Shadow Nemesis: An Implementation

Alok Singh

Department of Computer Science - Rutgers University

16th December 2016

Course Project under the guidance of Prof. David Cash and Prof. Vinod Ganapathy

CS-546 Computer System Security

Department of Computer Science - Rutgers University

Abstract

The domain of Internet Communication has seen end-to-end encryption as a means to mitigate trust issues between the communicating parties. ShadowCrypt is one such tool that utilises "efficiently deployable efficiently searchable encryption" (EDESE) that allows standard full-text search techniques on encrypted data. In achieving EDESE, there is often a "efficiency-leakage" trade-off. Shadow Nemesis attempts a query-recovery attack on the leakage profile assumed by the ShadowCrypt model. This project aims at recreate the Shadow Nemesis attack on ShadowCrypt by implementation and understand the practical implications of such an inference based attack.

Introduction

Data privacy is and always has been of paramount importance in almost all kinds of Internet communication. With the advent of Cloud technologies, there is a move to harness the Cloud without having to trust the service provider or the security of their infrastructure. Searchable Encryption attempts to address this need by encrypting the outbound data with a key known only to the user while still allowing the user to search on encrypted text "efficiently".

ShadowCrypt[2] proposed a method of appending hashes of keywords to the encrypted document and store this "complete" document on the cloud. This allows legacy applications, viz. Facebook, Gmail, etc. to support Searchable Encryption without any additional requirements to their infrastructure. ShadowCrypt places itself between the user and the web-server and converts the messages by building on the Shadow DOM, a web technology that isolates elements of a web page from each other in the browser.

EDESE schemes leak information to achieve practical deployment on large datasets. The leakage profile is precisely defined and formally verified. CGPR[1] categorises the leakage profiles from most leaky - L4: Full Plaintext under deterministic word-substitution cipher, to least leaky L1: Query Revealed occurrence pattern. Because ShadowCrypt aims at legacy applications, it has a particularly high leakage profile. This is captured as the L2 leakage, where

the occurrence pattern of keywords (queries) is fully revealed to the mail server but not the order of the keywords (as in L3).

Owing to such leakage, inference attacks are possible on the scheme's construction. It is possible to learn the hidden data by observing information about the number, frequency of documents accessed per query. IKK[8] showed empirically that a user's queries can be recovered with high accuracy in a honest-but-curious threat model. These attacks are possible even when the EDESE as confirmed to be provably secure under standard assumptions.

This project attempts to reconstruct the inference attack used by [2]. The threat model used is of a honest-but-curious adversary. The adversary is given access to the ciphertext messages and search tags. The adversary also knows a set of plaintext keywords, referred to as Auxiliary data, that he uses to create his keyword universe and tries to associate a tag to a keyword in the universe using graph matching algorithms.

Background and Related Work

Inverted Index Search.

Inverted indexes allow for a full-text search based on keywords. For each keyword, an inverted index consists of all the documents that have that keyword. Common English language words like *a, the, or, in,* etc. known as "stop-words" are usually excluded from the inverted index. This lowers space requirements without losing any significant searching power as the stop-words would be present in almost every document. EDESE schemes, like ShadowCrypt, work by including tags in the indexes.

The ShadowCrypt client converts the search query sent by the user to its corresponding tag and then asks the server to search for that tag in its database of indexes. The server returns appropriate documents from the matching indexes and the ShadowCrypt client then decrypts the returned files and presents the respective plaintexts to the user.

Inference Attacks on EDESE.

As defined by [2]:

> "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."

L4 leakage profile is trivial to attack. L3 can also easily attacked with a simple inference attack given by [1]. The adversary will look at order of plaintext keywords from known documents. It will then construct keyword vocabulary, i.e it can observe all hashed values of indexed keywords in a known document. The adversary can then use this information to extract more keywords from other documents and so on. The fact that words in english follow a Zipfian power law i.e. approximately, top 80% of words are present in 20% documents, helps this attack.

IKK, Shadow-Nemesis and the Count attack [1] function on L2 leakage profiles, Shadow-Nemesis being the strongest. We re-implement this attack on the Enron email dataset [4] and find empirical results about the strength of the attack.

Isam, Kuzu and Kantarcioglu [8] uses the co-occurrence frequency for each pair of keywords in a corpus of training data and the uses this information to associate tags to plain text keywords. They use a simulated annealing approach for this, proved to be an NP-Complete optimization problem, task to recover top few hundred keywords. The IKK attack works when the adversary has the full knowledge of plaintext keywords document set.

The Count attack uses the observation that a large fraction of keywords matches a unique number of documents in addition to the co-occurrence frequency for pairs of keywords. This attack initially finds the information of tags with unique document counts. It then uses this information along-with co-occurrence frequencies to further disambiguate keywords having non-unique counts.

Shadow-Nemesis also utilises co-occurrence frequencies to convert the task to the Weighted Graph Matching problem, which is a well established NP-Complete problem with efficient solvers. They particularly use Umeyama and PATH algorithm provided by the graphm package [5].

The Shadow Nemesis Inference Attack

Naive frequency Analysis.

Given a list of keywords from the auxiliary dataset and a list of cryptographic tags, the adversary sorts both the lists as $\mathbf{w} = \{\mathbf{w}1, \ldots, \mathbf{w}n\}$ as the list of keywords and $\mathbf{t} = \{\mathbf{t}1, \ldots, \mathbf{t}n\}$ as the list of tags. The adversary now concludes that $\mathbf{w}i \Leftrightarrow \mathbf{t}i$.

Graph matching.

Given a plaintext corpus as auxiliary information and a target corpus of encrypted text and tags, the adversary first extracts the keywords from the auxiliary dataset. S/he then removes the stop-words and stems the remaining words, assuming the server also follows a similar pre-processing. The adversary now sorts the top n keywords from auxiliary corpus and tags from target corpus to get lists $\mathbf{w}$ and $\mathbf{t}$ respectively. They then create two graphs G and H to represent the auxiliary and target data respectively. For each $i, j \in [1..n]$, the adversary sets the weight of 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 they construct the graph H for co-occurrence of tags over the target data. This construction is is sufficient to reduce the attack to the Weighted Graph Matching Problem.

The adversary can further create an instance of the Labeled Graph Matching problem by creating a similarity matrix C for the nodes. LGM is a generalisation of WGM problem where the nodes as well as edges of the graphs have weights and the goal is to simultaneously minimize the difference in edge weights while maximizing the similarity of node weights. When each entry c_{ij} of the similarity matrix C gives the similarity of node i's weight in G to node j's weight in H. For the purpose of this project, though we do not implement this approach. Also, there does not seem to be corresponding data to match with in the Shadow Nemesis results.

Implementation

The code-base[3] for the Shadow Nemesis attack uses python scripts to process the Enron email corpus [4] having the emails of 150 Enron employees. The graph matching algorithms used are the Umeyama Algorithm[6] and PATH Algorithm[7] from the *graphm* package[5]. Umeyama algorithm, takes $O(n^3)$ time to compute a permutation but it gives near optimal solution when graphs are almost isomorphic. The PATH algorithm is approximately two orders of magnitude slower than Umeyama, but it is more robust and works for imperfect matchings. In the first experiment, we assume that the auxiliary data and testing data correspond to the same dataset.

Scripts and Steps.

1. *keywords.py*: Extracts the keywords from an email and writes in a corresponding file. In the process of extraction, stop-words are removed using the *nltk.corpus* library in python. We use *SnowballStemmer* to stem the words. Stemming is necessary because even though it limits an attack to reconstruct only the stems of plaintext words, e.g. *cat* instead of *cats*, stemming reduces total vocabulary size and increases repetition of terms, making the attack easier. The extracted keywords are then written to disc using the same directory and file structure as the original corpus.
2. *divide_keywords_percentage.py*: This script basically divides the whole corpus into two sets, auxiliary and target. We use 50% division. The keywords per file are randomly chosen and divided. The results are stored in two directories *keys/* and *tags/*, which still have the same directory and file structure as the original corpus.
3. *make_tags.py*: This script is run on the *tags/* directory to create tags from keys. The tags are SHA-256 HMAC representations of the plaintext tags, with a secret key as defined in the file. This step hashes the *tags/* directory. Now we have both the target as well as the auxiliary corpus. The document contents are ignored from both the corpora for faster processing. Tags created after hashing are unique per file and randomly shuffled so as to mimic L2 leakage profile.

4. *freq_dict.py*: This script sorts the document frequencies of keywords/tags and writes top 1000 words in a file corresponding to that particular user. The data stored is a json dictionary of the word, its document frequency as *df*, total frequency as *tf* and the list of documents in which it appears as *did*. This list contains the document id and the number of times that word is present in the document. The words are sorted by their document frequencies, i.e. number of document a particular words appears per user. Tags will incidentally have same *df* and *tf*. A single key frequency dictionary looks like the following:

{"asset": {"df": 20, "tf": 42, "did": {"241504": 1, "242145": 1, "242338": 2, "241811": 1, "242503": 1, "241416": 1, "241738": 4, "242763": 2, "242546": 1, "242448": 3, "242600": 3, "242514": 2, "241331": 2, "242398": 11, "242647": 1, "242266": 1, "242754": 2, "242588": 1, "241438": 1, "242047": 1}}}

5. *read_freq_and_make_graph.py*: Reads these top frequencies, and creates a *1000x1000* graph for each user. We find the probability of occurrence of two keywords as the set difference in the *did* list of the two keywords divided by the total number of files for that user. The output after this step is a *1000x1000* for each user. Doing this step twice produces large graphs *G_large* and *H_large* as binaries of the *numpy* library.
6. *get_subgraphs.py :* This step creates text version of a subset of the adjacency matrix. For this experiment we use *100x100* and *200x200* matrixes.
7. These matrixes are then sent to the *graphm* package, which gives the permutations based on the applied algorithms: Umeyama and PATH in this case.
8. *permute.py :* reads the output from the graphm package, applies the permutations on the tags to get a list $\mathbf{t'} = \{\mathbf{t'}1, .., \mathbf{t'}n\}$.

   From here we conclude that:

   $$\mathbf{t'}i \cong \mathbf{w}i$$

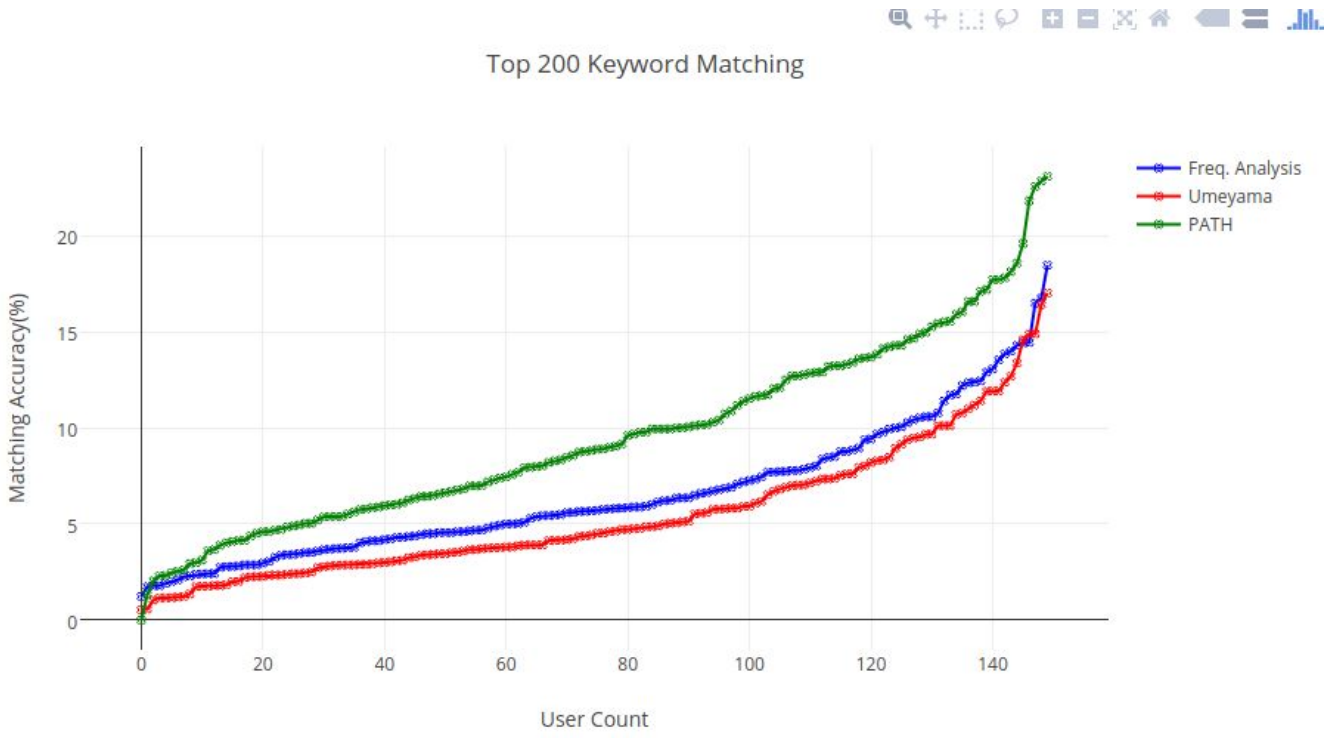
   For all $i \in [1..n]$

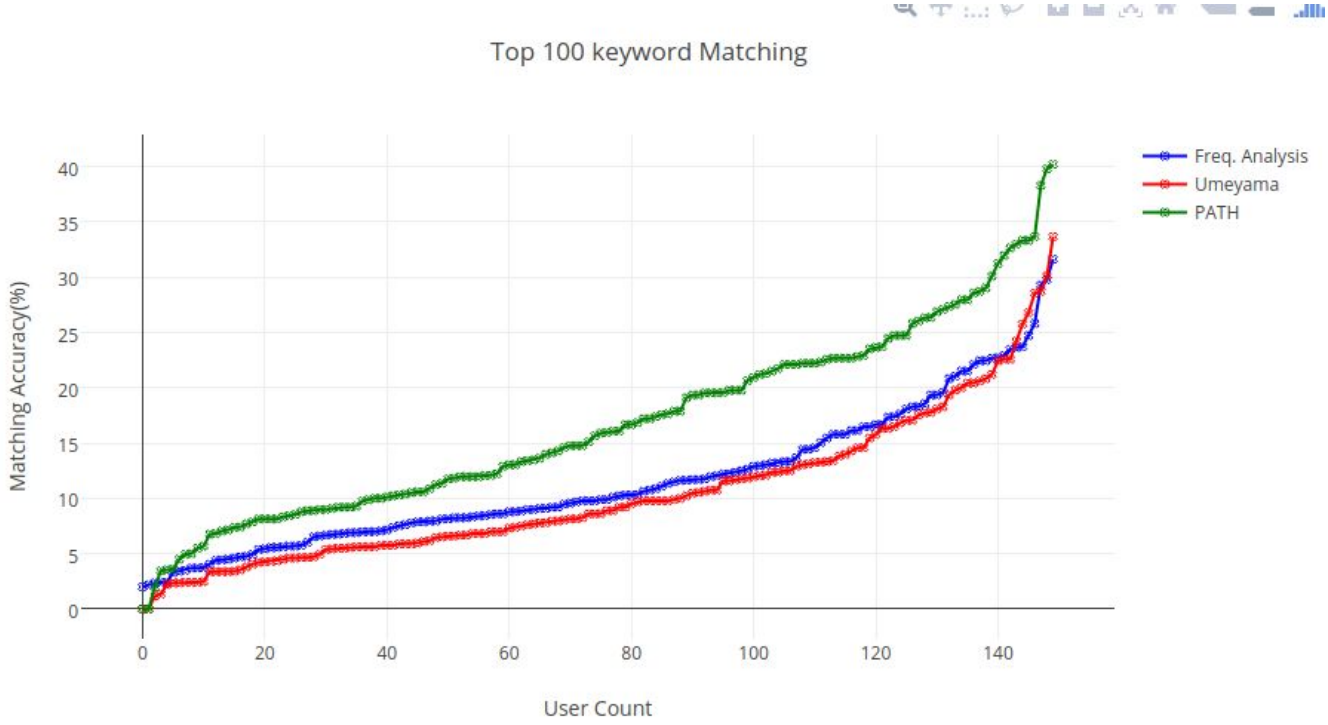
This script then generates the sorted list of percentage accurate matchings for each of *Naive Frequency Analysis*, *Umeyama Algorithm* and *PATH Algorithm*.

Results

The initial experiment with same auxiliary and target data gave 100% accuracy with all the three approaches: Simple Frequency Analysis, Umeyama and PATH algorithms. This is to be expected.

When the corpus is divided by 50% into non-overlapping sets of auxiliary data and target data, the results derived are much different and deteriorated than the ones presented by the authors of Shadow-Nemesis. A PATH Graph Matching for top 200 keywords gives nearly 17% accuracy for 10% of Enron user emails as compared to the claimed greater than 80% for half the users.





Conclusions and further work

The huge discrepancy in results might be attributed to:

1. Lack of usage of Similarity matrix and convert to an instance of LGM problem instead of WGM problem. The authors imply the use WGM instances but that may not be the case.
2. Stemming the data: The authors do not stem any keywords in their implementation, as far as mentioned in the paper. Whereas, we stem the keywords assuming it be a standard in creating inverted indexes.
3. Possibility in usage of different parameters in the graphm package: This project uses the default parameters as provided by the graphm package. Its has various parameters like the threshold of similarity, weights given to similarity matrix and adjacency matrices (which would be useful onlt in an LGM instance). The authors however, do not explicitly mention modifying any of parameters but it still remains a possibility

Further work will be needed to address this discrepancy in a conclusive manner. The attack can then, when found strong enough, be used to probe into L1 leakage profiles. That would be a major contribution as it would help determine whether EDESE as we use it is still useful to be deployed on a large scale for various organisations who have security in their best interests.

References

[1] 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

[2] D. Pouliot, C. V. Wright. The Shadow Nemesis: Inference Attacks on Efficiently Deployable, Efficiently Searchable Encryption. *Proceedings of the 2016 ACM SIGSAC Conference on CCS* .

[3] Project source code: https://github.com/alok-sin/shadow-nemesis

[4] The Enron dataset: https://www.cs.cmu.edu/~./enron/enron_mail_20150507.tgz

[5] Graphm Package: http://cbio.mines-paristech.fr/graphm/

[6] Umeyama Algorithm. S. Umeyama: An eigendecomposition approach to weighted graph Matching Problems. *IEEE Trans. Pattern Anal. Mach. Intell.*, 10(5):695aAS703, 1988

[7] M. Zaslavsky, F. Bach, and J.-P. Vert. A path following algorithm for the graph matching Problem. *IEEE Trans. Pattern Anal. Mach. Intell.,* 31(12):2227-2242, 2009

[8] M.S Islam, M. Kuzu, and M. Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *19th Annual Network and Dis. Sys. Sec. Sym., NDSS 2012.* The Internet Society, 2012