization problem that has been studied for nearly 30 years [11]. Given two weighted graphs G and H with n nodes each, the problem is to find the

permutation that re-labels the nodes in H so that the permuted graph most closely resembles G. More formally, let $A_G = [g_{ij}]$ and $A_H = [h_{ij}]$ be the adjacency matrices of G and H, respectively. Here, $g_{ij} \geq 0$ gives the weight of the edge connecting nodes i and j in G, and $h_{ij} \geq 0$ gives the weight in H. Further, let X be an $n \times n$ permutation matrix, and let $A'_H = XA_HX^T$ be the adjacency matrix for the permuted version of H, with edge weights h'_{ij}. The goal of the optimization problem is then to find the permutation matrix that minimizes the matrix distance between A_G and A'_H. For example, using the Euclidean distance as our matrix distance, the problem can be stated as

$$\begin{aligned}
\textbf{minimize} \quad & ||A_G - XA_HX^T||_2 = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}(g_{ij} - h'_{ij})^2} \\
\textbf{subject to} \quad & \sum_{i=1}^{n} X_{ij} = 1, \qquad 1 \leq j \leq n \\
& \sum_{j=1}^{n} X_{ij} = 1, \qquad 1 \leq i \leq n \\
& X_{ij} \in \{0,1\}, \qquad 1 \leq i,j \leq n.
\end{aligned}$$

The WGM problem is in NP. There exist many algorithms for efficiently finding approximate solutions, including an influential 1988 paper by Umeyama [29] that uses eigendecomposition of the adjacency matrices to find a nearly-optimal solution in $O(n^3)$ time. Umeyama's algorithm works especially well when the two input graphs are nearly perfectly isomorphic. The PATH algorithm [33] is more robust, using an adaptive path-following strategy; it also runs in $O(n^3)$ time, but with a larger constant factor than Umeyama's algorithm. A powerful linear programming (LP) technique from Almohamad and Duffuua [2] has complexity $O(n^7)$, so we do not consider it for use in practical inference attacks.

Labeled Graph Matching.

Labeled graph matching (LGM) is a further generalization of WGM. Whereas in WGM the similarity of two graphs is computed as a function of their edge weights, in LGM the nodes may also have weights. The best matching is the one that simultaneously minimizes the difference in edge weights while maximizing the similarity of the node weights. For example, if C_{ij} gives the similarity of node i's weight in G with node j's weight in H, and $\mathbb{P}$ is the set of all permutation matrices, then the permutation that maximizes the similarity of the node weights is

$$\max_{X \in \mathbb{P}} \text{tr}(C^T X) = \max_{X \in \mathbb{P}} \sum_{i=1}^{n}\sum_{j=1}^{n} C_{ij}X_{ij}.$$

A natural way to include both the edge weights and the node weights in a single objective function is with a simple linear combination where the parameter α designates the weight given to the terms of the linear combination. The full optimization problem can then be stated as

$$\begin{aligned}
\textbf{minimize} \quad & (1-\alpha)\ ||A_G - XA_HX^T||_2 - \alpha\ \text{tr}(C^T X) \\
\textbf{subject to} \quad & \sum_{i=1}^{n} X_{ij} = 1, \qquad 1 \leq j \leq n \\
& \sum_{j=1}^{n} X_{ij} = 1, \qquad 1 \leq i \leq n \\
& X_{ij} \in \{0,1\}, \qquad 1 \leq i,j \leq n.
\end{aligned}$$

Both the Umeyama algorithm and PATH can be easily adapted to solve the labeled graph matching problem. The GraphM software package [33] includes efficient implementations of these and other algorithms.

4. ATTACKS ON SC-ESE

The idea of using word co-occurrence frequencies for inference attacks against symmetric searchable encryption was first proposed by Islam, Kuzu, and Kantarcioglu [18]. Here we improve on this attack strategy for EDESE and formalize it as an instance of the graph matching problems described above. We also compare our attack to previous inference attacks, including classical frequency analysis and the ℓ_p-optimization technique from Naveed et al [23].

4.1 Frequency Analysis and ℓ_p-Optimization

The application of frequency analysis to attack EDESE is straightforward. Given an auxiliary corpus of plaintext messages, a target corpus of encrypted messages and their tags, the adversary simply counts the number of times that each keyword appears in the auxiliary data and the number of times each tag appears in the target data. He sorts both the list of keywords $\mathbf{w} = \{\mathbf{w}_1, ..., \mathbf{w}_n\}$ and the list of tags $\mathbf{t} = \{\mathbf{t}_1, ..., \mathbf{t}_n\}$ by their frequency. So, for example, $\mathbf{w}_1$ is the most common keyword and $\mathbf{w}_2$ is the second-most common, and so forth. Finally, the adversary concludes that the i^{th} most common tag corresponds to the i^{th} most common keyword; that is, $\mathbf{t}_i \equiv \mathbf{w}_i$, for all $i \in [1, n]$.

Naveed, Kamara, and Wright [23] also use frequency information in an inference attack. Rather than simply matching up plaintexts to ciphertexts in order of decreasing frequency, they pose the problem as a *linear sum assignment problem* to find the optimal matching that minimizes the total difference in frequencies. For $p \geq 2$, they found that ℓ_p-optimization produced results that are identical to frequency analysis; this strongly suggests that the two approaches may in fact be equivalent.

Both of these previous attacks rely solely on the frequencies of individual plaintexts and ciphertexts. Although they were highly effective against categorical data in encrypted databases, our experimental results in Section 4.3 show that they are much less accurate against natural language data, where the frequencies of plaintext keywords are much noisier. Instead, as in the IKK attack [18], we use the frequencies of *pairs* of words occurring together to drive our graph matching attacks for much greater accuracy.

4.2 Graph Matching Attacks

We now give the polynomial-time reduction of the inference attack on EDESE to the graph matching problems, WGM and LGM. Given a plaintext corpus for use as the adversary's auxiliary information and an EDESE-encrypted

corpus as his target, he first removes the most common "stop" words (e.g. *a, the, and, of, ...*) from the auxiliary data, because the victim system almost certainly stripped them from the target data before generating the tags. Then the adversary selects the top n most common remaining keywords $\mathbf{w} = \{\mathbf{w}_1, ..., \mathbf{w}_n\}$ from the auxiliary data and the top n most common tags $\mathbf{t} = \{\mathbf{t}_1, ..., \mathbf{t}_n\}$ from the target data. He creates two graphs G and H to represent the auxiliary and target data, respectively, as follows. For each $i, j \in [1..n]$, he sets the weight of the edge g_{ij} in G to be the probability, over the auxiliary corpus, that keywords $\mathbf{w}_i$ and $\mathbf{w}_j$ occur in the same document. Similarly, he sets the weight h_{ij} in H to be the probability, over the target data, that tags $\mathbf{t}_i$ and $\mathbf{t}_j$ are attached to the same encrypted document. This is sufficient to reduce the attack to the WGM problem.

To yield an instance of the LGM problem, the adversary must create a similarity matrix C for the nodes. Intuitively, each cell C_{ij} in the matrix should give the similarity of the frequency of word $\mathbf{w}_i$ compared to the frequency of tag $\mathbf{t}_j$. There are many ways to capture this similarity. For example, we might set the node weights as in [23] to minimize the overall difference in frequencies. Here, we opt instead for a slightly different approach based on the method of maximum likelihood. Let g_i be the fraction of auxiliary documents that contain word $\mathbf{w}_i$ and h_j be the frequency of tag $\mathbf{t}_j$ in the target data. Let D be the number of documents in the target. Let $k_j = h_j \cdot D$. Then the adversary sets the similarity C_{ij} as the likelihood that word $\mathbf{w}_i$ appears in k_j out of D documents. The permutation matrix X that maximizes the objective function is therefore the maximum likelihood solution.

$$
\begin{aligned}
D &= \text{Number of Documents} \\
g_i &= \text{Frequency of word } \mathbf{w}_i \\
h_j &= \text{Frequency of tag } \mathbf{t}_j \\
k_j &= h_j \cdot D \\
C_{ij} &= \text{Binom}(k_j, D, g_i) \\
C_{ij} &= \binom{D}{k_j} (g_i)^{k_j} (1 - g_i)^{D - k_j}
\end{aligned}
$$

The adversary solves the graph matching problem to find the optimal permutation matrix X' that most closely maps H to G. He then applies the same permutation to the list of tags to obtain the permuted list $\mathbf{t}' = X'\mathbf{t}$. Finally, the adversary concludes that each tag $\mathbf{t}'_j$ in the encrypted corpus represents keyword $\mathbf{w}_j$ from the auxiliary data, for all $j \in [1, n]$

4.3 Empirical Evaluation

ShadowCrypt and Mimesis aim to support email and other messaging applications, including Gmail, Twitter, WhatsApp, and others. To evaluate the practical impact of our attacks, we use two data sets of real email and chat messages. The Enron email corpus [20] includes real emails from the mailboxes of 150 employees of Enron Corporation, received between 2000 and 2002. It was originally made public as part of the federal government's investigation into the company's collapse, and it has since been used in several studies on the practicality of searchable encryption schemes [19] and the effectiveness of inference attacks [18, 9]. The Ubuntu Chat Corpus [30] is composed of archived chat logs from Ubuntu's Internet Relay Chat technical support channels. This corpus comes from the logs between July 2004 and October 2012. Table 1 summarizes the data sets that we use to evaluate our attacks on real data.

4.3.1 Initial Experiments

For a direct comparison with prior work that tests and trains on the same data [18] or that gives the adversary access to the plaintext corpus [9], we performed a small initial experiment with the Enron corpus. We note that normally, testing and training on the same data is considered exceptionally bad practice. However, for tools like ShadowCrypt and Mimesis Aegis, there is one real scenario where this might give an appropriate model for the adversary's capabilities. Suppose a user has a large corpus of messages stored on a service like Gmail, and she decides to encrypt all of her old messages using EDESE. At the moment when she finishes uploading the encrypted messages, the server has perfect knowledge of both the old plaintext corpus and its new EDESE tags. Since the server already has the old plaintext corpus, the point of performing the attack at this stage is to learn information about the tags. If the server can match tags to keywords at that point in time, it can recover the keywords in each new encrypted message almost for free.

In this initial experiment, we divide each Enron user's mails randomly into a training set and a testing set. Here we ignore the testing set, and we use the training set as both the adversary's auxiliary information and the target data. We use the training set to construct an adjacency matrix as described in Section 4.2 and we use this matrix as both A_G and A_H. We use the open source `graphm` tool with the Umeyama and PATH algorithms to find the permutation that most closely matches A_G to A_H.

In this easy attack scenario, the weighted graph matching attack performs extremely well. The Umeyama WGM algorithm achieves perfect 100% accuracy for every user in the corpus, as does simple frequency analysis. This is to be expected, as both algorithms are optimized for the case where the auxiliary and target data have very few differences. The Umeyama algorithm matches each pair of graphs in under 40 seconds. The PATH algorithm is designed to handle greater variation in the graphs, so it runs roughly two orders of magnitude slower than Umeyama. Its accuracy on this experiment is also somewhat reduced compared to the naive algorithms. Figure 1 shows the complementary cumulative distribution function (CCDF) of the PATH WGM algorithm's accuracy across all 150 users in the Enron data set.

A point at position (x, y) on the graph means that the attack correctly matched at least x% of the keywords for y% of the users in the corpus. The attack recovers about 95% of the keywords for more than 90% of the users, with some slight degradation in accuracy as we expand the attack to target a larger number of keywords.

Previous attacks on standard SSE require *a priori* knowledge of the both target corpus and some number of the queries. Islam, Kuzu, and Kantarcioglu report that their simulated annealing attack could analyze up to 150 queries and 2500 keywords in under 14 hours, and it could recover more than 80% of the queries. That is, the IKK attack can recover between 120 and 150 of the top keywords, as long as they are used in a query by some user. Given a similar experimental setup with 10% of queries known *a priori*, Cash

Corpus	Type	Date	Messages	Keywords per Message
Enron	Email	2000–2002	517446	101
Ubuntu	IRC chat	2004–2012	26360715	6.57

Table 1: Email and chat corpora

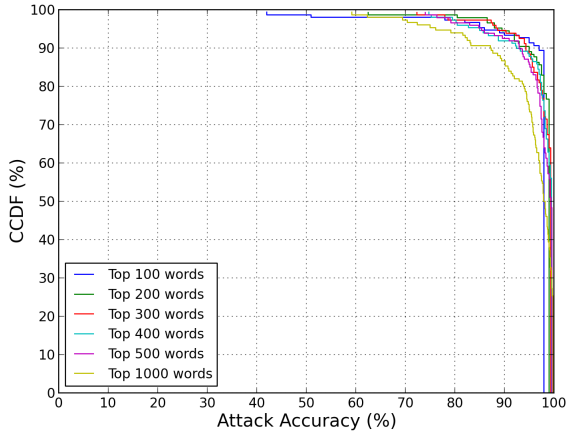

Figure 1: Accuracy of weighted graph matching attack (PATH algorithm) against SC-ESE for users in the Enron email corpus, using perfect auxiliary information

et al's *count attack* recovers 100% of the queries. The accuracy of our attack is similar to the related work, even when we have no known queries. But because EDESE gives us access to the tags for *all* the keywords, the practical impact is greater. Whereas a 95%-accurate attack on SSE might retrieve more than 140 keywords, our attack on EDESE recovers more than 950.

On the other hand, these results may give an overly pessimistic estimate of the security of SSE and EDESE. When given access to only 50% of the target data, both of the attacks from prior work achieve accuracy very near to zero (c.f. Fig. 6 in [9]). Next we look at what happens when we run our attack with no access to the target data.

4.3.2 Experiments with Imperfect Auxiliary Info

Here we consider a more realistic scenario, where the adversary does not have any specific knowledge of the messages in the encrypted corpus, but he still has very good estimates for the keyword frequencies. We conducted experiments in this model using data from the Enron email corpus and the Ubuntu chat corpus.

For each user in the Enron corpus, we randomly divided the user's emails into two non-overlapping sets. We took one half of the user's emails as the training set and used them to construct the adjacency matrix A_G for the adversary's auxiliary information. We took the other half of the user's emails as the test set and used them to construct the adjacency matrix A_H for the target data. We did this for several values of n between 100 and 1000.

To match the top n keywords, we first ran the attack using Frequency Analysis / ℓ_p-optimization and the Weighted Graph Matching attack with the Umeyama algorithm [29]. Figure 2 shows the complementary cumulative distribution function (CCDF) of the accuracy for the Frequency Analysis attack and the Weighted Graph Matching with the Umeyama algorithm. For the top 500 words, the accuracy for these two attacks is less than 10% for almost all users.

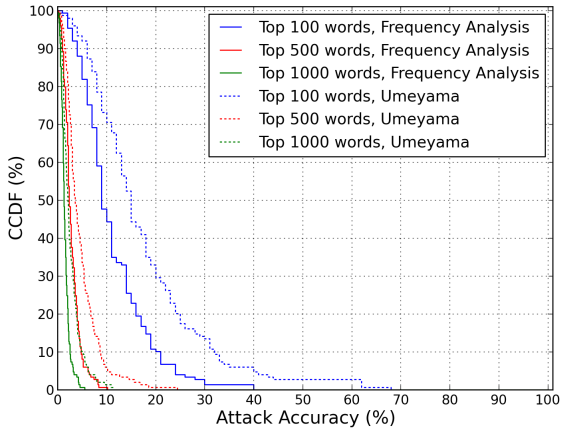

Figure 2: Accuracy of Frequency Analysis and Weighted Graph Matching (Umeyama) attacks for Enron data, with imperfect auxiliary information.

We then performed the Weighted Graph Matching attack with the PATH algorithm [33]. Figure 3 shows the (CCDF) of the attack's accuracy across the 150 users in the Enron corpus. Over all, the accuracy of the attack decreases as we increase the number of keywords targeted. But even when attempting to match the top 1000 words, the adversary still achieves over 90% accuracy for about 10% of the users. To reiterate—for these 15 unlucky users, the adversary recovers more than 900 of the top 1000 words in their email. If the adversary is only interested in the top 200 words, he achieves greater than 80% accuracy for half the users in the corpus. Note that, unlike previous work [18, 9], this attack succeeds given *no access to the target data*, *zero known keywords*, and *zero known documents*.

Further work will be required to understand why the attack's effectiveness varies so much from user to user. Our working hypothesis is that the variation stems from differences in the users' *topic model*. Like the earlier IKK attack, our adversary assumes that the probability of seeing each word—or each pair of words—in a document is constant across the entire corpus. For natural language text, this assumption does not really hold. Instead, in more accurate models of text, such as latent Dirichlet allocation [5], the distribution of words is fixed for each of several *topics*, and the mix of topics can vary greatly from document to document. We suspect that users for whom the attack is very successful have a more stable distribution of topics in their email. It might be possible for a future attack to learn both the topic model and the word frequencies at the same time.

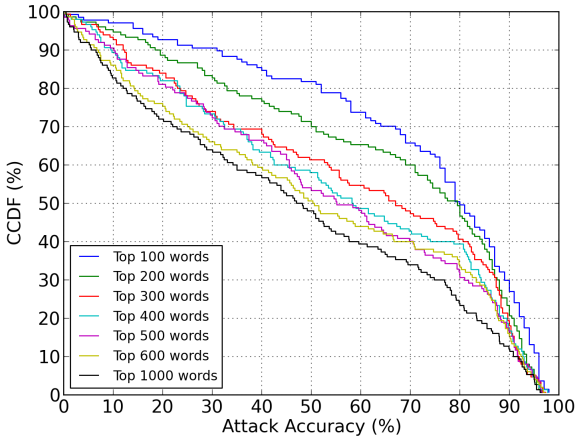

Figure 3: Accuracy of Weighted Graph Matching attack (PATH Algorithm) against SC-ESE for Enron data, with imperfect auxiliary information.

Following a similar procedure, we ran the experiment for each month of IRC chat logs from the Ubuntu corpus. We randomly assigned each day in the month to either the adversary's auxiliary information (ie, the training set) or the target data (ie, the test set). For each value of n, we created the adjacency matrices A_G and A_H as above, and we ran the `graphm` experiment for each month of the Ubuntu corpus, just as we did with each user in the Enron data.

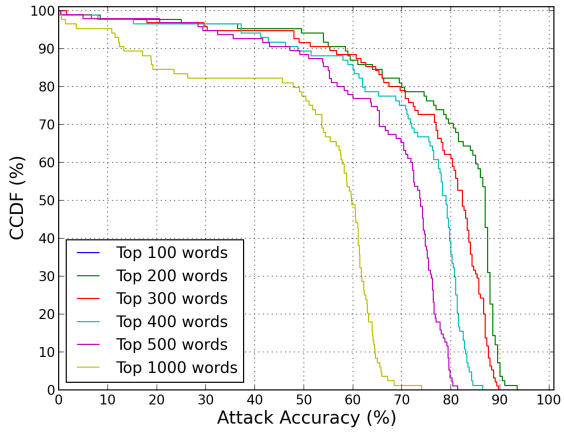

Figure 4: Accuracy of Weighted Graph Matching attack (PATH Algorithm) for Ubuntu data

Figure 4 shows the results for the Ubuntu experiment. Compared to the email data, overall the adversary's accuracy degrades more quickly as we increase the number of keywords targeted, but the attack is still many times more accurate than random guessing. For 10 percent of the months, the adversary correctly recovers almost 400 of the top 500 keywords. He recovers more than half of the top 500 keywords for nearly 90% of the corpus.

4.3.3 *Runtime Performance*

We ran all experiments on a cluster of HP Proliant servers with Intel Xeon L5520 processors at 2.26GHz, running CentOS Linux 6 and version 0.52 of the graphm software. The Umeyama algorithm required less than 40 seconds to run each attack in Section 4.3.1 with perfect 100% accuracy. However, its accuracy was substantially reduced when testing and training on different data.

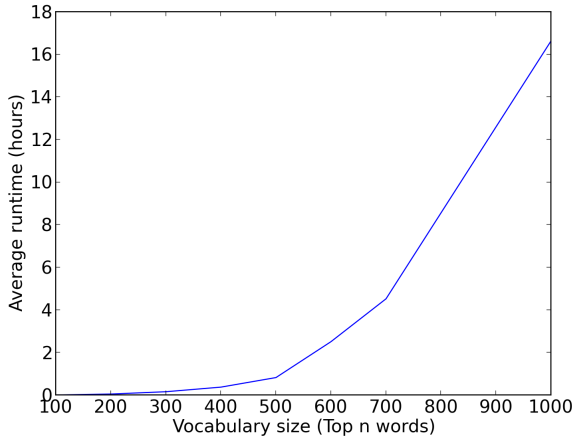

Figure 5: Runtime performance of the Weighted Graph Matching attack (PATH Algorithm) for the Enron email corpus

Figure 5 shows the average runtime for matching the top 100,200,...,1000 keywords for each Enron user with the PATH algorithm. Matching the top few hundred keywords is very fast; even our older 2009-era CPUs can match the top 500 words for a user in under one hour. The Umeyama and PATH algorithms are $O(n^3)$ in the number of keywords to be matched, so attacking thousands of keywords becomes increasingly expensive. However, matching several thousand keywords would not be beyond the capacity of a large corporation or a nation state. It is also possible that a much faster solver could be implemented using graphics processing units or other specialized hardware. Memory does not appear to be a limiting factor: even when matching the top 1000 keywords, the `graphm` process uses less than 250MB of memory.

4.3.4 *Experiments with Time Delay*

Figure 6 shows the accuracy of our attack when the adversary's auxiliary information is from the previous month before the target data. Comparing this graph to Figure 4, the attack performance is considerably lower. The cause for this degradation is likely due to the differences in topics, and thus words from one month to the next. The result is our auxiliary training data is not as close to the actual data compared to the attack on messages within the same month.

5. ATTACKS ON MA-ESE

MA-ESE is a more difficult target than SC-ESE. We cannot apply our graph matching attack directly, because MA-ESE does not reveal a one-to-one correspondence between keywords and tags. Where SC-ESE uses a single

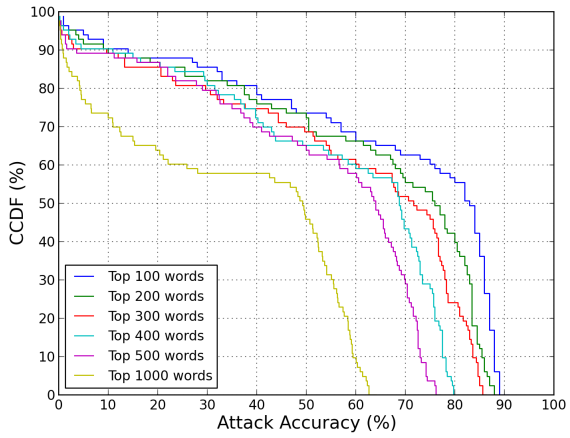

Figure 6: Accuracy of Weighted Graph Matching attack (PATH Algorithm) for Ubuntu data; 1 month delay between auxiliary and target

PRF to generate a single search tag for a given keyword, MA-ESE uses a family of k PRFs and generates up to k distinct tags for each keyword. If the Bloom filter is sufficiently small, there may be some collisions in the PRFs, so some tags may correspond to more than one keyword.

As we will show in Section 6, the MA-ESE construction provides the opportunity to make inference attacks much more difficult by carefully tuning the parameters of the Bloom filter. On the other hand, a naive choice of BF parameters such as those proposed in [22] allows the adversary to mount the same graph matching attack with only a small amount of additional work and a variable decrease in accuracy.

Our inference attack on MA-ESE proceeds in two steps. First, we analyze the Bloom filters to identify sets of bits that likely represent plaintext keywords. Then we use the graph matching attack to match each set of bits to the best-fitting keyword.

5.1 Recovering tags from Bloom filters

Our general strategy to discover the groups of bits that represent keywords in the Bloom filters begins with a simple frequency-based analysis. If the Bloom filters use k hash functions, then for each keyword we expect to see a group of k bits that (1) have the same bit counts where the bit count is the number of documents that the bit is set to one and (2) appear together in the same set of documents. For example, if a keyword w sets the bits 10, 20, 30 and 40, we expect each of these bits to have similar counts and appear together in the same documents. To find these bits for each keyword, we begin by counting the number of documents in which each bit is set, and we group together all the bits that have the same count.

For example, suppose we have a collection of Bloom filters with the parameters used in Mimesis Aegis: $m = 2^{24}$ bits and $k = 10$ hash functions. Figure 7 gives some example counts that arise for one Enron user with these parameters. Each row represents a set of bits that all occur in the same number of documents. The first column gives the count of the documents where these bits appeared, and the 2nd column gives the number of bits with that count.

Doc Count	Set Size
238	10
226	10
219	11
212	9
211	10
206	10
186	10
173	10
169	10
143	10
129	1

Doc Count	Set Size
113	10
101	10
99	9
98	10
89	10
87	10
84	20
82	10
81	10
80	20
79	40

Figure 7: Example Bloom filter counts

Exact Matching.

Sometimes it is easy to identify the bits for many keywords. In our example, there are exactly 10 bits that appear in 238 documents, another 10 bits that appear in 226 documents, and other sets of 10 that occur in 211, 206, 186 documents, respectively. It is very likely that these five sets of 10 bits correspond to five keywords that appear 238, 226, 211, 206, and 186 times in the plaintext corpus. Similarly, the other sets of 10 bits probably represent one plaintext keyword each. Each of these sets of 10 bits are equivalent to the tags from Section 4.

Algorithm 1 Bloom Filter Tag Extraction

Let S be the set of sets of bits
Let D be the set of encrypted documents
Let B be the set of Bloom filters $B = \{B_d : d \in D\}$
Let k be the number hash functions in each bloom filter

function FindTags(S, D, B, k)
    Let $T \leftarrow \varnothing$ ▷ T will be the set of extracted tags
    for $b \in S$ **do**
        if $|b| == k$ **then**
            $T \leftarrow T \cup b$
        else
            $T \leftarrow T \cup$ Split(b)
    return T

function Split(b)
    for $d \in D$ **do**
        Let $s_1 \leftarrow b \cap B_d$
        Let $s_2 \leftarrow b \setminus s_1$
        if $|s_1| == k$ **then return** $\{s_1\} \cup$ Split(s_2)
        else if $|s_2| == k$ **then return** $\{s_2\} \cup$ Split(s_1)
        else if $k < |s_1| < |b| - k$ **then return** Split(s_1) $\cup$ Split(s_2)
        else
            Continue
    return b

Other sets of bits likely include multiple plaintext keywords. For example in Figure 7, there are 20 bits that appear in 84 documents and 40 bits that appear in 79 documents. These probably represent two keywords that each appear 84 times and four keywords that each appear 79 times. We can identify the 10 bits that correspond to each distinct keyword if we can find an encrypted document that contains the given

keyword but none of the other keywords that have the same count. Algorithm 1 gives a more formal specification of our technique for finding such a document.

In some cases there is no such document in the encrypted corpus, and so our algorithm fails to split a larger set of bits into individual keywords. This means that the words in that set always appear together in the same documents, and therefore they will be indistinguishable under the graph matching attack anyway. The important thing is that we have identified a set of keywords that always appear together; this is sufficient for setting up the graph matching attack.

Inexact Matching.

Finally, there are some sets whose sizes are not nice multiples of k. Some sets have extra bits, and some sets appear to be missing bits. In our example, there is only one bit with a count of 129, and the set with count 99 has only 9 bits in it. It is likely that we are seeing the results of a collision in one of the PRFs. The bit with count 129 probably belongs with the bits in the set with count 99, representing a plaintext keyword that appears in 99 documents. This same bit must also go with one or more other keywords that collectively occur in 30 other documents to bring its total count up to 129.

In cases like this example, it is tempting to treat the set of 9 bits as a "good enough" match for a plaintext keyword. Then we can ignore the left-over singleton bits like the one above. But how likely is it that those 9 bits are a real word and not a false positive? To evaluate this, we look at the false positive formulas [8] for Bloom filters. The probability that a specific bit is zero after all elements are entered into the Bloom filter is:

$$p' = (1 - \frac{1}{m})^{kn} \approx e^{\frac{-kn}{m}}$$

where k is the number of hash functions, m is the size of the Bloom filter, n is the number of words added to the Bloom filter. The probability of a false positive is:

$$(1 - p')^k \approx (1 - e^{\frac{-kn}{m}})^k$$

We can modify this formula to calculate the false positive rate when we only require a match on $\ell \leq k$ bits:

$$(1 - e^{\frac{-kn}{m}})^\ell$$

Mimesis uses a Bloom filter with $m = 2^{24}, k = 10$. The Enron emails contain on average $n = 101$ unique keywords. The false positive rate with these parameters when matching on 10 bits $= 6.25 \times 10^{-43}$. The false positive rate with 9 bits $= 1.04 \times 10^{-38}$ and 8 bits $= 1.72 \times 10^{-34}$

It appears that we can safely create a node in our graph matching step whenever we find a group of ℓ bits that tend to appear together, even if ℓ is smaller than k. We present experimental results for 8 and 10 bits in Section 5.2.

Graph Matching on Extracted Tags.

After finding the sets of bits that we believe correspond to each of the top keywords in the corpus, we again use the graph matching attack to match each set of Bloom filter bits to its best-fitting plaintext keyword.

5.2 Empirical Evaluation

We evaluate our attacks on Mimesis in two parts. First, we measure the ability of the tag recovery algorithm to extract the correct set of bits from the Bloom filter for each keyword. Then, we measure the accuracy of the graph matching attack when its list of tags comes not from ground truth, but from the (possibly incorrect) set of tags extracted by the attack on MA-ESE.

We created a Bloom filter for each email in the Enron corpus. The Bloom filter parameters we used are the same parameters from [22], $m = 2^{24}$ bits and $k = 10$ hashes. HMAC-SHA-256 is the PRF with each hash in k receiving a unique key.

5.2.1 Bloom Filter Attack

An attack on the Bloom filter is successful when a set of k bits is successfully identified as a tag for the corresponding keyword. Figure 8 shows the average accuracy of our attack across all users in the Enron corpus changes as we increase the number of words targeted. With the majority of users, we were able to recover over 80% of the tags up to the top 1000 words used.

In addition to our experiments that required us to identify all 10 bits for each tag, we also ran experiments relaxing this restriction. The loosened restriction means if we can identify as few as 8 bits that have similar counts and belong to the same set of documents, those 8 bits are considering a tag. Figure 8 compares the Tag finding accuracy on the Enron corpus matching 10 bits for each Tag compared to the less restrictive matching on 8 bits. The dashed lines show the accuracy of the top 10% of the attack results from the test. The dotted lines reflect the bottom 10% of the attack results.

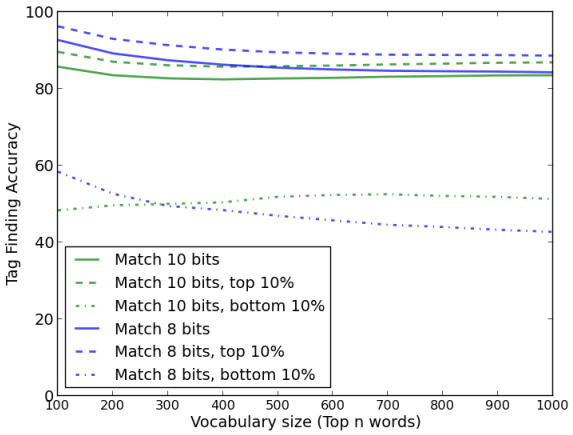

Figure 8: Tag Finding accuracy on Enron corpus Bloom filters with parameters: k=10 and Bloom filter size = 2^{24}, matching on 8 and 10 bits

5.2.2 Graph Matching Attacks

The real evidence if matching on less than k bits is effective is to compare the results from graph matching. Figure 9 shows the `graphm` accuracy of the SC-ESE tags compared to the accuracy of the MA-ESE tags matching on all 10 bits verses matching on 8 bits. As with the previous graph, the dashed lines show the accuracy of the top 10% of the

attack performances from the test. The dotted lines reflect the bottom 10% of the attack performances.

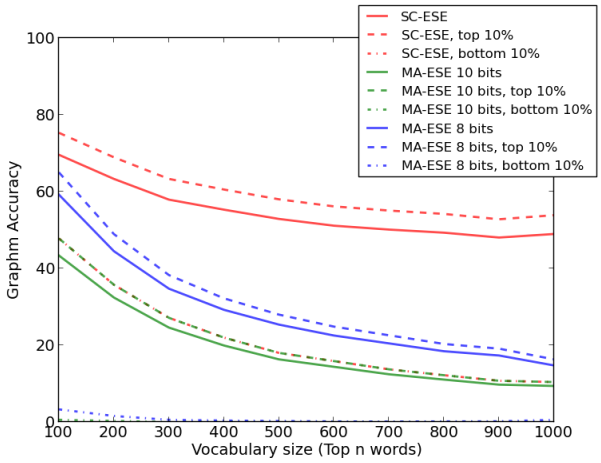

Figure 9: Accuracy of Weighted Graph Matching attack (PATH Algorithm) for MA-ESE and SC-ESE on Enron Data

The accuracy on these `graphm` attacks for the Bloom filters appears to degrade quickly between the 100 and 300 word count. Discovering an improved tag finding attack would certainly help this. We believe it also makes a significant difference which tags are not found from the Tag Finding attack. A tag that has a high rate of occurrence will have a larger effect on the `graphm` algorithm than a tag with a lower rate. Missing the tag with the highest occurrence would certainly have an increased adverse effect compared to missing the tag with the 100th highest count.

Future work might involve analyzing the relationship among the occurrence count of the tags that are not found from the graph matching and working on algorithms to maximize the attack on discovering those specific tags.

The success rate of the `graphm` attack with the tags recovered from the Bloom filter is fairly low with a vocabulary size of 1000. However even an attack that has only a 10% success rate is still much better than random guessing and reveals much to an adversary.

6. MITIGATION

The most obviously effective defense against our attacks is to use an encrypted search construction that reveals much less information to the adversary, e.g. [10]. However, for many real use cases the operational requirements only admit efficiently-searchable schemes [16]. Here we describe a novel strategy for defending EDESE schemes against inference attacks by carefully tuning the Bloom filter parameters.

When deciding which Bloom filter parameters to use for encrypted search, many previous works [14, 22, 26] discuss how the parameters affect the false positive rate. Until now, little or no attention has been paid to how the choice of parameters may affect security. Our experiments in the previous section demonstrate that the parameters used in Mimesis Aegis are susceptible to attack. Our analysis in this section reveals that the attacks are possible because the Bloom filter used in Mimesis is much larger than necessary. With careful tuning of the parameters, we can make inference attacks much more difficult.

Broder and Mitzenmacher [8] describe a technique for picking the optimal number of hash functions in a Bloom filter to minimize its false positive rate. Given a Bloom filter with m bits and a document length of n keywords, the FP rate is minimized at

$$k = ln(2 * (m/n))$$

As a side effect of this parameter choice, it happens that each bit in the Bloom filter will be set with probability 50% in each document. Intuitively, this will make it more difficult for the adversary to extract information about which keywords appear in which documents.

After removing stop words, the Enron corpus has an average of 101 unique keywords per document. Applying this formula with $m = 2^{24}$ and $n = 101$ results in an optimal value of $k = 115,139$. Clearly, using more than 100,000 hash functions is not practical. So instead we looked at modifying m, the size of the bloom filter. Applying the same formula, but instead using $k = 10$, $n = 101$ and solving for m, we get a value in between 2^{11} and 2^{12}.

To test this approach, we re-ran our tag finding attack against the Enron corpus, using a constant value of $k = 10$ hash functions, but varying the size of the Bloom filter from 2^{10} to 2^{22}. For each configuration, we computed the average accuracy of the tag finding attack and the expected false positive rate offered by the Bloom filter. The results of this experiment are shown in Figure 10. Setting the size of the Bloom filter close to the value derived from the formula above is very effective in reducing the accuracy of the attack. For $m = 2^{12}$, the attacker is unable to find the tags for even the top 100 most common keywords. Without tags, the attacker cannot even attempt the graph matching attack. Moreover, with these parameters the Bloom filter still offers a very low false positive rate of 2.5×10^{-5}. A side benefit of this approach is that it is also more space efficient. With a Bloom filter of only 2^{12} bits, each tag can be much smaller than in the default Mimesis configuration.

But as m grows large relative to $k = 10$, the success of our attack grows very quickly. With 2^{22} bits, we can find tags for more than 60% of the top 1000 keywords. At the same time, the tags must also grow to encode more bits.

Analysis.

With the naive choice of Bloom parameters, there is nearly a 1:1 correspondence between the bits in the Bloom filters and the keywords that generated those bits. The modified Bloom filter parameters weaken this relationship, and as a result, the attack is much less successful.

To make this more precise, let q be the "baseline" probability that any given bit, b, is set in any given Bloom filter. Using the equations from [8], we can compute q from the number of words per document, n, the number of hash functions, k, and the size of the Bloom filter, m:

$$q = 1 - (1 - \frac{1}{m})^{kn} \approx 1 - e^{\frac{-kn}{m}}.$$

Using the default parameters from Mimesis Aegis [22], we obtain our original q value $q_0 = 6.01\text{x}10^{-5}$.

On the other hand, if some keyword w sets bit $b = 1$, and w occurs in fraction p of the documents, then we should expect to see bit b set in about $p \cdot 1 + (1-p) \cdot q$ of the encrypted

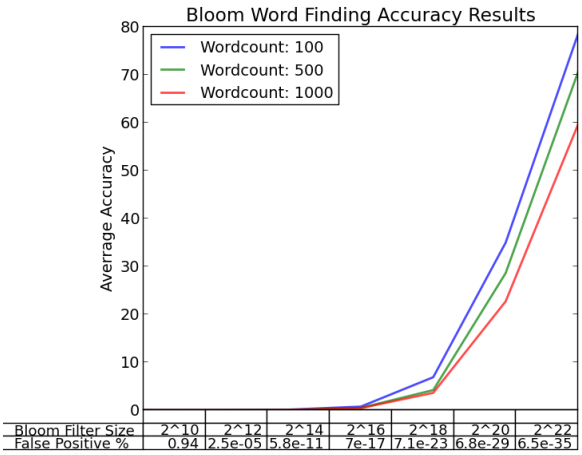

Figure 10: Tag finding accuracy on Enron corpus with variable sized Bloom filters

corpus. If the resulting frequency is differs significantly from q, then the adversary can easily tell which bits go together. Table 2 illustrates this effect. With $q = q_0$, we expect the bits for each keyword to have a unique frequency very close to the frequency, p, of the keyword itself. For example, bits for the most common keyword should appear in about 54% of the encrypted corpus, and bits for the 100^{th} most common keyword should appear in about 9.4% of the documents.

By increasing q, we can make the attack more difficult. The optimized parameters from [8] set q at about 0.5, but it is also possible to drive q even higher while maintaining a low false positive rate. For example, with $m = 2^{11}$ and $k = 25$, we get $q = 0.709$, and the probability of a false positive is less than 10^{-3}. Table 2 shows how the bit frequencies change as we increase q. With $q = 0.7$, the frequencies for all bits belonging to the top 300–1000 words will be roughly similar. This makes it increasingly likely that large numbers of bits will be grouped together in the first phase of our attack, and increasingly likely that Algorithm 1 will fail to find the unique documents it needs in order to identify the groups of bits for individual keywords.

Word Rank	p	Pr[bit $b = 1$]			
		$q = q_0$	$q = 0.5$	$q = 0.7$	$q = 0.9$
1	0.540	0.540	0.770	0.862	0.954
10	0.311	0.311	0.656	0.793	0.931
50	0.135	0.135	0.568	0.741	0.914
100	0.094	0.094	0.547	0.728	0.909
200	0.062	0.062	0.531	0.719	0.906
300	0.048	0.048	0.524	0.714	0.905
400	0.039	0.039	0.520	0.712	0.904
500	0.034	0.034	0.517	0.710	0.903
600	0.029	0.029	0.515	0.709	0.903
700	0.026	0.026	0.513	0.708	0.903
800	0.023	0.023	0.512	0.707	0.902
900	0.021	0.021	0.511	0.706	0.902
1000	0.019	0.019	0.510	0.706	0.902

Table 2: Impact of Bloom filter parameters on bit frequency; Parameters from [22] give $q_0 = 6.01\text{x}10^{-5}$.

We caution the reader that these results do not in any way constitute a proof of security, and it is possible that new attacks might still be devised against the improved Bloom filter parameters. Also, the analysis above depends on a few critical simplifying assumptions that do not necessarily hold in practice. First, the equations from [8] assume that the words in the documents are uniformly random; this is certainly not true for natural language texts. Second, our simplified analysis here assumes that all documents contain the same number of words; this is also untrue for any non-trivial text corpus. These complications make further analysis more difficult and beyond the scope of this paper. For the present time, we urge continued caution with systems that rely on Bloom filters for encrypted search.

7. CONCLUSIONS AND FUTURE WORK

We presented new inference attacks on two recent schemes for efficiently deployable, efficiently searchable encryption. Unlike earlier attacks, ours do not require special knowledge of the documents in the target encrypted corpus. Our analysis of Bloom filters in Mimesis Aegis illustrates the importance of Bloom filters on the security of the system. We believe our attack would also be effective against other searchable encryption schemes that rely on Bloom filters, such as the SADS anonymous encrypted database [26]. This validates the SADS author's decision to use different hash functions for each document in a later version of the system [25]. Similarly, Goh [14] briefly discusses the possibility of using Mimesis-style tags for efficient searchability. Our results also validate his decision to apply a second layer of protection to his Bloom filters before uploading them to the untrusted server.

Although we have shown that careful tuning of the BF parameters breaks the attacks presented here, we do not yet have a proof that this defense will be effective against *all* such attacks. In future work, we will provide a more formal analysis of the security that is possible for EDESE schemes using various defenses.

Acknowledgements

The authors thank Seny Kamara for early discussions about the (in)security of ESE schemes. We also thank the anonymous reviewers and our shepherd, Elaine Shi, for many helpful suggestions that improved the presentation of the paper.

8. REFERENCES

[1] M. AbuTaha, M. Farajallah, R. Tahboub, and M. Odeh. Survey paper: cryptography is the science of information security. *International Journal of Computer Science and Security (IJCSS)*, 5(3):298, 2011.

[2] H. A. Almohamad and S. O. Duffuaa. A Linear Programming Approach for the Weighted Graph Matching Problem. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(5):522âĂŞ525, 1993.

[3] G. Amanatidis, A. Boldyreva, and A. OâĂŹNeill. Provably-secure schemes for basic query support in outsourced databases. In *Data and Applications Security XXI*, page 14âĂŞ30. Springer Berlin Heidelberg, 2007.

[4] M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In A. Menezes, editor, *CRYPTO 2007*, volume 4622 of

LNCS, pages 535–552. Springer, Heidelberg, Aug. 2007.

[5] D. Blei, A. Ng, and M. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993âĂŞ1022, January 2003.

[6] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.

[7] T. Brekne, A. Årnes, and A. Øslebø. Anonymization of ip traffic monitoring data: Attacks on two prefix-preserving anonymization schemes and some proposed remedies. In *PETs*, pages 179–196, 2006.

[8] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. *Internet mathematics*, 1(4):485–509, 2004.

[9] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart. Leakage-abuse attacks against searchable encryption. In I. Ray, N. Li, and C. Kruegel:, editors, *ACM CCS 15*, pages 668–679. ACM Press, Oct. 2015.

[10] 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*. The Internet Society, Feb. 2014.

[11] D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. *International journal of pattern recognition and artificial intelligence*, 18(03):265–298, 2004.

[12] 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. ACM Press, Oct. / Nov. 2006.

[13] B. A. Fisch, B. Vo, F. Krell, A. Kumarasubramanian, V. Kolesnikov, T. Malkin, and S. M. Bellovin. Malicious-client security in blind seer: A scalable private DBMS. In *2015 IEEE Symposium on Security and Privacy*, pages 395–410. IEEE Computer Society Press, May 2015.

[14] E.-J. Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2003. http://eprint.iacr.org/2003/216.

[15] S. Goldwasser and S. Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, pages 365–377, New York, NY, USA, 1982. ACM.

[16] P. Grubbs. On deploying property-preserving encryption. Presented at Real World Crypto, January 2016.

[17] 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*, CCS '14, pages 1028–1039, New York, NY, USA, 2014. ACM.

[18] M. S. Islam, M. Kuzu, and M. Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *NDSS 2012*. The Internet Society, Feb. 2012.

[19] 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. ACM Press, Oct. 2012.

[20] B. Klimt and Y. Yang. The enron corpus: A new dataset for email classification research. In *Machine learning: ECML 2004*, pages 217–226. Springer, 2004.

[21] M.-S. Lacharité and K. G. Paterson. A note on the optimality of frequency analysis vs. ℓ_p-optimization. Cryptology ePrint Archive, Report 2015/1158, 2015. http://eprint.iacr.org/2015/1158.

[22] B. Lau, S. Chung, C. Song, Y. Jang, W. Lee, and A. Boldyreva. Mimesis aegis: A mimicry privacy shield–a system's approach to data privacy on public cloud. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 33–48, San Diego, CA, Aug. 2014. USENIX Association.

[23] M. Naveed, S. Kamara, and C. V. Wright. Inference attacks on property-preserving encrypted databases. In I. Ray, N. Li, and C. Kruegel:, editors, *ACM CCS 15*, pages 644–655. ACM Press, Oct. 2015.

[24] V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S. G. Choi, W. George, A. D. Keromytis, and S. Bellovin. Blind seer: A scalable private DBMS. In *2014 IEEE Symposium on Security and Privacy*, pages 359–374. IEEE Computer Society Press, May 2014.

[25] V. Pappas, M. Raykova, B. Vo, S. M. Bellovin, and T. Malkin. Private search in the real world. In *Proceedings of the 27th Annual Computer Security Applications Conference*, ACSAC '11, pages 83–92, New York, NY, USA, 2011. ACM.

[26] M. Raykova, B. Vo, S. M. Bellovin, and T. Malkin. Secure anonymous database search. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 115–126. ACM, 2009.

[27] T. Sanamrad, L. Braun, D. Kossmann, and R. Venkatesan. Randomly partitioned encryption for cloud databases. In *DBSec XXVIII*, pages 307–323. 2014.

[28] 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. IEEE Computer Society Press, May 2000.

[29] S. Umeyama. An Eigendecomposition Approach to Weighted Graph Matching Problems. *IEEE Trans. Pattern Anal. Mach. Intell.*, 10(5):695âĂŞ703, 1988.

[30] D. C. Uthus and D. W. Aha. The ubuntu chat corpus for multiparticipant chat analysis. In *AAAI Spring Symposium: Analyzing Microtext*, 2013.

[31] S. Whittaker and C. Sidner. Email overload: exploring personal information management of email. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 276–283. ACM, 1996.

[32] I. H. Witten, A. Moffat, and T. C. Bell. *Managing gigabytes: compressing and indexing documents and images*. Morgan Kaufmann, 1999.

[33] M. Zaslavskiy, F. Bach, and J.-P. Vert. A path following algorithm for the graph matching problem. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(12):2227–2242, 2009.