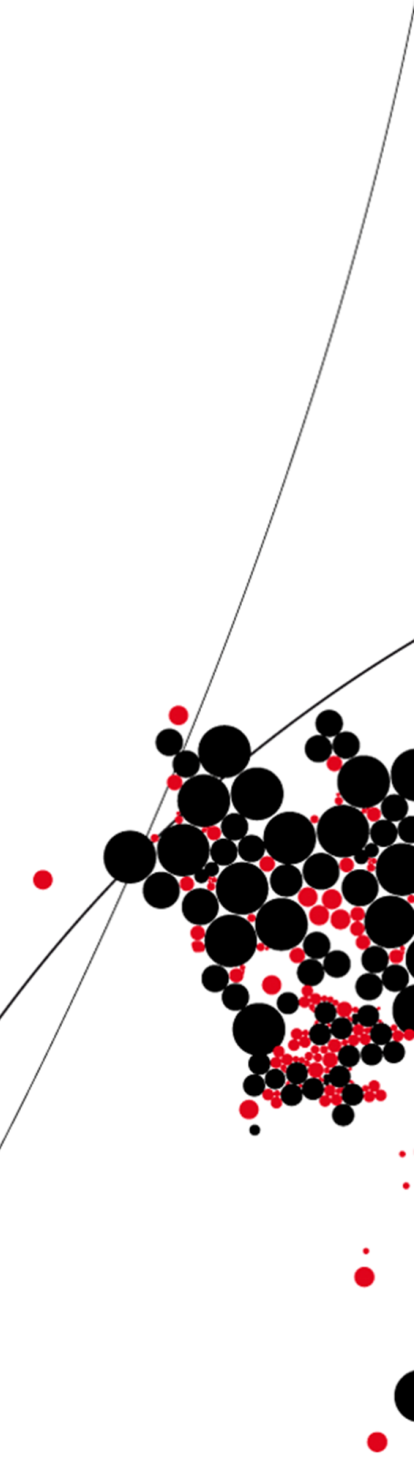
UNIVERSITY OF TWENTE.

**Faculty of Electrical Engineering,
Mathematics & Computer Science**

Thwarting File-Injection Attacks on Searchable Encryption via Client-side Detection

C. H. M. van den Bogaard

M.Sc. Thesis
December 2020



Supervisors:
Dr. A. Peter
Dr. Ing. F.W. Hahn
Dr. D. Bucur

Services and Cyber-Security
Faculty of Electrical Engineering,
Mathematics and Computer Science
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Thwarting File-Injection Attacks on Searchable Encryption via Client-side Detection

C.H.M. van den Bogaard
Faculty of Electrical Engineering, Mathematics and Computer Science
University of Twente
Enschede, The Netherlands
c.h.m.vandenbogaard@student.utwente.nl

***Abstract*—The concept of *Searchable Encryption* is a promising solution to protect the contents of outsourced data and search queries to the data from unauthorized access by Cloud Service Providers and other external adversaries. Recently *File Injection Attacks* have been proposed which can break *query privacy*. In these attacks, the adversary sends files to the client, which are then encrypted and stored. This adversary can break query privacy using these injected files. Solutions to these attacks have been proposed, such as *Forward Private Searchable Encryption*; however, these SE schemes do not completely mitigate the problem. We propose client-side detection schemes that can be applied to all Searchable Encryption Schemes and have low overhead. We show that we can reduce the attack success of three attacks to 0 and for one attack to a maximum of 0.1 while obtaining 0.99 detection accuracy for benign files. Furthermore, we show that modifications to the first three attacks do not improve the attack success, and for the last attack, we show that the adversary can obtain an attack success maximum of 0.45 under the worst-case scenario while obtaining 0.80 detection accuracy for benign files.**

***Index Terms*—searchable encryption, file injection attacks, detection, mitigation**

I. Introduction

With the rise of cloud computing and storage, multiple issues start to arise: Can we trust our private data on third-party servers? To mitigate these privacy concerns, one could encrypt its data before uploading it to the third-party services, but that raises even more issues in terms of usability. To search or edit the data, the user should download all data and decrypt it before searching or editing operation. He then would encrypt and upload the changes again. This trivial solution does not scale well when the dataset is large, and thus it is not the right solution to mitigate these privacy issues. Another solution would be to have the server decrypt the data and run the search query on the dataset. Here, the server learns the data's content, which leads to a problem if the contents need to be hidden from the server.

The concept of Searchable Encryption (SE) is a promising solution to protect outsourced data from unauthorized access by Cloud Service Providers or other external adversaries. It was first introduced by Song et al. [1]. An SE scheme allows the server to search in encrypted data without learning the plaintext data contents. For example, a user may encrypt his email, store it at the server, and later search for this email on the server without learning this email's content. Much research has been done to construct these SE schemes and improve them in search and storage complexity. However, all these SE schemes expose some information about the plaintext of the underlying plaintext, which we call *leakage*. Such leakage can be abused to learn the distribution of keywords or frequency of queries on a specific keyword.

A malicious server can exploit this leakage to learn the contents of the encrypted data (*plaintext recovery*) or the plaintext keyword of the query (*query recovery*). Recently proposed attacks show that it is possible to (partially) recover the plaintext or the query leveraging this private information [2]–[4]. The IKK attack by Islam et al. and the Count attack by Cash et al. use statistical techniques to recover the queries using the (partial) knowledge of the plaintext [2], [3]. These two attacks are passive in contrast with Zhang et al., who propose active attacks on Searchable Encryption using File Injections, which significantly outperform the former attacks [4]. It works by linking newly inserted documents with previous search queries, recovering queries with a small number of injected files. They consider both adaptive and non-adaptive attacks, where the adaptive attacks achieve higher recovery rates with less injected files.

To prevent these adaptive file injection attacks, Bost et al. came up with a new SE scheme with *Forward Privacy*. Forward Privacy ensures that newly inserted files can not be linked to previous search queries [5]. Further extending this research, multiple authors proposed other SE schemes with Forward Privacy, which perform better in search and storage complexity. The adaptive attack from Zhang et al. can not be executed on Forward Private schemes; however, the non-adaptive attack remains possible [4]. One issue with these Forward Privacy schemes is that updates are only unlinkable until a new search query is performed. After a new search query, these updates are linkable again, meaning that the adaptive file injection attacks are still possible: It would need a query replay from the user, or the adversary has to wait until the new search token is generated by the client again leading to the fact that adaptive file injections are still possible over more time. Furthermore, Forward Privacy introduces higher search and storage complexity for

Searchable Encryption schemes.

Other research also proposed several mitigations, but the adversary can easily circumvent these by slightly modifying the attack. These incomplete solutions lead to an interesting question on how to mitigate these File Injection attacks. We reason that, while other solutions are not entirely mitigating the problem, another solution should be proposed. Up to now, there has been no research on the detection of File Injection attacks. We propose client-side detection schemes that can detect the files used in these File Injection attacks and mitigate the attack by suppressing these injected files. We furthermore show that modified File Injection attacks are still not successful in evading our proposed detection methods. We first describe Searchable Encryption schemes. Furthermore, we describe the attacks proposed by Zhang et al. [4]. Based on the characteristics of the attacks, we propose new detection methods. Then, attack modifications are proposed to determine whether the adversary can evade the detection schemes.

Our contributions

- We show that our detection methods can reduce the attack performance of the Hierarchical Search Attack, Binary Search Attack, and Partial Knowledge Attack to 0 while classifying 99% of the benign files correctly.
- We show that our detection methods can reduce the attack performance of the Multi Token Attack to a maximum of 0.10
- We provide improved variants of the attacks that are tailored to evade the proposed detection methods.
- We show that the attack success of the improved versions of the Binary Search Attack, Hierarchical Search Attack, and the Partial Knowledge Attack can be reduced to 0 while classifying 95% of the benign files correctly.
- We show that the detection obtains a maximum of 0.45 under the worst-case scenario for the improved Multi Token Attack while classifying 95% of the benign files correctly.

II. Attacks on Searchable Encryption

A. Searchable Encryption

At a high level, a Searchable Encryption (SE) scheme allows a client to store encrypted documents on a server such that at a later point, the client can retrieve files containing a certain keyword or collection of keywords. Assume a set of keywords $K = \{w_0, w_1, \ldots, w_n\}$. The client computes a token t corresponding to w and secret key sk and sends t to the server. The server computes and sends back the file identifiers containing the keyword w. While this is a high-level overview of a general SE scheme and many variants exist, we will use this implementation of a Searchable Encryption scheme for this research. Table I shows an overview of the used symbols, which will be introduced in other sections.

K	Keyword universe, set of keywords
T	Keyword threshold in files
sk	Secret key used in SSE scheme
F_j	File F_j
w_i	Keyword w_i
t_i	Token t_i derived using keyword w and key sk
τ_i	Set of tokens
$f(t_i)$	Frequency of token t_i in database
$f^*(w_i)$	Estimated frequency of keyword w_i given the leaked documents
$f(t_i, t_j)$	Co-frequency of token t_i, t_j in database
$f^*(w_i, w_j)$	Estimated co-frequency of keywords w_i, w_j given the leaked documents
β	Fraction of leaked files to the adversary
α	Threshold value used in the detection method
ρ	Dummy keyword universe fraction
r	Attack success

TABLE I
Overview of used notations

B. File Injection Attacks

The goal of File Injection attacks is to break *query privacy*, i.e., the adversary tries to find the keyword w_i matching to some observed token t_i. In these File Injection attacks, the server, who acts as the adversary, sends files to the client, which are then encrypted and stored. The adversary then tries to reveal search tokens sent by the client by using these injected files' structure and contents. File Injections turn out to be easy when taking Searchable Encryption applications into account, such as a cloud-based e-mail archive. In this case, the adversary only has to send e-mails to the client to inject her malicious files.

1) Binary Search Attack: In the Binary Search Attack (BSA), the server does not require knowledge about the clients' plaintext files and recovers all the keywords being searched by the client with 100% accuracy. Assume keyword universe K. The adversary injects $log(|K|)$ files where each file consists of $\frac{1}{2}|K|$ keywords. The adversary generates a set of $\{F_1, \ldots, F_{log(|K|)}\}$ where F_i contains exactly the keywords in K where the index of these keywords i-th bit is 1. The combination of files returned in response to a search token is used to compute the keyword. A visual example using $|K| = 8$ on how the keyword is determined given a token using the malicious files is given in Figure 1.

The injected files are generated non-adaptively and are independent of a search token. The same injected files can be re-used to compute the keyword of an associated search token. A keyword universe of 50000 unique keywords would only need 16 files of 25000 keywords each. The drawback of this attack is that the adversary needs to have knowledge of the used keyword universe. A possibility to overcome this limitation would be to inject all existing keywords or use leaked or related data.

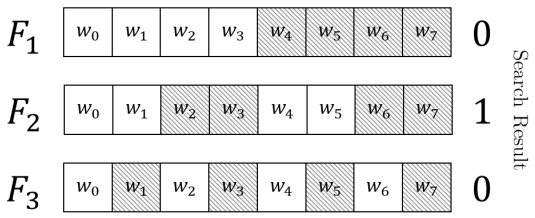


Fig. 1. Binary Search Attack with $|K| = 8$. Each file injected by the attacker contains four keywords, which are shaded in the figure. If F_2 is returned in response to some token but F_1 and F_3 are not, the keyword corresponding to that token is w_2

2) Hierarchical-Search Attack: The Hierarchical Search Attack (HSA) improves the aforementioned Binary Search Attack when some keyword threshold T is defined, where the threshold defines the maximum amount of keywords per file. It first partitions the keyword universe into $\frac{|K|}{T}$ subsets. These subsets are all injected. Then, the Binary Search Attack is executed on two adjacent subsets for all subsets. Based on the access pattern on one of these subsets, the adversary learns which subset the keyword is in. Then, the adversary can determine the exact keyword using the files generated with the Binary Search Attack with the two adjacent subsets. The number of injected files therefore is

$$\frac{|K|}{T} + \frac{1}{2} * \frac{|K|}{T} * log(2T) = \frac{|K|}{2T} * (log(2T) + 2) \quad (1)$$

The number of files can be reduced to fewer injections as in each iteration, the first file generated by the Binary Search Attack overlaps with one of the keyword universe subsets. It is further not necessary that the last subset is injected. If it is not in any of the other subsets, it must be in the last subset. Therefore, the total amount of files can be found in Equation 2.

$$\frac{|K|}{2T} * (log(2T) + 1) - 1 \quad (2)$$

The complexity of this attack in terms of files injected is significantly larger than the Binary Search Attack. Using the same keyword universe of 50000 unique keywords and a threshold of 200 keywords per file, the adversary needs to inject approximately 1200 files. The drawback of both the HSA and BSA is that the adversary needs knowledge of the keyword universe she will inject. However, the attack will always succeed if the adversary knows the keyword universe.

3) Partial Knowledge Attack: For the Partial Knowledge Attack (PKA), the knowledge of the underlying plaintext is used to attack query privacy. In this attack, the adversary uses leaked files to attack one observed token. The server first learns the frequencies of tokens in the data. Let $f(t_i)$ denote the frequency of token t_i corresponding to keyword w_i in the dataset. The adversary can further generate $f^*(w_j)$, which is the frequency of keyword w_j in the known leaked plaintext. After observing a token t_i the adversary constructs a candidate universe K' with $2T$ keywords whose estimated frequencies are closest to $f(t_i)$. The adversary then uses the Binary Search Attack with these $2T$ keywords, which results in a total of $log(2T)$ files. Zhang et al. report a 70% recovery rate if 20% of the files are leaked, and a 30% recovery rate if only 1% of the files are leaked. However, if we want to learn more tokens, we would need to inject $m * log(2T)$ files.

4) Multitoken Attack: We can further improve the attack to learn m tokens while injecting less than $m * log(2T)$ files in the Multitoken Attack. The adversary tries to recover a set τ of m tokens while minimizing the amount of injected files. We split the set of tokens τ in τ_1, τ_2 where τ_1 is the set of tokens of length n where $n < m$, by taking the n tokens with the highest observed frequency $f(t)$ for all $t \in \tau$. For each token t_i in τ_1, we compute $2T/n$ keywords and add it to the set K' with the estimated frequency $f^*(k)$ nearest to $f(t)$ such that the total amount of keywords to be injected is $2T$. We can then execute the Binary Search Attack with the $2T$ keywords and recover the n tokens in τ_1. The set of recovered tokens from τ_1 is called the ground truth G.

Using G, we can recover the other tokens in τ_2. We define the function $f(t, t')$ to be the exact observed joint frequency of tokens t, t' and $f^*(k, k')$ as the estimated joint frequency of two keywords k, k' measured from the leaked documents. To recover the tokens t' in τ_2 we add the keywords k' from the leaked documents to K such that if keyword k' corresponds to t', then $f^*(k, k')$ should be close to $f(t, t')$ for each of the pairs (t, k) in G. The closeness is defined using the parameter δ. An extended description of how this parameter is determined is enclosed in the research by Zhang et al. [4]. However, the adversary can increase δ to obtain higher attack success with the downside of generating more files needed to inject.

Using the generated keyword universe K, the Hierarchical Search Attack or the Binary Search Attack is executed. Compared to the Binary Search Attack and the Hierarchical Search Attack, we are using the leaked knowledge to compute a keyword universe instead of assuming we have knowledge of the keyword universe. Zhang et al. report a 60% recovery rate when 50% of the plaintext is leaked, and only a total of 40 files are injected when $|K| = 5000$. Furthermore, it achieves a 100% recovery rate when the complete plaintext is leaked while only injecting 17 files on average [4].

III. Detection Methods

In the field of Searchable Encryption and File Injection attacks, no client-side detection methods have been proposed so far. We propose two detection techniques targeting the four proposed File Injection attacks.

A. Security Model

Figure 2 shows an overview of the Security Model. The adversary Mallory controls the Storage Provider and thus knows

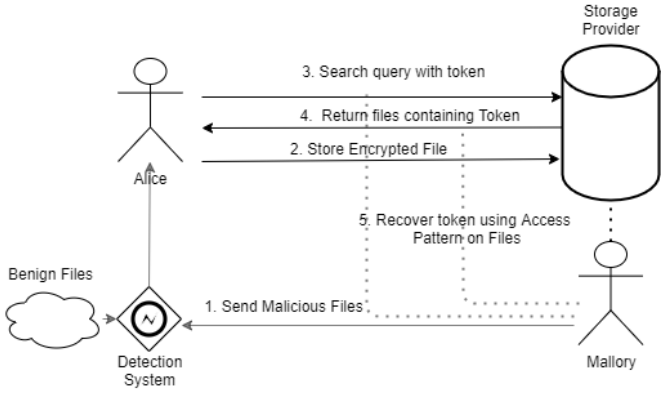


Fig. 2. Overview of Security Model where the adversary Mallory tries to break the token issued by Alice by executing the file injection attack. Alice has a detection scheme that analyses the received files

the access pattern on files. She furthermore crafts malicious files and sends this to Alice. Alice encrypts all received files and sends them to the server. We propose detection schemes on the client-side to detect these malicious files such that the client discards these malicious files, and therefore these are not stored on the external server.

B. Keyword Co-occurrence Detection (KCD)

Each of the four attacks generates the malicious files using the computed keyword universe to attack the targeted tokens. In essence, these files consist of binary combinations of the attacked keyword universe. It is highly likely that the combination of keywords within a file makes no sense to the human reader. Therefore, these seemingly arbitrary combinations of keywords in files differ from the combination of keywords in regular or benign files the client receives. We propose a detection method that utilizes the composition of keywords. It takes for every adjacent word pair the co-frequency $f(w_1, w_2)$ for adjacent keywords w_1, w_2 and averages this for the file. Let F be a file the client receives. Then, the *Keyword Co-Frequency Score (KCS)* of a received file F where $|F|$ denotes the amount of keywords in F is computed using Equation 3.

$$KCS = \frac{1}{|F|} \sum_{i=1}^{|F|} f(w_i, w_{i+1}) \tag{3}$$

The KCS value is compared to a threshold value determined by computing the KCS over a set of benign files where we take the $\alpha\%$ smallest of these KCS values and set it as the threshold. In the maliciously selected keywords, the pairs of keywords are more likely to never occur or occur with small probability; therefore, the KCS value should be low, in contrast with benign files. We computed the averaged KCS over sampled benign files, files generated with the HSA and the BSA and listed them in Table II. We observe that the files generated with the BSA are significantly smaller than the files used in the HSA.

File	Score
Benign	$1.71 * 10^{-2}$
BSA	$1.61 * 10^{-7}$
HSA	$6.96 * 10^{-3}$

TABLE II
AVERAGED KCS SCORES FOR GENERATED FILES

C. Variance Score Detection (VCD)

In the partial knowledge attack, a token t_i derived from keyword w_i and key sk is observed, and the BSA is executed with the $2T$ keywords with the closest observed frequency known to the adversary computed from the (partial) known documents. If the adversary has 100%, i.e., $\beta = 1$ knowledge of the underlying plaintext, then the keywords injected will be exact $2T$ keywords closest to the frequency of w_i. When the known document frequency is smaller, then the $2T$ injected keywords will *approximately* match with the keywords closest to the frequency of w_i. In contrast with benign files, it is not necessarily the case that all words have approximately the same frequency.

These characteristics of the attack can be used by the client to detect malicious files used in the partial knowledge attack. We argue that if all keywords $w_i \in F$ have approximately the same frequency $f(w)$, then the Variance Score (VS) given in Equation 4 is low.

$$VS = var(\{f(w)|w \in F\}) \tag{4}$$

To get an idea of these values, we computed an average VS for a malicious file and a benign file, respectively, $1.51 * 10^{-3}$ and $1.12 * 10^{-2}$.

The VS value is compared to a threshold value, determined by computing the VS over a set of benign files where we take the $\alpha\%$ smallest of these VS values computed from the benign set. The optimal α value will be determined experimentally to minimize the attack's success and maximize the detection method's accuracy.

We note that computing the frequency $f(w_i)$ and co-frequency $f(w_i, w_j)$ should be done without leaking any knowledge to the adversary. If the client would compute these values by querying the token derived from the keyword and, based on the query result, compute the frequency, the adversary learns all the tokens' access patterns in the injected file. Computing $f(w_i)$ and $f(w_i, w_j)$ without leaking the access pattern could, for example, achieved by client-side storage of a co-frequency matrix or the use of a secondary encrypted index stored on the server where the co-frequency values are updated and encrypted by the client.

IV. EXPERIMENTATION

To replicate the attack results published by Zhang et al., the data is pre-processed in the same fashion. For this ,the Enron dataset, which consists of 30109 emails, is used [6]. We use the Porters stemming algorithm, and we remove stopwords.

Zhang et al. use the top 5000 keywords of the dataset in their experiments, and we will adhere to that [4]. For validation purposes, the Amazon dataset is used [7]. This dataset is pre-processed in the same fashion. For the benign, files we will use the data from the same datasets.

To determine the impact of the detection methods, we will need to quantify the attack success. As the adversary's goal is to break query privacy, we will define the attack success as the number of recovered tokens concerning the sampled tokens. Note that the four different attacks differ in requirements and assumptions, and therefore simulations will be different in terms of quantification of attack success. As we are interested in the impact of the detection method and the evasion methods on the attack success rate, we will define the success as r, r^d, r^e where these values represent the attack success using the attack without detection, with detection, and with detection and evasion respectively.

We are also interested in the quality of the proposed detection scheme. We define a True Positive (TP) as a correctly classified malicious file and a True Negative (TN) as a correctly classified benign file. We furthermore define a False Positive (FP) as a benign file classified as malicious and a False Negative (FN) as a malicious file classified as benign. To assess the quality of detection schemes, we will use recall, which defines the fraction of correctly classified malicious files shown in Equation 5, and specificity shown in Equation 6, which defines the fraction of correctly classified benign files.

$$recall = \frac{TP}{TP + FN} \tag{5}$$

$$specificity = \frac{TN}{TN + FP} \tag{6}$$

As each of the four attacks has different assumptions to execute the attack, we will propose four different setups to gain insight into the attack's feasibility with detection schemes and evasion methods in place. For each attack, we will define the attack success metric and how it is computed.

A. Binary Search Attack

To fully prove our proposed detection's success, we will simulate the worst-case scenario for this attack, i.e., the best case for the adversary. This worst-case scenario is defined under the assumption that the adversary has full knowledge of K and has knowledge of each token t_i. Therefore, the adversary will execute the attack using the whole keyword universe. The adversary generates malicious files according to the attack specification and tries to inject them. The client will execute the detection scheme and discards all files which are classified as malicious. The adversary then tries to recover the keyword w_i matching the token t_i for all tokens. We will define r_{BSA} as the fraction of successfully recovered keywords w from K.

B. Hierarchical Search Attack

We will use the same strategy as is done in the Binary Search Attack to report on the impact of the Hierarchical Search Attack's detection. Zhang et al. propose to use $T = 200$ [4].

C. Partial Knowledge Attack

The adversary uses leaked knowledge to attack the token t_i to find w_i. To determine the proposed detection methods' impact, we will sample 100 tokens uniformly at random and mount the attack 100 times. r_{PKA} is the fraction of correctly recovered tokens. We determine a threshold value of α to minimize the attack success and minimize the amount of incorrectly classified benign files. The Partial Knowledge Attack success depends on both the fraction of leaked knowledge β and the applied detection scheme's performance, instead of the Binary Search Attack and Hierarchical Search Attack, where the detection scheme only influences the attack success. We will vary the fraction of leaked knowledge β, and the used leaked knowledge is sampled uniformly at random from the dataset.

D. Multitoken Attack

In the Multitoken Attack, the adversary attacks a set of tokens while minimizing the number of files injected using leaked files. To simulate the attack, we will use the same parameters as is done by Zhang et al. We sample 100 tokens uniformly at random and set $n = 10$. We will define r_{MTA}, r^d_{MTA} as the amount of correctly recovered tokens out of the attacked 100, without and with detection applied. We will experimentally determine α and vary the fraction of leaked files β.

V. Results

A. Binary Search Attack

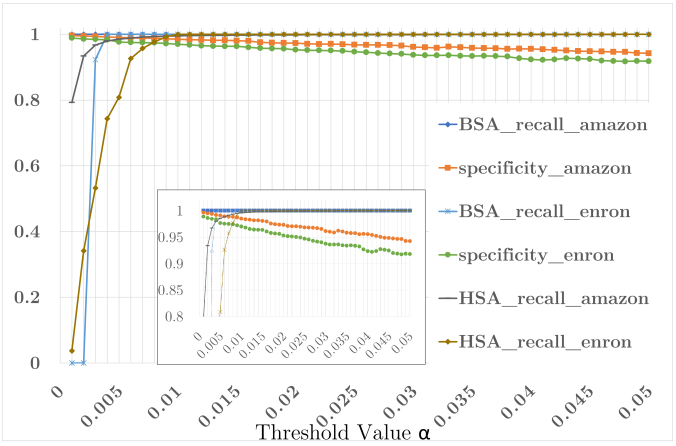


Fig. 3. Recall and specificity for different values of α for the BSA and HSA. The subgraph shows a zoom-in

The KCD detection results on the Binary Search Attack can be found in Figure 3. We observe that this detection works in detecting malicious files. We see that for $\alpha \geq 0.004$ the

$recall = 1.00$ and therefore the $r^d_{BSA} = 0.00$. For values $\alpha < 0.004$, the recall proliferates from 0 to 1 for the Enron dataset. Furthermore, the specificity for $\alpha = 0.004$ is 0.994 and 0.984 for the Amazon and Enron dataset respectively. This means that for this attack, we can mitigate it entirely while still correctly classifying at least $99.4\%, 98.4\%$ of the benign files for the Amazon and Enron dataset, respectively, if $\alpha \geq 0.004$.

B. Hierarchical Search Attack

The KCD detection results on the Hierarchical Search Attack can be found in Figure 3. This detection's specificity with the appropriate threshold value does not change when using another attack as the benign files do not change. We see that $recall \geq 0.99$ when $\alpha \geq 0.01$ where for this α the specificity is 0.986 and 0.983 for Amazon and Enron, respectively. Therefore we see that the α value needs to be slightly higher in contrast with the Binary Search Attack to reduce r^d_{HSA} to 0, i.e., $recall \approx 1$. Conclusively, we can mitigate this attack entirely while still correctly classifying at least 98.6% and 98.3% of the benign files for the Amazon and Enron dataset.

C. Partial Knowledge Attack

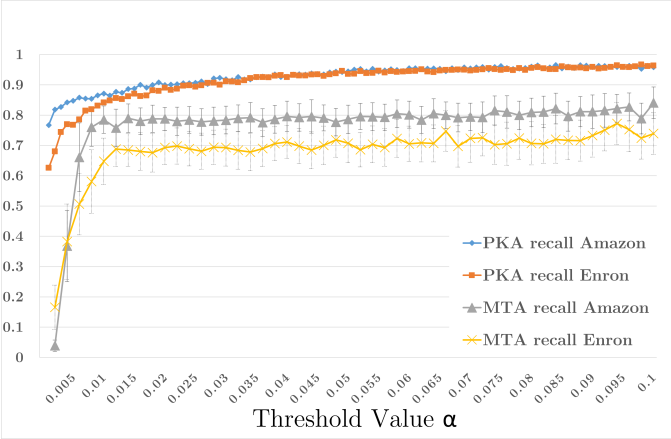


Fig. 4. Detection results for KCD detection where PKA and MTA are used for both the Enron and Amazon datasets

For the Partial Knowledge Attack, the attack success is dependent on the fraction of file leakage β and the performance of the used detection method. We note that for different values of β, the detection method's recall remains the same. Therefore it is averaged over all values of β. Figure 4 shows the recall for different values α. For this detection performance with $\alpha = 0.01$ we were able to reduce the attack success r^d_{PKA} when $\beta = 1.00$ from 1.00 to 0.104 for Amazon and 1.00 to 0.168 for Enron. For higher values of α, the attack success is reduced even further. To achieve $r^d_{PKA} < 0.01$, we did not find a suitable threshold, but it should be larger than $\alpha = 0.2$, which was the upper bound in the experiments. When computing the attack success with detection for $\alpha = 0.2$ we find r^d_{PKA} of 0.017 and 0.014 where the average over the different values for β is used for the Amazon and Enron dataset, respectively. Using this α value comes at the cost of decreasing the specificity to 0.82 and 0.80. For $\alpha = 0.05$, the attack success is reduced to 0.056 and 0.066 while still correctly classifying 95.3% and 93.4% benign files for Amazon and Enron, respectively.

We see that in comparison with the BSA and HSA, the used threshold value, α, needs to be set higher such that we can obtain the same recall scores, i.e., $recall \approx 1$, and thus low attack success. For example, the HSA has $recall = 1.00$ when $\alpha \geq 0.01$ whereas the PKA has $recall = 0.87$ and $recall = 0.84$ when $\alpha = 0.01$ for Amazon and Enron, respectively. For example, the difference in specificity for the HSA and PKA is 98.6% and 95.4% for the Amazon dataset.

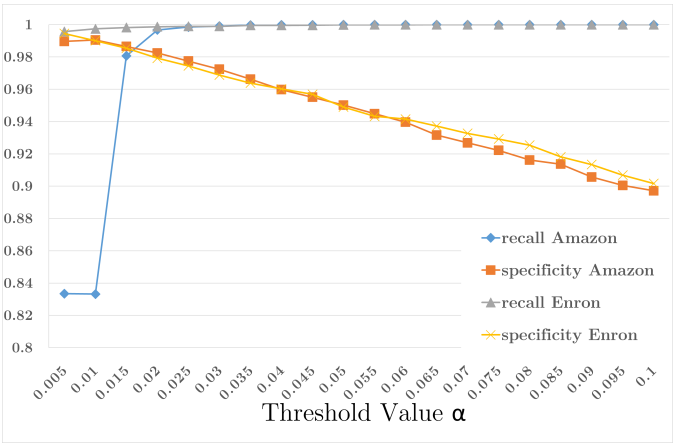


Fig. 5. Detection results for VCD detection where we compute the recall and specificity based on a threshold value α for both the Enron and Amazon datasets

The performance of VCD can be found in Figure 5. We observe a steep increase in recall starting from $\alpha = 0.01$, and the optimum threshold value is determined at $\alpha = 0.02$ with a recall value of 0.999 and 0.997 and specificity 0.979 and 0.982 for the Amazon and Enron dataset, respectively.

This method outperforms KCD for this attack. We can mitigate PKA entirely while still correctly classifying at least 97.9% and 98.2% of the benign files for the Amazon and Enron dataset. The downside of this detection method is that it only applies to the PKA, while KCD is applicable for all four attacks.

D. Multitoken Attack

Figure 4 shows the detection results for KCD for the Multitoken Attack. Inspection on these lines show high errors in the measurements; however, we can safely deduce that the recall for $\alpha > 0.02$ is more than 0.77 and 0.69 for Amazon and Enron, respectively. For $\alpha = 0.02$ we measured a specificity of 0.979 and 0.982 as described in Section V-C. Figure 6 shows the attack success r_{MTA} and r^d_{MTA} for the leaked knowledge

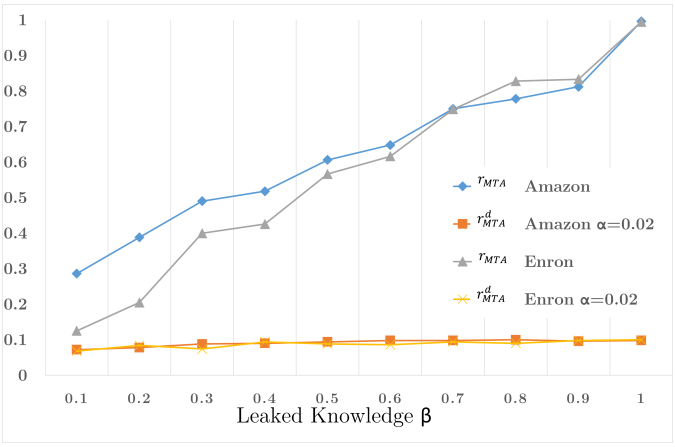


Fig. 6. Attack success of MTA for Amazon and Enron dataset with KCD applied with $\alpha = 0.02$ and without detection scheme

parameter β. The attack success is reduced to roughly 0.10 for every value of β. This 10% is due to the first part of the attack where the ground truth G is determined, and these files are not detected in KCD. We can not conclude whether a higher value for α increases the recall. Increasing α comes at the cost of decreasing the specificity. While these recall values are not as promising as is determined in the other three attacks, this detection performance should significantly impact the attack success r^d_{MTA}. The attack success parameter r is dependent on the leaked knowledge β and the performance of the used detection method. However, we can conclude we can significantly reduce the MTA to 0.10 while still correctly classifying at least 97.9% and 98.2% of the benign files for the Amazon and Enron dataset.

VI. Improved Attacks

The adversary's goal is to inject the malicious files, while the goal of the client is to detect these files before they are encrypted and injected. Depending on the detection technique, the file injection attack might succeed or not. We now assume the adversary is aware of which detection scheme is present. We will propose several modifications for the original attacks, which try to evade the detection schemes to increase the attack success.

A. Keyword Threshold Change Evasion (KTCE)

As the previous evasion techniques require partial knowledge, there are no evasion techniques applicable to the HSA. We propose an evasion technique to reduce the decrease in attack success rate by decreasing the file sizes and their initial created $\frac{|K|}{T}$ subsets. To execute this evasion method, the adversary needs to change the Keyword Threshold parameter T_{adv} to be smaller than T set by the client in the setup of the Searchable Encryption scheme. The downside of this evasion technique is the increase in files needed to inject.

B. Optimal Keyword Co-occurrence Matching Evasion (OKCME)

KCD works by computing the probability of two adjacent keywords appearing together in the known corpus. In the case of malicious files, this is less likely to happen, and therefore this can be used to distinguish benign and malicious files. Furthermore, the HSA subdivides the keyword universe into smaller keyword universes, and the BSA is executed on these smaller universes. The adversary should aim to generate word pairs such that the KCS is maximized for the whole keyword universe, and therefore, the likelihood of having a higher KCS for the individual malicious files is higher.

To obtain the optimal keyword co-occurrence matching, the adversary wants to generate the keyword universe K such that KCS is maximized using the leaked knowledge to the adversary.

$$max(\frac{1}{|K|}\sum_{i=1}^{|K|} f^*(w_i, w_{i+1})) \tag{7}$$

To achieve this she generates a complete Graph $G(V, E)$ where the vertices represent the keywords in K following the mapping $y(v_i) = w_i$ and the edges between vertices (v_i, v_j) are edges with weight $f^*(y(w_i), y(w_j))$. She wants to achieve the maximum weight matching of two vertices where each vertex can only be present in one pair of two vertices. To achieve this, the Blossom algorithm by Edmonds et al. [8] is used. The algorithm runs in high computation time derived in Equation 8.

$$O(|E||V|^2) = O(|\frac{|K| * |K| - 1}{2}|K|^2) \subseteq O(|K|^4) \tag{8}$$

The algorithm's result is a set of word pairs such that Equation 7 is satisfied. This altered keyword universe is then used in the attacks.

C. Keyword Co-occurrence Dummy Keyword Evasion (KCDKE)

To evade the KCD, the adversary can insert dummy keywords in the keyword universe to increase the KCS, with the downside of increasing the number of malicious files needed to inject. We define a set of dummy keywords K_d and fraction ρ such that $|K_d| = \rho * |K|$. Then, for each keyword w_i in K, the adversary computes the co-occurrence of every other keyword w_j in the leaked knowledge, and takes the $|K_d|$ keyword pairs (w_i, w_j) such that she obtains the maximum increase of the KCS given K and ρ and she adds the keywords with the highest KCS to the keyword universe K. This keyword universe is then used in the attack. In this evasion method, there is a trade-off between an increase in attack success, given a detection method, and an increase in malicious files needed to inject. For the PKA, these keywords are not added but substituted with the $\rho * |K|$ keywords in the attacked keyword universe K with the keywords with the frequency $f(t)$ not closest to the attacked token t, as the attack uses the BSA with $2 * T$ keywords which is the maximum when some keyword threshold T is set.

D. Variance Score Dummy Keyword Evasion (VSDEE)

As is described in the previous section for OKCME for the PKA, the adversary substitutes dummy keywords with potential keywords to evade the variance score detection method. She achieves this by increasing the variance of the injected files. We define a set of dummy keywords K_d and fraction ρ such that $|K_d| = \rho * |K|$. The adversary substitutes the $|K_d|$ keywords with the highest difference in frequency $f^*(w) - f(t)$ for each keyword w with the keywords in K that have the lowest probability of matching with attacked token t. In this case, there is a trade-off between removing potential candidates, thus lowering the attack success and evading the VCD.

VII. Improved Attack Results

We will apply these improved attacks and report on the impact of the results.

A. Hierarchical Search Attack

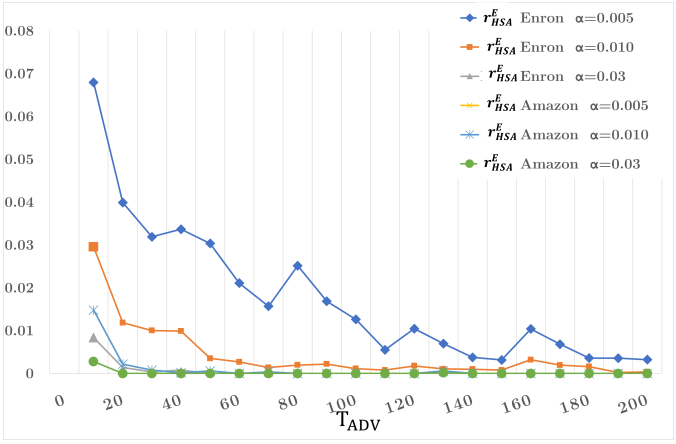


Fig. 7. KTCE technique applied to KCD for HSA with multiple values for α

We will use the KTCE with different values for T to observe the impact of the detection and evasion on r^d_{HSA}, as r^e_{HSA}. Again, r_{HSA} in this attack is always 1.0 as if the keyword matching with some token is in the attacked Keyword Universe K it always is recovered correctly. After modifying the attack with the KTCE technique, we observe the impact on the recall of the used detection method with the used threshold value of α. We see that for $\alpha = 0.01$ the recall decreases to 0.959 and 0.91 and thus resulting in 0.015 and 0.041 attack rate for Amazon and Enron dataset respectively when $T_{adv} = 10$. The downside is that the amount of files that need to be injected is 1750, in contrast with 141 when $T_{adv} = 200$. Figure 7 shows the attack success for different values of T_{adv} with different values for α. We see that when T_{adv} is smaller, we achieve higher success rates, but this comes at the cost of more files needed to be injected. We can achieve $r^e_{HSA} < 0.01$ for both the datasets with $\alpha \geq 0.03$. We can safely conclude that KCD completely mitigates the BSA and HSA even when KTCE is applied while still achieving 0.986 and 0.983 specificity for Amazon and Enron.

B. Partial Knowledge Attack

We will modify the PKA with OKCME, KCDKE, and VSDEE techniques to evade the KCD and VCD schemes and report on the impact on the attack success r^e_{PKA}

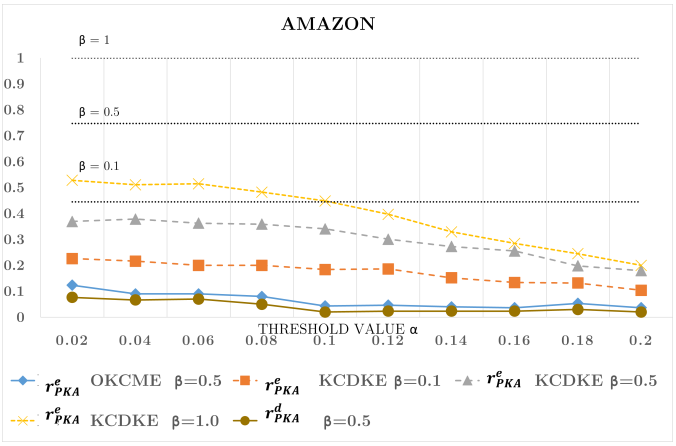


Fig. 8. Attack success for PKA with KCD applied for $\beta = 0.1, \beta = 0.5, \beta = 1$, KCDKE applied with $\rho = 0.5$ and OKCME applied. The dotted lines represent r_{PKA} given β for Amazon Dataset

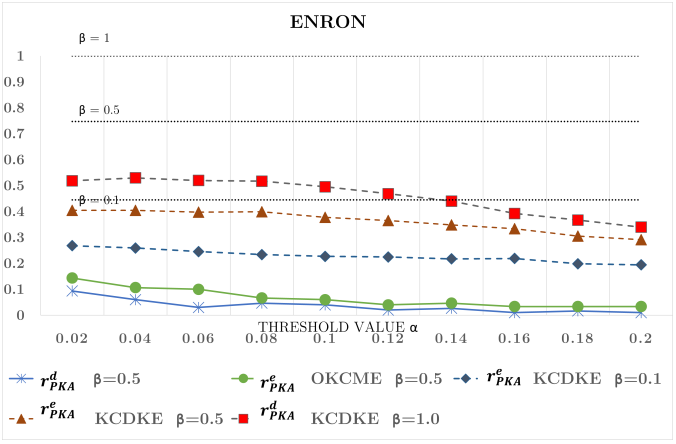


Fig. 9. Attack success for PKA with KCD applied for $\beta = 0.1, \beta = 0.5, \beta = 1$, KCDKE applied with $\rho = 0.5$ and OKCME applied. The dotted lines represent r_{PKA} given β for Enron Dataset

1) KCD: We will apply the KCD detection on the PKA and modify the PKA with OKCME and KCDKE. Figure 8 and Figure 9 show the attack success without detection, with KCD using the threshold $\alpha = 0.05$ and with the OKCME and KCDKE modifications applied for the Amazon dataset and the Enron dataset, respectively. For KCDKE, we have chosen to use $\rho = 0.5$ as this gives the highest increase in r^e_{PKA}. The dotted lines represent the attack success without detection schemes in place for $\beta = 0.1, \beta = 0.5$ and $\beta = 1.0$. r^d_{PKA} is reduced to a maximum of 0.10 for $\alpha = 0.02$ and $\beta =$

0.5. After applying the improved attacks, we find an increase in attack success. We find that OKCME only has a minimal increase of r^e_{PKA} for all threshold values. It only increases r^e_{PKA} with a maximum 0.02. For KCDKE, see that for $\alpha = 0.02$ and $\beta = 0.5$, the adversary can achieve success rates of 0.36 and 0.40 for Amazon and Enron. We can conclude that using KCDKE for PKA benefits the adversary in evading KCD; however, the adversary can not completely evade this detection scheme. Increasing α helps in reducing r^e_{PKA}, for $\beta = 0.5$ the client is able to reduce r^e_{PKA} from 0.36 and 0.40 to 0.15 and 0.30 by increasing $\alpha = 0.02$ to $\alpha = 0.20$. This comes at the cost of decreasing the specificity to 0.81 versus 0.98. Conclusively, the adversary can partially evade the detection scheme by using $\rho = 0.5$.

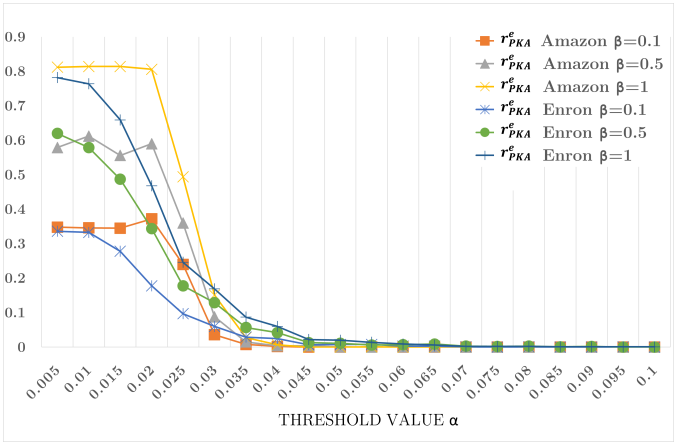


Fig. 10. Attack success for PKA with VCD applied for $\beta = 0.1, \beta = 0.5, \beta = 1$ and VSDEE applied with $\rho = 0.2$. The dotted lines represent r_{PKA} given β.

2) VCD: In Section V-C we have shown that for $\alpha > 0.02$ we obtain a recall of 0.999 and 0.997 for Amazon and Enron respectively using VCD. This means that r^d_{PKA} is reduced to 0. Figure 10 shows the results after modifying the PKA with VSDEE for different α values. The experiments have shown that $\rho = 0.2$ maximizes r^e_{PKA}. Therefore, other ρ values are omitted. Furthermore, r^d_{PKA} is as this is 0 for $\alpha > 0.02$. Figure 10 shows the attack success r_{pka} without detection and r^d_{pka} with VCD applied. Furthermore, VSDEE is applied. We observe that for the initial threshold value of $\alpha = 0.02$, this detection scheme is evaded when the optimal fraction value $\rho = 0.2$ from the adversary perspective is chosen. However, when using a higher α value, this attack modification has no significant impact on the attack success and is reduced to $r^e_{pka} < 0.02$. Therefore, we can conclude that the VCD works in mitigating the attack even with the VSDEE modification applied while keeping a specificity of 0.950 and 0.949 for Amazon and Enron.

C. Multitoken Attack

The MTA is modified with OKCME and KCDKE to evade the KCD. We use $\rho = 0.5$ as this gives the best results from the adversary perspective for the values $0 < \rho \leq 1$. For the

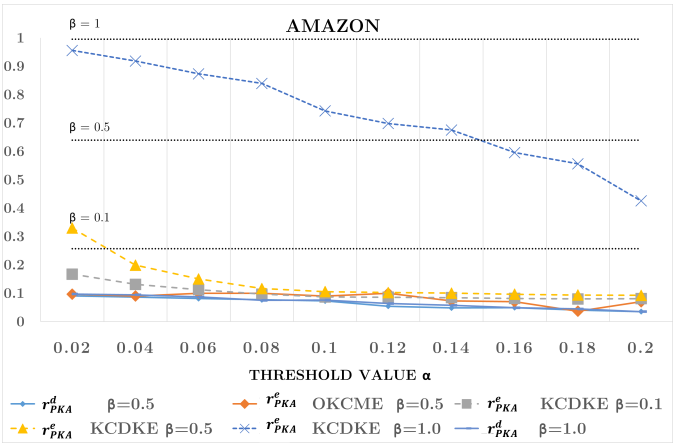


Fig. 11. Attack success for MTA without detection for the Amazon dataset, with KCD applied and OKCME and KCDKE applied with $\rho = 0.5$. Straight dotted lines represent r_{MTA} when no detection method is applied

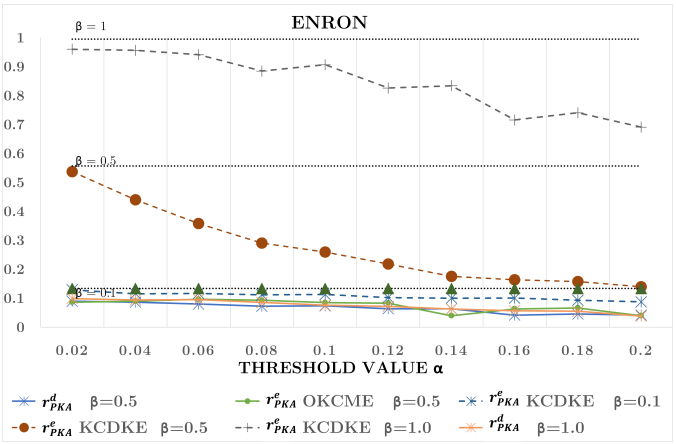


Fig. 12. Attack success for MTA without detection for the Enron dataset, with KCD applied and OKCME and KCDKE applied with $\rho = 0.5$. Straight dotted lines represent r_{MTA} when no detection method is applied

OKCME modification, we see that this has not much impact on the attack success. Closer inspection shows that OKCME increases the attack success by 0.015 on average. KCDKE has a significant impact, and r^e_{MTA} is, in all cases, larger than r^d_{MTA}. For the Amazon dataset for $\beta = 0.5$ and $\alpha = 0.02$ we see that applying KCDKE increases r^d_{MTA} from 0.091 to 0.33. For $\beta = 1.0$ this increases from 0.097 to 0.956 which is nearly the same as r_{MTA}. Increasing α for $\beta = 1.0$ does decrease r^e_{MTA} to 0.42 but comes at the cost of decreasing the specificity from 0.98 to 0.81. We note that r^e_{MTA} is high for $\beta = 1.0$ as closer inspection on $\beta = 0.1$ and $\beta = 0.5$ shows that from $\alpha > 0.1$ r^e_{MTA} is roughly equal to r^d_{MTA} for the Amazon dataset. This is not the case for the Enron dataset, and the threshold needs to be set to $\alpha = 0.2$ to reduce to approximately r^d_{MTA}. The downside of using OKCME with $\rho = 0.5$ is that the amount of malicious files is doubled on average. The amount of files is doubled instead of the smaller increase of 50% due to the generation of the Keyword Universe after selecting the ground truth G. If the amount of

correctly recovered tokens in the ground truth is smaller due to the detection method's impact, then the keyword universe injected in the second stage of the attack is larger.

VIII. Conclusion

Our paper shows that file-injection attacks can be successfully mitigated using the KCD or VCD on the client-side. For the four attacks proposed by Zhang et al. [4], we managed to reduce the attack success to 0.0 for the HSA and BSA, therefore completely mitigating the attacks while keeping at least 0.98 specificity. For the PKA, we managed to mitigate the attack to a maximum of 0.02 while keeping at least 0.95 specificity even with attack modifications in place by using VCD. We managed to reduce the attack success to a maximum of 0.10 while keeping at least 0.98 specificity for the MTA. However, when modifying the attack, the adversary can increase attack success. Under the assumption that the adversary has full knowledge, i.e., $\beta = 1.0$, we could only reduce the attack success to 0.38 and 0.70 for the Amazon and Enron dataset, respectively. When the adversary has less knowledge, we can reduce the modified MTA's attack success to a maximum of 0.11 and 0.22 for $\beta = 0.1$ and $\beta = 0.5$ while keeping 0.90 specificity. The downside of the attack modification from the adversary perspective is that the amount of files that need to be injected is approximately doubled.

IX. Related work

The first attacks on Searchable Encryption were proposed by Islam et Al. and Cash et al. The IKK attack by Islam et al. and the Count attack by Cash et al. use statistical techniques to recover queries using the (partial) knowledge of the plaintext [3] [2]. Zhang et al. further improved the attacks on Searchable Encryption using File Injection attacks, which out-performs the former attacks significantly [4].

Zhang et al. proposed several possible countermeasures to thwart these File Injection Attacks [4]. The injected files by Zhang et al. will look like semantically incorrect documents with mostly unrelated keywords. The client can easily filter these documents by closely inspecting the injected files. However, the attacker can make these injected files look semantically correct by adding keywords. Uchimoto et al. propose a method of generating semantically correct text from a given set of keywords based on n-grams from leaked files [9]. Adding keywords will result in extra overhead for the file injection, but it does not make it impossible. This possibility, however, is not experimentally proven for detection in File Injection attacks. This approach differs from our approach as we detect the already stemmed and pre-processed files and only use the combination of keywords in the keyword universe.

Zhang et al. further propose Batching Updates [4]. Initially, the assumption is made that the server knows whether its File is injected or not and which file maps to which encrypted File: She injects it and finds an encrypted file uploaded by the server. This is a reasonable assumption, but this can be circumvented by applying batching updates. In this method, the client waits for a batch of B documents before inserting them into the server, meaning that if the server injects one File before it is batched, the injected File is one out of B possibilities. The server can circumvent this by injecting B files before other files arrive or inject the same File multiple times to determine the injected File. Again this does not prevent the attack but generates more overhead for the attacker. This countermeasure is not experimentally tested and is left for further research.

Bost et al. proposed Forward Privacy to thwart adaptive File Injection attacks [5]. Forward Private schemes ensure that the server does not learn that the updated document matches a previously queried keyword. Forward Privacy does not completely mitigate the attack as *passive* File Injection Attacks such as the Binary Search Attack and the Hierarchical Search Attack remain possible. Furthermore, *adaptive* File Injection attacks remain possible if the client generates a new search token. Therefore, Forward Privacy does not necessarily imply total mitigation of the File Injection attack by Zhang et al. It makes it harder or more time-consuming as the adversary needs to receive an updated search token.

The file injection attack by Zhang mainly relies on the leakage of the access pattern. The only known solutions to hide the access pattern are SE schemes based on Oblivious ram (ORAM). ORAM schemes induce large bandwidth overhead and massive storage complexity. Naveed et al. show that the use of ORAM for SSE is unrealistic [10]. They show that due to the massive storage complexity and the communication performance, this approach worse than the trivial approach of streaming all of the outsourced data for each query. These limitations emphasize the need for mitigation for the File Injection Attacks, which have low complexity and bandwidth overhead.

A method of mitigating file injection attack on SE schemes was proposed by Liu et al. [11]. They propose a method where a new file is generated and sent to the server alongside the possibly maliciously injected File each time the client receives a file. This self-injected file is generated by taking the same amount of keywords with no intersection with the received File and, when encrypted, has the same size. The chosen keywords combined have no semantic correctness, and thus the client can see which files were self-injected. The server sees two injected files of the same size and can not distinguish the maliciously injected File, and therefore it breaks the access pattern. However, we reason that the adversary can inject the same File twice and thus bypass this countermeasure, as the access pattern of a search token on those same malicious injected files would always be consistent, but its access pattern over those self-injected files would be various. This can be suppressed by ignoring repeated emails and only adding unique emails to the index. However, the adversary can change one or more keywords

and still distinguish his malicious File from the self-injected files.

REFERENCES

[1] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proceedings of the 2000th IEEE Symposium on Security and Privacy. S&P 2000*. IEEE, 2000, pp. 44–55.

[2] M. S. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation," in *NDSS*, 2012.

[3] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-abuse attacks against searchable encryption," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. ACM, 2015, pp. 668–679.

[4] Y. Zhang, J. Katz, and C. Papamanthou, "All your queries are belong to us: The power of file-injection attacks on searchable encryption," in *Proceedings of the 25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 707–720.

[5] R. Bost, " $o\varphi o\varsigma$: Forward secure searchable encryption," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 1143–1154.

[6] Enron dataset. https://www.cs.cmu.edu/enron/.

[7] Amazon industrial and scientific review dataset.

[8] J. Edmonds, "Paths, trees, and flowers," *Canadian Journal of Mathematics*, vol. 17, p. 449–467, 1965.

[9] K. Uchimoto, H. Isahara, and S. Sekine, "Text generation from keywords," in *Proceedings of the 19th international conference on Computational linguistics-Volume 1*. Association for Computational Linguistics, 2002, pp. 1–7.

[10] M. Naveed, "The fallacy of composition of oblivious ram and searchable encryption," *IACR Cryptol. ePrint Arch.*, vol. 2015, p. 668, 2015.

[11] H. Liu, B. Wang, N. Niu, S. Wilson, and X. Wei, "Vaccine:: Obfuscating access pattern against file-injection attacks," in *2019 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2019, pp. 1–9.